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

Lab Report On:

Software Portfolio: Problem Fixer

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 : September 02, 2025

Contents

1 Introduction

The *Problem Fixer* system is a proposed complaint management platform designed to improve the efficiency of reporting and resolving issues within a university. The system will facilitate structured and transparent handling of complaints, ensuring that problems are addressed in a timely manner.

1.1 Project Scope

The **Problem Fixer** system is a web-based complaint management platform that enables university students and teachers to efficiently report and track facility issues through a structured digital workflow. Users can submit categorized complaints—including malfunctioning office equipment, electrical faults, sanitation deficiencies, and AC malfunctions—via an intuitive React.js interface. The system's admin dashboard allows administrators to assign specialized employees to specific complaint categories (e.g., Computer, Electrical, Cleaning), monitor resolution progress in real-time, and manage departmental resources. An integrated notification system, powered by Node.js and Prisma ORM, delivers automated status updates through both in-app alerts and email notifications, ensuring transparent communication throughout the complaint lifecycle from submission to resolution."

1.2 Objectives and Goals

The key objectives and goals of the *Problem Fixer* system include:

- Developing an **automated and structured** complaint management system.
- Allowing **teachers and students to submit complaints** and track their progress.
- Enabling **administrators to assign employees** to complaints for quick resolution.
- Ensuring **timely notifications** through in-system alerts and email updates.
- Providing users with the ability to **mark complaints as resolved** once satisfactorily addressed.
- Improving **efficiency and accountability** in complaint resolution.
- Designing the system with **scalability** to handle increasing users and complaints without performance degradation.
- Ensuring **modularity** in the system architecture, allowing easy updates, feature additions, and maintenance.
- Adhering to **software engineering best practices**, including clean code, version control, and thorough testing for robust system development.

2 Defining Requirements

This section defines the essential components, functionalities, and performance expectations for the *Problem Fixer* system.

2.1 System Requirements

The system will be composed of the following key modules:

- **User Module (Teachers, Students or other professionals):** Enables users to submit complaints, track their status, and mark them as resolved.
- **Admin Module:** Allows administrators to manage complaint categories, assign employees, resolve complaints, and send notifications.
- **Employee Management Module:** Enables the admin to add employees and assign them to specific complaint categories.
- **Notification System:** Sends real-time notifications and emails to users upon complaint updates.
- **Database Management:** Securely stores all complaint records, user details, and resolution history.

2.2 Functional Requirements

The following functionalities will be supported:

1. **User Features:**
    - Users can **log in** and **submit complaints**.
    - Users can **select a complaint category** (e.g., Computer, Electrical, Cleaning, AC).
    - Users can **track the status** of their complaints.
    - Users can **mark complaints as done** once resolved.
2. **Admin Features:**
    - Admin can **log in** and **manage complaint categories**.
    - Admin can **add departments**.
    - Admin can **add employees** and **assign them to specific complaint categories**.
    - Admin can **view and manage complaints** submitted by users.
    - Admin can **resolve complaints** by providing resolution details.
    - Admin can **assign an employee** to handle a complaint if necessary.
    - Admin can **send notifications and emails** to users when a complaint is resolved.

2.3 Non-Functional Requirements

1. **Performance Requirements:**
   - The system should be **scalable and capable of handling multiple complaints concurrently**.
   - Complaint submission and updates should be processed **in real-time**.
2. **Security Requirements:**
   - Secure authentication mechanisms must be implemented.
   - Only **authorized admins** should be able to manage complaints and assign employees.
3. **Usability Requirements:**
   - The interface should be **intuitive and easy to navigate**.
   - Complaint submission should be **simple and user-friendly**.
4. **Reliability and Maintainability:**
   - The system should ensure **high availability with minimal downtime**.
   - Data should be **regularly backed up** to prevent loss.

3 Requirement Analysis

The requirement analysis for the ”Problem Fixer” project identifies key system components and user interactions. It focuses on understanding how admins, employees, and teachers interact with the system, automating processes, and managing complaints from submission to resolution. The following use case diagram and descriptions outline the primary system requirements.

3.1 Scenario-Based Models

3.1.1 Use Case Diagram

The Use Case Diagram for the ”Problem Fixer” system models the interactions between the users and the system. The actors in the system include the following.

- **Primary Actor:** User - The individual who submits complaints and marks them as resolved.
- **Secondary Actor:** Admin - The person who manages departments, categories, employees, and resolves complaints.

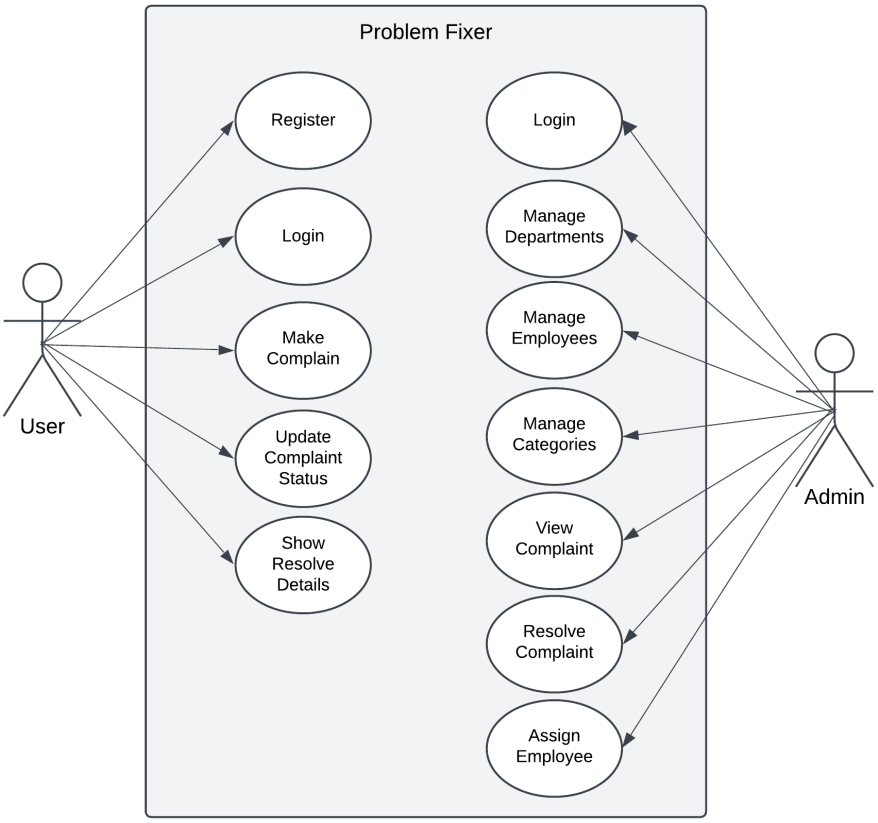

Figure 1: Use Case Diagram

3.1.2 Use Case Descriptions

Use Case 1: Submit a Complaint

- **Actor:** User
- **Description:** The user submits a complaint related to a broken item, service request, or facility issue (e.g., a broken office computer). The user selects a category for the complaint (e.g., Computer, Electrical, Cleaning) and provides a description of the issue.
- **Preconditions:** The user is logged into the system.
- **Postconditions:** The complaint is successfully submitted and recorded in the system.

Use Case 2: Manage Employees, Departments, and Categories

- **Actor:** Admin
- **Description:** The admin manages employees, departments, and categories to ensure smooth operation of the system. The admin assigns employees to specific departments, creates categories for complaints, and ensures that each department has the appropriate resources to handle complaints efficiently.
- **Preconditions:** The admin is logged into the system and has access to management features.
- **Postconditions:** The employees, departments, and categories are successfully managed, and resources are allocated to handle complaints effectively.

Use Case 3: Assign Employees to Complaints

- **Actor:** Admin
- **Description:** The admin assigns employees to complaints based on their expertise in relevant categories (e.g., a computer technician for computer-related issues).
- **Preconditions:** The complaint is in the ”due” state, and there is an employee available to handle it.
- **Postconditions:** The employee is successfully assigned to the complaint, and they can begin working on resolving the issue.

3.2 Behavioral Models

Behavioral models help visualize the dynamic interactions within the system for better understanding.

3.2.1 Activity Diagram

The Activity Diagram depicts the flow of activities in two key processes of the system, showing how various actions and decisions occur in a sequential manner.

