****

**GRAVITYEDU**

**INCIDENT MANAGEMENT SYSTEM**

**FOR BANKING SECTOR**

**GROUP 11**

**Master of Information Technology (MIT)**

**Department of Industrial Management** - **Faculty of Science**

**University of Kelaniya**

**September 2025**

# Team Members

|  |  |
| --- | --- |
| **Name** | **Registration Number** |
| **AE Jayakodi** | **FGS/MIT/2024/014** |
| **DGAV Senarathna** | **FGS/MIT/2024/039** |
| **HMDC Bandara** | **FGS/MIT/2024/056** |
| **KHA Ayesha** | **FGS/MIT/2024/057** |

# Abstract

GravityEdu is a private educational institute located in Nugegoda, providing academic support for Advanced Level (A/L) students. While the institute has established a strong reputation for quality teaching and personalized guidance, its administrative processes are still managed manually using spreadsheets and paper-based records. This reliance on traditional methods has resulted in inefficiencies, errors, and delays in key tasks such as student enrollment, fee management, class scheduling, and performance tracking.

To address these challenges, this project proposes the development of the GravityEdu Management System, a user-friendly and scalable application powered by a Python-based technology stack with a MySQL database. The system is designed to centralize and automate core administrative operations, including user authentication, teacher and student management, class scheduling, payment tracking, attendance, and academic resource distribution. Additional features such as bulk data uploads and automated report generation further enhance the system’s efficiency and usability.

By streamlining administrative workflows, minimizing manual errors, and providing real-time insights through reporting, the system reduces operational overhead and improves overall accuracy. The implementation of this system not only enhances transparency and accountability but also allows the institute to focus more on its primary mission—delivering high-quality education and personalized support to students.

# Table of Contents

[Team Members 2](#_Toc209793631)

[Abstract 3](#_Toc209793632)

[Table of Contents 4](#_Toc209793633)

[Chapter 1: INTRODUCTION 6](#_Toc209793634)

[1.1 About GRAVITY 6](#_Toc209793635)

[1.2 Problem Statement 6](#_Toc209793636)

[1.3 Motivation for the Project 7](#_Toc209793637)

[1.4 Objectives 8](#_Toc209793638)

[1.5 Project Scope 9](#_Toc209793639)

[Chapter 2: ANALYSYS 10](#_Toc209793640)

[2.1 Requirement gathering techniques 10](#_Toc209793641)

[2.2 Proposed System 11](#_Toc209793642)

[2.3 Process model for the proposed system 13](#_Toc209793643)

[ Iterative 13](#_Toc209793644)

[ simple prototype to make final Application 13](#_Toc209793645)

[2.4 project management 13](#_Toc209793646)

[2.4.1 Backlog 14](#_Toc209793647)

[2.4.2 communication and regular meetings 16](#_Toc209793648)

[2.4.3 Timeline 17](#_Toc209793649)

[2.5 HIGH LEVEL USE CASE 18](#_Toc209793650)

[2.5.1 Use Case Analysis 18](#_Toc209793651)

[2.5.2 Activity diagram 19](#_Toc209793652)

[2.7 Functional Requirements 23](#_Toc209793653)

[2.8 Non-Functional Requirements 23](#_Toc209793654)

[Chapter 3: DESIGN 24](#_Toc209793655)

[3.1 Development Tools 24](#_Toc209793656)

[3.1.1 Programming Languages and Frameworks 24](#_Toc209793657)

[3.2 System Architecture 25](#_Toc209793658)

[3.2.1 MVC Architecture 25](#_Toc209793659)

[3.3 Security Implementation 26](#_Toc209793660)

[3.4 Performance Considerations 26](#_Toc209793661)

[3.6 Testing 27](#_Toc209793662)

[3.7 Deployment Strategy 28](#_Toc209793663)

[3.8 Design architecture 29](#_Toc209793664)

[3.9 Class Diagram 30](#_Toc209793665)

[3.10 Database Design 31](#_Toc209793666)

[3.11 iNTERFACE Design 31](#_Toc209793667)

[3.11.1 User interface 35](#_Toc209793668)

[Chapter 4: Conclusion 50](#_Toc209793669)

[List of Figures 50](#_Toc209793670)

# Chapter 1: INTRODUCTION

## About GRAVITYEDU

GravityEdu is a well-established private educational institute located in Nugegoda, dedicated to providing high-quality after-school academic support for Advanced Level (A/L) students. Over the years, it has built a strong reputation for academic excellence, personalized guidance, and a nurturing learning environment, making it a trusted choice among both students and parents.

However, despite its strengths in teaching and student engagement, the institute still relies heavily on traditional methods such as spreadsheets, manual records, and paper-based processes for critical administrative functions. Student enrollment, fee management, and class scheduling are currently handled manually, leading to inefficiencies, errors, and delays in day-to-day operations. These challenges have highlighted the need for a more efficient and reliable management system.

## Problem Statement

The reliance on manual processes has created several operational challenges:

* High risk of data entry errors and inconsistencies.
* Increased administrative workload, resulting in delays.
* Limited scalability to manage a growing student base.
* Difficulty in tracking student performance and generating accurate reports.

Such inefficiencies not only slow down workflows but also affect the overall service quality offered by the institute.

## Motivation for the Project

Educational institutes like GravityEdu handle a wide range of administrative tasks from student enrollments and class scheduling to fee collection and performance tracking. Despite its strong academic capabilities, the institute currently relies heavily on manual, paper-based processes and spreadsheets for these operations.

This traditional approach leads to delays, data inconsistencies, difficulty in monitoring student progress, and increased administrative workload. Managing growing student numbers and expanding academic programs becomes increasingly complex under such a system. Without automation, tracking information accurately, generating reports, and ensuring smooth coordination between departments can be cumbersome and error-prone.

The motivation behind developing the GravityEdu Management System is to modernize administrative operations and reduce reliance on manual processes. By providing a centralized, user-friendly platform, the system aims to automate routine tasks, improve data accuracy, and streamline workflows.

Additionally, the system seeks to enhance transparency, accountability, and efficiency in day-to-day operations. With real-time access to student records, class schedules, fee status, and academic performance, administrators can make timely and informed decisions, transforming administrative management into a structured, efficient, and strategic function.

## Objectives

The primary objective of the GravityEdu Management System project is to design and implement a **user-friendly, scalable, and efficient application** that automates and streamlines administrative processes. The system aims to reduce manual workload, improve data accuracy, and provide a centralized platform for managing day-to-day operations.

The specific objectives include:

1. **Automate Student Enrollment** – Simplify registration and maintain accurate student records.
2. **Simplify Class Scheduling** – Create conflict-free timetables and optimize resource usage.
3. **Efficient Fee Management** – Track payments, discounts, and pending fees with transparency.
4. **Performance Monitoring** – Systematically track student academic progress and generate reports.
5. **Centralized Data Management** – Consolidate student, teacher, and class information for better accessibility.
6. **Enhance Operational Efficiency** – Automate repetitive tasks, reduce errors, and streamline administrative workflows.

## Project Scope

The GravityEdu Management System is designed to automate and streamline the institute’s academic and administrative operations. The system scope includes:

**1. User Authentication**

* Secure login for administrators and users.
* Role-based access control to ensure proper authorization.

**2. Teacher Management**

* Register teachers with details such as Name, Subject, Contact Number, and Email.
* View and edit teacher information.

**3. Student Management**

* Register students with Name, Registration Year & Month, Contact Number, Discount %, Email, and Stream.
* View and edit student information.

**4. Class Management**

* Manage classrooms and class types (Group or Hall).
* Categorize classes as Theory or Revision.
* Schedule class sessions, assigning Teachers, Subjects, Class Type, Category, and Time Slots.

**5. Payment Management**

* Record monthly tuition fee payments, including discounts and exemptions.
* Verify and track class fee payments.

**6. Student and Tutte Tracking**

* Update tutte distribution.
* Track student attendance.

**7. Bulk Operations**

* CSV upload of student details for efficient data entry.
* CSV upload for monthly payment status updates.

**8. Reports and Monitoring**

* Generate outstanding payment reports.
* Provide total income and outstanding balance reports.

