Riphah International University Lahore, Pakistan

Riphah School of Computing & Innovation

Final Year Project

PROJECT REPORT (Part-1)

HawkEye

Project Team

Student Name	Student ID	Program	Contact Number	Email Address
Daniyal Wajid (Leader)	48528	BSSE	03091840367	48528@students.riphah.edu.pk
Uzair Hassan	48525	BSSE	03254039356	48525@students.riphah.edu.pk

Ma'am Uzma Kiran
Senior Lecturer

Project Report

HawkEye

Change Record

Author(s)	Version	Date	Notes	Supervisor's Signature
	1.0		<Original Draft>	
			<Changes Based on Feedback from Supervisor>	
			<Changes Based on Feedback From Faculty>	
			<Added Project Plan>	
			<Changes Based on Feedback from Supervisor>	

APPROVAL

PROJECT SUPERVISOR

Comments: _____

_____

Name:_____

Date:_____ Signature:_____

PROJECT MANAGER

Comments: _____

_____

Date:_____ Signature:_____

HEAD OF THE DEPARTMENT

Comments: _____

_____

Date:_____ Signature:_____

Dedication

This work is dedicated to my . . . . . .

Acknowledgements

I am really thankful to my supervisor who has . . . . . .

Executive Summary

[12 pt, Calibri, Justified]

[*An executive summary summarizes a longer report or proposal or a group of related reports in such a way that readers can rapidly become acquainted with a large body of material without having to read it all. This section summarizes the overall document, and should include the important highlights from the document. It should be concise. It is NOT an introduction, index or table of contents, it is a summary. The Executive Summary should not make any reference to other parts of the document. You have to write one page to let reader understand an overview of the project.]*

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 1: Introduction

This chapter provides an overview of the **HawkEye** system, an intelligent AI-based discipline monitoring solution designed for educational institutions. It explains the project's background, motivation, objectives, and the need for an automated surveillance system to ensure campus safety. The chapter discusses existing manual systems, their limitations, and the technological gap that HawkEye aims to fill. It also outlines the proposed solution, project goals, and implementation plan that combine artificial intelligence, computer vision, and data analytics to maintain discipline and accountability across the campus.

1.1. Background

Maintaining discipline and safety within educational institutions has become increasingly challenging due to large student populations and limited human supervision. Traditional monitoring methods rely heavily on manual observation by security staff, which is often inefficient, time-consuming, and prone to human error. With advancements in artificial intelligence and computer vision, it is now possible to automate campus surveillance and behavior detection. **HawkEye** leverages these technologies to continuously monitor student activities through CCTV cameras, automatically detect violations such as fighting, smoking, or improper uniforms, and record them for administrative action. This AI-driven approach ensures a safer, more disciplined, and transparent campus environment.

1.2. Motivations and Challenges

Motivations:

- The need for a **smart and automated monitoring system** to maintain discipline in educational institutions without continuous manual supervision.
- To **reduce human bias and errors** that occur during manual observation of student behavior.

- To **enhance campus safety** by instantly detecting and reporting incidents such as fights, smoking, or possession of prohibited items.
- To create a **digital record management system** for tracking student violations and penalties over time.
- To **utilize AI and computer vision** technologies for real-time detection, faster decision-making, and improved administrative efficiency.

Challenges:

- Ensuring **real-time and accurate detection** of multiple behaviors under different lighting and crowd conditions.
- Maintaining **data privacy and ethical use** of surveillance footage in compliance with institutional policies.
- **Integrating AI models** with existing CCTV infrastructure and ensuring compatibility with various camera types.
- Handling **large volumes of video data** efficiently using cloud-based or GPU-enabled servers.
- Minimizing **false detections** and ensuring the system performs reliably in diverse environmental scenarios.

1.3. Goals and Objectives

The main goal of the **HawkEye** project is to develop an intelligent, AI-driven system that automates campus discipline monitoring and enhances overall safety. It aims to replace traditional manual supervision with a reliable, real-time surveillance and violation management solution. The project focuses on improving efficiency, accuracy, and transparency within educational institutions.

Objectives:

- To design and implement an **AI-based video analysis system** capable of detecting rule violations such as fighting, smoking, or possession of prohibited items.
- To **automatically log incidents** with timestamps, captured images, and violation types in a secure database.

- To create **digital student disciplinary profiles** that store and track violation history.
- To **automate penalty and fine management** based on predefined institutional rules and policies.
- To develop a **web-based administrative dashboard** for real-time monitoring, analytics, and incident review.
- To ensure **data security, privacy, and access control** through encrypted storage and role-based permissions.

1.4. Literature Review/Existing Solutions

System / Vendor	Manual Monitoring	Violence / Weapon Detection	Uniform / ID Detection	Multi-Task Behaviour Analytics	Integrates with Existing Cameras / VMS	Student-Record Integration	Policy-Based Penalties
Traditional CCTV	✅	❌	❌	❌	✅	❌	❌
Genetec Security Center	✅	✅	❌	✅	✅	❌	❌
Avigilon (Motorola)	✅	✅	❌	✅	✅	❌	❌
BriefCam Analytics	✅	✅	❌	✅	✅	❌	❌
Irisity (School Guard)	✅	✅	❌	✅	✅	❌	❌
Nyckel AI (Uniform Classifier)	❌	❌	✅	❌	✅	❌	❌
ZeroEyes (Gun Detection)	✅	✅	❌	❌	✅	❌	❌
Omnilert (AI Gun Detection)	✅	✅	❌	❌	✅	❌	❌
Intelgic Vision AEye	✅	✅	❌	✅	✅	❌	❌
HawkEye	✅	✅	✅	✅	✅	✅	✅

1.5. Gap Analysis

Most existing surveillance systems primarily focus on general security or crime prevention rather than educational discipline management. Solutions like Genetec Security Center, Avigilon, and ZeroEyes offer limited capabilities such as weapon or violence detection but lack integration with student data, uniform monitoring, or automated penalty assignment. These systems also depend heavily on manual verification, leading to delays and inconsistencies in disciplinary actions. There is a significant gap in having a **dedicated, AI-powered platform** that combines behavior recognition, facial identification, and policy-based penalty management tailored specifically for academic institutions. **HawkEye** addresses this gap by offering an end-to-end automated solution that integrates detection, recordkeeping, and analytics to ensure fairness, accountability, and improved campus safety.

1.6. Proposed Solution

Hawkeye provides an AI-driven automated surveillance and discipline management solution for educational campuses. It integrates live CCTV feeds with advanced computer vision models to detect violations such as fighting, possession of weapons, smoking, or improper uniform usage in real time. Once a violation is detected, the system captures image evidence, identifies the students involved through facial recognition or student-ID matching, and automatically logs the incident in the database. Each record includes details such as date, time, location, and violation type.

The system then updates the respective student's disciplinary profile and, based on predefined institutional policies, calculates penalties or fines automatically. Administrators can review all incidents through a centralized dashboard, where they can verify, approve, or adjust penalties. The dashboard also provides analytics to identify recurring offenders and common violation types. Hawkeye aims to minimize manual monitoring, reduce bias, and ensure consistent enforcement of campus rules. The overall solution enhances safety, accountability, and administrative efficiency through AI-powered automation.

1.7. Project Plan

The Hawk Eye project will be executed in multiple well-defined phases to ensure systematic progress and successful completion. The entire project will be managed using an Agile methodology, allowing iterative development, continuous improvement, and timely stakeholder feedback. The project spans from October 2025 to April 2026, and is divided into two main phases: Documentation Phase and Implementation Phase.

1. Documentation Phase (October 2025 – January 2026)

This phase focuses on the research, planning, and design aspects of the Hawk Eye system. The key objectives include defining project scope, gathering functional and non-functional requirements, and creating detailed design specifications.

Major tasks in this phase include:

- **Requirement Gathering and Analysis:** Identifying project goals, stakeholders, and system needs.
- **Project Charter & Proposal Creation:** Defining objectives, scope, and constraints.
- **Feasibility Study & Research:** Evaluating technical feasibility and suitable technologies (YOLO/OpenCV for detection, TensorFlow for AI training).
- **System and UML Design:** Developing system architecture, data flow diagrams, and use case models.
- **UI/UX Prototyping:** Designing user-friendly interfaces to ensure smooth interaction.
- **Final Documentation:** Preparing the documentation for approval before moving to implementation.

2. Implementation Phase (February 2026 – April 2026)

This phase involves developing, integrating, and testing all major components of the Hawk Eye system. The development process will follow Agile sprints to ensure incremental progress and quick adaptation to changes.