(a) Activity Diagram for Making a Complain Process by User

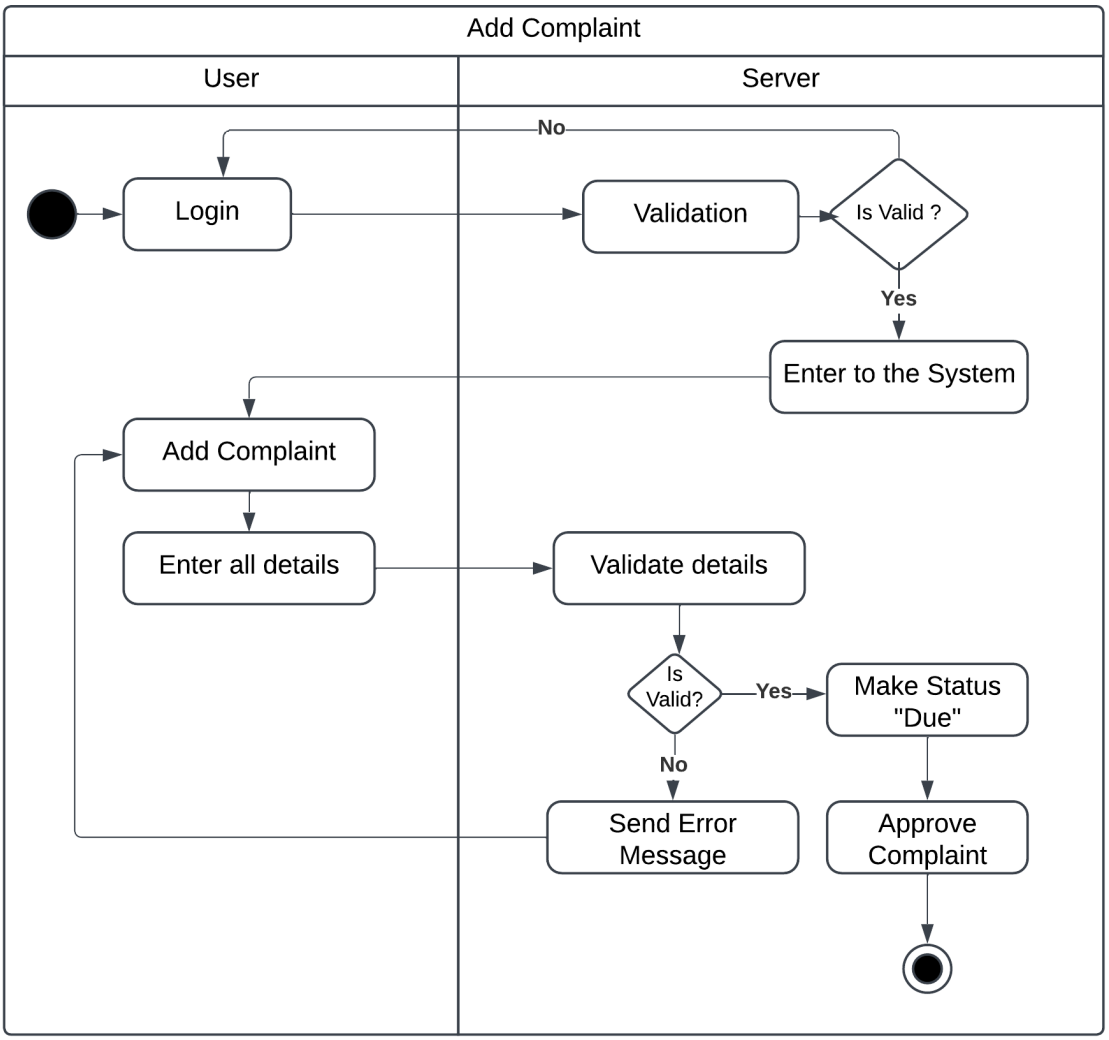

Figure 2: Activity Diagram of Making Complain Process by User

(b) Activity Diagram of the Complaint Management Process by Admin

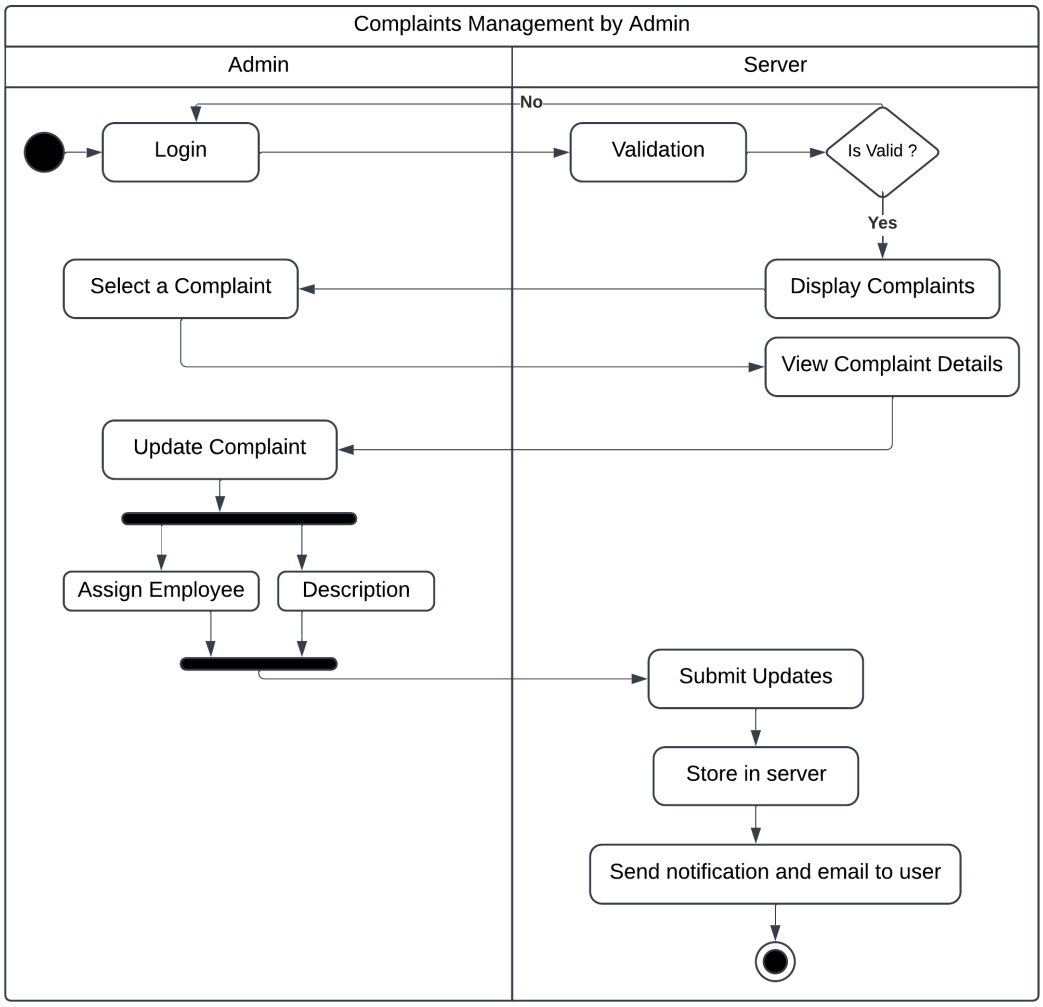

Figure 3: Activity Diagram of Complaint Management Process by Admin

3.2.2 Sequence Diagram

The Sequence Diagram illustrates the interaction between system components for two important processes, showing the order of messages exchanged between the objects in the system.