By implementing these functionalities, the system aims to centralize operations, enhance efficiency, reduce errors, and provide administrators with a reliable platform to manage GravityEdu’s academic and financial activities effectively.

# Chapter 2: ANALYSYS

## 2.1 Requirement gathering techniques

To ensure that the GravityEdu Management System meets the specific needs of its users and stakeholders, a combination of requirement gathering techniques was employed during the initial stages of development. These techniques helped collect accurate and relevant data regarding system functionality, user interface design, and alignment with the institute’s business objectives. The following methods were used:

**(a) Interviews**

One-on-one interviews were conducted with administrators, teachers, and support staff to understand their current challenges, expectations, and preferences. These interviews provided valuable insights into the difficulties faced when managing student enrollments, class scheduling, fee collection, and performance tracking. The feedback also highlighted the importance of transparency, ease of use, and automation in the proposed system.

**(b) Document Analysis**

Existing documents such as student enrollment records, class schedules, fee receipts, and manual attendance sheets were reviewed. This analysis helped in identifying recurring administrative issues, data redundancies, and areas where manual errors frequently occurred. Reviewing these records also provided clarity on the structure of data that needs to be digitized and streamlined within the new system.

**(c) Observation**

Observation sessions were carried out by shadowing administrative staff during routine tasks such as registering students, scheduling classes, and collecting payments. These observations revealed real-time bottlenecks, such as delays in cross-referencing records, difficulties in updating schedules, and challenges in handling bulk data entry. Insights from these sessions were directly used to design features that minimize workload and improve efficiency.

## 2.2 Proposed System

The proposed GravityEdu Management System is designed to overcome the limitations of the current manual processes by introducing an automated, centralized, and user-friendly solution. The system will integrate key administrative and academic functions into a single platform, ensuring efficiency, accuracy, and scalability. The major components of the proposed system are as follows:

* **User Authentication**

The system will provide secure login facilities for both administrators and staff members. Role-based access control will be implemented, ensuring that users can only access features relevant to their responsibilities, thereby maintaining data security and integrity.

* **Teacher Management**

Administrators will be able to register new teachers by entering details such as Name, Subject, Contact Number, and Email. The system will also allow authorized users to view and edit teacher information when updates are required, ensuring that records remain accurate and up to date.

* **Student Management**

The system will streamline student enrollment by capturing essential details, including Name, Registration Year & Month, Contact Number, Discount Percentage, Email, and Stream. Administrators will be able to view and edit student information, ensuring that student records are easily maintained and updated.

* **Class Management**

The system will provide features to manage classrooms, class types (Group or Hall), and categories (Theory or Revision). It will allow scheduling of class sessions by assigning a teacher, subject, class type, category, and time slot. This will help avoid scheduling conflicts and ensure optimal use of resources.

* **Payment Management**

The system will support the recording of monthly tuition fee payments while handling discounts and exemptions as needed. Administrators will be able to verify fee payments and track outstanding balances, ensuring financial accuracy and transparency.

* **Student and Tutte Tracking**

The system will enable updates on tutte distribution and record student attendance. This ensures that academic resources are distributed fairly and that participation levels are systematically tracked.

* **Bulk Operations**

To minimize data entry workload, the system will allow bulk uploads through CSV files. Student details and monthly payment status updates can be imported in batches, saving time and reducing errors.

* **Reports and Monitoring**

The system will generate key reports to support decision-making and monitoring. These include outstanding payments reports, total income and outstanding balances, and student registration reports. Such reports provide administrators with real-time insights into financial and academic trends.

## 2.3 Process model for the proposed system

* Iterative

The InciTrack system follows an iterative development process, allowing for continuous improvement and refinement throughout the development lifecycle. Each iteration focuses on specific features and functionalities, starting with core incident management capabilities and gradually expanding to include advanced features like analytics and reporting. This approach enables the development team to gather feedback early and often, making necessary adjustments to meet user requirements effectively. The iterative process also facilitates better risk management, as potential issues can be identified and addressed in early stages, reducing the likelihood of major problems in later phases. Regular testing and validation ensure that each iteration meets quality standards and business objectives.

* simple prototype to make final Application

The development of InciTrack began with a simple prototype focusing on basic incident reporting and tracking functionalities. This initial version served as a foundation for gathering user feedback and understanding core requirements. As the system evolved, additional features were incrementally added, including role-based access control, advanced analytics, and comprehensive reporting capabilities. This evolutionary approach allowed for continuous refinement of the user interface and system architecture while maintaining stability and performance. The final application emerged through multiple iterations of enhancement and optimization, resulting in a robust and user-friendly incident management system that meets all specified requirements and industry standards

## 2.4 project management

Effective project management was crucial to the success of the GravityEdu system. Key

aspects of the project management contribution are outlined below.

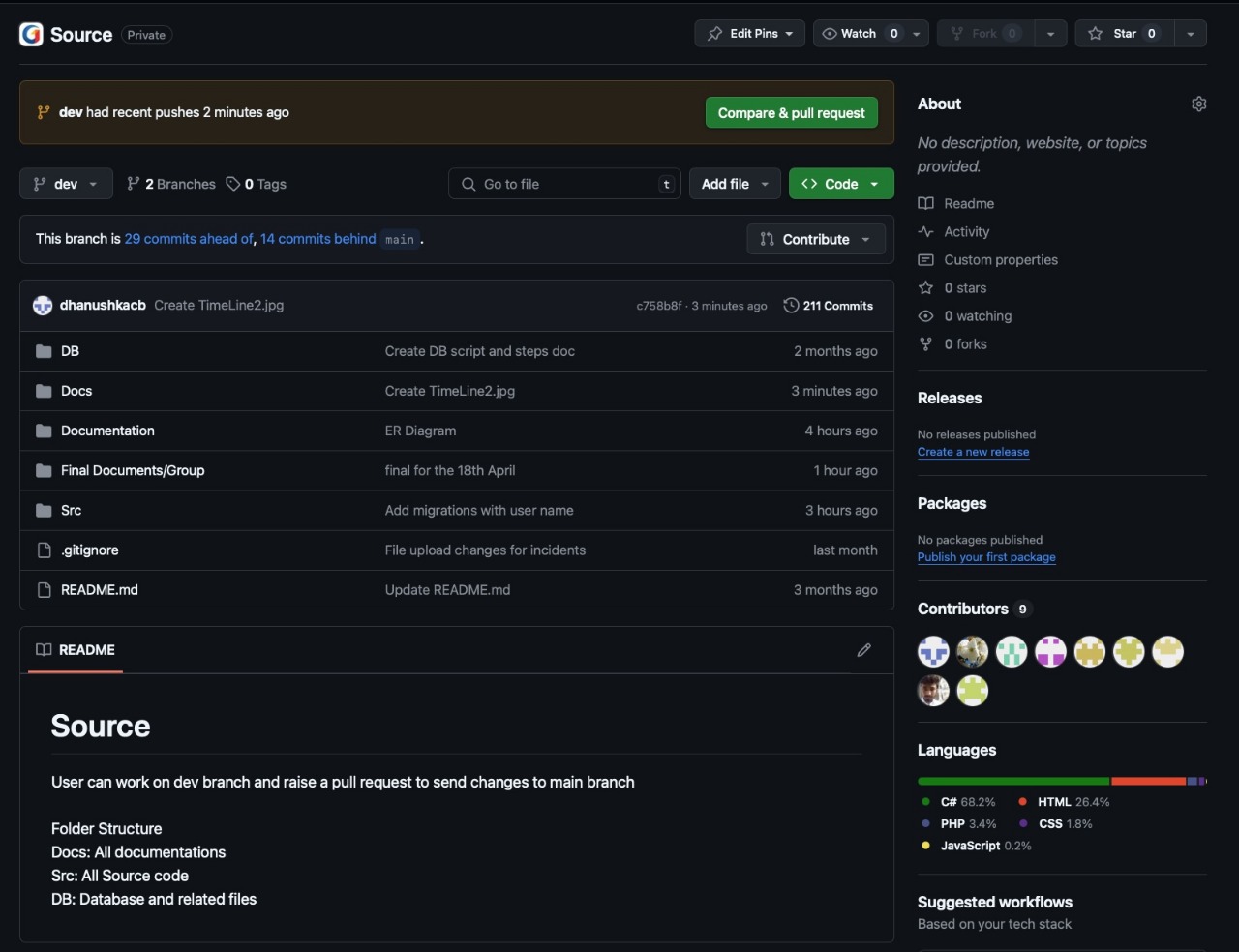


Figure : Git Hub Repository

### 2.4.1 Backlog

To facilitate transparent and efficient workflow tracking, we utilized GitHub’s project

management features. Tasks were organized into project boards, with each board representing a specific development phase. Issues were created for individual tasks, assigned to relevant team members, and tracked using GitHub’s timeline and status indicators. This system allowed the entire team to stay updated on progress, address bottlenecks quickly, and

maintain clear accountability.