Key activities include:

- **Module Development:** Implementing backend and frontend components.
- **AI Model Training and Integration:** Training object detection and tracking models using YOLO/OpenCV and integrating them into the system.
- **Database Configuration:** Setting up data storage and retrieval mechanisms.
- **API Development & Integration:** Linking modules and ensuring seamless communication between components.
- **System Testing and Debugging:** Conducting unit testing, integration testing, and performance evaluation.
- **Deployment and Evaluation:** Deploying the final system and ensuring it meets the defined requirements.

3. Final Phase (April 2026)

The last phase includes documentation, presentation, and evaluation of the final system.

Tasks include:

- User Manual Preparation
- Final Report Compilation
- Project Presentation and Demonstration

1.7.1. Work Breakdown Structure

A **Work Breakdown Structure (WBS)** provides a hierarchical decomposition of the entire Hawk Eye project into manageable sections, covering all essential deliverables from documentation to AI-based system development, testing, and deployment. The tasks are distributed between two team members Uzair and Daniyal ensuring a balanced workload, accountability, and smooth workflow.

WBS Hierarchy

1. Project Management

1.1 Project Charter & Scope Definition

1.2 Work Breakdown Structure (WBS) Creation

1.3 Roles & Responsibility Matrix

1.4 Schedule & Resource Allocation

1.5 Risk Assessment and Mitigation Planning

1.6 Progress Monitoring and Reporting

2. Reports / Documentation

2.1 Introduction & Problem Statement

2.2 Literature Review / Related Work

2.3 Feasibility Study (Technical & Operational)

2.4 Requirement Gathering and Analysis

2.5 System Design Documentation

2.6 Implementation Strategy & Plan

2.7 Testing and Evaluation Plan

2.8 Final Report and Conclusion

2.9 End User Guide

2.10 Technical & Administrator Documentation

3. System Development

3.1 Development Environment Setup

3.1.1 IDE and Tools Installation (VS Code, Anaconda, etc.)

3.1.2 Version Control Configuration (Git/GitHub)

3.1.3 AI Libraries and Dependencies Installation (YOLO, OpenCV, TensorFlow)

3.1.4 Database Setup and Configuration (Firebase / SQL)

3.2 Detection & Tracking Engine

3.2.1 Dataset Collection & Preprocessing

3.2.2 Object Detection Model Training (YOLO / TensorFlow)

3.2.3 Real-time Tracking Implementation (OpenCV)

3.2.4 Accuracy and Performance Optimization

3.2.5 Model Testing and Validation

3.3 Backend Development

3.3.1 API Design and Development (FastAPI / Flask)

3.3.2 Model Integration with Backend

3.3.3 Data Storage and Retrieval Logic

3.3.4 Authentication and Access Control

3.3.5 Logging and Error Handling Mechanism

3.4 Frontend Development

3.4.1 UI/UX Design and Wireframing

3.4.2 Dashboard Development (React.js / Flutter Web)

3.4.3 Real-time Detection Visualization

3.4.4 User Authentication & Role Management

3.4.5 Reporting and Alerts Module

3.5 Testing and Evaluation

3.5.1 Unit and Integration Testing

3.5.2 Model Accuracy Evaluation

3.5.3 Performance and Load Testing

3.5.4 Usability and UI Testing

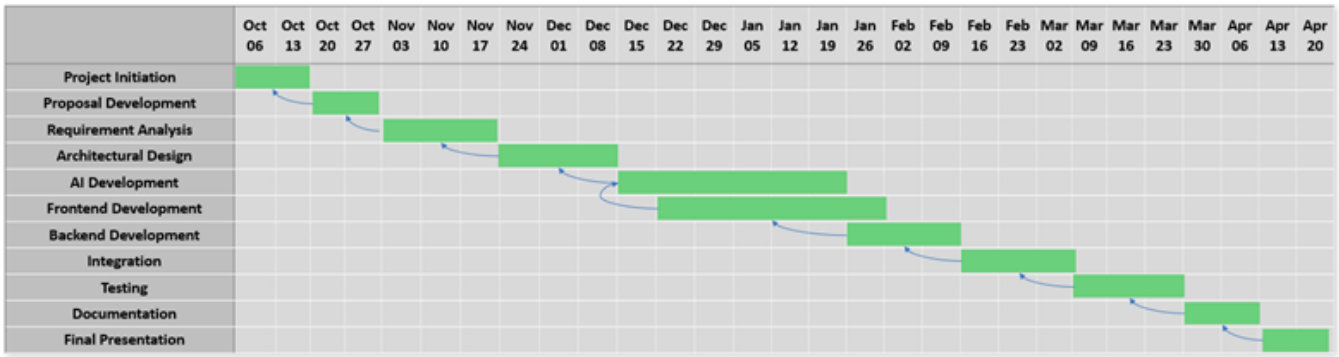
3.5.5 Bug Fixing and Improvements

1.7.2. Roles & Responsibility Matrix

WBS #	WBS Deliverable	Activity #	Activity to Complete the Deliverable	Duration (# of Days)	Responsible Team Member(s) & Role(s)
1	Project Management	1.1-1.4	Define WBS, assign roles, and manage project schedule	7	Uzair & Daniyal
2	Documentation Introduction	2.1	Write project overview, objectives, and problem definition	8	Uzair
3	Literature Review / Feasibility Study	2.2	Research existing AI surveillance systems and methods	10	Uzair
4	Requirements Analysis	2.3	Identify functional, non-functional, and technical requirements	10	Uzair & Daniyal
5	System Design	2.4	Create system architecture diagrams, data flow, and ER models	12	Daniyal
6	Implementation Planning	2.5	Define development phases, module dependencies, and timeline	6	Both
7	Development Environment Setup	3.1	Install IDEs, configure Git/GitHub, set up YOLO, OpenCV, and database	8	Daniyal
8	Detection & Tracking Engine	3.2	Collect dataset, train YOLO model, and integrate OpenCV	25	Daniyal
9	Backend Development	3.3	Build FastAPI backend, integrate AI model, and manage APIs	20	Uzair
10	Frontend Development	3.4	Design user interface, implement dashboard, and display live detection	20	Uzair
11	Testing & Evaluation	3.5	Conduct unit testing, model accuracy testing, and UI evaluation	15	Both

12	Final Documentation & Reports	2.7-2.10	Prepare final report, admin guide, and technical documentation	15	Uzair
13	Presentation & Submission	----	Prepare final demo presentation and report submission	5	Both

1.7.3. Gantt Chart



1.8. Report Outline

This report is organized into several chapters, each addressing a specific aspect of the **HawkEye** project. **Chapter 1** introduces the project by explaining its background, motivations, goals, and the proposed AI-driven solution for campus discipline monitoring.

Chapter 2

Software Requirement Specifications

Chapter 2: Software Requirement Specifications

2.1. Introduction

2.1.1. Purpose

The purpose of this Software Requirement Specification (SRS) is to define the functional and non-functional requirements of the HawkEye system—an AI-based discipline monitoring solution for educational institutions. It outlines the system's scope, including real-time violation detection, student record management, and an administrative dashboard. This document serves as a guideline for developers, testers, and stakeholders to ensure accurate implementation of all modules and features across the complete system.

2.1.2. Document Conventions

This SRS follows the **IEEE standard** for Software Requirement Specifications. All headings and subheadings are numbered according to their hierarchy for easy navigation. Important terms and system modules are written in bold for emphasis. Each functional requirement is labeled uniquely (e.g., *REQ-SF1-1*) for clear identification and traceability. High-priority requirements are explicitly marked and do not inherit priorities from higher-level requirements. The document maintains consistent formatting using Calibri, 12 pt, and 1.5 line spacing throughout.

2.1.3. Intended Audience and Reading Suggestions

This document is intended for all stakeholders involved in the HawkEye project, including developers, project supervisors, testers, and system administrators. Developers can use it to understand functional and technical requirements for implementation, while testers can refer to it for validating system behavior and performance. Supervisors and administrators can review it to ensure the system aligns with institutional goals.

The SRS is organized into sections that begin with an overview of the system, followed by detailed functional and non-functional requirements. Readers are advised to start with Chapter 1 (Introduction) for context, then proceed to Chapter 2 (SRS) for specific requirements, and finally review subsequent chapters for system design and implementation details.

2.1.4. Product Scope

Scope of the Project

The Hawkeye system is designed to automate campus discipline monitoring through AI-based video analysis and smart data management. The project consists of several major modules, each focusing on a specific functionality that contributes to the overall goal of maintaining a safe and well-disciplined environment.

1. Real-Time Monitoring Module:

This module connects to live CCTV cameras installed across the campus. It uses AI and computer vision algorithms to continuously analyze video feeds for suspicious or prohibited activities such as fighting, possession of weapons, or improper uniforms. It ensures 24/7 surveillance without the need for constant human monitoring.