(a) Sequence Diagram of Registration Process

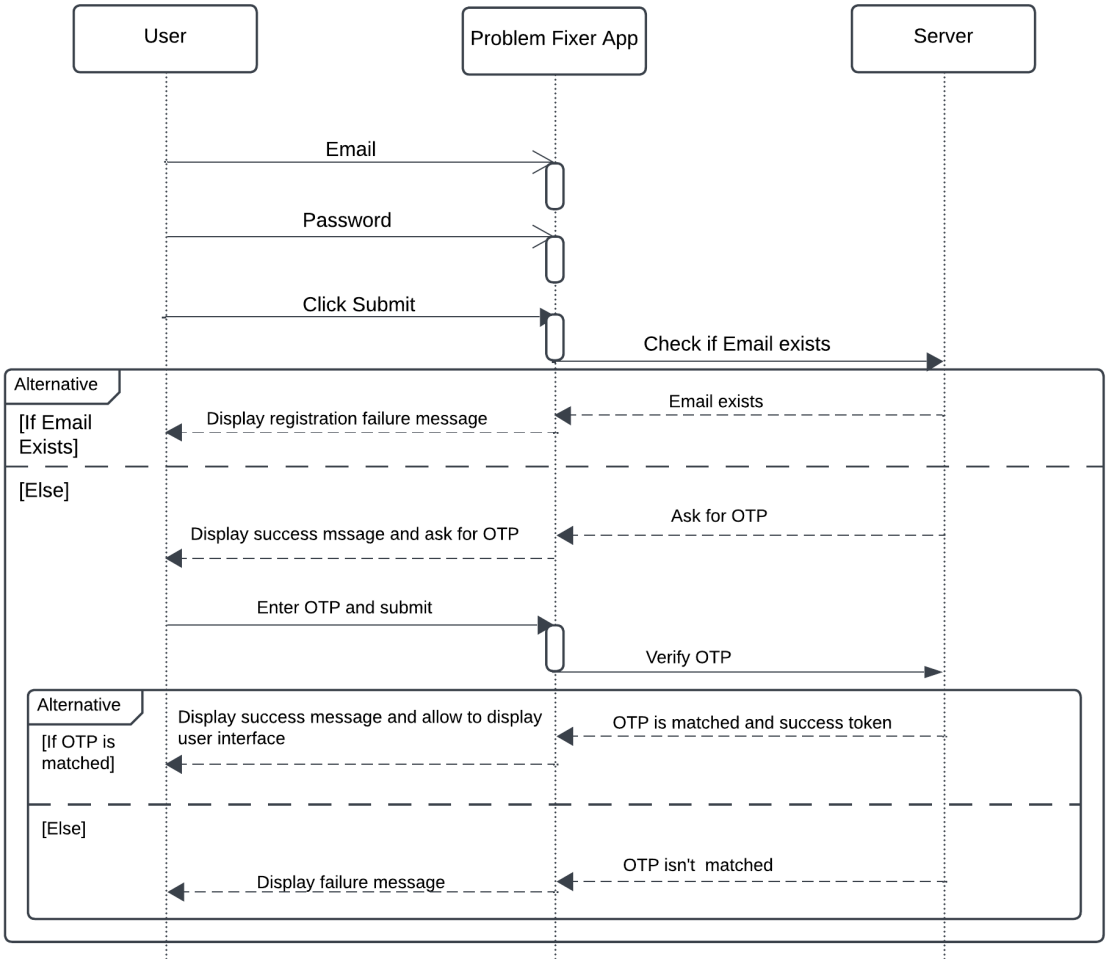

Figure 4: Sequence Diagram of Registration Process

(b) Sequence Diagram of Adding Employee

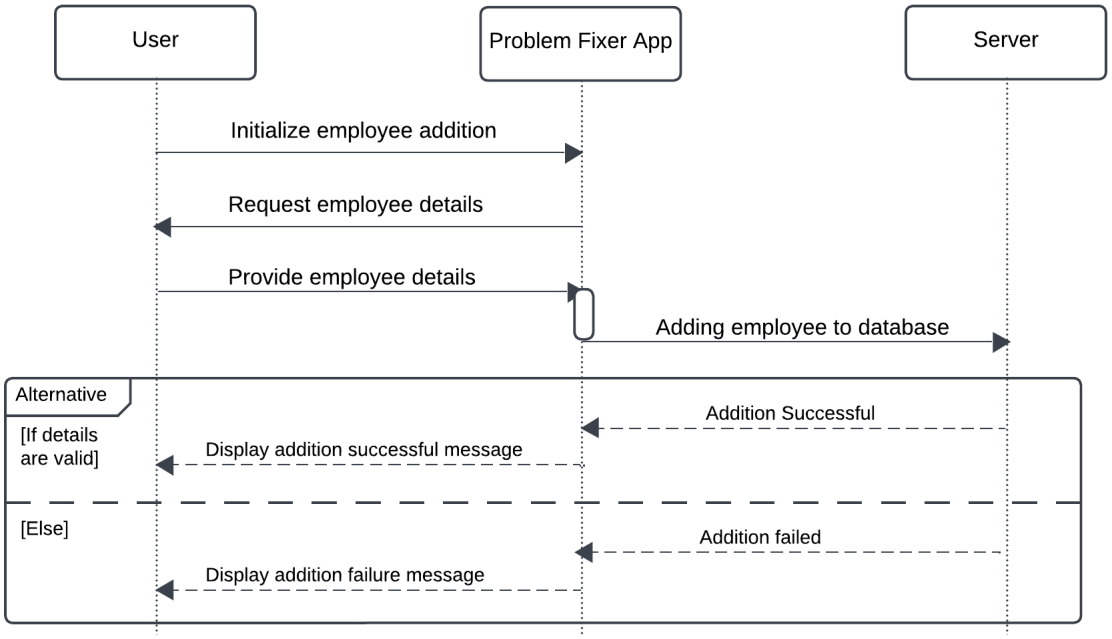

Figure 5: Sequence Diagram of Adding Employee

3.3 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).

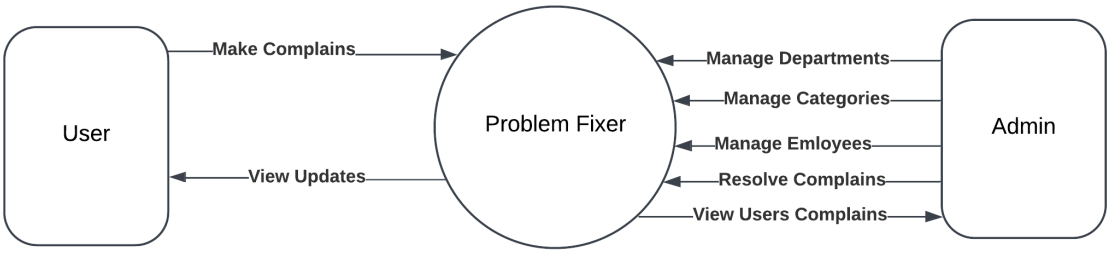

Figure 6: Level 0 DFD

(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.