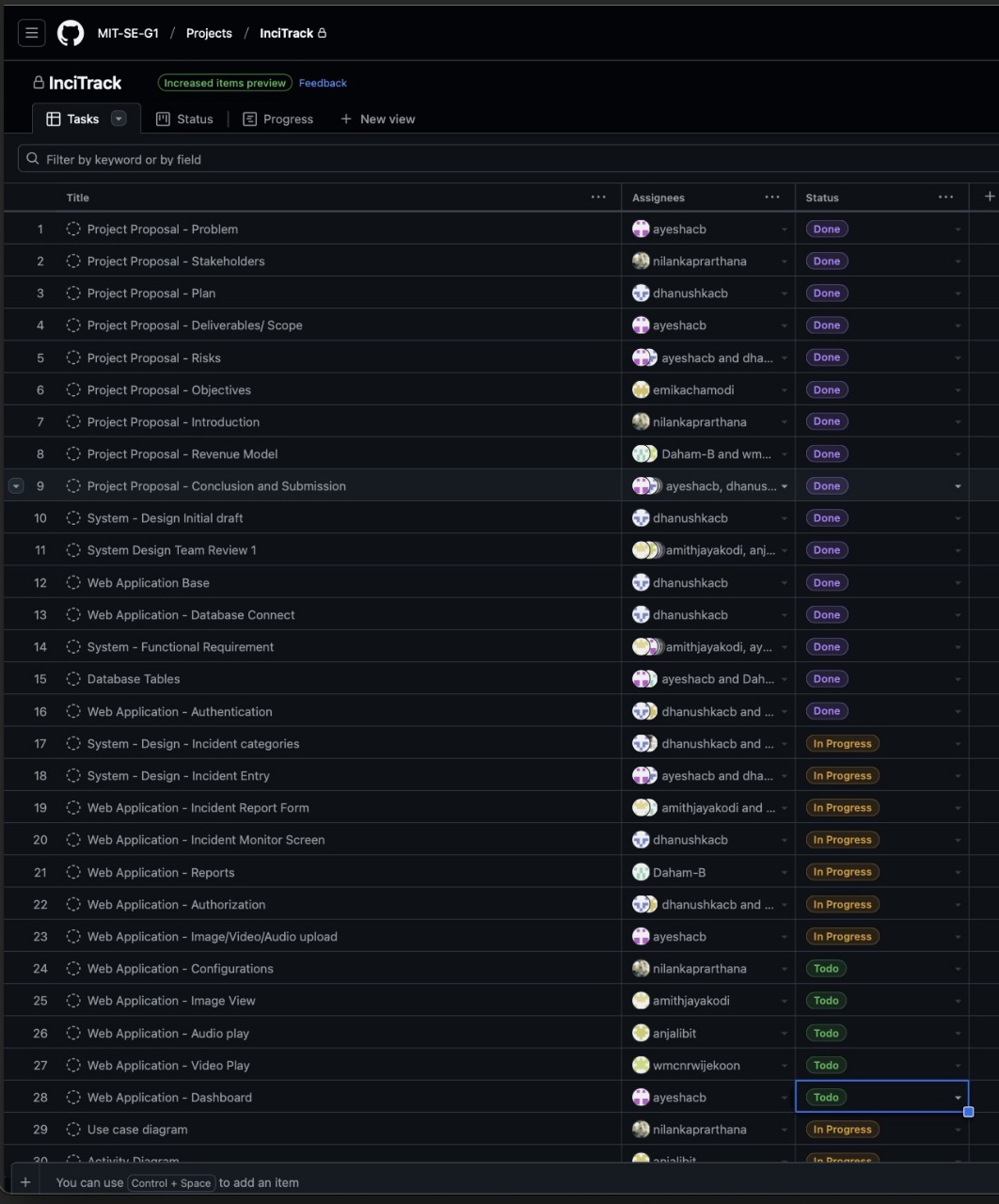


Figure : Backlog

### 2.4.2 communication and regular meetings

These sessions were conducted regularly to monitor progress, resolve challenges, and ensure alignment among all contributors.

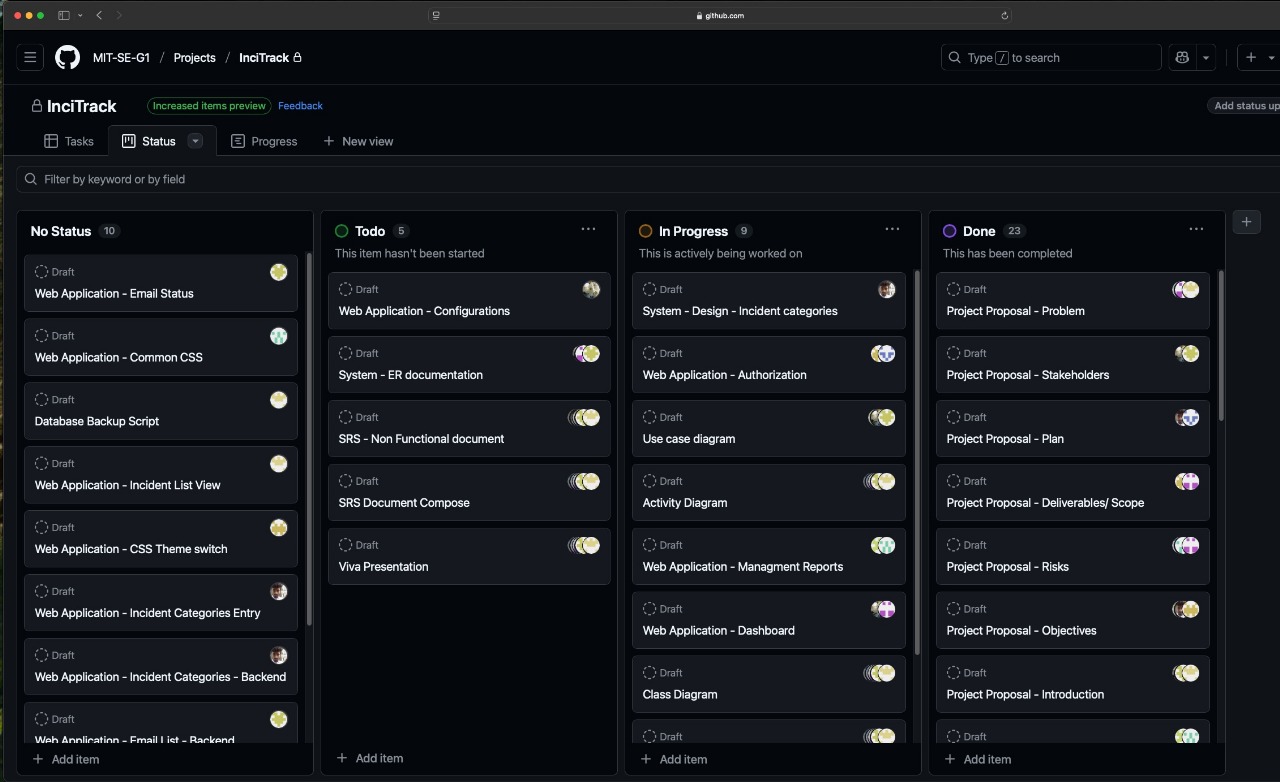


Figure : Task Progress

### 2.4.3 Timeline

This included defining major milestones such as design finalization, backend and frontend

integration, user testing, and deployment. We updated the timeline based on ongoing progress and feedback, ensuring that deadlines remained realistic while keeping the team focused on timely delivery.