2. Violation Detection and Logging Module:

Once a violation is detected, this module captures relevant image frames, timestamps, and location details. It automatically logs the incident in the system database along with the type of violation detected. The data is securely stored and can be accessed later for verification and recordkeeping.

3. Student Record and Penalty Management Module:

This module maintains digital profiles for all students, tracking their violation history. Based on predefined rules and policies, it calculates appropriate fines or penalties automatically. Administrators can review and modify penalties and maintain transparent disciplinary records.

4. Administrator Dashboard Module:

This web-based interface allows authorized staff to monitor real-time activity, review logged incidents, and generate analytical reports. The dashboard provides graphical summaries of violations, recurring offenders, and campus hotspots to support data-driven decisions.

5. Notification and Alert Module:

This component generates instant alerts for administrators or security personnel when serious violations are detected. It helps ensure timely intervention and enhances the overall response efficiency of campus authorities.

Excluded Features:

- Integration with national ID or police databases.
- Audio-based violation detection or voice recognition.
- Online payment processing for fines or penalties.

2.1.5. References

- Redmon, J., & Farhadi, A. (2018). *YOLOv3: An Incremental Improvement*. arXiv preprint arXiv:1804.02767. Retrieved from https://arxiv.org/abs/1804.02767
- Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. arXiv preprint arXiv:2004.10934. Retrieved from https://arxiv.org/abs/2004.10934
- Bradski, G. (2000). *The OpenCV Library*. Dr. Dobb's Journal of Software Tools. Retrieved from https://opencv.org

2.2. Overall Description

2.2.1. Product Perspective

HawkEye is a new, self-contained system designed to automate discipline monitoring and safety management within educational institutions. It is not a follow-up to any existing system but rather an innovative solution that integrates artificial intelligence, computer vision, and cloud technologies into a single unified platform. The system connects to existing CCTV or IP camera networks and processes live video streams to detect violations such as fighting, smoking, or improper uniforms.

The product functions as an independent application with multiple interconnected modules, including the AI Processing Engine, Backend Server, Cloud Database, and Web-Based Dashboard. These components communicate through secure APIs to ensure real-time data flow and system reliability. HawkEye can also integrate with future institutional systems, such as student information systems or attendance databases, for enhanced functionality.

2.2.2. Product Functions

- The HawkEye system provides several key functions designed to automate campus discipline monitoring and streamline administrative processes. The main functions of the system are summarized below:
- **Real-Time Video Analysis:** Continuously monitors CCTV feeds to detect violations such as fighting, smoking, or possession of prohibited items using AI and computer vision.
- Violation Logging: Automatically captures image evidence, timestamps, and location data for each detected incident and stores it in the cloud database.
- **Student Identification:** Recognizes students involved in violations using facial recognition and links incidents to their digital disciplinary profiles.
- **Penalty Management:** Calculates and assigns fines or penalties automatically based on institutional policies and violation types.
- **Administrative Dashboard:** Provides a web-based interface for administrators to view live camera feeds, review incidents, manage student records, and generate reports.
- **Notification System:** Sends real-time alerts to authorized staff for serious violations, ensuring quick response and corrective actions.
- **Data Analytics and Reporting:** Generates analytical summaries, graphs, and insights to identify frequent offenders and recurring violation trends.

2.2.3. User Classes and Characteristics

The HawkEye system is designed for multiple categories of users, each with distinct responsibilities, access privileges, and system interaction levels. The main user classes are described below:

- **Administrator / Discipline Incharge:**
  Responsible for overall system management, reviewing reported incidents, approving or modifying penalties, and generating analytical reports. They have the highest access privileges and can manage users, policies, and configurations.
- **Security Officer:**
  Monitors live CCTV feeds through the dashboard and responds to alerts generated by the system. They have access to violation logs, notifications, and real-time camera monitoring but limited administrative control.
- **Faculty Member / Class Incharge:**
  Can view disciplinary records of students under their supervision and receive notifications of incidents involving their class or department. They have read-only access to reports and profiles relevant to their area.
- **Student:**
  Can access their personal disciplinary record to view warnings, fines, or penalties. They have limited access and cannot modify system data.
- **System Developer / Technical Staff:**
  Responsible for system maintenance, database management, and AI model updates. They have back-end access for debugging and performance optimization but limited access to user-sensitive data.

Among these, Administrators and Security Officers are the most critical user classes, as they directly interact with the core features of the HawkEye system for campus monitoring and violation management.

2.2.4. Operating Environment

The HawkEye system operates in a cloud-based and networked environment, integrating hardware and software components to support real-time surveillance and data processing. The

primary hardware includes CCTV or IP cameras, GPU-enabled servers, and administrator devices such as desktop PCs or tablets. The AI models and backend services run on cloud platforms like AWS, Google Cloud, or Microsoft Azure, utilizing GPU instances for high-performance video analysis.

The system's backend is developed using Flask, FastAPI, or Node.js, while the frontend dashboard operates on React.js for web access and optionally Flutter for mobile support. The database layer uses Firebase Firestore or MongoDB Atlas for secure and scalable data storage. The system runs on modern operating systems such as Windows 10/11, Ubuntu 22.04, or equivalent Linux distributions. It coexists smoothly with existing campus networks, CCTV infrastructure, and browsers like Google Chrome, Edge, or Firefox for administrative access.

2.2.5. Design and Implementation Constraints

The design and implementation of the HawkEye system are subject to several constraints that influence the development approach, technology choices, and operational boundaries.

- **Hardware Limitations:** Real-time video analysis requires GPU-enabled servers with sufficient processing power and memory. The system performance may degrade on low-end hardware or limited-bandwidth networks.
- **Technology Constraints:** The system must be implemented using Python (YOLO, OpenCV, TensorFlow) for AI processing and Flask/FastAPI or Node.js for backend development. The frontend must be developed using React.js to ensure compatibility with the web-based dashboard design.
- **Integration Constraints:** The system must integrate seamlessly with existing CCTV or IP camera networks and support standard video stream formats such as RTSP.
- **Security and Privacy Policies:** All video data and student information must comply with institutional data protection policies and use SSL/TLS encryption and JWT authentication to secure access.
- **Database Constraints:** Only Firebase Firestore or MongoDB Atlas are to be used for cloud-based storage, as defined in the project architecture.

- **Network Dependency:** Since HawkEye processes live video streams, it requires stable internet connectivity and low-latency communication between the AI engine, backend, and dashboard.
- Development Standards: The project follows Agile methodology, modular coding practices, and version control using Git/GitHub to maintain consistency and collaboration among developers.

2.2.6. User Documentation

The HawkEye system will be delivered with comprehensive user documentation to assist administrators, security officers, and technical staff in operating and maintaining the platform effectively. The documentation components include:

- **User Manual:** A detailed guide explaining system setup, login procedures, dashboard navigation, and feature usage for administrators and security staff.
- **Installation Guide:** Step-by-step instructions for deploying the backend server, AI engine, and database configuration on cloud or local servers.
- **Online Help Section:** Integrated help tips and tooltips within the dashboard to assist users with key operations and troubleshooting.
- **Video Tutorials:** Short instructional videos demonstrating how to monitor live feeds, review incidents, and manage records.
- **Technical Documentation:** Developer-level documentation covering API endpoints, database structure, and AI model integration for system maintenance.

2.2.7. Assumptions and Dependencies

The design and performance of the HawkEye system are based on several assumptions and dependencies that influence its successful operation and deployment.

Assumptions:

- The institution has an existing CCTV or IP camera network with compatible video output formats (e.g., RTSP or HTTP streams).
- A stable internet connection with sufficient bandwidth is available to support real-time video transmission and cloud communication.
- The institution's IT infrastructure supports integration with cloud platforms such as Firebase, AWS, or Google Cloud.
- Authorized staff members possess basic computer literacy to operate the web-based dashboard and respond to alerts.
- Adequate GPU-enabled servers or cloud resources are available for AI processing and object detection tasks.

Dependencies:

- The system depends on third-party AI frameworks such as YOLO, OpenCV, and TensorFlow for real-time detection and image processing.
- Cloud services like Firebase Firestore, MongoDB Atlas, and Google Cloud Storage are required for database and media storage operations.
- Notification services such as Firebase Cloud Messaging (FCM) or Twilio API are needed to deliver alerts and updates to administrators.
- Any changes to external APIs, cloud policies, or third-party service availability may affect the system's functionality or require code updates.

2.3. External Interface Requirements

2.3.1. User Interfaces

The HawkEye system features a web-based user interface designed to provide administrators, security officers, and faculty members with an intuitive and responsive experience. The interface follows modern design standards for usability, accessibility, and consistency across all screens.