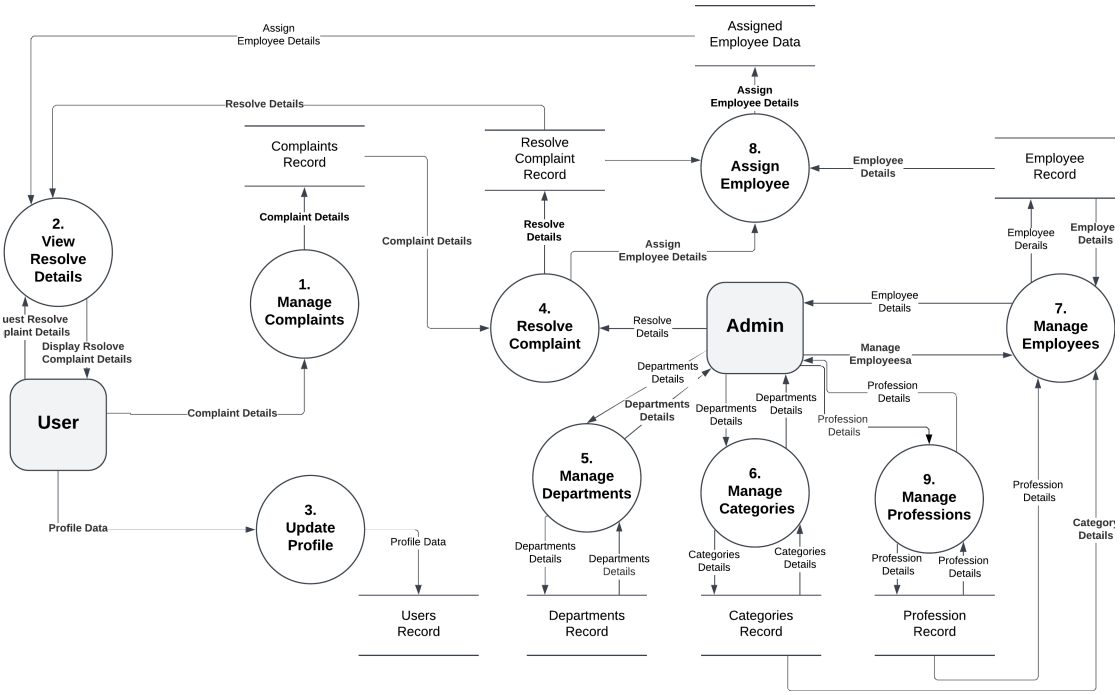

Figure 7: Level 1 DFD

(iii) Level 2 DFD

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

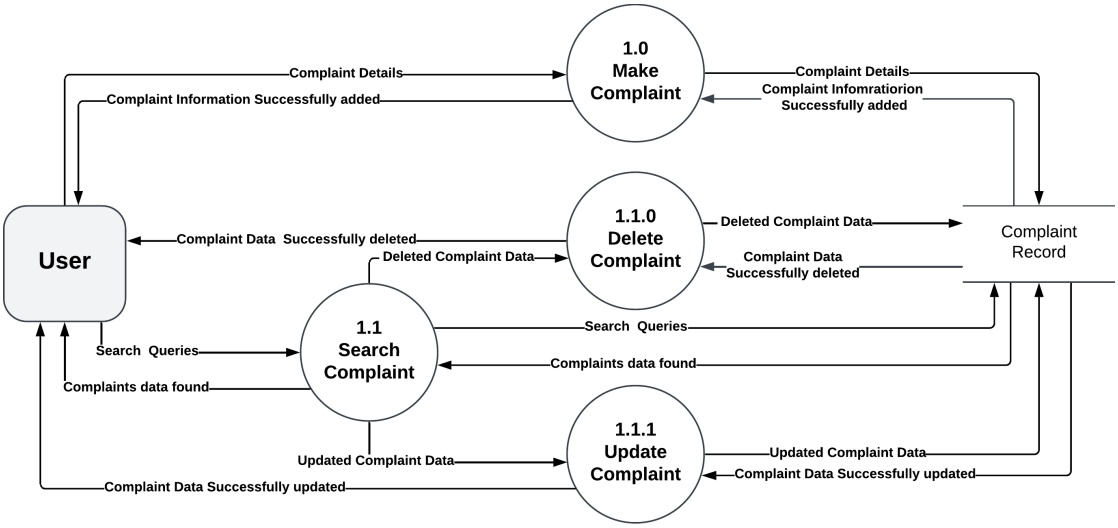

Figure 8: Label 2 DFD of Complaints Management by User

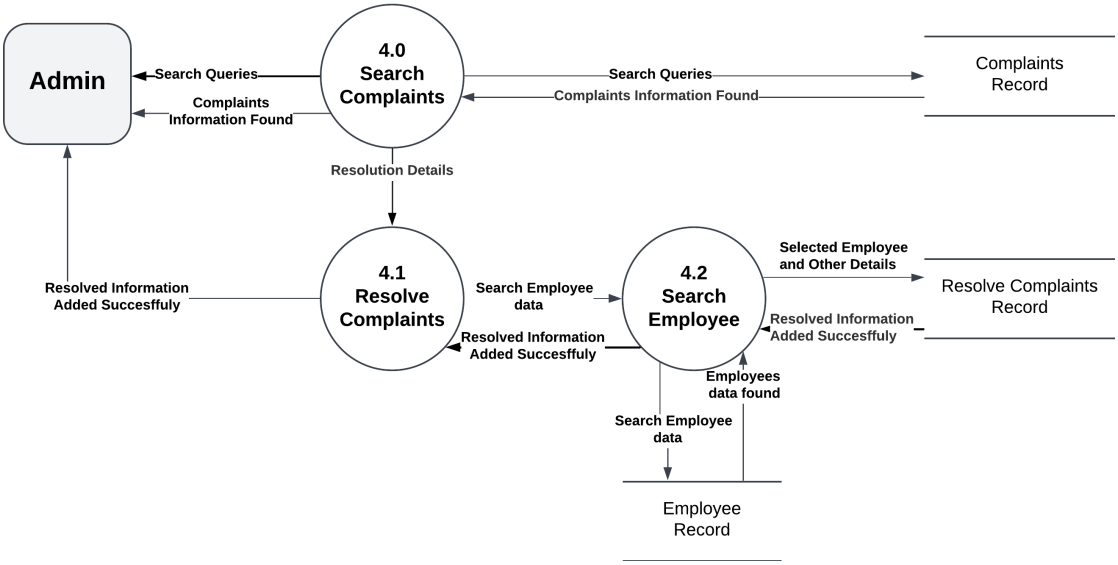

Figure 9: Level 2 DFD of Resolving Complaints by Admin

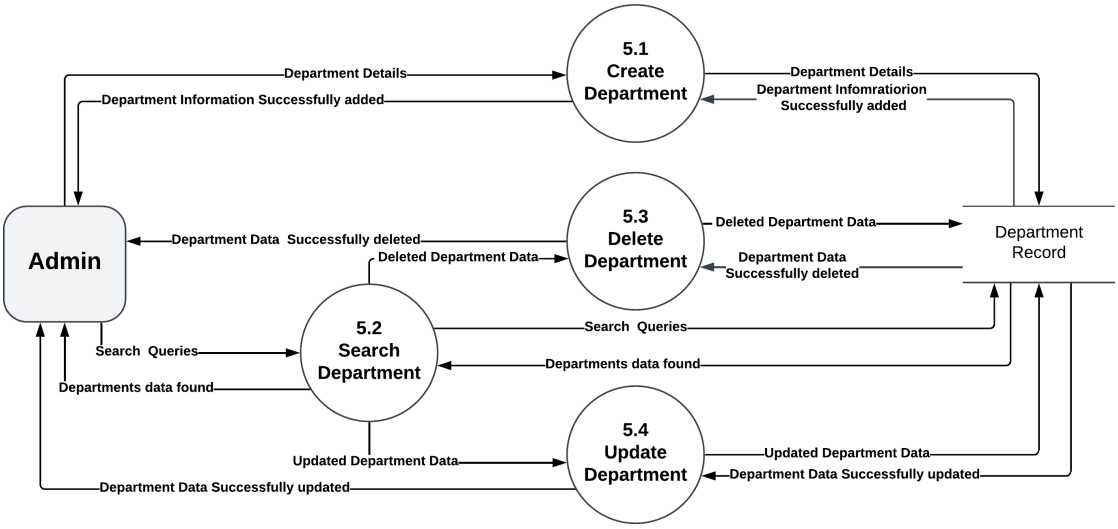

Figure 10: Level 2 DFD of Departments Management by Admin

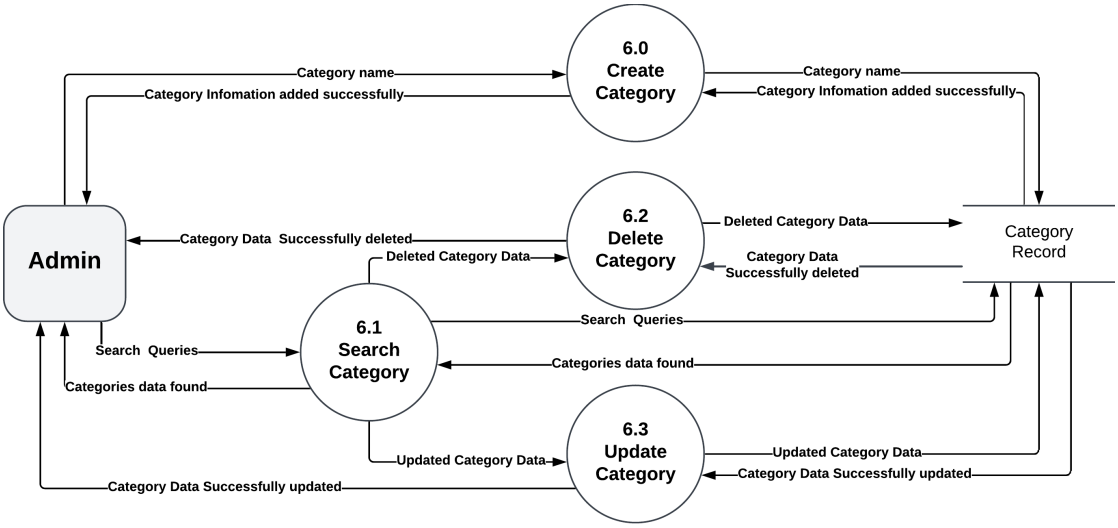

Figure 11: Level 2 DFD of Category Management by Admin