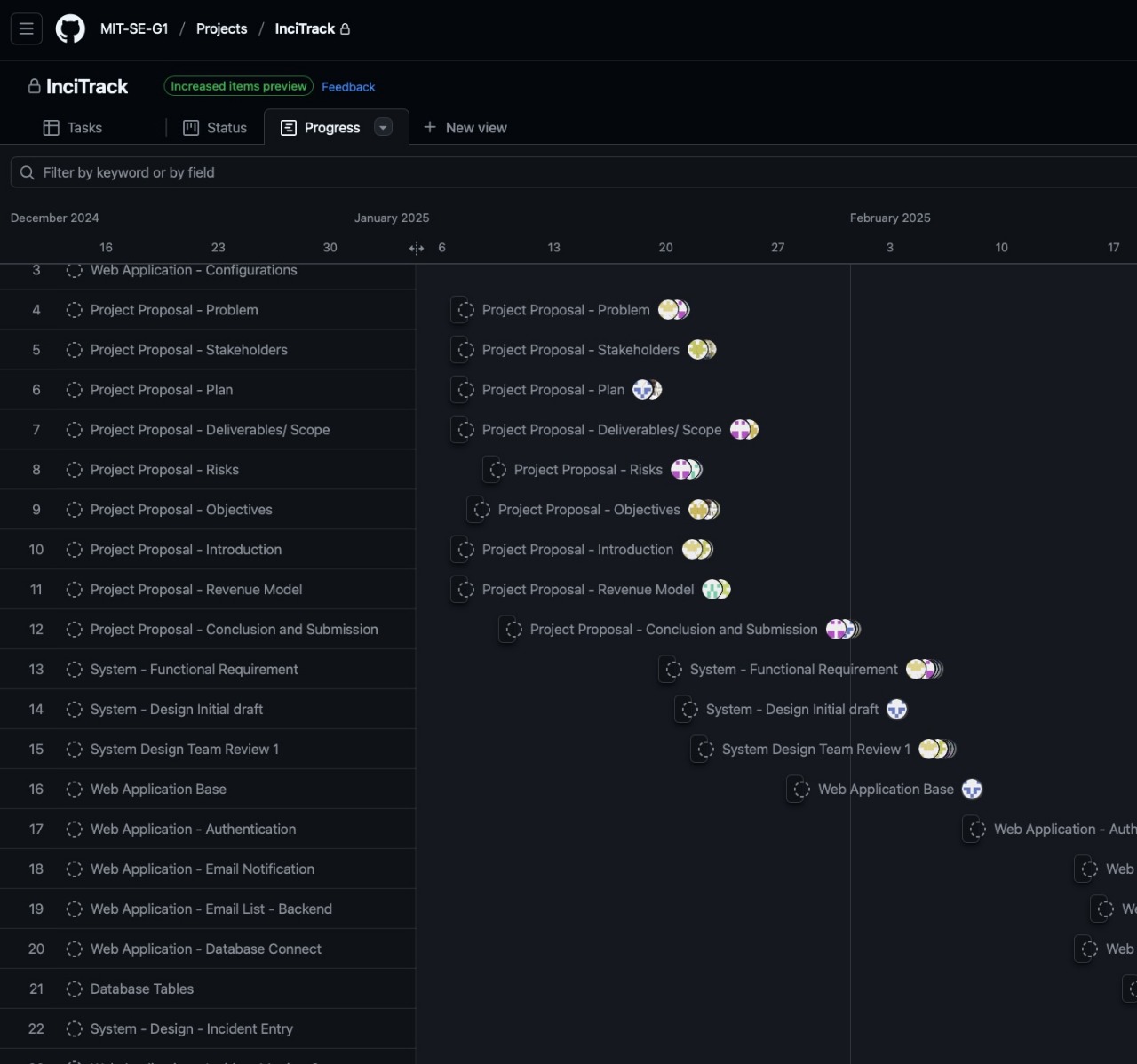


Figure : Task Timeline

## 2.5 HIGH LEVEL USE CASE

### 2.5.1 Use Case Analysis

The use case diagram illustrates the core functionalities of the Institute Management System with two primary actors, Staff and Admin. Staff manage academic and operational activities such as student registration (including stream assignment), teacher registration (with subject assignment), class scheduling (assigning teacher, classroom, and time slot), setting fees, recording and verifying student payments through multiple methods (cash, card, online), and bulk data imports. Admin oversees system-level tasks, including creating and managing user roles. The diagram highlights how various processes are interconnected through include relationships, ensuring smooth integration of registration, scheduling, payment, and role management within the system.

A diagram of a teacher

AI-generated content may be incorrect.

Figure : Highlevel Use Case Diagram

### 2.5.2 Activity diagram

The activity diagrams in this section detail the step-by-step flow of processes within the InciTrack system. These diagrams break down complex operations into manageable sequences of actions, showing the decision points and parallel activities that occur during incident management. The diagrams help understand the workflow of various processes, such as incident reporting, category management, and resolution procedures. They provide a clear visualization of how different system components interact and how data flows through the system during various operations.