The dashboard is developed using React.js with a clean and minimal layout, incorporating key UI components such as navigation menus, live camera feeds, incident logs, and analytical charts. Each page maintains consistent elements, including a header bar with quick navigation options, a notification icon, and a help section for user assistance.

Standard UI features include:

Login Screen: Secure authentication for users based on their roles (Admin, Security Officer, Faculty).

Dashboard Home: Displays live video feeds, detected incidents, and recent alerts.

Incident Management Screen: Allows users to review, verify, or delete logged violations with supporting image evidence.

Student Profile Screen: Displays student disciplinary records, violation history, and penalties.

Analytics Page: Graphical representation of frequent violations, offenders, and time-based trends.

Settings Page: Enables configuration of system policies, user accounts, and camera management.

The interface is designed with responsive layouts compatible with desktops, laptops, and tablets. It uses standard colors, icons, and typography following accessibility guidelines for readability. Error messages, tooltips, and confirmation dialogs are integrated to assist users during operations.

2.3.2. Hardware Interfaces

The HawkEye system interfaces with multiple hardware components to enable real-time monitoring and data processing. The primary hardware components include CCTV/IP cameras, GPU-enabled servers, and user devices such as desktop computers, laptops, or tablets.

- **CCTV/IP Cameras:**

  These cameras act as the primary data source for the system. They continuously capture live video streams of campus areas and transmit them to the AI Processing Engine. The system supports standard camera models using RTSP or HTTP streaming protocols.

- **GPU-Enabled Server / Cloud Infrastructure:**

  This server hosts the AI Processing Engine that performs video analysis, object detection, and behavior recognition in real time. Communication between the cameras and the server occurs over a secure TCP/IP network.

- **Administrator and Security Devices:**

  Authorized users access the HawkEye dashboard through personal computers, tablets, or other network-connected devices using modern browsers. Data visualization and control interactions are handled through secure HTTPS requests.

2.3.3. Software Interfaces

The HawkEye system interacts with several software components and external services to achieve seamless functionality across its AI, backend, and frontend modules. The software interfaces define how data flows between these components to support detection, logging, and reporting operations.

- **Operating Systems:**

  The system is compatible with Windows 10/11 and Linux (Ubuntu 22.04 or later) for both development and deployment environments.

- **AI Frameworks and Libraries:**

  HawkEye integrates YOLO (You Only Look Once) for object and behavior detection, OpenCV for image processing, and TensorFlow for deep learning operations. These libraries handle incoming video streams, process frames, and output detection data such as object type, confidence score, and bounding box coordinates.

- **Backend Framework:**

  The backend is developed using Flask, FastAPI, or Node.js and communicates with the AI engine through RESTful APIs or WebSockets. It processes detection events, manages user authentication, and stores structured data in the cloud database.

- **Database Systems:**

  The system uses Firebase Firestore or MongoDB Atlas for storing incident records, student profiles, and system configurations. Communication between the backend and database occurs over secure connections using JSON-based data exchange.

- **Cloud Storage:**

  Image and video evidence from detected violations are stored using AWS S3 or Google Cloud Storage, ensuring scalability and fast retrieval.

- **Frontend Interface:**

  The web-based dashboard built with React.js interacts with the backend through secure HTTPS API calls, displaying real-time camera feeds, analytics, and incident logs.

2.3.4. Communications Interfaces

The HawkEye system relies on secure and efficient communication interfaces to ensure real-time data exchange between its modules, servers, and users. All communication occurs over encrypted channels to maintain data integrity and confidentiality.

- **Network Communication:**

  The system uses TCP/IP protocols for stable data transmission between CCTV cameras, the AI Processing Engine, and the backend server. Real-time video feeds are transmitted using RTSP or HTTP streaming protocols.

- **Web Communication:**

  Communication between the frontend dashboard and backend server occurs via HTTPS (HTTP over SSL/TLS) to ensure secure data transfer. RESTful APIs and WebSockets are used for real-time updates, such as live detection alerts and incident logging.

- **Cloud Communication:**

  The system interacts with Firebase, MongoDB Atlas, and cloud storage services like AWS S3 or Google Cloud Storage using secure API endpoints. All requests and responses are formatted in JSON for consistency and easy parsing.

- **Notification Services:**

  Alerts and updates are sent through Firebase Cloud Messaging (FCM) or Twilio API, using secure communication protocols to notify administrators via SMS, email, or push notifications.

- **Security and Encryption:**

  Every communication channel uses SSL/TLS encryption, JWT authentication tokens, and role-based access control to prevent unauthorized access. Sensitive data such as user credentials and incident evidence is encrypted before transmission.

2.4. System Features

The HawkEye system consists of several integrated features designed to provide intelligent, automated campus discipline monitoring through AI and computer vision. Each feature focuses on a core functionality that collectively contributes to maintaining safety, transparency, and administrative efficiency. The major system features are outlined below:

2.4.1.1 Description and Priority

This feature enables the system to continuously monitor live CCTV or IP camera feeds and automatically detect prohibited activities such as fighting, smoking, or unauthorized gatherings using AI-based object and behavior recognition models.

Priority: High

Benefit: 9 **Penalty:** 8 **Cost:** 6 **Risk:** 5

2.4.1.2 Stimulus/Response Sequences

- **Stimulus:** CCTV cameras stream live video to the AI Processing Engine.
- **Response:** The system analyzes frames in real-time and triggers an alert if a violation is detected.
- **Stimulus:** A violation event is identified.
- **Response:** The system captures evidence (image snapshot), logs the incident, and updates the admin dashboard.

2.4.1.3 Functional Requirements

REQ-SF1-1: The system shall continuously process live feeds and detect predefined violation types using AI models.
REQ-SF1-2: The system shall capture snapshots with metadata (time, location, camera ID) for every violation.
REQ-SF1-3: The system shall notify the administrator in real-time upon detection.
REQ-SF1-4: The system shall process up to 10 camera streams simultaneously.
REQ-SF1-5: The system shall filter false positives using a confidence threshold.

2.4.2 Violation Logging and Record Management

2.4.2.1 Description and Priority

Automatically logs every detected violation with its evidence, timestamp, and camera source. Data is securely stored in the cloud for administrative review.
Priority: High
Benefit: 8 **Penalty:** 7 **Cost:** 5 **Risk:** 4

2.4.2.2 Stimulus/Response Sequences

- **Stimulus:** A violation is detected.
- **Response:** The backend logs data into the database.
- **Stimulus:** Administrator opens the log panel.
- **Response:** System retrieves and displays violation records.

2.4.2.3 Functional Requirements

REQ-SF2-1: The system shall store each violation with a unique ID, time, and location.
REQ-SF2-2: The system shall allow filtering and searching of recorded incidents.
REQ-SF2-3: Only authorized users can modify or delete logs.

REQ-SF2-4: The system shall back up logs to cloud storage automatically.

REQ-SF2-5: The system shall prevent data loss during disconnection.

2.4.3 Student Identification and Profile Linking

2.4.3.1 Description and Priority

Uses facial recognition to identify students involved in incidents and links violations to their digital profiles.

Priority: High

Benefit: 9 **Penalty:** 8 **Cost:** 7 **Risk:** 6

2.4.3.2 Stimulus/Response Sequences

- **Stimulus:** Violation detected with a clear facial image.
- **Response:** The system matches the face with student records.
- **Stimulus:** A match is confirmed.
- **Response:** The violation is added to the student's profile.

2.4.3.3 Functional Requirements

REQ-SF3-1: The system shall identify students with ≥85% accuracy.

REQ-SF3-2: The system shall link the student to the violation record.

REQ-SF3-3: The system shall alert the student's class incharge of the violation.

REQ-SF3-4: The admin can manually verify or edit mismatched IDs.

REQ-SF3-5: Biometric data shall be encrypted and stored securely.

2.4.4 Automated Penalty and Fine Calculation

2.4.4.1 Description and Priority

Automatically calculates and assigns penalties or fines based on institutional rules and the severity of the violation.

Priority: Medium

Benefit: 8 **Penalty:** 6 **Cost:** 4 **Risk:** 5

2.4.4.2 Stimulus/Response Sequences

- **Stimulus:** A verified violation is logged.
- **Response:** The system retrieves predefined penalty rules.
- **Stimulus:** Violation type and student record are matched.
- **Response:** The system updates the student profile with penalty details.

2.4.4.3 Functional Requirements

REQ-SF4-1: The system shall apply penalties according to violation type and severity.

REQ-SF4-2: The system shall calculate monetary fines automatically.

REQ-SF4-3: The system shall prevent duplicate penalty entries.