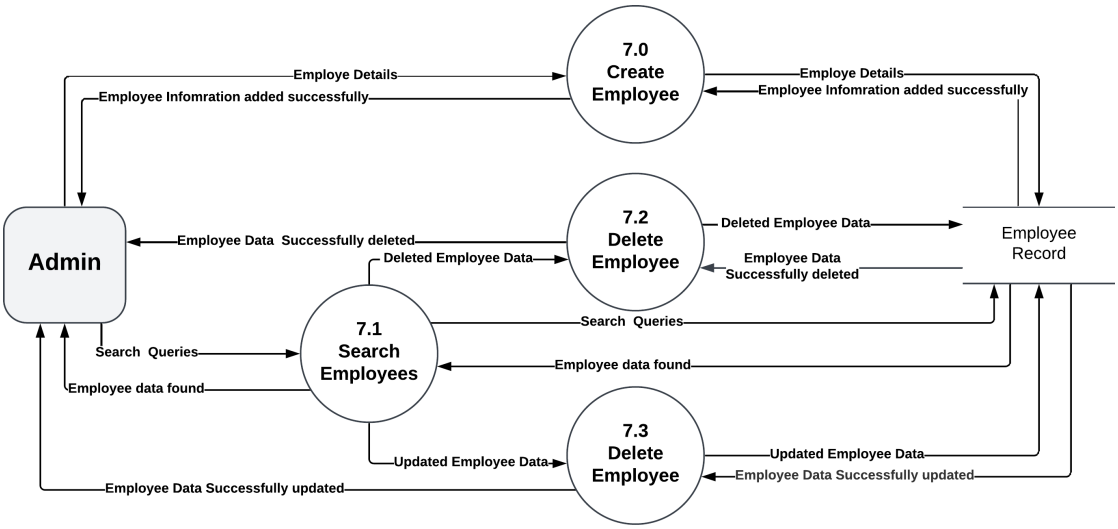

Figure 12: Level 2 DFD of Employees Management by Admin

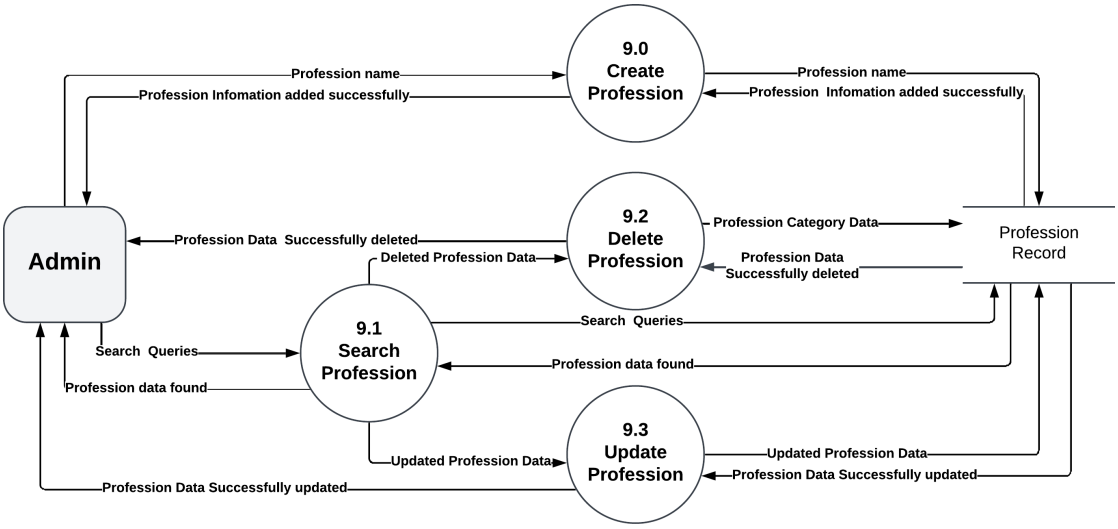

Figure 13: Level 2 DFD of Professions Management by Admin

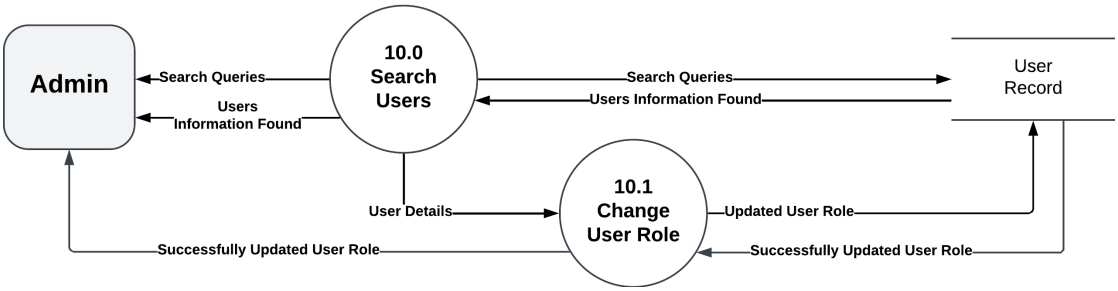

Figure 14: Level 2 DFD of Changing Roles of User by Admin

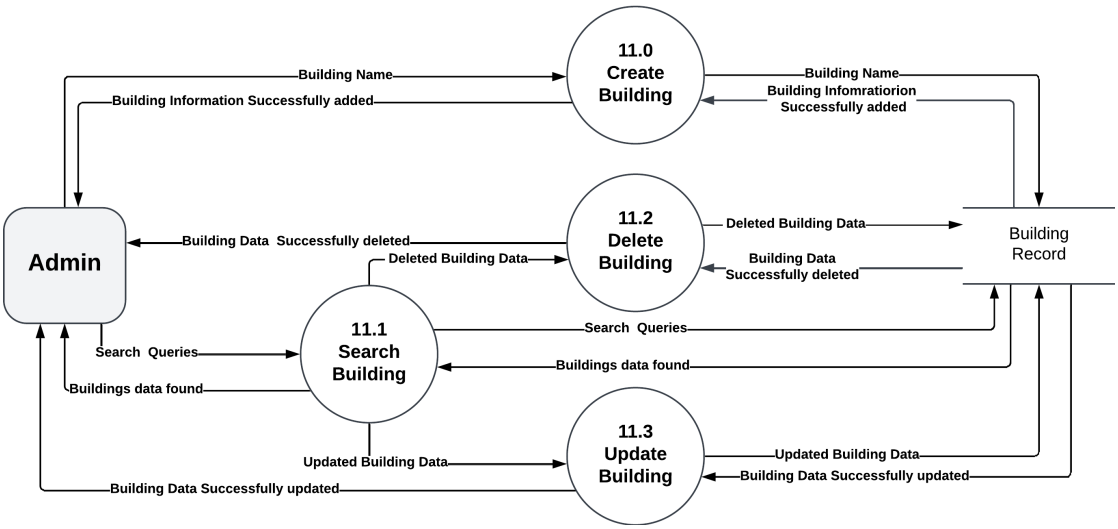

Figure 15: Level 2 DFD of Buildings Management by Admin

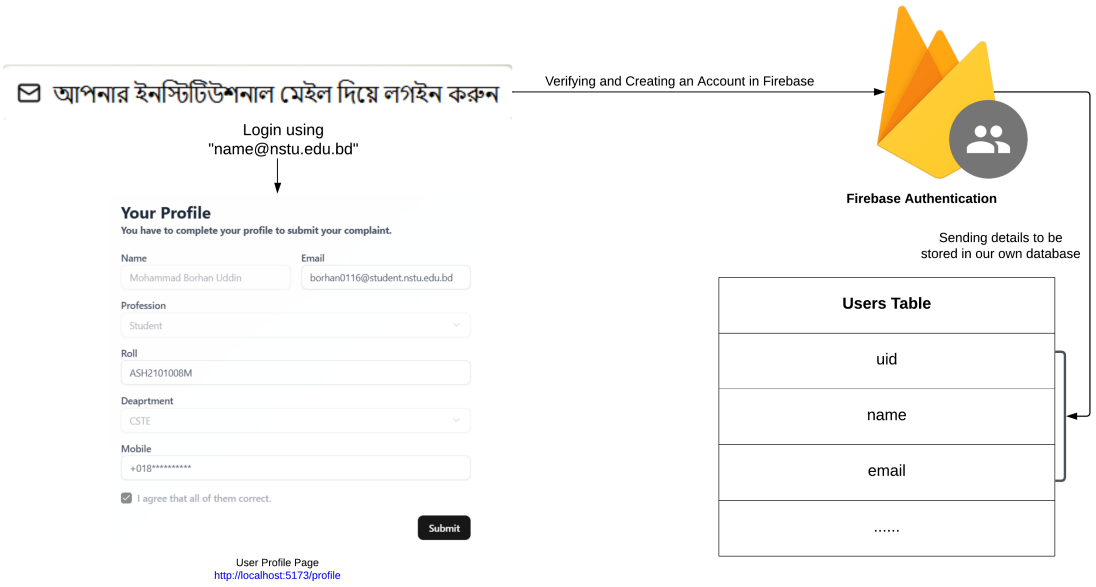

Figure 16: Level 2 DFD of Authentication System

3.4 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.

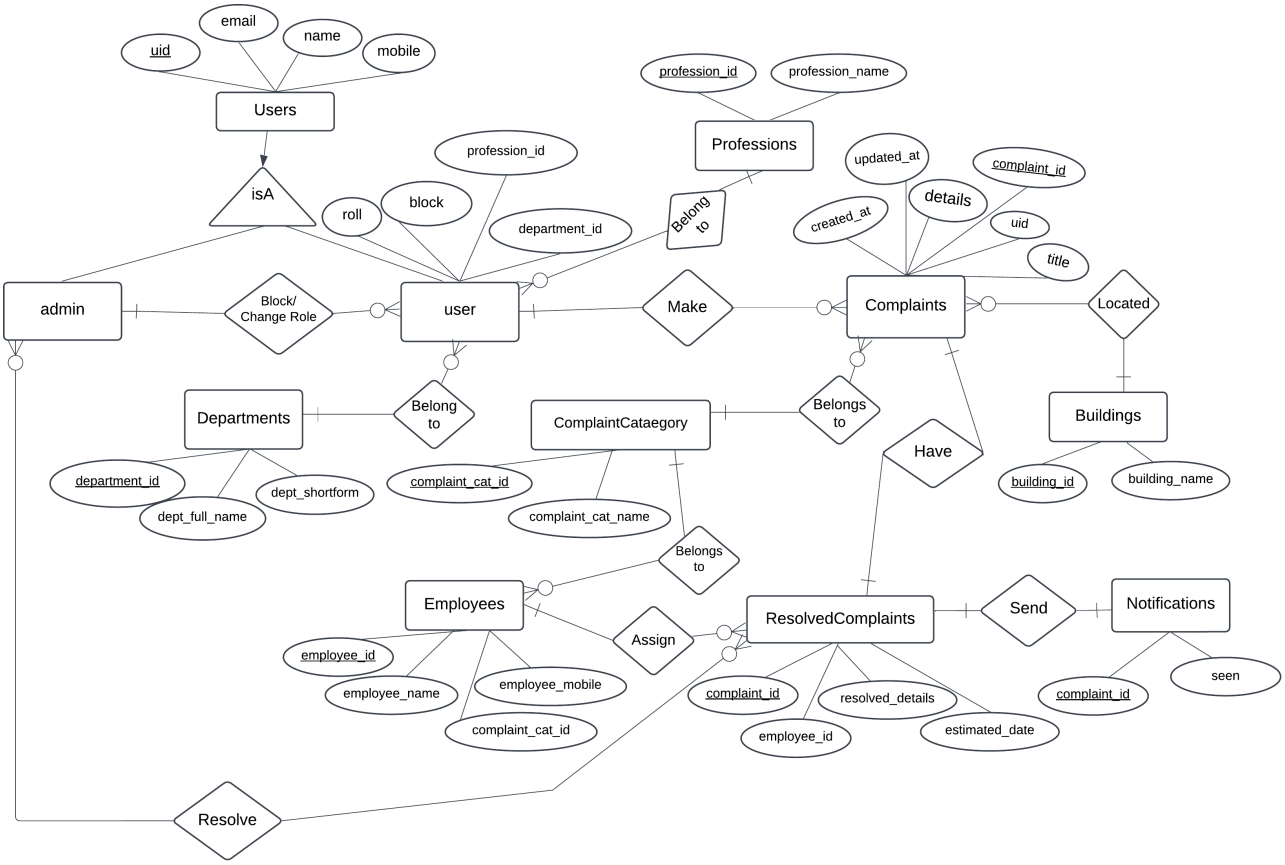

Figure 17: ER Diagram

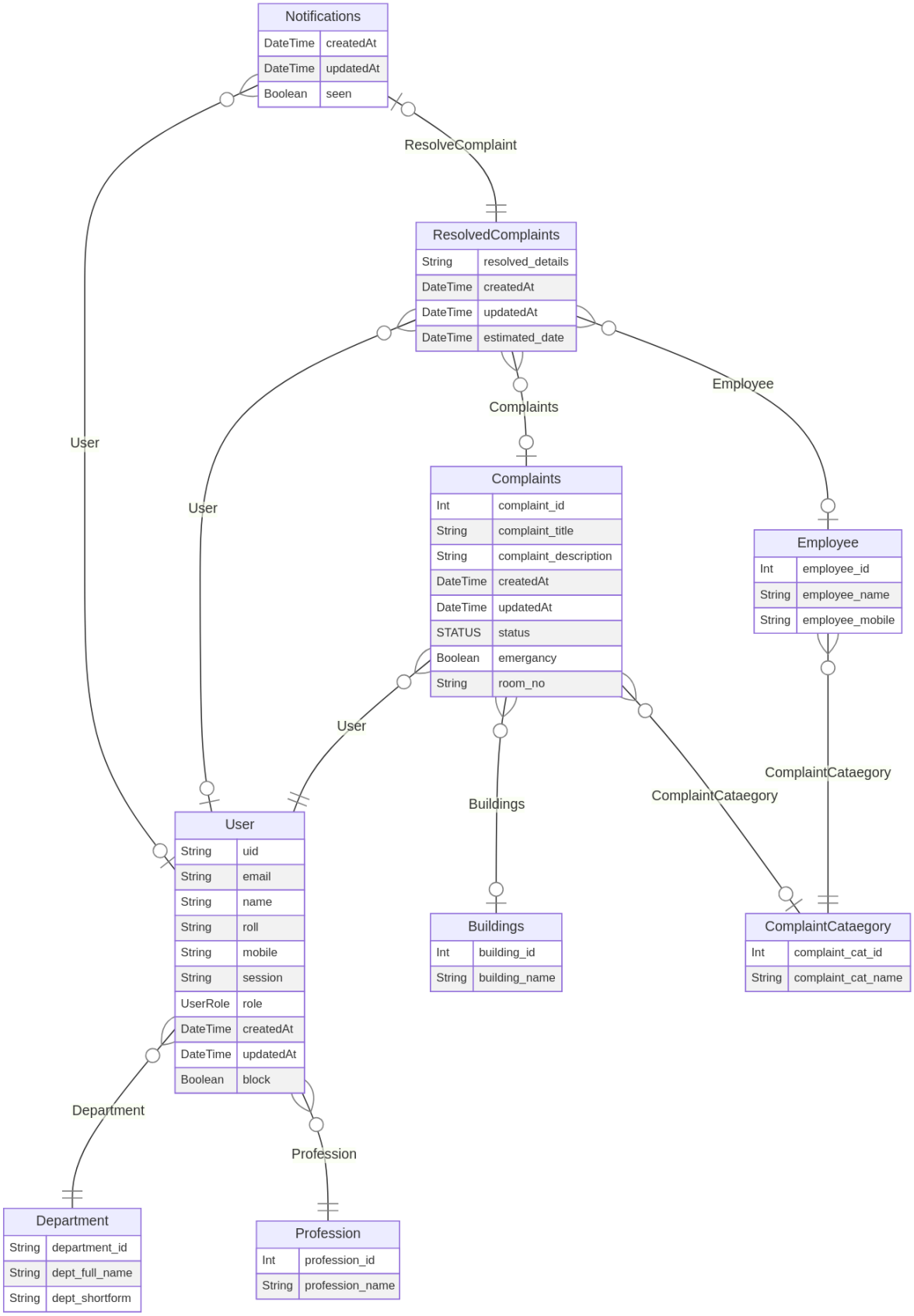

Figure 18: Entity-Relationship Schema Diagram

(i) Class Diagram

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

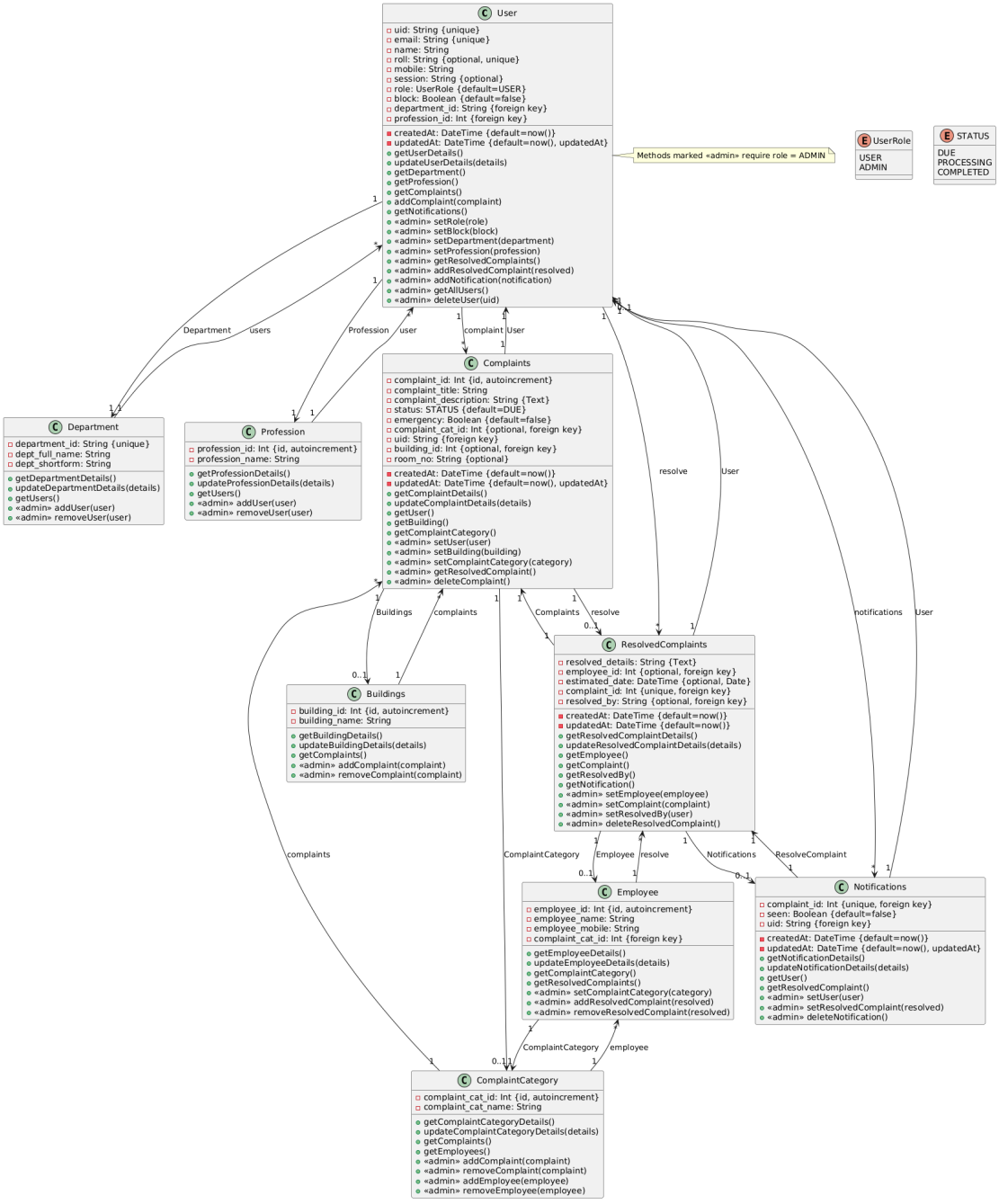

Figure 19: Class Diagram

(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).

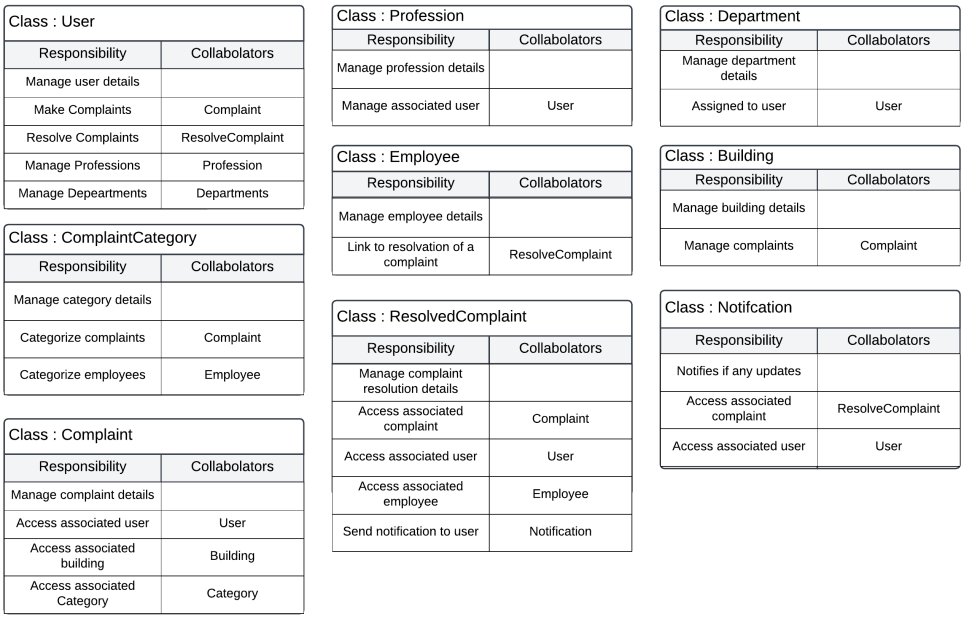

Figure 20: Class Diagram

4 Project Planning

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

4.1 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

4.2 Project Scheduling

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

4.2.1 CPM (Critical Path Method)

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

Task	Task Name	Duration (Days)	Dependencies	Critical Path	Start Date	End Date
A	Define scope/objectives	3	None	Yes	May 1	May 3
B	Identify stakeholders	2	A	Yes	May 4	May 5
C	Create requirements doc	4	B	Yes	May 6	May 9
D	Design DB schema	5	C	Yes	May 10	May 14
E	Develop UML/CRC	4	C	No	May 10	May 13
F	Define UI mockups	3	C	No	May 10	May 12
G	Implement User module	10	D	No	May 15	May 24
H	Implement Complaints	12	D	Yes	May 15	May 26
I	Implement Admin module	10	G	No	May 25	Jun 3
J	Integrate Dept/Prof	6	G	No	May 25	May 30
K	Develop Notification	5	G	No	May 25	May 29
L	Unit testing	8	A,..,J	Yes	Jun 4	Jun 11
M	Integration testing	6	L	Yes	Jun 12	Jun 17
N	User acceptance test	5	M	Yes	Jun 18	Jun 22
O	Deploy to production	3	N	Yes	Jun 23	Jun 25
P	Train users/admins	4	O	Yes	Jun 26	Jun 29
Q	Initial support	5	P	Yes	Jun 30	Jul 4
R	Monitor performance	3	O	No	Jun 26	Jun 28
S	Bug fixes/updates	5	R	No	Jun 29	Jul 3

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

Critical Path Method (CPM) Diagram

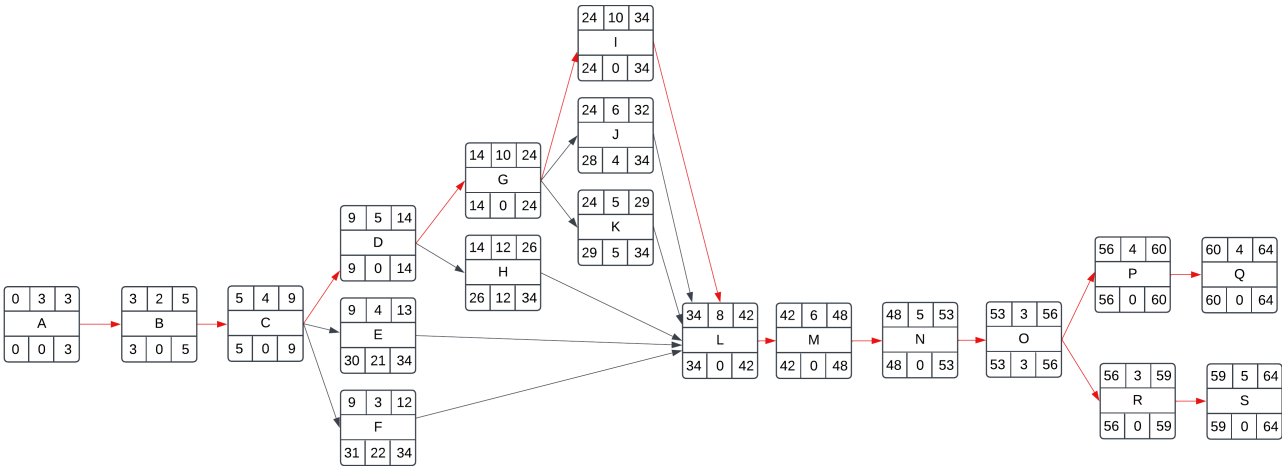

Figure 21: Critical Path Method (CPM) Diagram

The CPM analysis identifies two critical paths:

- A → B → C → D → G → H → I → J → L → M → N → O → P → Q
- A → B → C → D → G → H → I → J → L → M → N → O → R → S

4.2.2 Gantt Chart

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

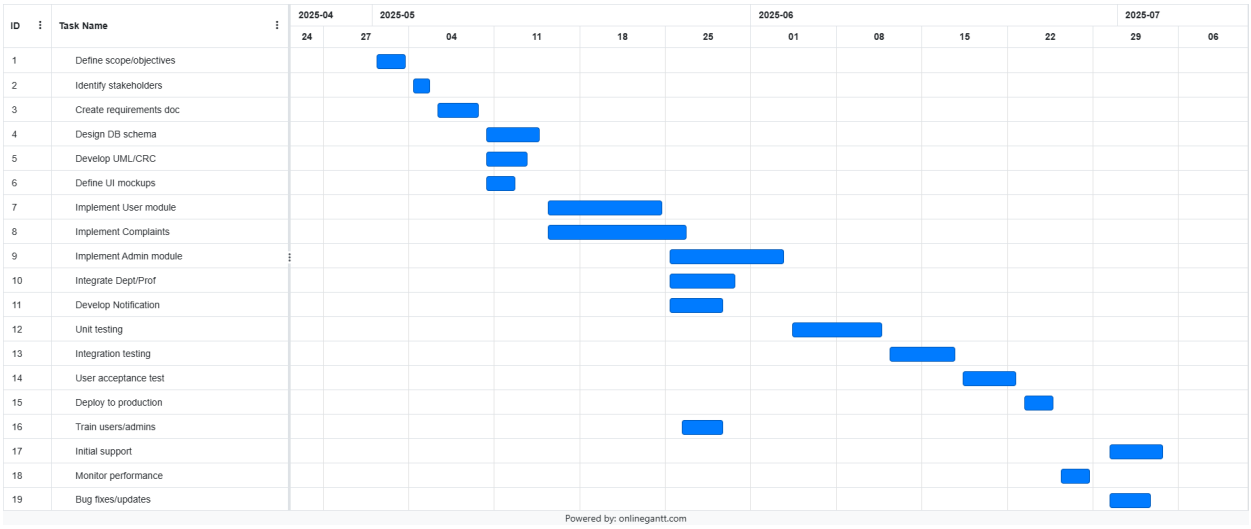

Figure 22: Grantt Chart

4.3 Software Measurements

Direct Measures

- **Lines of Code (LOC):** 4,000–5,000 (derived from 9 classes with approximately 50 methods and 50 attributes)
- **Function Points (FP):** 30–40 (based on system entities and core functionalities)
- **Defect Density:** 0 defects/KLOC (all test cases passed)
- **Number of Test Cases:** 5 formal test cases covering:
  - User authentication (Test ID 001–002)
  - Complaint management (Test ID 003, 005)
  - Admin operations (Test ID 004)
- **Number of Modules:** 11 core modules:
  - User Module
  - Admin Module
  - Authentication Management
  - Employee Management
  - Category Management
  - Complaints Management
  - Notification Management

  - Complaints Management
  - Building Management
  - Professions Management
  - Database Management

Indirect Measures

- **Productivity:**

$$\text{Productivity} = \frac{\text{Size (35 FP)}}{\text{Effort (87 person-hours)}} = 0.40 \text{ FP/person-hour}$$

- **Quality:** 100% test pass rate (validated through black-box, white-box, unit, integration, subsystem, and acceptance testing)
- **Schedule Adherence:** 64-day timeline met (all critical path tasks completed without delays)
- **Cost Efficiency:**

$$\text{Total Cost} = 87 \text{ hours} \times \$15/\text{hour} = \$1,305$$

$$\text{Cost per FP} = \frac{\$1,305}{35 \text{ FP}} = \$37.29$$

4.4 Software Metrics

Product Metrics

- **Size Metrics:**
  - LOC: 4,000–5,000
  - FP: 30–40
  - Complexity: Moderate (9 classes, 50 methods)
- **Quality Metrics:**
  - Defect Density: 0 defects/KLOC
  - Reliability: High availability with minimal downtime
  - Usability: Intuitive UI (complaint submission requires <5 steps)
- **Functional Metrics:** 100% of functional requirements implemented:
  - User login/complaint tracking
  - Admin resolution/employee assignment
  - Real-time notifications

Process Metrics

- **Effort Metrics:**
  - Total Effort: 87 person-hours
  - Effort Distribution:
    * Design: 20% (DB schema, UML diagrams)
    * Development: 50% (modules implementation)
    * Testing: 30% (unit to acceptance testing)
- **Schedule Metrics:**
  - Planned Duration: 64 days
  - On-Time Completion: All critical path tasks finished by deadline
- **Cost Metrics:**
  - Total Cost: $1,305
  - Cost per FP: $37.29

Project Metrics

- **Progress Tracking:**
  - WBS Compliance: All 6 phases completed:
    * Initiation → Design → Development → Testing → Deployment → Maintenance
  - Milestone Achievement: 100% (UML design completed by May 14, deployment by June 25)
- **Risk Management:**
  - Risks Identified: 4 (workload imbalance, integration errors, deadline pressure, resource constraints)
  - Mitigation Success: 100% (parallel task completion resolved deadline pressure)
- **Resource Utilization:**
  - Team Size: 2 members
  - Tool Efficiency:
    * GitHub/Vercel for CI/CD
    * React.js, Node.js, MySQL for development
- **Stakeholder Satisfaction:**
  - Acceptance Testing: 100% success
  - User Feedback: Positive (user-friendly interface)

4.5 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 $\times$ (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 64-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 64 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:

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

5 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.

5.1 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.

5.2 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.

6 Testing

Testing is the process of evaluating software to ensure it meets requirements and works correctly.

i. Test Cases

Test Id	001
Test Condition	Verify login with valid institutional email using "Login with your institutional email" button
Test Steps	1. Open login page. 2. Click "Login with your institutional email" button. 3. Google login popup appears. 4. Select a valid @nstu.edu.bd email.
Test Input	student0116@nstu.edu.bd
Test Expected Result	User should be successfully authenticated and redirected to dashboard/homepage.
Actual Result	Pass
Status	Pass
Remarks	Positive test case

Table 2: Test Case for Student Roll/ID Validation

Test Id	002
Test Condition	Verify student roll/ID validation when adding their ID. Roll must match the format and department code derived from email.
Test Steps	1. Open student profile page. 2. Click on "Add Roll/ID" field. 3. Enter roll in the format: HallCode (3 letters) + SessionID (2 digits) + DepartmentCode (2 digits) + ClassSerial/Roll (3 digits) + Gender (M/F). 4. Submit. 5. System validates roll format and department code against email. 6. Observe result.
Test Input	Email: student*01*16@cste.nstu.edu.bd Roll: ASH21*02*00*M
Test Expected Result	System should reject the roll and show an error: *Department code does not match your email. Please enter the correct code (01 for CSTE).*
Actual Result	Pass
Status	Pass
Remarks	Negative test case: incorrect department code. Checks proper validation against email.

Table 3: Test Case for Student Roll/ID Validation

Test Id	003
Test Condition	Verify that a user can successfully create a complaint by providing all required details.
Test Steps	1. Open the complaint submission page. 2. Click on **“Make a Complaint”** button. 3. Enter a short title in the “Title” field. 4. Select a building from the “Building” dropdown. 5. Select a valid category from the “Category” dropdown. 6. Enter the room number in the “Room No” field. 7. Enter detailed description in the “Details” field. 8. Click the **Submit** button. 9. Observe result.
Test Input	Title: “Computer not working” Building: “Academic 1” Category: “Computer” Room No: 305 Details: “The office computer is not turning on despite repeated attempts.”
Test Expected Result	Complaint should be successfully submitted and a confirmation message displayed: *“Your complaint has been submitted successfully.”*
Actual Result	Pass
Status	Pass
Remarks	Positive test case for complaint creation.

Table 4: Test Case for Creating a Complaint

Test Id	004
Test Condition	Verify that an Admin can add a new category from the Admin panel.
Test Steps	1. Login to Admin panel. 2. Navigate to **Category** section. 3. Click on **Add Category** button. 4. Enter a category name in the input field. 5. Click **Create** button. 6. Observe result.
Test Input	Category Name: *“Electrical”*
Test Expected Result	New category should be added successfully and a confirmation message displayed: *“Category created successfully.”*
Actual Result	Pass
Status	Pass
Remarks	Positive test case for adding a new category in Admin panel.

Table 5: Test Case for Adding a Category by Admin

Test Id	005
Test Condition	Verify that an Admin can resolve a complaint by providing solution details, selecting an estimated date, and assigning an employee.
Test Steps	1. Login to Admin panel. 2. Navigate to **Complaints** section. 3. Select a complaint from the list. 4. Click **"Resolve this"** button. 5. Enter solution details in the description field. 6. Select an estimated completion date. 7. Assign an employee from the dropdown list. 8. Click **Submit** button. 9. Observe result.
Test Input	Complaint: *"Computer not working"* Solution Details: *"Replaced power cable and restarted system."* Estimated Date: *2025-09-10* Employee: *"ABC"*
Test Expected Result	Complaint status should be updated to **Resolved**, employee assigned, and a confirmation message displayed: *"Complaint resolved successfully."*
Actual Result	Pass
Status	Pass
Remarks	Positive test case for complaint resolution by Admin.

Table 6: Test Case for Resolving a Complaint by Admin

ii. Black-box Testing

Definition: Black-box testing is a software testing method in which the tester evaluates the system without knowledge of its internal code or logic, focusing on inputs and expected outputs.

Example: In the Problem Fixer project, entering an invalid student roll number or a non-institutional email correctly triggers a validation error, ensuring only valid users can log in.

iii. White-box Testing

Definition: White-box testing involves examining the internal logic, code structure, and flow of the software to ensure that all paths, conditions, and loops function correctly.

Example: The student roll validation module was tested to verify that all conditional checks for department code, session ID, and gender are implemented correctly.

iv. Unit Testing

Definition: Unit testing is the process of testing individual components or modules of a software in isolation to ensure that each unit functions correctly.

Example: The login module, complaint submission module, admin resolution module, and category creation module were tested individually to verify they perform their specific functions.

v. Integration Testing

Definition: Integration testing checks whether multiple modules or components of the system work together correctly.

Example: End-to-end workflows such as *Complaint submission → Admin assignment → Employee resolution* and *Student login → Roll validation → Complaint submission* were executed to ensure proper interaction between modules.

vi. Subsystem Testing

Definition: Subsystem testing involves testing a complete subsystem as a whole to ensure all integrated components function correctly and data flows consistently.

Example: The Student Module, Admin Module, and Employee Module were tested to verify consistent data flow and correct subsystem behavior.

vii. Acceptance Testing

Definition: Acceptance testing is performed from the end-user perspective to determine if the system meets the specified requirements and is ready for deployment.

Example: Both students and admins successfully performed tasks like logging in, submitting complaints, resolving issues, and managing categories, confirming the system meets the project requirements.

7 Deployment

The deployment of the **Problem Fixer** project involves specifying the software environment, platform requirements, and version control management used during development and deployment.

1. Software Specification

- **Project Name:** Problem Fixer
- **Purpose:** A complaint management system for students, teachers, and administrators to efficiently submit, track, and resolve complaints.
- **Programming Language:** JavaScript
- **Frameworks and Libraries:** React.js for frontend, Node.js and Express.js for backend, Prisma ORM for database management, TailwindCSS and shadcn/ui for UI styling and components.
- **Database:** MySQL
- **Key Features:**
  - Login using institutional email.
  - Student roll/ID validation against email and department code.
  - Complaint creation and submission.
  - Admin complaint resolution with employee assignment.
  - Category management for complaints.
  - Real-time notifications for updates.

2. Platform Specification

- **Operating System:** Windows
- **Deployment Platform:** Vercel (for frontend and backend hosting)
- **Web Browsers Tested:** Google Chrome, Microsoft Edge
- **Server Requirements:** Node.js runtime environment for executing backend services.
- **Client Requirements:** Modern web browser with JavaScript enabled.

3. Version Control Management

- **Version Control System:** Git, with code hosted on GitHub.
- **Branching Strategy:**
  - *main* branch: Stable, production-ready code.
  - *development* branch: Features under development and testing.
- **Commit Practices:** Descriptive commit messages were maintained to track changes effectively.
- **Deployment Workflow:** Code changes were pushed to GitHub, integrated and deployed automatically to Vercel, ensuring continuous delivery and version tracking.

Additional Notes

The deployment setup ensures cross-browser compatibility, responsive UI, and a smooth user experience. Using Vercel provides fast hosting and continuous integration with GitHub, while Prisma ORM ensures secure and efficient database operations. TailwindCSS and shadcn/ui facilitate modern and consistent design across all pages.

8 Discussion and Conclusion

The development of **Problem Fixer** has successfully demonstrated the implementation of a comprehensive complaint management system for students, teachers, and administrators.

Key functionalities, such as login using institutional emails, student roll/ID validation, complaint creation, category management, and complaint resolution by administrators, were implemented and thoroughly tested.

Through extensive testing, including black-box, white-box, unit, integration, subsystem, and acceptance testing, the system was verified to function correctly and meet the specified requirements. Positive and negative test cases were handled effectively, ensuring the reliability and robustness of the system.

The deployment on Vercel, along with version control using GitHub, provided a stable and accessible environment, compatible with modern web browsers such as Google Chrome and Microsoft Edge. Overall, **Problem Fixer** provides a reliable, user-friendly platform that streamlines the complaint management process in an academic environment.

Outcome

- Students can log in using their institutional email and add validated roll/ID information.
- Users can submit complaints with all required details, including building, category, room number, and description.
- Admins can resolve complaints by providing solutions, assigning employees, and updating statuses.
- Administrators can manage complaint categories efficiently.
- Notifications and confirmation messages improve user engagement and awareness.

Limitation

- The system requires an active internet connection as it is hosted online on Vercel.
- Currently, the system supports only institutional email login (*@nstu.edu.bd*).
- Role-based features are limited to students and admins; employee access is basic.
- Some advanced reporting or analytics features are not implemented in this version.

Future Scope

- Integration of multi-role support including employees with detailed dashboards.
- Mobile application development to improve accessibility for students and administrators.
- Advanced reporting and analytics for complaint statistics and departmental performance.
- Integration with SMS/email notifications for real-time updates.
- Implementation of AI-based categorization or prioritization of complaints to enhance efficiency.
- Addition of an emergency mental health support field to allow students to quickly report urgent psychological or well-being concerns.