**INCIDENT CREATION**

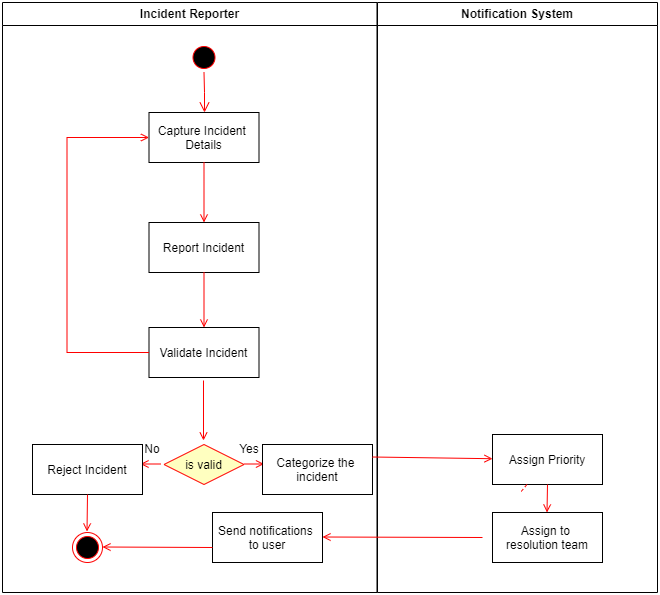


Figure : Incident Creation

**INCIDENT VIEW**

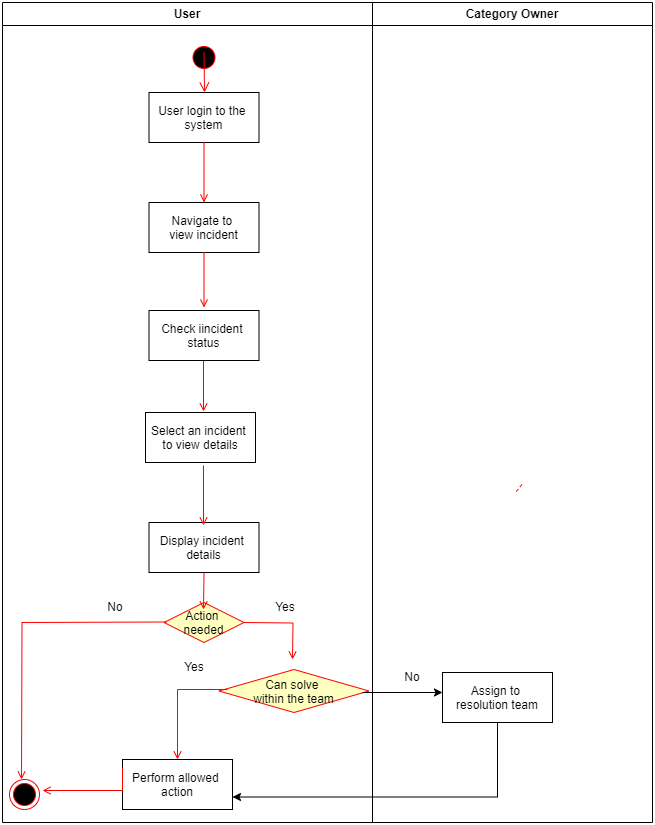


Figure : Incident View

**INCIDENT CLOSE**

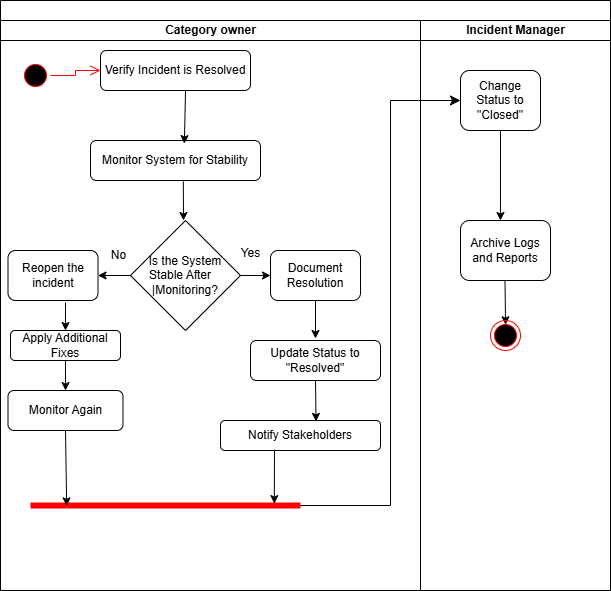


Figure : Incident Close

Each diagram type serves a specific purpose in documenting the system's design and functionality, providing different perspectives on how the InciTrack system operates and interacts with its users.

2.7 Functional Requirements

* The system must provide secure login with role-based access control.
* The system must allow registration and management of teacher details.
* The system must allow student registration and record updates.
* The system must support class creation, scheduling, and categorization.
* The system must handle monthly tuition fee payments with discounts and exemptions.
* The system must record student attendance and tutorial distribution.
* The system must support bulk uploads of student data and payment records via CSV.
* The system must generate reports on payments, income, and student registration.

2.8 Non-Functional Requirements

* **Performance**: The system **shall** be capable of supporting 100+ concurrent users efficiently.
* **Scalability**: The system **shall** support multiple tier levels to accommodate growth and increased demand.
* **Security**: The system **shall** incorporate encrypted credentials, role-based access control, and ensure GDPR compliance.
* **Usability**: The system **shall** feature a responsive user interface with intuitive and easy navigation.
* **Maintainability**: The system **shall** adopt modular code design to facilitate smooth updates and enhancements.
* **Reliability**: The system **shall** achieve 99.9% uptime and perform daily backups to ensure data safety and availability.

# Chapter 3: DESIGN

The InciTrack system is built on MVC architectural design pattern using ASP.NET Core 8, MySQL, and responsive frontend technologies. The design emphasizes security, scalability, and user experience while maintaining clean separation of concerns. The system architecture supports efficient incident management, real-time monitoring, and comprehensive reporting capabilities through carefully selected technologies and design patterns.

## 3.1 Development Tools

**1.Visual Studio Code (VSCode)**

- Primary IDE for development

- Extensions used:

- C# Dev Kit

- .NET Core Tools

- Git Integration

**2. MySQL Workbench**

- Database design and management

- Query development and optimization

- Schema visualization

**3. GitHub**

- Version control system

- Code repository management

- Collaboration platform

### 3.1.1 Programming Languages and Frameworks

**1. C#.NET 8**

- Backend development

- ASP.NET Core MVC framework

- Entity Framework Core

- Authentication and authorization

**2. HTML5/CSS3**

- Frontend structure and styling

- Responsive design

- Modern UI components

- Cross-browser compatibility

**3. JavaScript**

- Client-side interactivity

- AJAX requests

- Dynamic content updates

- Chart.js integration

- Form validation

**4. MySQL**

- Relational database management

- Data storage and retrieval

- Transaction management

- Query optimization

## 3.2 System Architecture

### 3.2.1 MVC Architecture

**1. Model**

- Entity Framework Core

- MySQL database

- Stored procedures

- Data migrations

**2. View**

- ASP.NET Core MVC

- Razor views

- JavaScript/jQuery

- Bootstrap framework

**3. Controller**

- C# services

- Repository pattern

- Business rules implementation

- Data validation

## 3.3 Security Implementation

**Authentication**

- ASP.NET Core Identity

- Role-based access control

- Password hashing

**Authorization**

- Role-based authorization

- Policy-based authorization

## 3.4 Performance Considerations

**Database Optimization**

- Indexing strategy

- Query optimization

- Connection pooling

- Caching implementation

**Application Performance**

- Response compression

- Static file caching

- Lazy loading

## 3.6 Testing

| **Test Case ID** | **Module** | **Test Scenario** | **Test Steps** | **Expected Result** |
| --- | --- | --- | --- | --- |
| UAT-01 | User Authentication | Login with valid admin credentials | 1. Open login page2. Enter valid admin username and password3. Click Login | Admin is successfully logged in and redirected to the dashboard |
| UAT-02 | User Authentication | Login with invalid credentials | 1. Enter incorrect username or password2. Click Login | System displays an error message and prevents login |
| UAT-03 | Teacher Management | Add a new teacher | 1. Navigate to Teacher Management2. Click Add Teacher3. Enter Name, Subject, Contact No, Email4. Click Save | New teacher is added and visible in the teacher list |
| UAT-04 | Teacher Management | Edit teacher details | 1. Select a teacher from the list2. Click Edit3. Update details4. Save changes | Teacher information is updated correctly |
| UAT-05 | Student Management | Add a new student | 1. Navigate to Student Management2. Click Add Student3. Enter Name, Registration Year & Month, Contact No, Discount %, Email, Stream4. Save | Student is added and visible in the student list |
| UAT-06 | Student Management | Edit student information | 1. Select a student2. Click Edit3. Update details4. Save | Student information is updated successfully |
| UAT-07 | Class Management | Schedule a new class | 1. Navigate to Class Management2. Enter Classroom, Class Type, Category, Teacher, Subject, Time Slot3. Save | Class is scheduled and appears in the timetable without conflicts |
| UAT-08 | Payment Management | Record student fee | 1. Go to Payment Management2. Select student3. Enter payment details and discount if any4. Save | Payment is recorded and balance updated correctly |
| UAT-09 | Attendance Tracking | Mark attendance | 1. Go to Student and Tute Tracking2. Select class session3. Mark present/absent for each student4. Save | Attendance is recorded and updated for the session |
| UAT-10 | Bulk Operations | CSV upload for students | 1. Go to Bulk Operations2. Upload CSV containing multiple student records3. Submit | System successfully imports student records and displays them in the list |
| UAT-11 | Reports | Generate outstanding payments report | 1. Navigate to Reports2. Select Outstanding Payments3. Generate report | Report is displayed correctly with accurate outstanding balances |

## 3.7 Deployment Strategy

**Development Workflow**

1. Version Control

* Feature branching
* Pull requests
* Code review process
* Continuous integration

2. Deployment Pipeline

* Database migrations
* Environment configuration

**Deployment Environments**

* Development
* Production
* Backup and recovery

This technology stack and design approach ensures a robust, scalable, and maintainable incident management system that meets modern development standards and best practices.

## 3.8 Design architecture



Figure 9: Design Architecture

## 3.9 Class Diagram

The class diagrams presented here show the static structure of the InciTrack system, illustrating the relationships between different classes and their attributes. These diagrams provide a detailed view of the system's object-oriented design, showing how different entities like incidents, categories, and users are related to each other. The diagrams help understand the system's data model and the inheritance relationships between different classes. They serve as a blueprint for the system's implementation, showing how different components interact at the code level.

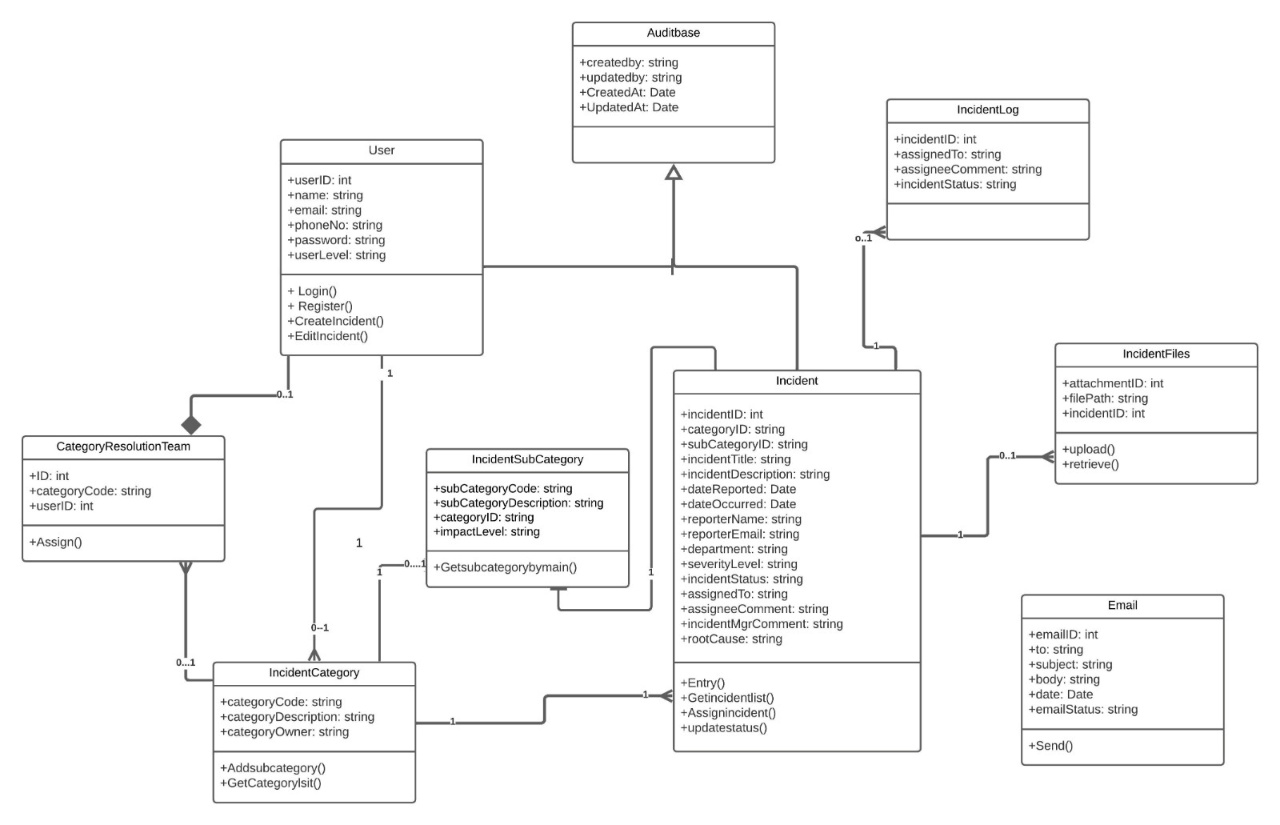


Figure 10 :Class diagram for InciTrack

## 3.10 Database Design

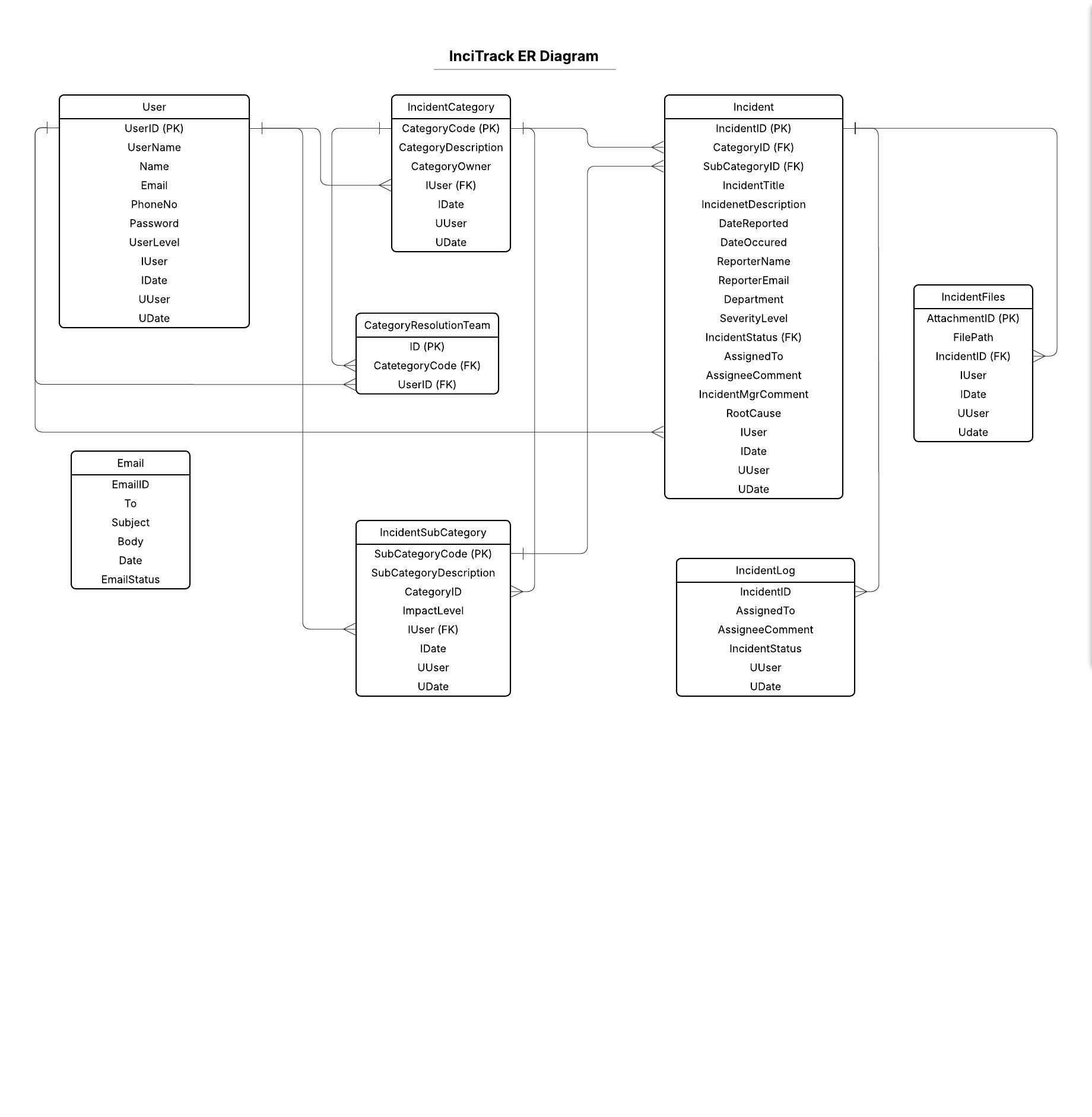


Figure 11: ER Diagram InciTrack

## 3.11 iNTERFACE Design

**CRAP stands for:**

1. Contrast: Creating visual hierarchy and emphasis

2. Repetition: Consistent use of design elements

3. Alignment: Proper organization and structure

4. Proximity: Grouping related elements together

**Applying CRAP to InciTrack**

**1. Contrast in InciTrack**

The InciTrack system implements contrast through:

**Color Usage :**

* Critical incidents: Red highlights
* Normal incidents: Blue/neutral colors
* Resolved incidents: Green indicators

**Typography :**

* Headers: Bold, larger font
* Body text: Regular weight
* Important alerts: Highlighted text

**Visual Hierarchy :**

* Dashboard cards with varying sizes
* Status indicators with different colors
* Priority levels with distinct visual cues

**2. Repetition in InciTrack**

Consistent design elements throughout the system:

**- Navigation :**

* Uniform menu structure
* Consistent button styles
* Standard icon usage

**- Forms :**

* Consistent input field styling
* Uniform validation messages
* Standard button placement

**- Tables :**

* Consistent column headers
* Uniform row styling
* Standard pagination controls

**3. Alignment in InciTrack**

Proper organization of interface elements:

**- Grid System :**

- Consistent spacing between elements

- Aligned form fields

- Organized dashboard layout

**- Content Structure :**

- Left-aligned text for readability

- Centered headings

- Left-aligned action buttons

**- Data Presentation :**

- Aligned table columns

- Consistent chart layouts

- Organized report formats

**4. Proximity in InciTrack**

Logical grouping of related elements:

**- Dashboard Layout :**

- Related metrics grouped together

- Status indicators near relevant data

- Action buttons close to related content

**- Form Design :**

- Related fields grouped in sections

- Labels close to input fields

- Validation messages near relevant fields

**- Navigation :**

- Related menu items grouped

- Contextual actions near content

- Related tools clustered together

**Benefits of CRAP in InciTrack**

**1. Improved Usability :**

- Clear visual hierarchy

- Intuitive navigation

- Easy-to-scan interfaces

**2. Enhanced User Experience :**

- Consistent interaction patterns

- Predictable interface behavior

- Reduced cognitive load

**3. Professional Appearance :**

- Polished, cohesive design

- Professional look and feel

- Trust-building interface

**4. Efficient Information Processing :**

- Quick data scanning

- Easy pattern recognition

- Clear information hierarchy

The application of CRAP principles in InciTrack results in a professional, user-friendly interface that enhances usability and user satisfaction while maintaining a consistent and organized appearance throughout the system.