REQ-SF4-4: The system shall notify students via email or portal about new penalties.

REQ-SF4-5: Administrators can edit or waive penalties if needed.

2.4.5 Administrative Dashboard

2.4.5.1 Description and Priority

Provides an intuitive interface for monitoring live feeds, reviewing incidents, and managing reports.

Priority: High

Benefit: 9 **Penalty:** 8 **Cost:** 6 **Risk:** 4

2.4.5.2 Stimulus/Response Sequences

- **Stimulus:** Admin logs into the dashboard.
- **Response:** The system displays a summary of live cameras and recent violations.
- **Stimulus:** Admin clicks a specific violation record.
- **Response:** The system shows full incident details with evidence and actions.

2.4.5.3 Functional Requirements

REQ-SF5-1: The dashboard shall display live video feeds and incident notifications.

REQ-SF5-2: The dashboard shall support filtering by date, type, or student.

REQ-SF5-3: The dashboard shall provide export options (PDF, CSV).

REQ-SF5-4: Only authorized users can access dashboard features.

REQ-SF5-5: The interface shall update data in real-time.

2.4.6 Notification and Alert System

2.4.6.1 Description and Priority

Sends instant alerts to relevant users when serious violations occur, ensuring rapid response.

Priority: High

Benefit: 9 **Penalty:** 8 **Cost:** 5 **Risk:** 5

2.4.6.2 Stimulus/Response Sequences

- **Stimulus:** A high-priority violation is detected.
- **Response:** The system sends a notification to administrators and security officers.
- **Stimulus:** Admin acknowledges the alert.
- **Response:** System marks the alert as “resolved” in the database.

2.4.6.3 Functional Requirements

REQ-SF6-1: The system shall send alerts via email, SMS, or dashboard notification.

REQ-SF6-2: The system shall categorize alerts by severity level.

REQ-SF6-3: The system shall maintain a history of sent notifications.

REQ-SF6-4: The system shall retry sending failed alerts.

REQ-SF6-5: Admins can customize alert preferences.

2.4.7 Data Analytics and Reporting

2.4.7.1 Description and Priority

Generates analytical reports and visual summaries to identify patterns in violations and support decision-making.

Priority: Medium

Benefit: 8 **Penalty:** 6 **Cost:** 5 **Risk:** 4

2.4.7.2 Stimulus/Response Sequences

- **Stimulus:** Admin selects a date range and report type.
- **Response:** The system fetches and analyzes relevant data.
- **Stimulus:** Admin requests visualization.
- **Response:** System displays graphs and downloadable reports.

2.4.7.3 Functional Requirements

REQ-SF7-1: The system shall generate reports based on violations, time, and users.

REQ-SF7-2: The system shall display data visually using charts and graphs.

REQ-SF7-3: Reports shall be downloadable in PDF and Excel formats.

REQ-SF7-4: The system shall allow filtering by department, date, or violation type.

REQ-SF7-5: The system shall store generated reports for future access.

2.4.8 Security and Access Control

2.4.8.1 Description and Priority

Ensures data protection through encrypted communication, authentication, and role-based permissions.

Priority: High

Benefit: 9 **Penalty:** 9 **Cost:** 6 **Risk:** 5

2.4.8.2 Stimulus/Response Sequences

- **Stimulus:** User attempts to log in.
- **Response:** The system verifies credentials via JWT authentication.
- **Stimulus:** Unauthorized access attempt.
- **Response:** System blocks access and records a security log entry.

2.4.8.3 Functional Requirements

REQ-SF8-1: The system shall use JWT tokens for session management.

REQ-SF8-2: The system shall encrypt all communications with SSL/TLS.

REQ-SF8-3: The system shall restrict access based on user roles.

REQ-SF8-4: The system shall log all login and access attempts.

REQ-SF8-5: The system shall lock accounts after multiple failed login attempts.

2.5. Other Nonfunctional Requirements

2.5.1. Performance Requirements

The HawkEye system must maintain high performance and responsiveness to ensure real-time surveillance and accurate violation detection without delays. The following performance requirements define system expectations under typical and peak operating conditions:

- The system shall process and analyze video streams from up to 10 cameras simultaneously with minimal latency (less than 2 seconds delay).
- The AI engine shall detect and classify violations with at least 85% accuracy under standard lighting and visibility conditions.
- The system shall log each violation and update the dashboard within 3 seconds of detection.
- The backend server shall handle at least 100 concurrent user sessions without performance degradation.
- Database queries and report generation operations shall complete within 5 seconds under normal load.
- The notification service shall deliver alerts to administrators within 2 seconds after a critical event is detected.
- The system shall utilize optimized GPU processing and asynchronous data handling to minimize lag during real-time analysis.
- All modules (AI, backend, and dashboard) shall maintain 99% uptime, ensuring continuous monitoring and data availability.

2.5.2. Safety Requirements

The HawkEye system is designed to operate safely within institutional environments without causing harm to individuals, property, or data. Since it handles surveillance and disciplinary data, the following safety requirements are established to ensure secure and responsible system usage:

- The system shall not cause any physical harm to users or interfere with existing campus security equipment.
- All video feeds and data storage processes shall comply with institutional safety and privacy regulations to prevent misuse of personal or visual data.
- The system shall implement fail-safe mechanisms to prevent data loss or corruption in case of network or hardware failure.
- Access to sensitive information, such as student identities and violation evidence, shall be restricted to authorized personnel only.
- The system shall prevent unauthorized deletion or modification of violation records that could compromise investigation integrity.
- Regular data backups and integrity checks shall be performed to minimize the risk of information loss.
- The system shall conform to IT safety standards such as ISO/IEC 27001 for data security and GDPR-like institutional policies for data protection and ethical monitoring.
- In the event of a detected malfunction or anomaly, the system shall log the error, trigger an alert, and revert to a safe operational mode.

2.5.3. Security Requirements

The HawkEye system must ensure strong protection of sensitive surveillance data, user credentials, and institutional information. Given its role in monitoring and recording disciplinary incidents, the system must comply with strict data security and privacy standards.

- All user access shall be protected through multi-level authentication, including secure login with email/password and JWT (JSON Web Token) session management.
- Communication between the frontend, backend, and cloud servers shall be encrypted using SSL/TLS protocols to prevent data interception.
- The system shall enforce role-based access control (RBAC) to ensure that users only access data relevant to their responsibilities.

- Sensitive data such as facial recognition results, student information, and evidence images shall be encrypted at rest and in transit.

- The system shall automatically log and audit all user activities, including login attempts, data edits, and deletions.
- The system shall lock user accounts after multiple failed login attempts to prevent brute-force attacks.
- Regular security patches and updates shall be applied to AI libraries, backend frameworks, and databases to minimize vulnerabilities.
- The system shall comply with institutional IT security guidelines and international standards such as ISO/IEC 27001 and GDPR-equivalent privacy principles.
- Database and file storage services shall have restricted administrative access and use firewall protection to prevent unauthorized data exposure.
- The system shall notify administrators immediately if suspicious or unauthorized access attempts are detected.

2.5.4. Software Quality Attributes

The HawkEye system is designed to meet high standards of software quality to ensure reliability, maintainability, and user satisfaction. The following attributes define key quality expectations for the system:

- **Availability:**

  The system shall maintain 99% uptime, ensuring continuous real-time monitoring and uninterrupted access to all features.
- **Reliability:**

  The system shall consistently detect and record violations with a false-positive rate below 10%, ensuring dependable results even under varying lighting and network conditions.
- **Usability:**

  The web dashboard shall be intuitive and easy to navigate, allowing new users to perform key operations (view logs, manage data) within 10 minutes of training.
- **Maintainability:**

  The system’s modular design shall enable updates, bug fixes, and enhancements with minimal downtime and without affecting other modules.

- **Scalability:**

  The architecture shall support expansion to additional cameras, user accounts, or locations without major code modification or performance degradation.

- **Interoperability:**

  HawkEye shall seamlessly integrate with external systems such as institutional databases, attendance records, or notification services via standard REST APIs.

- **Security:**

  Implements end-to-end encryption, secure authentication, and data anonymization to ensure data confidentiality and system protection.

- **Portability:**

  The system shall operate on both Windows and Linux-based environments and be deployable on local servers or cloud infrastructures.

- **Adaptability:**

  The AI models and rules can be updated or retrained to adapt to new violation categories or institutional policies.

- **Testability:**

  Each module shall include unit and integration tests to verify correctness before deployment, achieving at least 90% code coverage.

2.5.5. Business Rules

The HawkEye system operates under a defined set of business rules that govern how different users and roles interact with the platform. These rules ensure accountability, data integrity, and consistent enforcement of institutional disciplinary policies.

- **Role-Based Access:**

  Only authorized users such as Administrators, Security Officers, and Faculty Members can access specific system features according to their defined roles.

- **Incident Verification:**

  All automatically detected violations must be verified by an administrator or security officer before being finalized in the system records.

- **Penalty Enforcement:**

  Penalties and fines are automatically calculated based on institutional policies and must be approved by the administrator before being applied to a student's record.

- **Data Confidentiality:**

  Student disciplinary data and violation evidence are confidential and can only be viewed by authorized personnel. Sharing this data outside the system is strictly prohibited.

- **Audit and Accountability:**

  Every user action, such as editing, deleting, or verifying a record, is logged automatically for auditing purposes.

- **Record Retention:**

  All violation and disciplinary records must be retained for a minimum period (e.g., one academic year) before archival or deletion, as per institutional policy.

- **Notification Rules:**

  Notifications are automatically triggered for severe violations and sent only to relevant authorities (e.g., discipline incharge, principal).

- **System Maintenance:**

  Only technical staff or system developers are permitted to modify system settings, perform backups, or update AI models.

- **Policy Updates:**

  Institutional authorities can update disciplinary policies or penalty rules, which automatically reflect in future detection and penalty calculations.

2.6. Other Requirements

The following additional requirements apply to the HawkEye system. These aspects, while not directly tied to specific functionalities, are crucial for ensuring compliance, scalability, and proper long-term operation.

- **Database Requirements:**

  The system shall use a cloud-based NoSQL database such as Firebase Firestore or MongoDB Atlas to store structured and unstructured data, including violations, user

profiles, and configuration details. All database transactions shall ensure atomicity, consistency, isolation, and durability (ACID) principles for reliability.

- **Backup and Recovery:**

  Automatic daily backups of the database and image evidence shall be maintained on a secure cloud server. The system shall support data restoration in case of accidental deletion or corruption.

- **Legal and Ethical Requirements:**

  The system must comply with institutional data protection laws and policies regarding student privacy and camera surveillance. Collected data shall be used solely for disciplinary purposes and may not be shared externally without authorization.

- **Localization and Internationalization:**

  The system shall support multi-language capability in the user interface, allowing easy adaptation for institutions in different regions if required.

- **Reuse Objectives:**

  The system's modular architecture shall allow future reuse of its AI detection module, notification system, or dashboard component in other campus management or security projects.

- **Scalability and Future Integration:**

  The system shall be designed for horizontal scalability, allowing integration with new cameras, users, or departments without altering core architecture. Future integration with attendance or access control systems should be supported via standard APIs.

- **Regulatory Compliance:**

  HawkEye shall adhere to ISO/IEC 27001 information security standards and GDPR-equivalent privacy practices to ensure lawful data collection and retention.

- **Data Retention Policy:**

  Recorded video evidence and disciplinary data shall be retained for a defined institutional period (e.g., one academic year) before secure archival or deletion.

Chapter 3

Use Case Analysis

Chapter 3: System Analysis

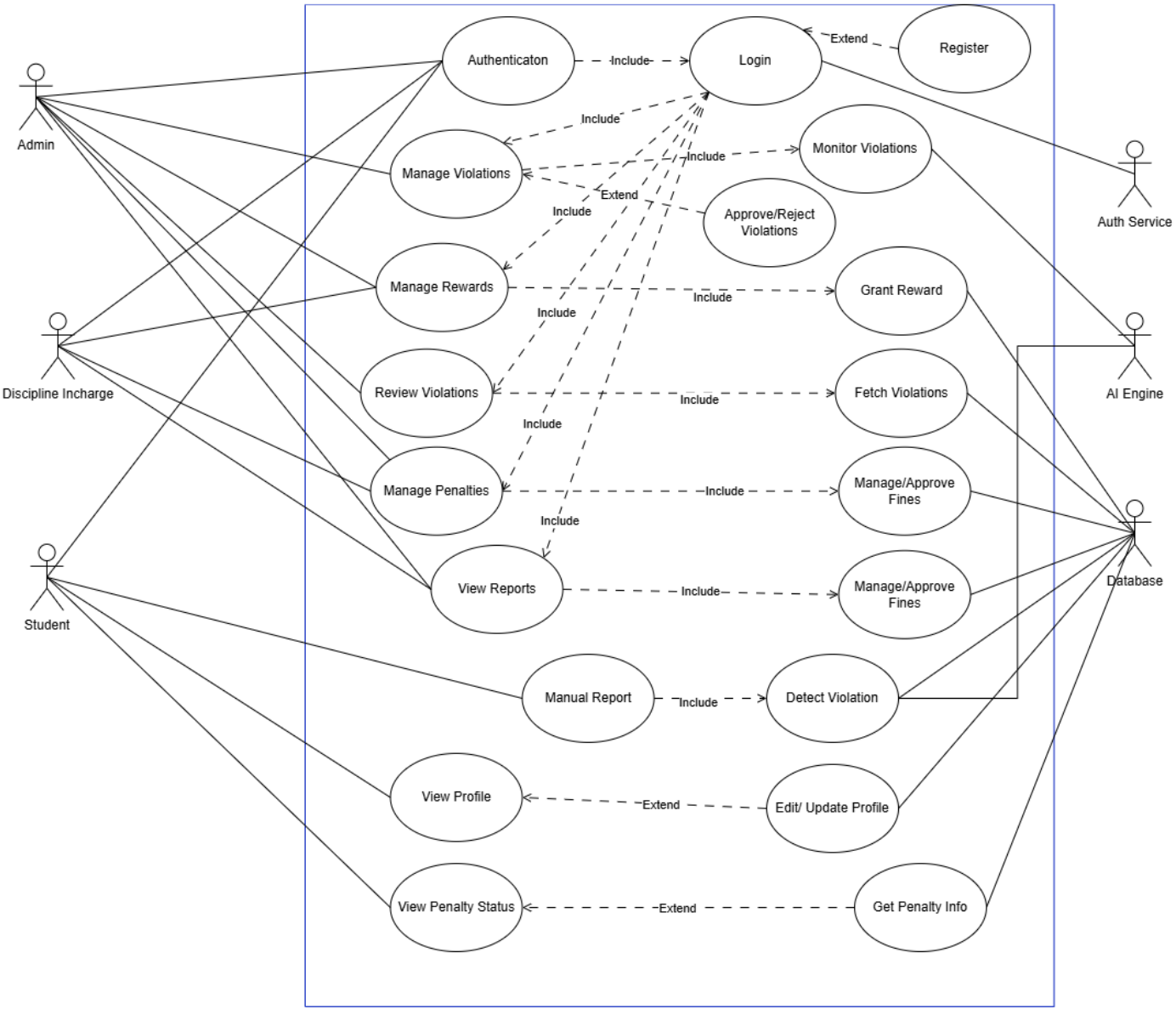
This chapter presents an in-depth analysis of the HawkEye system and outlines how the system behaves from the user's perspective. It explains the functional boundaries of the system, the interactions between stakeholders, and the core processes that define system behavior. The objective of this chapter is to transform project requirements into a structured set of use cases that guide the system's logical development. By analyzing actors, their responsibilities, and the scenarios they engage in, this chapter forms the foundation for the later design and implementation phases.

3.1. Use Case Model

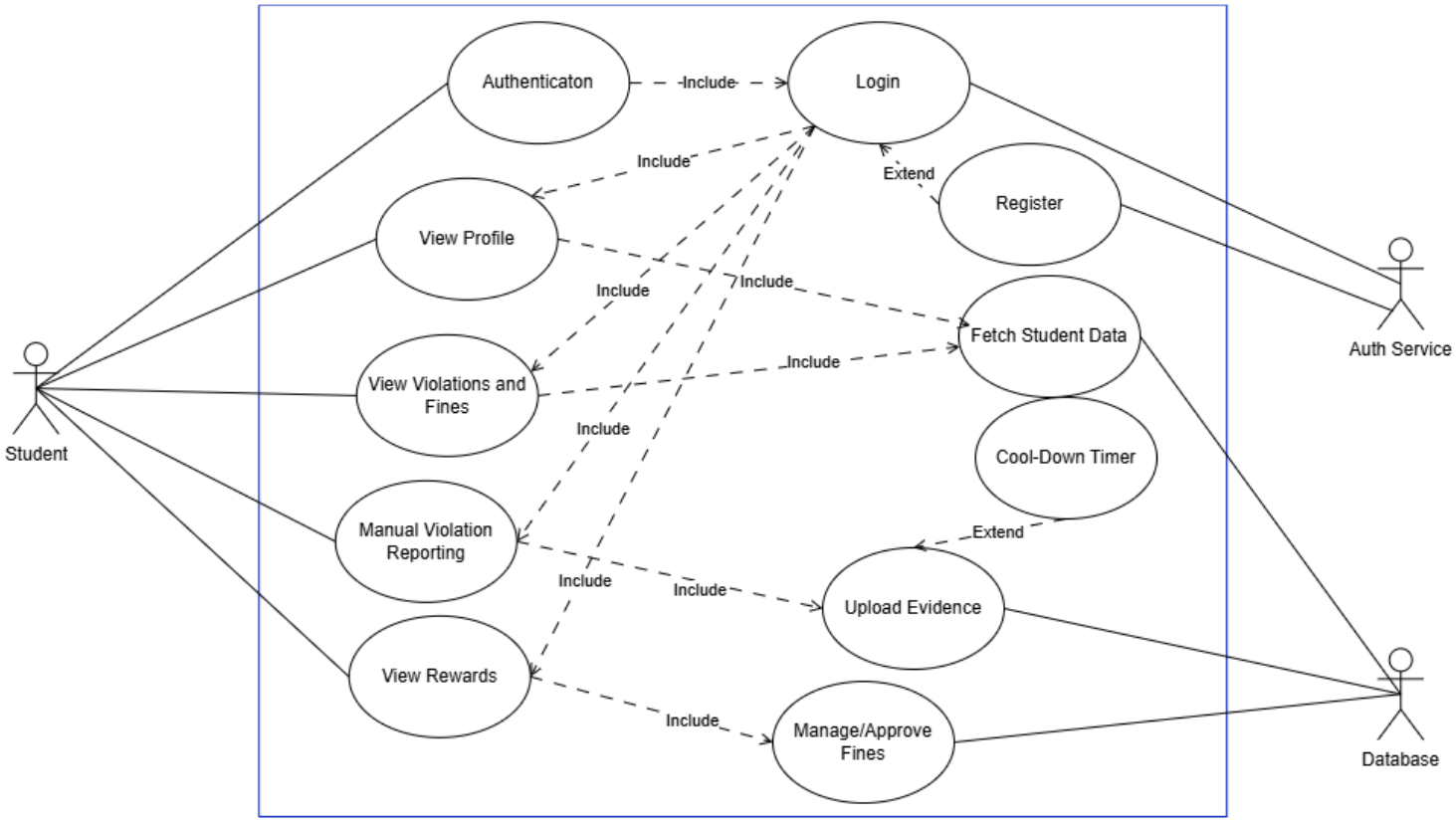
The use case model describes the high-level interactions between users and the HawkEye system. It identifies key actors—Admin, Discipline Incharge, Students, and the AI System—and outlines how each actor communicates with the system to achieve specific goals. These use cases provide a clear understanding of system functionality such as authentication, student registration, violation detection, manual reporting, fine management, reward distribution, and notification handling. The use case model ensures that all required functionalities are aligned with user expectations and system objectives.d

3.2. Fully Dressed Use Cases

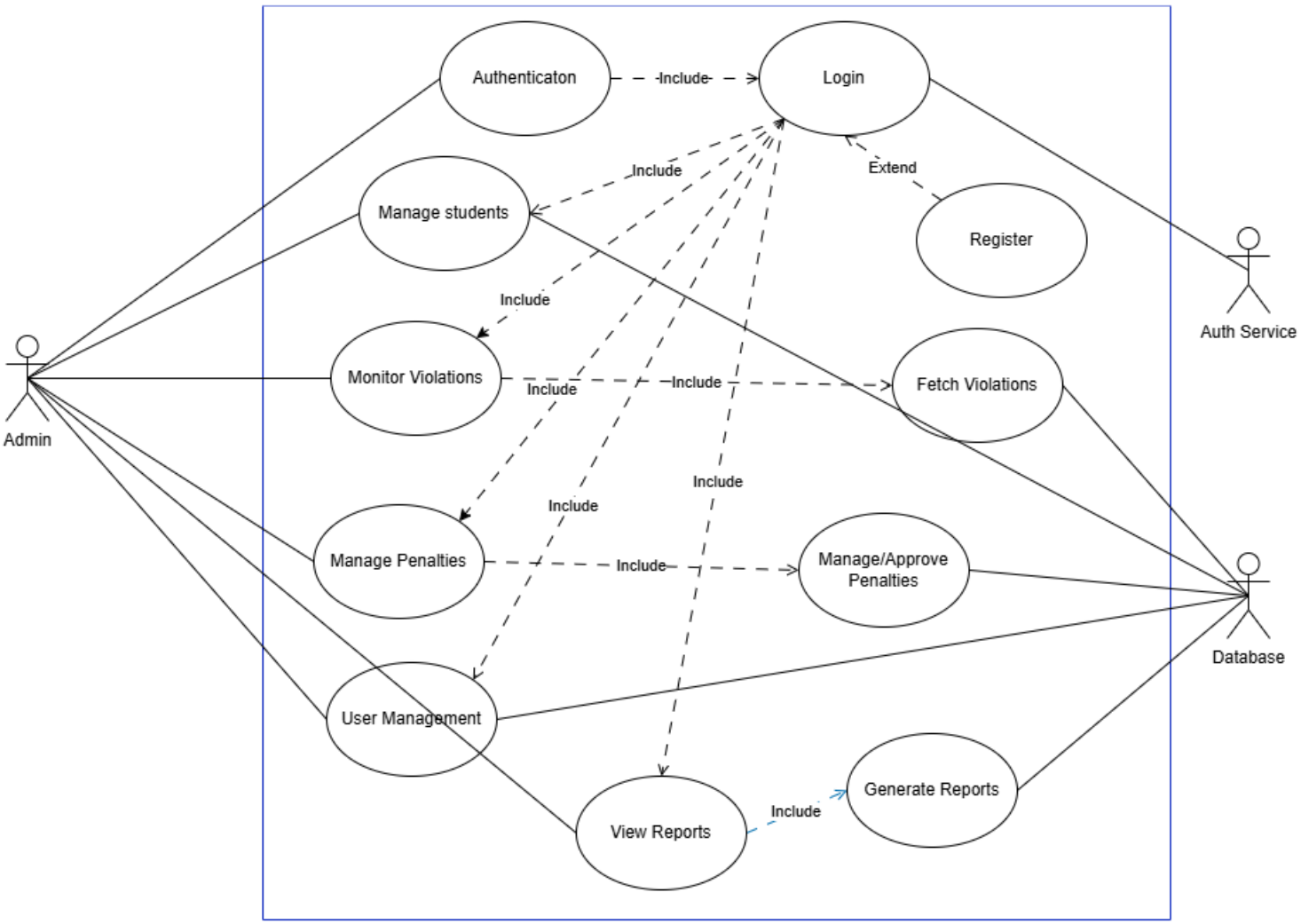
Use Case Diagram System



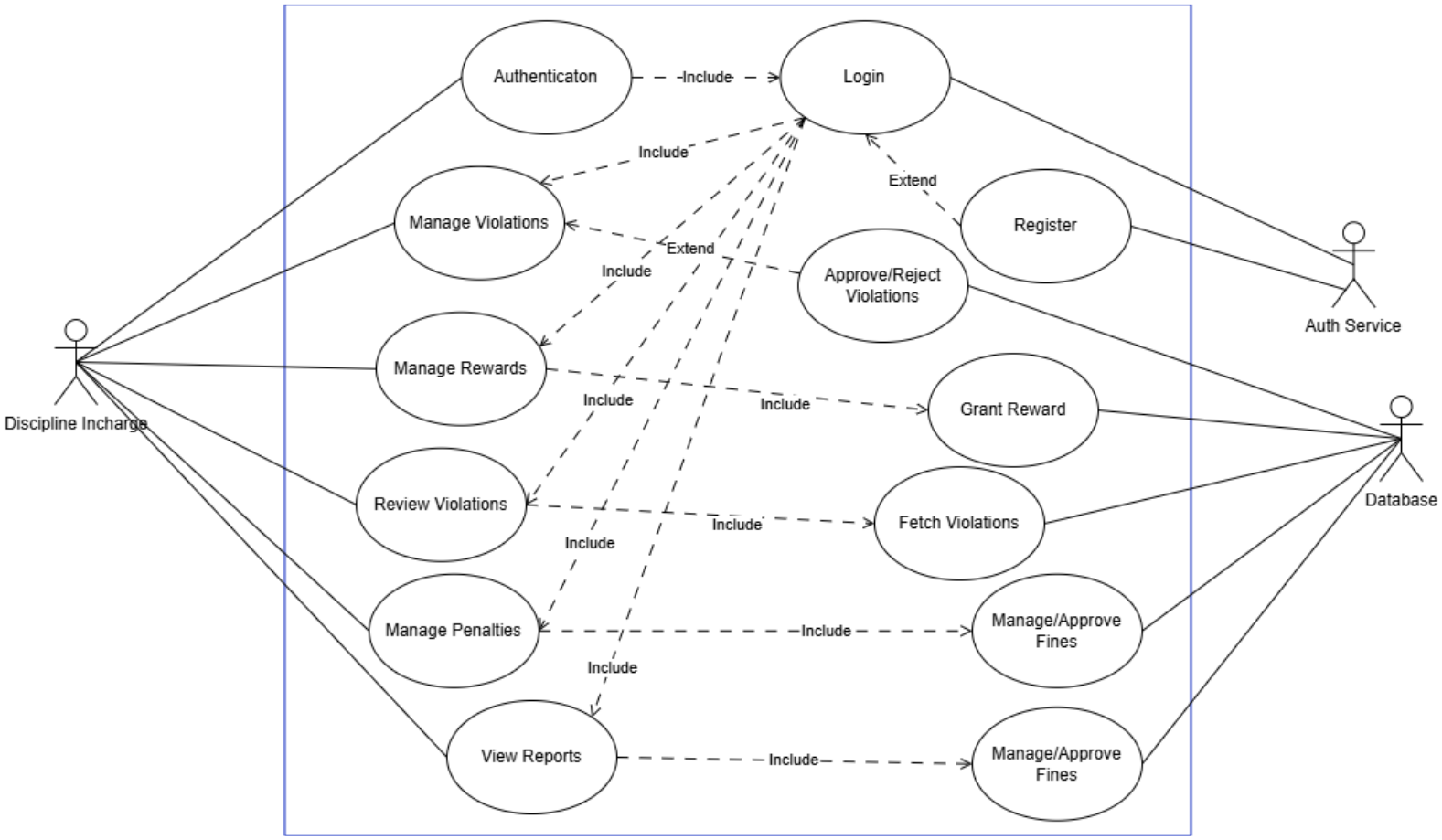
Use Case Diagram Student



Use Case Diagram Admin



Use Case Diagram Discipline Incharge



Use Case Diagram AI System

UC-01 — User Login

Field	Description
Use Case ID	UC-Login
Use Case Name	User Login
Primary Actor	Student, Teacher, Admin
Stakeholders & Interests	Users (secure access), Admin (security), System (audit)
Preconditions	User has an existing account; system online
Postconditions	User authenticated; session created; audit log updated
Trigger	User submits login form
Main Success Scenario	1. User enters credentials. 2. System validates. 3. System creates session. 4. Redirect to dashboard.
Alternate Flows	A1. Invalid password. A2. Account locked. A3. Forgot password flow triggered.
Special Requirements	Password hashing, MFA (optional), rate limiting
Frequency	High
Notes	Clear error messages required

UC-02 — Sign-Up & Email Verification

Field	Description
Use Case ID	UC-Signup
Use Case Name	User Sign-Up & Email Verification
Primary Actor	Student, Teacher, Admin
Preconditions	Email server available
Postconditions	Account created, email verified
Trigger	User fills registration form
Main Success Scenario	1. User enters details. 2. System validates. 3. Creates unverified account. 4. Emails verification link. 5. User clicks link → verified.
Alternate Flows	A1. Email already used. A2. Token expired → resend.
Special Requirements	Token expiry; CAPTCHA; secure hashing
Frequency	Medium

UC-03 — Student Registration & Face Image Capture

Field	Description
Use Case ID	UC-RegisterStudent
Use Case Name	Student Registration & Face Capture
Primary Actor	Admin
Preconditions	Admin logged in; camera functioning
Postconditions	Student profile created; face images stored
Trigger	Admin selects “Register Student”
Main Success Scenario	1. Admin enters student details. 2. Captures multiple face images. 3. System checks quality. 4. Saves images + metadata.
Alternate Flows	A1. Poor image → retake. A2. Upload instead of capture.
Special Requirements	Encryption of biometric data
Frequency	Periodic (new admissions)

UC-04 — Model Training

Field	Description
Use Case ID	UC-TrainModel
Use Case Name	Face Model Training
Primary Actor	System
Secondary Actor	Admin
Preconditions	New face data collected
Postconditions	Model trained & deployed
Trigger	New student registered or scheduled training
Main Success Scenario	1. System loads new images. 2. Preprocesses. 3. Trains model. 4. Evaluates accuracy. 5. Deploys model.
Alternate Flows	A1. Insufficient data. A2. Training fails, rollback.
Special Requirements	GPU/compute availability
Frequency	Scheduled or event-driven

UC-05 — AI Violation Detection

Field	Description
Use Case ID	UC-AIDetect
Use Case Name	AI Violation Detection
Primary Actor	System
Secondary Actors	Cameras, Admin, Security
Preconditions	Cameras active; detection model running
Postconditions	Violation event generated
Trigger	AI detects suspicious pattern in video
Main Success Scenario	1. AI reads live feed. 2. Detects violation (dress/weapon/fight). 3. Extracts frames. 4. Runs face recognition. 5. Sends event for logging.
Alternate Flows	A1. Low confidence → ignore. A2. Face not found.
Special Requirements	Real-time processing & alerts
Frequency	Continuous

UC-06 — Auto Violation Logging

Field	Description
Use Case ID	UC-LogViolation
Use Case Name	Automatic Violation Logging
Primary Actor	System
Secondary Actor	Admin, Discipline Incharge
Preconditions	Valid detection event exists
Postconditions	Violation saved in DB
Trigger	AI detection module sends event
Main Success Scenario	1. System receives event. 2. Adds metadata. 3. Saves to MongoDB. 4. Sends alert to admin.
Alternate Flows	A1. Storage error → retry queue.
Special Requirements	Encryption of evidence images
Frequency	High

UC-07 — Face Recognition

Field	Description
Use Case ID	UC-FaceRecognition
Use Case Name	Identify Student via Face Recognition
Primary Actor	System
Preconditions	Trained model deployed
Postconditions	Identity resolved or marked unknown
Trigger	AI detects a face in video
Main Success Scenario	1. Extract face region. 2. Generate embeddings. 3. Compare with DB. 4. Return best match.
Alternate Flows	A1. Unrecognized → label Unknown.
Special Requirements	Threshold tuning
Frequency	High

UC-08 — Manual Violation Reporting (Student)

Field	Description
Use Case ID	UC-SubmitReport
Use Case Name	Manual Report Submission
Primary Actor	Student
Secondary Actor	AI Validator, Admin
Preconditions	Student logged in; cooldown expired
Postconditions	Report stored, awaiting validation
Trigger	Student clicks “Submit Report”
Main Success Scenario	1. Student captures image/video. 2. Selects violation category. 3. System gathers metadata. 4. AI validates authenticity. 5. Sends to admin.
Alternate Flows	A1. Cooldown active. A2. Evidence fake → reject.
Special Requirements	Anti-spam and metadata verification
Frequency	Moderate

UC-09 — Report Validation

Field	Description
Use Case ID	UC-ValidateReport
Use Case Name	AI-Based Report Validation
Primary Actor	System
Secondary Actor	Admin
Preconditions	Student report submitted
Postconditions	Report marked Valid/Invalid
Trigger	New report arrives
Main Success Scenario	1. Validate metadata. 2. Check for duplicates. 3. Analyze evidence. 4. Apply AI rules. 5. Assign status.
Alternate Flows	A1. No match → manual review.
Special Requirements	Forensic checks (tampering)
Frequency	Medium

UC-10 — Automatic Fine Generation

Field	Description
Use Case ID	UC-AutoFine
Use Case Name	Auto Fine Assignment
Primary Actor	System
Secondary Actor	Admin, Student
Preconditions	Violation verified
Postconditions	Fine created; student notified
Trigger	Violation confirmed
Main Success Scenario	1. System checks rule. 2. Calculates fine. 3. Saves fine entry. 4. Notifies student/admin.
Alternate Flows	A1. No rule exists → admin intervention.
Special Requirements	Configurable fine tables
Frequency	High

UC-11 — Admin Review & Approval

Field	Description
Use Case ID	UC-AdminReview
Use Case Name	Admin Review of Violations
Primary Actor	Admin / Teacher
Preconditions	Violation or report exists
Postconditions	Approved, Rejected, or Pending status
Trigger	Admin selects violation to review
Main Success Scenario	1. Admin opens record. 2. Reviews evidence. 3. Approves/rejects. 4. System updates status.
Alternate Flows	A1. Insufficient evidence → request more info.
Special Requirements	Audit logging
Frequency	High

UC-12 — Notifications & