Noakhali Science & Technology University
Department of Computer Science & Telecommunication Engineering

Lab Report On:

Class modeling, Project planning, and Risk analysis

Course Code: CSTE 3210

Submitted by:

Group No. : 23

Mohammad Borhan Uddin
ID: ASH2101008M

Mohammad Billal Hossain
ID: MUH2101028M

Submitted to:

Dr. Nazia Majadi
Associate Professor
Department of Computer Science & Telecommunication Engineering

Date : May 18, 2025

Contents

1 Requirement analysis

This section focuses on analyzing the system requirements using various modeling techniques to understand and document how the system will function and interact with users and data.

1.1 Flow Models

Flow models use Data Flow Diagrams (DFDs) to represent the flow of data within the system, illustrating processes, data stores, external entities, and data flows.

(i) Level 0 DFD
A Level 0 DFD, also known as a context diagram, provides a high-level overview of the entire system as a single process interacting with external entities (e.g., users, admins, employees).

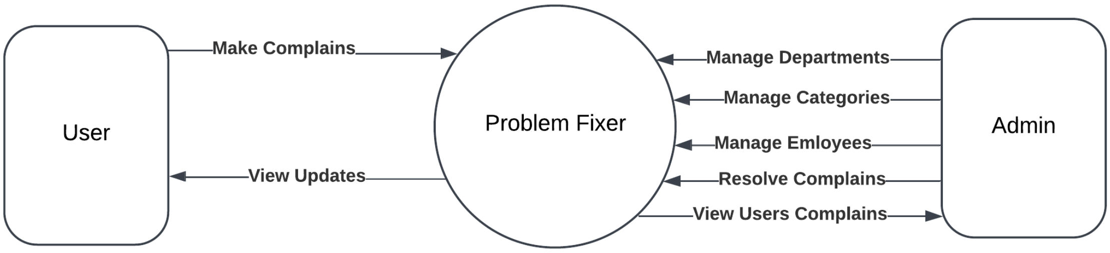


Figure 1: Level 0 DFD

(ii) Level 1 DFD
A Level 1 DFD decomposes the single process from the Level 0 DFD into major subprocesses, showing how data flows between them.

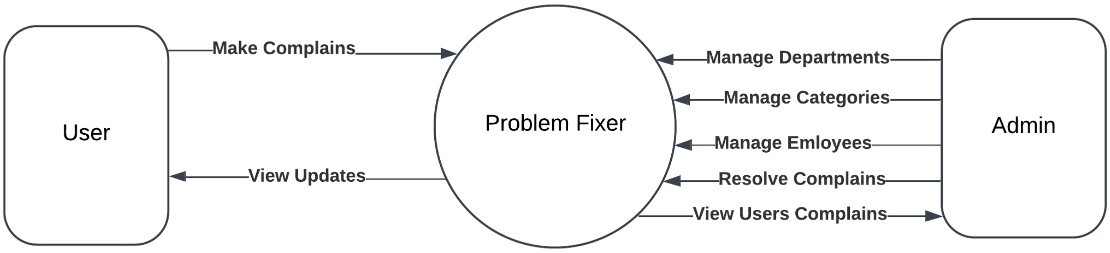


Figure 2: Level 1 DFD

(iii) Level 2 DFD
A Level 2 DFD further decomposes one or more processes from the Level 1 DFD into finer subprocesses.

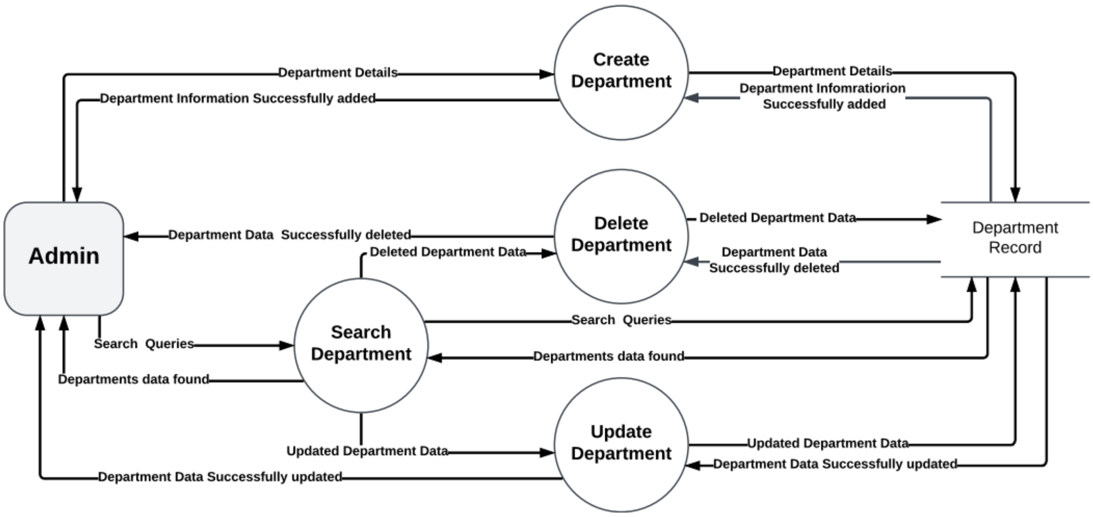


Figure 3: Level 2 Data Flow Diagram: Administrative Management of Departments

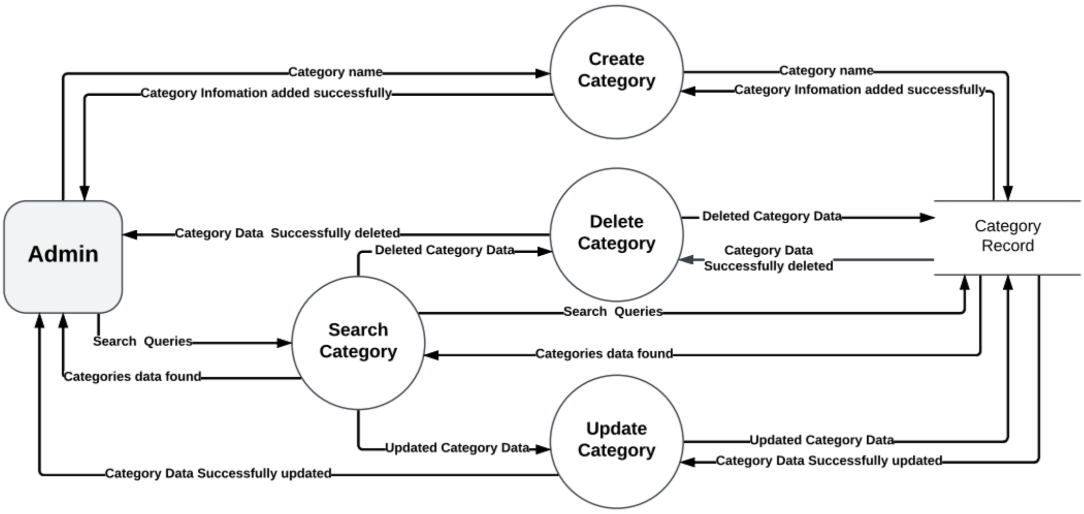


Figure 4: Level 2 Data Flow Diagram: Administrative Management of Categories

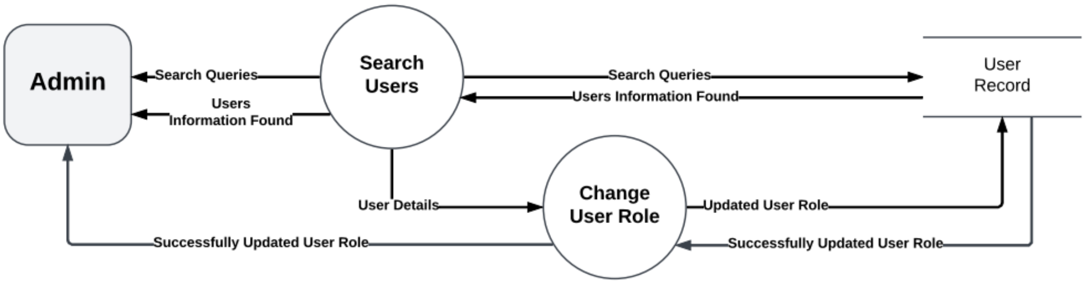


Figure 5: Level 2 Data Flow Diagram: Administrative Management of User

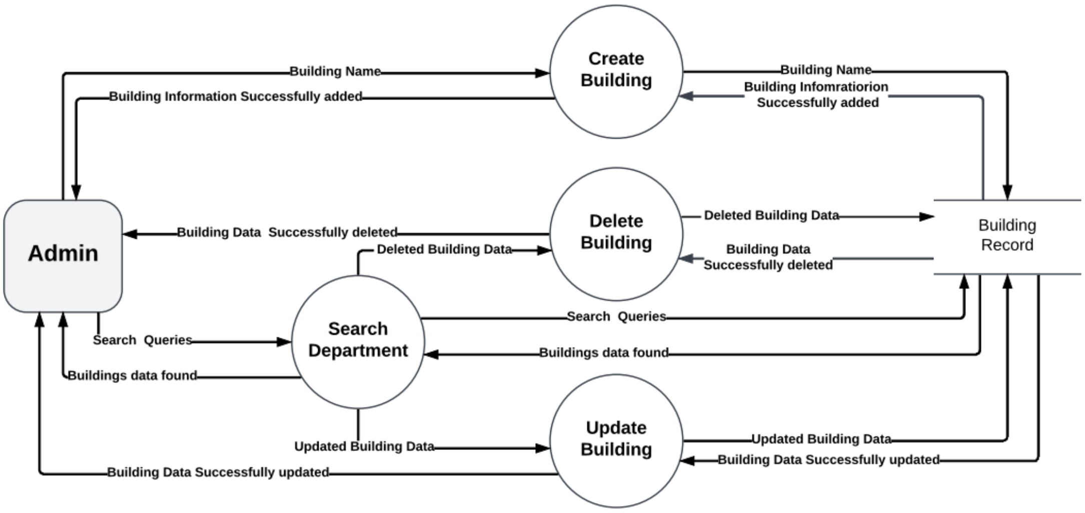


Figure 6: Level 2 Data Flow Diagram: Administrative Management of Buildings

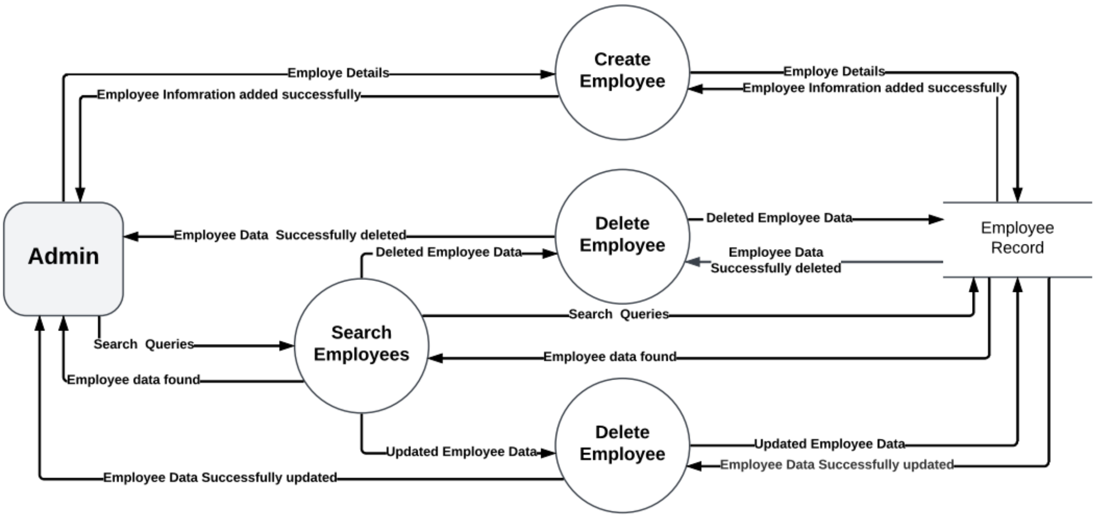


Figure 7: Level 2 Data Flow Diagram: Administrative Management of Employee

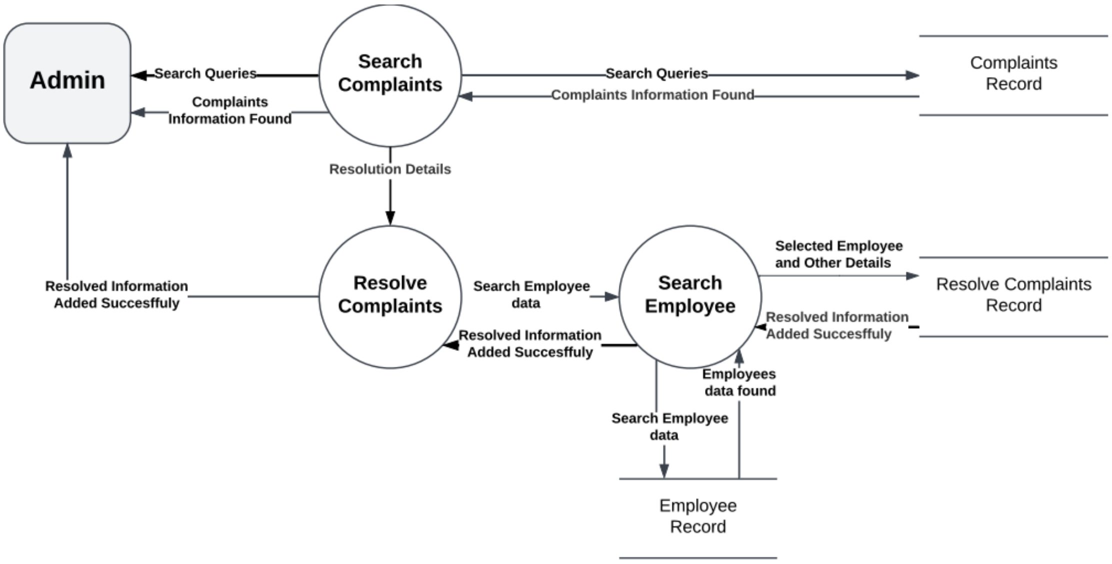
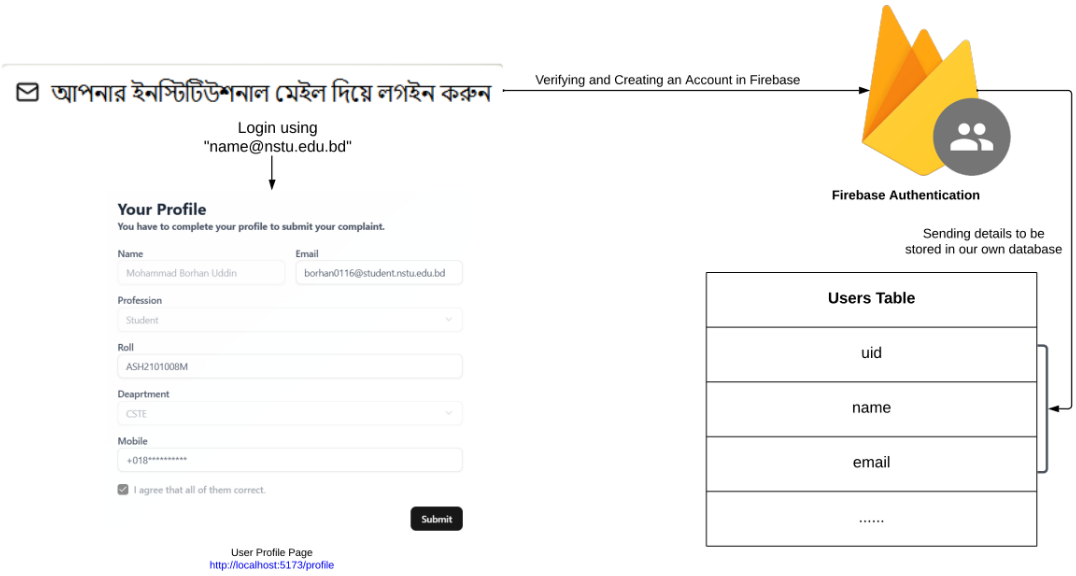

Admin
Search Queries
Complaints Information Found
Search Complaints
Search Queries
Complaints Information Found
Complaints Record
Resolution Details
Resolved Information Added Succesffuly
Resolve Complaints
Search Employee data
Resolved Information Added Succesffuly
Search Employee
Selected Employee and Other Details
Resolved Information Added Succesffuly
Resolve Complaints Record
Employees data found
Search Employee data
Employee Record

Figure 8: Level 2 Data Flow Diagram: Administrative Management of Complaint Resolution



আপনার ইনস্টিটিউশনাল মেইল দিয়ে লগইন করুন
Login using "name@nstu.edu.bd"
Verifying and Creating an Account in Firebase
Firebase Authentication
Sending details to be stored in our own database
Users Table
uid
name
email
......
User Profile Page
http://localhost:5173/profile

Your Profile

You have to complete your profile to submit your complaint.

Name: Mohammad Borhan Uddin

Email: borhan0116@student.nstu.edu.bd

Profession: Student

Roll: ASH2101008M

Deaprtment: CSTE

Mobile: +018**********

☑ I agree that all of them correct.

Submit

Figure 9: Level 2 Data Flow Diagram: Authentication System

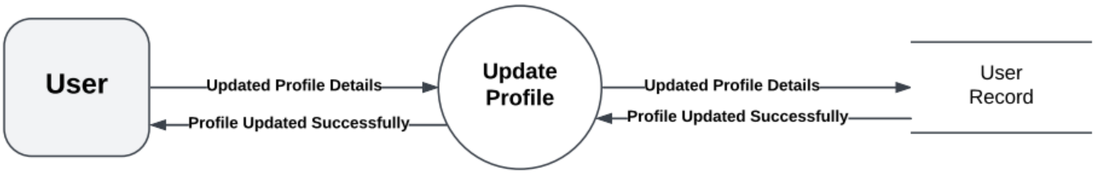


Figure 10: Level 2 Data Flow Diagram: User Profile Management

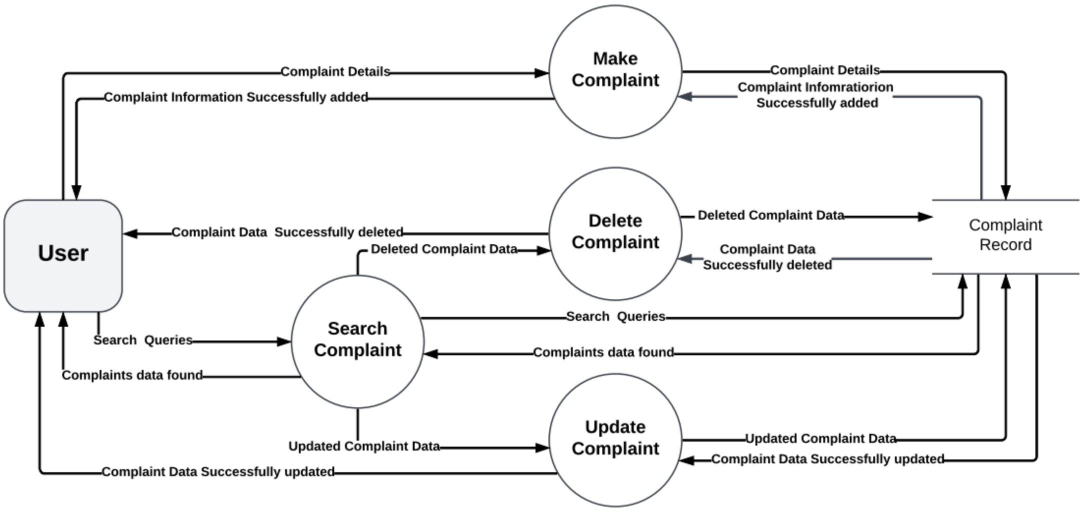


Figure 11: Level 2 Data Flow Diagram: User Management of Complaints

1.2 Class-Based Models

Class-based models focus on the structural aspects of the system, defining entities, their attributes, relationships, and responsibilities.

(i) Entity-Relationship Diagram

An Entity-Relationship (ER) Diagram represents the database structure of the system, showing entities , their attribute, and relationships.

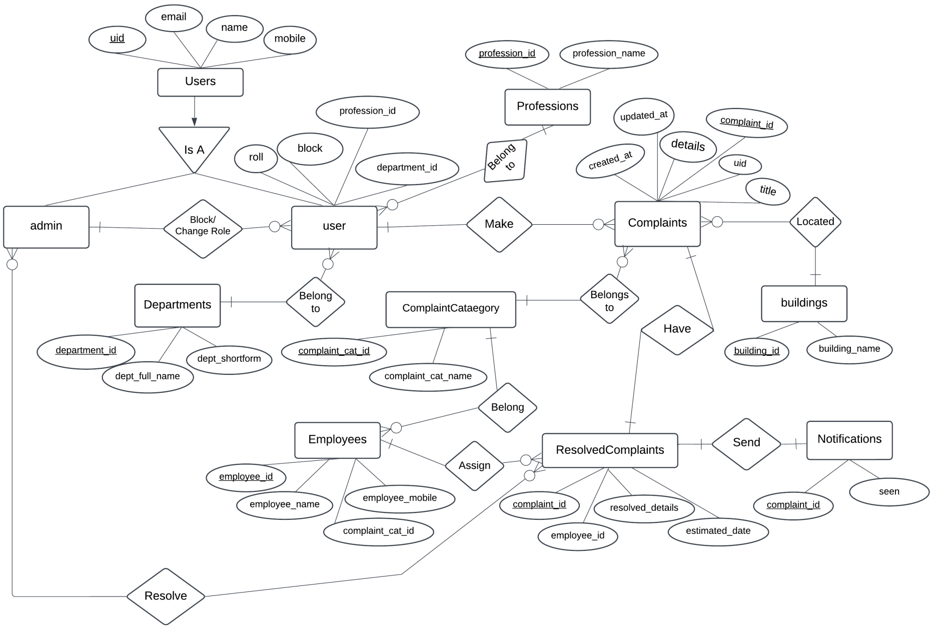


Figure 12: ER Diagram

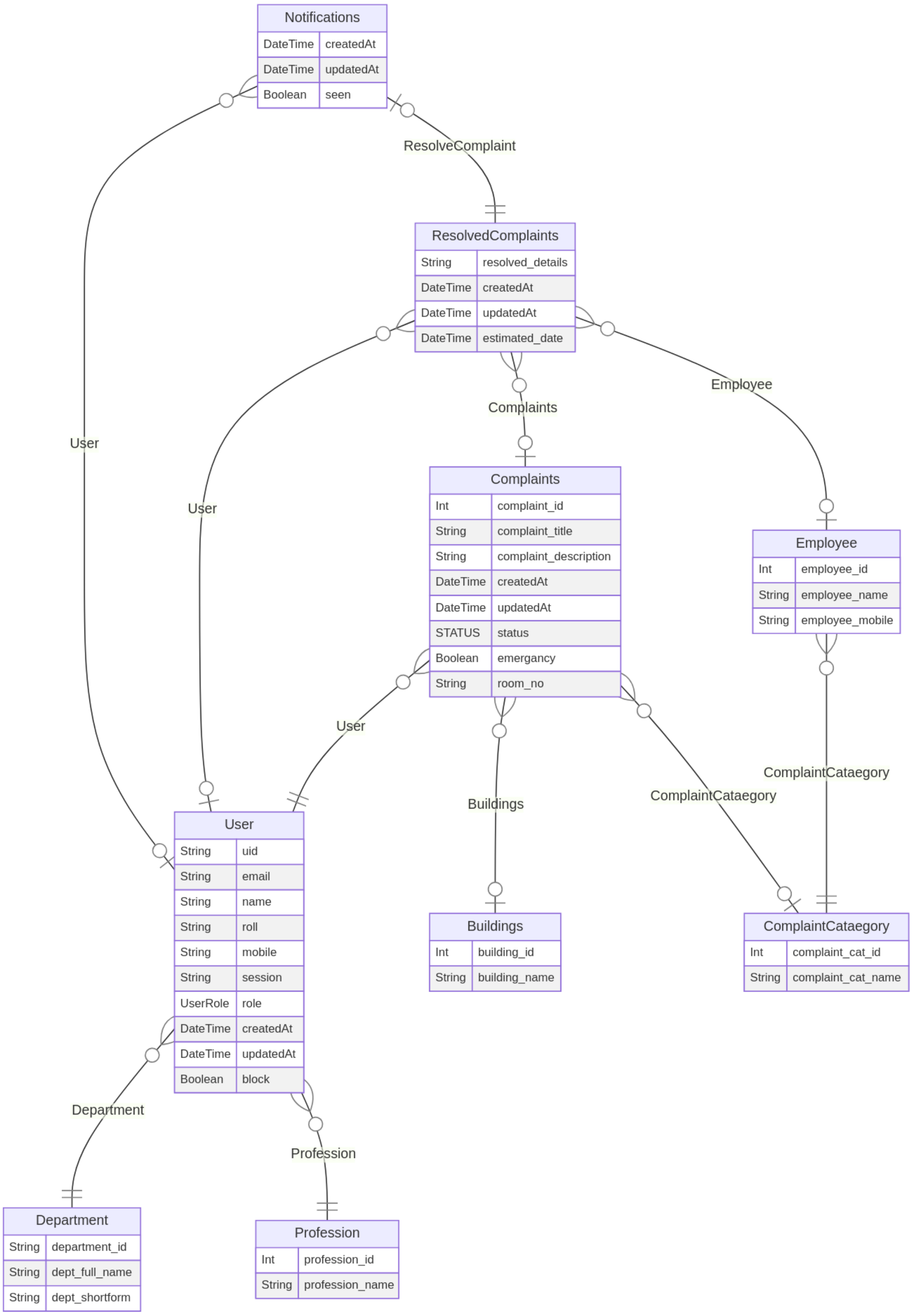


Figure 13: Entity-Relationship Schema Diagram

(i) Class Diagram

A UML Class Diagram models the system's classes, their attributes, methods, and relationship.

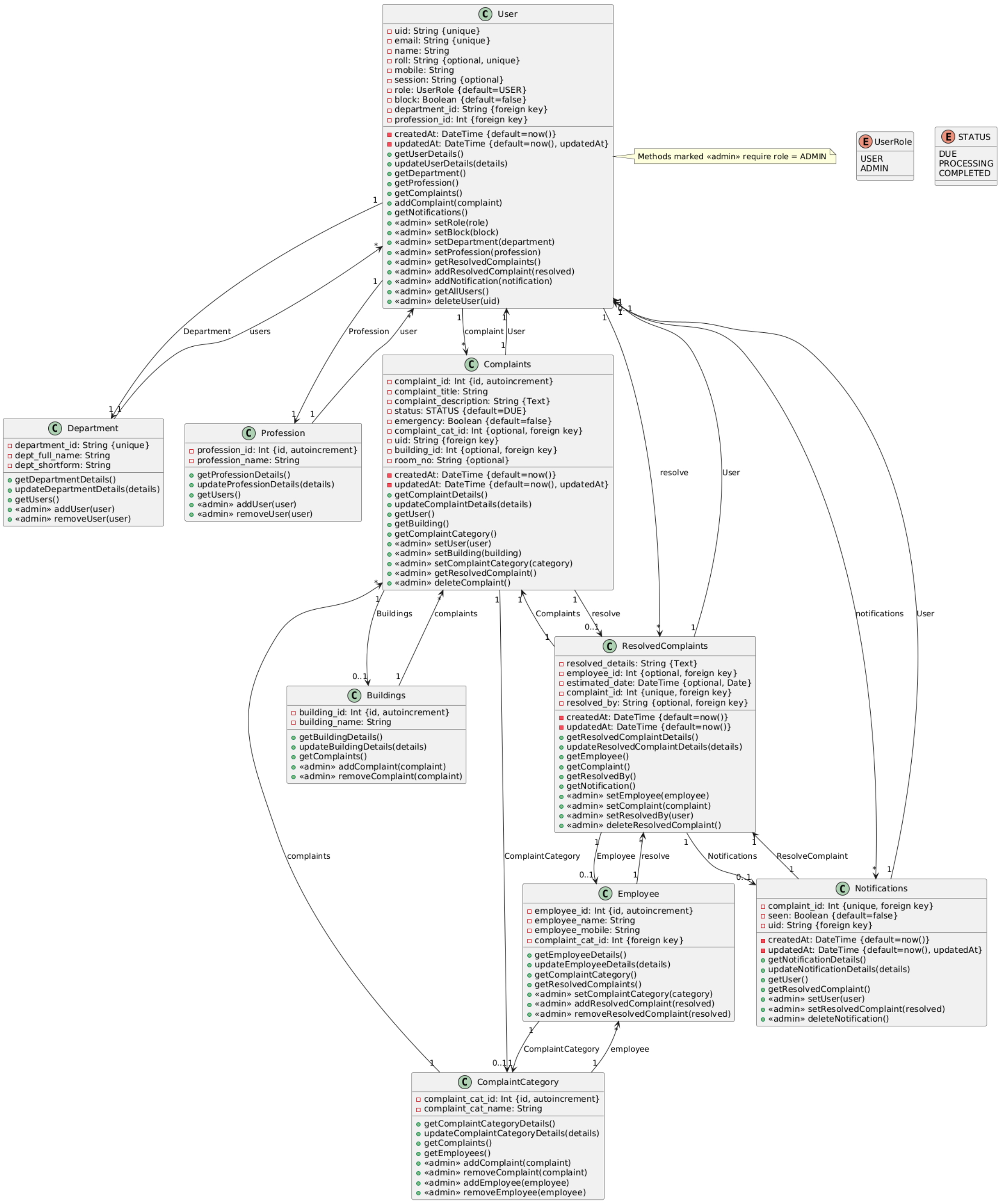
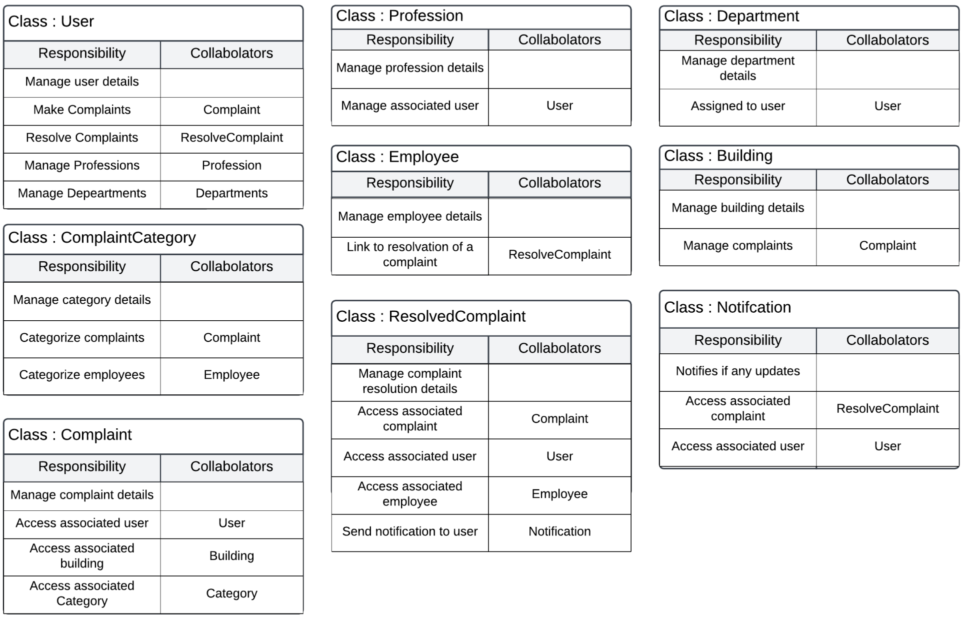

User
uid: String {unique}
email: String {unique}
name: String
roll: String {optional, unique}
mobile: String
session: String {optional}
role: UserRole {default=USER}
block: Boolean {default=false}
department_id: String {foreign key}
profession_id: Int {foreign key}
createdAt: DateTime {default=now()}
updatedAt: DateTime {default=now(), updatedAt}
getUserDetails()
updateUserDetails(details)
getDepartment()
getProfession()
getComplaints()
addComplaint(complaint)
getNotifications()
«admin» setRole(role)
«admin» setBlock(block)
«admin» setDepartment(department)
«admin» setProfession(profession)
«admin» getResolvedComplaints()
«admin» addResolvedComplaint(resolved)
«admin» addNotification(notification)
«admin» getAllUsers()
«admin» deleteUser(uid)
Methods marked «admin» require role = ADMIN
UserRole
USER
ADMIN
STATUS
DUE
PROCESSING
COMPLETED
Department
users
Profession
user
complaint
User
resolve
User
notifications
User
Department
department_id: String {unique}
dept_full_name: String
dept_shortform: String
getDepartmentDetails()
updateDepartmentDetails(details)
getUsers()
«admin» addUser(user)
«admin» removeUser(user)
Profession
profession_id: Int {id, autoincrement}
profession_name: String
getProfessionDetails()
updateProfessionDetails(details)
getUsers()
«admin» addUser(user)
«admin» removeUser(user)
Complaints
complaint_id: Int {id, autoincrement}
complaint_title: String
complaint_description: String {Text}
status: STATUS {default=DUE}
emergency: Boolean {default=false}
complaint_cat_id: Int {optional, foreign key}
uid: String {foreign key}
building_id: Int {optional, foreign key}
room_no: String {optional}
createdAt: DateTime {default=now()}
updatedAt: DateTime {default=now(), updatedAt}
getComplaintDetails()
updateComplaintDetails(details)
getUser()
getBuilding()
getComplaintCategory()
«admin» setUser(user)
«admin» setBuilding(building)
«admin» setComplaintCategory(category)
«admin» getResolvedComplaint()
«admin» deleteComplaint()
Buildings
complaints
Complaints
resolve
Buildings
building_id: Int {id, autoincrement}
building_name: String
getBuildingDetails()
updateBuildingDetails(details)
getComplaints()
«admin» addComplaint(complaint)
«admin» removeComplaint(complaint)
ResolvedComplaints
resolved_details: String {Text}
employee_id: Int {optional, foreign key}
estimated_date: DateTime {optional, Date}
complaint_id: Int {unique, foreign key}
resolved_by: String {optional, foreign key}
createdAt: DateTime {default=now()}
updatedAt: DateTime {default=now()}
getResolvedComplaintDetails()
updateResolvedComplaintDetails(details)
getEmployee()
getComplaint()
getResolvedBy()
getNotification()
«admin» setEmployee(employee)
«admin» setComplaint(complaint)
«admin» setResolvedBy(user)
«admin» deleteResolvedComplaint()
complaints
ComplaintCategory
Employee
resolve
Notifications
ResolveComplaint
Employee
employee_id: Int {id, autoincrement}
employee_name: String
employee_mobile: String
complaint_cat_id: Int {foreign key}
getEmployeeDetails()
updateEmployeeDetails(details)
getComplaintCategory()
getResolvedComplaints()
«admin» setComplaintCategory(category)
«admin» addResolvedComplaint(resolved)
«admin» removeResolvedComplaint(resolved)
Notifications
complaint_id: Int {unique, foreign key}
seen: Boolean {default=false}
uid: String {foreign key}
createdAt: DateTime {default=now()}
updatedAt: DateTime {default=now(), updatedAt}
getNotificationDetails()
updateNotificationDetails(details)
getUser()
getResolvedComplaint()
«admin» setUser(user)
«admin» setResolvedComplaint(resolved)
«admin» deleteNotification()
ComplaintCategory
employee
ComplaintCategory
complaint_cat_id: Int {id, autoincrement}
complaint_cat_name: String
getComplaintCategoryDetails()
updateComplaintCategoryDetails(details)
getComplaints()
getEmployees()
«admin» addComplaint(complaint)
«admin» removeComplaint(complaint)
«admin» addEmployee(employee)
«admin» removeEmployee(employee)

Figure 14: Class Diagram

(iii) Class-Responsibility Collaboration (CRC) Modeling

CRC (Class-Responsibility-Collaboration) modeling identifies classes, their responsibilities (what they d), and collaborators (other classes they interact with).



Class : User

Responsibility	Collabolators
Manage user details	
Make Complaints	Complaint
Resolve Complaints	ResolveComplaint
Manage Professions	Profession
Manage Depeartments	Departments

Class : ComplaintCategory

Responsibility	Collabolators
Manage category details	
Categorize complaints	Complaint
Categorize employees	Employee

Class : Complaint

Responsibility	Collabolators
Manage complaint details	
Access associated user	User
Access associated building	Building
Access associated Category	Category

Class : Profession

Responsibility	Collabolators
Manage profession details	
Manage associated user	User

Class : Employee

Responsibility	Collabolators
Manage employee details	
Link to resolvation of a complaint	ResolveComplaint

Class : ResolvedComplaint

Responsibility	Collabolators
Manage complaint resolution details	
Access associated complaint	Complaint
Access associated user	User
Access associated employee	Employee
Send notification to user	Notification

Class : Department

Responsibility	Collabolators
Manage department details	
Assigned to user	User

Class : Building

Responsibility	Collabolators
Manage building details	
Manage complaints	Complaint

Class : Notifcation

Responsibility	Collabolators
Notifies if any updates	
Access associated complaint	ResolveComplaint
Access associated user	User

Figure 15: Class Diagram

2 Project Planning

This section outlines the planning process for developing the system, ensuring it is completed on time, within budget, and with adequate resources.

2.1 WBS (Work Breakdown Structure)

The WBS breaks the project into smaller, manageable tasks organized hierarchically. For the Complaint Management System, this includes phases like Initiation, Design, Development,

Level 1: Complaint Management System Development

Level 2: Project Initiation

- Define project scope and objectives
- Identify stakeholders (users, admins, employees)
- Create system requirements document

Level 2: System Design

- Design database schema (e.g., User, Complaints tables)
- Develop UML class diagrams and CRC model
- Define user interface mockups

Level 2: System Development

- Implement User module (authentication, profile management)
- Implement Complaints module (filing, status tracking)
- Implement Admin module (user management, resolution)
- Integrate Department and Profession modules
- Develop Notification system

Level 2: Testing

- Unit testing (each module)
- Integration testing (cross-module interactions)
- User acceptance testing (UAT)

Level 2: Deployment

- Deploy to production environment
- Train users and admins
- Provide initial support

Level 2: Maintenance

- Monitor system performance
- Handle bug fixes and updates

2.2 Project Scheduling

The Project Scheduling section outlines the timeline and sequencing of tasks required to develop the Complaint Management System efficiently.

2.2.1 CPM (Critical Path Method)

Identifies the longest sequence of dependent tasks to determine the minimum project duration.

Task Name	**Duration (Days)**	**Dependencies**	**Critical Path**	**Start Date**	**End Date**
Define scope/objectives	3	None	Yes	May 1	May 3
Identify stakeholders	2	Scope	Yes	May 4	May 5
Create requirements doc	4	Stakeholders	Yes	May 6	May 9
Design DB schema	5	Requirements	Yes	May 10	May 14
Develop UML/CRC	4	Requirements	No	May 10	May 13
Define UI mockups	3	Requirements	No	May 10	May 12
Implement User module	10	DB schema	No	May 15	May 24
Implement Complaints	12	DB schema	Yes	May 15	May 26
Implement Admin module	10	User module	No	May 25	Jun 3
Integrate Dept/Prof	6	User module	No	May 25	May 30
Develop Notification	5	User module	No	May 25	May 29
Unit testing	8	All	Yes	Jun 4	Jun 11
Integration testing	6	Unit test	Yes	Jun 12	Jun 17
User acceptance test	5	Integration	Yes	Jun 18	Jun 22
Deploy to production	3	UAT	Yes	Jun 23	Jun 25
Train users/admins	4	Deployment	Yes	Jun 26	Jun 29
Initial support	5	Training	Yes	Jun 30	Jul 4
Monitor performance	3	Deployment	No	Jun 26	Jun 28
Bug fixes/updates	5	Monitoring	No	Jun 29	Jul 3

Table 1: Project Scheduling: CPM and Gantt Chart for Complaint Management System (Starting May 1,2025)

Path 1
Requirements (9 days) → Database schema (5 days) → User module (10 days) → Unit testing (8 days) → Integration testing (6 days) → UAT (5 days) → Deployment (3 days) → Training (4 days) → Initial support (5 days) = 55 days

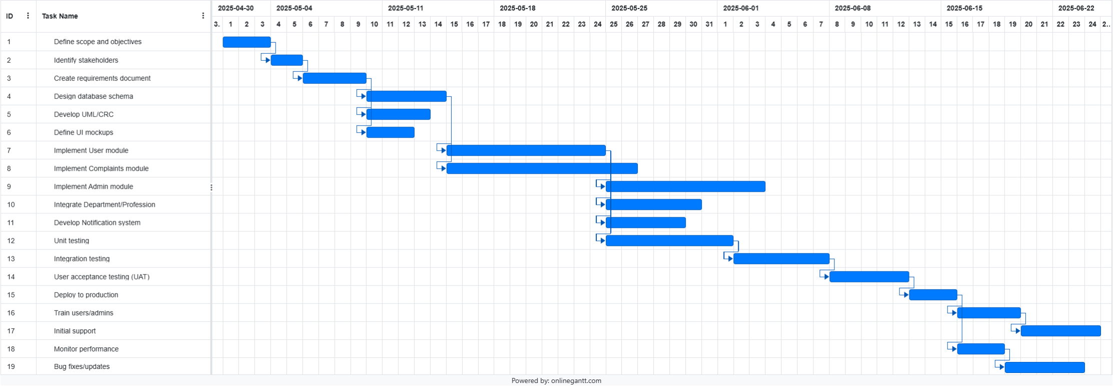
Path 2
Requirements (9 days) → Database schema (5 days) → Complaints module (12 days) → Unit testing (8 days) → Integration testing (6 days) → UAT (5 days) → Deployment (3 days) → Training (4 days) → Initial support (5 days) = 57 days

Path 3
Requirements (9 days) → Database schema (5 days) → Admin module (10 days) → Unit testing (8 days) → Integration testing (6 days) → UAT (5 days) → Deployment (3 days) → Training (4 days) → Initial support (5 days) = 55 days

Critical Path
Path 2 (57 days) is the longest, so the project minimum duration is 57 days. Tasks on this path (e.g., Complaints module) must be completed on time to avoid delays.

2.2.2 Gantt Chart

A visual timeline showing task start/end dates and dependencies.



ID	Task Name
1	Define scope and objectives
2	Identify stakeholders
3	Create requirements document
4	Design database schema
5	Develop UML/CRC
6	Define UI mockups
7	Implement User module
8	Implement Complaints module
9	Implement Admin module
10	Integrate Department/Profession
11	Develop Notification system
12	Unit testing
13	Integration testing
14	User acceptance testing (UAT)
15	Deploy to production
16	Train users/admins
17	Initial support
18	Monitor performance
19	Bug fixes/updates

Figure 16: Grantt Chart

2.3 Project Estimation

This section provides detailed estimates for the development of the Complaint Management System, tailored to a team of two members and a project of moderate complexity. The estimates include product size, effort, schedule, and cost, with justifications rooted in industry standards.

Product Size

The product size is estimated based on the system's scope, which includes 9 classes (e.g., User, Complaint, Department) with approximately 50 methods and 50 attributes, as derived from the class diagram and CRC model. Using the Lines of Code (LOC) metric, the system is projected to require 4000–5000 LOC, reflecting a moderate-sized application for a lab project. Alternatively, using Function Points (FP), an estimate of 30–40 FP is appropriate, considering the number of entities and basic functionalities such as complaint filing and user management. This range is justified by the system's limited scope, avoiding advanced features, and aligning with typical academic project sizes.

Effort

Effort is calculated using the COCOMO (Constructive Cost Model) basic model for organic projects, where Effort = a × (Size)^b, with $a = 2.4$ and $b = 1.05$. Assuming a size of 35 FP (midpoint of 30–40 FP), the effort is:

$$\text{Effort} = 2.4 \times (35)^{1.05} \approx 87 \text{ person-hours.}$$

For a team of two members, this translates to approximately 43.5 person-hours per member. Given a 20-hour workweek per member over the 57-day schedule, the total effort is adjusted to 5–6 person-months (assuming 160 hours per month), reflecting the project's moderate complexity and the team's small size. This adjustment accounts for learning curves and part-time commitment typical in a lab setting.

Schedule

The project schedule, determined by the Critical Path Method (CPM), is set at 57 days, starting from May 1, 2025, and ending on July 4, 2025. This duration encompasses all tasks, including requirement analysis, design, development, testing, and deployment, as outlined in the WBS and Gantt chart. The schedule is justified by the critical path's longest sequence (e.g., requirements to complaints module to testing), ensuring a realistic timeline for a two-member team with overlapping tasks like UML development and database design.

Cost
The cost is estimated based on the effort and an hourly rate suitable for a student project. Assuming a rate of $15 per hour (reflecting a reasonable student or intern wage in 2025), and a total effort of 87 hours, the cost is:

$$\text{Cost} = 87 \times 15 = \$1,305.$$

3 Risk Analysis

This section identifies potential risks associated with the development of the Complaint Management System and outlines a Risk Mitigation, Monitoring, and Management (RMMM) plan to address them, tailored to a two-member team in an academic lab setting.

3.1 Identify Risks (SWOT Analysis)

A SWOT analysis evaluates the project's internal Strengths and Weaknesses, as well as external Opportunities and Threats.

- **Strengths**: The team consists of two motivated students with basic software engineering knowledge, enabling efficient collaboration. The project's scope is well-defined through class modeling and project planning, providing a clear roadmap.
- **Weaknesses**: Limited team size (two members) may lead to workload imbalances or delays if one member is unavailable. Inexperience with complex integrations (e.g., notifications) could result in technical errors.
- **Opportunities**: The project offers a chance to gain practical experience with UML, DFDs, and project management tools, potentially enhancing skills for future courses or internships. Feedback from the instructor can improve the system design.
- **Threats**: Tight deadlines may pressure the team, risking incomplete testing. Unforeseen technical issues or resource constraints (e.g., limited access to tools) could hinder progress.

These factors highlight the need for proactive risk management to ensure successful project completion.

3.2 RMMM Plan

The RMMM plan addresses identified risks with specific strategies for mitigation, monitoring, and management.

- **Risk 1: Workload Imbalance Due to Small Team Size**
  - **Mitigation**: Divide tasks evenly based on the WBS (e.g., one member handles design, the other development), with weekly progress reviews to reallocate if needed.
  - **Monitoring**: Track task completion using a shared schedule (e.g., Gantt chart), checking progress every 3 days.
  - **Management**: If one member falls behind, adjust deadlines or seek assistance from the instructor by May 17, 2025.
- **Risk 2: Technical Errors in Integrations (e.g., Notifications)**
  - **Mitigation**: Conduct unit testing for each module (e.g., User, Notification) early, using mock data to simulate interactions.
  - **Monitoring**: Review integration test results by June 12, 2025, to identify issues.
  - **Management**: Allocate extra time (e.g., 2 days) during the testing phase (June 4–11, 2025) to debug and consult online resources or peers if errors persist.
- **Risk 3: Tight Deadline Pressure**
  - **Mitigation**: Prioritize critical path tasks (e.g., Complaints module) and complete non-critical tasks (e.g., UI mockups) in parallel.
  - **Monitoring**: Assess progress against the 57-day schedule weekly, with a final check by May 17, 2025.
  - **Management**: If delays occur, focus on core functionalities (e.g., complaint filing) and document limitations for submission.
- **Risk 4: Unforeseen Technical Issues or Resource Constraints**
  - **Mitigation**: Set up a basic development environment (e.g., LaTeX, Inkscape) and test database connectivity by May 10, 2025.
  - **Monitoring**: Check tool availability and system performance during the design phase (May 10–14, 2025).
  - **Management**: Request institutional support (e.g., software licenses) or extend testing time (e.g., June 12–17, 2025) if issues arise.

This RMMM plan ensures risks are addressed systematically, aligning with the project's timeline and team capacity.