### 3.11.1 User interface

1. User login

A screenshot of a login screen

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | User Login |
| **Trigger** | The user enters their credentials and clicks the "Login" button. |
| **Input** | - Username |
| - Password |
| **Processing** | 1. System checks that both fields are filled. |
| 2. Login credentials are securely sent to the authentication service. |
| 3. Authentication service compares credentials with the user database. |
| 4. If valid, generate a session/token and load user-specific data. |
| 5. If invalid, return an error message. |
| **Output** | - Success: User is redirected to the dashboard. |
| - Failure: Error message displayed. |
| **Preconditions** | - User is registered in the system. |
| - Authentication service is operational. |
| - User account is active (not locked or disabled). |
| **Postconditions** | - Secure session is created. |
| - User is marked as "logged in". |
| - Audit log entry is recorded. |

1. User Registration

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | User Registration |
| **Trigger** | Admin clicks **"Add User"** or **"Delete User"** in the User Management module. |
| **Input** | - Username |
| - Password |
| - User Role |
| **Processing** | 1. System validates mandatory fields and correct formats. |
| 2. Checks for duplicate username when adding a user. |
| 3. Encrypts the password. |
| 4. Saves new user details into the database OR removes user record when deleting. |
| 5. Updates the user list in the system. |
| **Output** | **Save:** User is created and visible in the list. |
| **Clear:** clear the data input fields. |
| **Delete:** User removed and no longer visible in the list. |
| **Preconditions** | - Admin is logged in. |
| - Database is operational. |

1. Classroom Registration

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Classroom Registration |
| **Trigger** | Admin clicks **"Add Classroom"**, **"Delete Classroom"**, or **"Clear Fields"** in the Classroom Management module. |
| **Input** | - Classroom Name/Number |
| - Capacity |
| - Location (optional) |
| **Processing** | 1. System validates mandatory fields and correct formats. |
| 2. **Add Classroom**: Checks for duplicate classroom entries → Saves classroom details into the database. |
| 3. **Delete Classroom**: Confirms classroom exists → Deletes classroom record from the database. |
| 4. **Clear Fields**: Resets all input fields on the form without saving. |
| 5. Updates classroom list in the system. |
| **Output** | - **Success (Add):** Classroom is created and visible in the list. |
| - **Failure (Add):** Duplicate classroom or validation errors. |
| - **Success (Delete):** Classroom removed and no longer visible in the list. |
| - **Failure (Delete):** Classroom not found. |
| - **Clear Fields:** Input form reset with empty fields. |

1. Add Student

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Add Student |
| **Trigger** | Admin clicks "Add Student" in the Student Management module. |
| **Input** | - Name |
| - Registration Year & Month |
| - Contact Number |
| - Discount % |
| - Email |
| - Stream |
| **Processing** | 1. System validates mandatory fields and correct formats. |
| 2. Checks for duplicate student entries. |
| 3. Saves the student details into the database. |
| 4. Updates student list in the system. |
| **Output** | - Success: Student is added and visible in the list. |
| - Failure: Validation errors are shown. |
| **Preconditions** | - Staff is logged in. |
| - Database is operational. |

1. Add Teacher

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Add/Edit Teacher |
| **Trigger** | Admin clicks "Add Teacher" or selects a teacher to edit in Teacher Management. |
| **Input** | - Name |
| - Subject |
| - Contact Number |
| - Email |
| **Processing** | 1. System validates mandatory fields. |
| 2. Checks for duplicate teacher entries. |
| 3. Saves or updates teacher information in the database. |
| 4. Updates teacher list view. |
| **Output** | - Success: Teacher information added/updated and visible in list. |
| - Failure: Validation error displayed. |
| **Preconditions** | - Admin is logged in. |
| - Database is operational. |
| **Postconditions** | - Teacher record is created or updated. |
| - Teacher list updated. |
| - Audit log entry recorded. |

1. Schedule Class

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Schedule Class |
| **Trigger** | Admin clicks "Schedule Class" in the Class Management module. |
| **Input** | - Teacher |
| - Class Type (Group/Hall) |
| - Category (Theory/Revision) |
| - Time Slot |
| - Classroom |
| - Fee (per student) |
| **Processing** | 1. System validates all fields are filled. |
| 2. Checks for conflicts in teacher assignment or classroom availability. |
| 3. Saves the class schedule in the database. |
| 4. Updates timetable view. |
| **Output** | - Success: Class is scheduled and visible in the timetable. |
| - Failure: Conflict message is shown. |
| **Preconditions** | - Admin is logged in. |
| - Teachers and classrooms are available. |
| **Postconditions** | - Class schedule is recorded. |
| - Timetable is updated. |
| - Audit log entry is recorded. |

1. Student Enrollment Management

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Student Enrollment Management |
| **Trigger** | Admin clicks **"Add Enrollment"**, **"Delete Enrollment"**, or **"Clear Fields"** in the Enrollment module. |
| **Input** | - Student |
| - Class |
| - Enrollment Date |
| **Processing** | 1. System validates mandatory fields and correct formats. |
| 2. **Add Enrollment**: Checks if the same student is already enrolled in the class → Saves enrollment details into the database. |
| 3. **Delete Enrollment**: Confirms enrollment exists → Removes the enrollment record from the database. |
| 4. **Clear Fields**: Resets all input fields on the form without saving. |
| 5. Updates the enrollment list in the system. |
| **Output** | - **Success (Add):** Enrollment is created and visible in the list. |
| - **Failure (Add):** Duplicate enrollment or validation errors. |
| - **Success (Delete):** Enrollment removed and no longer visible in the list. |
| - **Failure (Delete):** Enrollment not found. |
| - **Clear Fields:** Input form reset with empty fields. |
| **Preconditions** | - Staff is logged in. |
| - Database is operational. |

1. Record Payment

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Record Payment |
| **Trigger** | Admin clicks **"Save Payment"**, **"Delete Payment"**, or **"Clear Fields"** in the Payment Management module. |
| **Input** | - Student ID |
| - Class |
| -Month |
| -Year |
| - Amount Pay |
| - Payment Method |
| - Remarks |
| - **Class Fee (uneditable)** |
| - **Discount (uneditable)** |
| **Processing** | 1. System validates mandatory fields and correct formats. |
| 2. **Save Payment**: Checks for duplicate payment entry for the same student, class, and month → Calculates net payable (Class Fee – Discount) → Saves payment record in the database. |
| 3. **Delete Payment**: Confirms payment record exists → Removes the payment entry from the database. |
| 4. **Clear Fields**: Resets all input fields on the form (editable only). |
| 5. Updates the payment list in the system. |
| **Output** | - **Success (Save):** Payment is recorded and visible in the list. |
| - **Failure (Save):** Duplicate payment or validation errors. |
| - **Success (Delete):** Payment removed and no longer visible in the list. |
| - **Failure (Delete):** Payment not found. |
| - **Clear Fields:** Input form reset with empty editable fields. |
| **Preconditions** | - Staff is logged in. |
| - Database is operational. |
| - Student and Class records must exist. |

1. Payment Verification

A screen shot of a payment verification screen

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Payment Verification |
| **Trigger** | Admin clicks **"Verify Payment"** in the Payment Management module. |
| **Input** | - Student ID |
| - Class |
| - Month |
| - Year |
| **Processing** | 1. System validates mandatory fields and correct formats. |
| 2. Searches the database for an existing payment record for the same **Student ID + Class + Month + Year**. |
| 3. **If payment exists** → mark payment as verified, block duplicate entry for same combination. |
| 4. **If no payment exists** → return failure message. |
| **Output** | - **Success:** Payment is verified and flagged as completed. |
| - **Failure:** No payment found, or duplicate payment attempt is blocked. |
| **Preconditions** | - Admin is logged in. |
| - Database is operational. |
| - Student and Class records must exist. |

1. Import Student Record

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Import Student Details |
| **Trigger** | Staff clicks **"Process Import"** after selecting a CSV file in the Student Management module. |
| **Input** | - CSV File containing: |
| - Student Name |
| - Registration Year |
| - Registration Month |
| - Contact Number |
| - Discount Percentage |
| - Stream |
| **Processing** | 1. System validates the CSV file format and mandatory fields. |
| 2. Reads each row from the CSV file. |
| 3. Checks for duplicate student entries in the database. |
| 4. Inserts new student records into the database. |
| 5. Logs errors for any invalid rows and skips them. |
| 6. Updates the student list in the system after import. |
| **Output** | - **Success:** All valid student records are added and visible in the student list. |
| - **Partial Success:** Some records failed validation; errors logged. |
| - **Failure:** CSV file invalid or no valid records found. |
| **Preconditions** | - Admin is logged in. |
| - Database is operational. |
| - CSV file must be in correct format. |

1. Import Monthly Payments

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Import Monthly Payments |
| **Trigger** | Staff clicks **"Process Import"** after selecting a CSV file in the Payment Management module. |
| **Input** | - CSV File containing: |
| - Class ID |
| - Student ID |
| - Year |
| - Month |
| - Amount |
| - Payment Method |
| - Remarks |
| **Processing** | 1. System validates the CSV file format and mandatory fields. |
| 2. Reads each row from the CSV file. |
| 3. Checks for duplicate payment entries for the same **Student ID + Class + Month + Year**. |
| 4. Inserts new payment records into the database for valid rows. |
| 5. Logs errors for invalid rows (e.g., duplicate payment or missing fields) and skip them. |
| 6. Updates the payment list in the system after import. |
| **Output** | - **Success:** All valid payment records are added and visible in the payment list. |
| - **Partial Success:** Some records failed validation; errors logged. |
| - **Failure:** CSV file invalid or no valid records found. |
| **Preconditions** | - Admin is logged in. |
| - Database is operational. |

1. Import Attendance Record

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Import Attendance Records |
| **Trigger** | Staff clicks **"Process Import"** after selecting a CSV file in the Attendance Management module. |
| **Input** | - CSV File containing: |
| - Student ID |
| - Class ID |
| - Session Date |
| - Status (Present/Absent) |
| - Tute |
| **Processing** | 1. System validates the CSV file format and mandatory fields. |
| 2. Reads each row from the CSV file. |
| 3. Checks for duplicate attendance records for the same **Student ID + Class + Session Date**. |
| 4. Inserts new attendance records into the database for valid rows. |
| 5. Logs errors for invalid rows (e.g., missing fields or duplicates) and skips them. |
| 6. Updates the attendance list in the system after import. |
| **Output** | - **Success:** All valid attendance records are added and visible in the attendance list. |
| - **Partial Success:** Some records failed validation; errors logged. |
| - **Failure:** CSV file invalid or no valid records found. |
| **Preconditions** | - Admin is logged in. |
| - Database is operational. |
| - CSV file must be in correct format. |
| - Students and Classes referenced in CSV must exist in the system. |

1. Reports

Payment Outstanding Report

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Outstanding Payment Report |
| **Trigger** | User clicks **"Generate Report"** after selecting Class, Month, and Year in the Payment Report module. |
| **Input** | - Class |
| - Month |
| - Year |
| **Processing** | 1. System validates mandatory fields and correct formats. |
| 2. Retrieves all payment records for the selected Class, Month, and Year. |
| 3. Calculates outstanding amounts for each student (Class Fee – Paid Amount – Discount). |
| 4. Aggregates data into a report format (list or table). |
| 5. Displays the report to the user. |
| **Output** | - **Success:** Outstanding payment report displayed with student details and due amounts. |
| - **Failure:** Validation errors or no records found. |
| **Preconditions** | - User is logged in. |
| - Database is operational. |
| - Class, Month, and Year exist in the system. |

Attendance Sheet Report

A screenshot of a computer

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Attendance Sheet Report |
| **Trigger** | User clicks **"Generate Sheet"** after selecting Class ID and Date Range in the Attendance Report module. |
| **Input** | - Class ID |
| - Start Date |
| - End Date |
| **Processing** | 1. System validates mandatory fields and correct date range. |
| 2. Retrieves attendance records for the selected Class ID and date range. |
| 3. Aggregates and formats the attendance data (Present/Absent, Tute). |
| 4. Displays the attendance sheet to the user. |
| 5. **Optional Export:** If user clicks **"Export to File"**, system generates CSV or Excel file for download. |
| **Output** | - **Success:** Attendance sheet displayed with student attendance details. |
| - **Export Success:** Data exported to file successfully. |
| - **Failure:** Validation errors or no records found. |
| **Preconditions** | - User is logged in. |
| - Database is operational. |
| - Class and attendance data exist for the selected date range. |

Teacher Income Report

A screenshot of a report

AI-generated content may be incorrect.

|  |  |
| --- | --- |
| **Element** | **Description** |
| **Process Name** | Teacher Income Report |
| **Trigger** | User clicks **"Generate Report"** after selecting Month and Year in the Teacher Income module. |
| **Input** | Month |
| Year |
| **Processing** | 1. System validates mandatory fields and correct format. |
| 2. Retrieves payment records related to teachers for the selected Month and Year. |
| 3. Calculates total income for each teacher. |
| 4. Displays the report to the user. |
| 5. **Optional Export:** If user clicks **"Export"**, system generates CSV or Excel file for download. |
| **Output** | - **Success:** Teacher income report displayed with income details per teacher. |
| - **Export Success:** Data exported to file successfully. |
| - **Failure:** Validation errors or no records found. |
| **Preconditions** | - User is logged in. |
| - Database is operational. |
| - Payment records for teachers exist for the selected Month and Year. |

# Chapter 4: Conclusion

The GravityEdu Management System provides a practical solution to streamline the administrative and academic operations of the institute. By automating student enrollment, class scheduling, payment management, and performance tracking, the system reduces manual effort, minimizes errors, and ensures accurate record keeping. It also enhances transparency and improves the overall experience for administrators, teachers, and students. With its user-friendly design and scalability, the system lays a strong foundation for efficient educational management and supports GravityEdu’s mission of delivering quality education with improved operational efficiency.

# List of Figures

[Figure 1: Git Hub Repository 12](#_Toc196496602)

[Figure 2: Backlog 13](#_Toc196496603)

[Figure 3: Task Progress 14](#_Toc196496604)

[Figure 4: Task Timeline 15](#_Toc196496605)

[Figure 5: High-level Use Case Diagram 16](#_Toc196496606)

[Figure 6: Incident Creation 17](#_Toc196496607)

[Figure 7: Incident View 18](#_Toc196496608)

[Figure 8: Incident Close 19](#_Toc196496609)

[Figure 9: Design Architecture 26](#_Toc196496610)

[Figure 10 : Class diagram for InciTrack 27](#_Toc196496611)

[Figure 11: ER Diagram InciTrack 28](#_Toc196496612)

[Figure 12 : User Login 31](#_Toc196496613)

[Figure 13: Create User 34](#_Toc196496614)

[Figure 14: Incident Entry 35](#_Toc196496615)

[Figure 15: Register Incident Category 37](#_Toc196496616)

[Figure 16 : Incident List View 38](#_Toc196496617)

[Figure 17: Incident Feedback Form 39](#_Toc196496618)

[Figure 18: Email Notification 41](#_Toc196496619)

[Figure 19 : Dashboard 42](#_Toc196496620)

[Figure 20: Incident Report 44](#_Toc196496621)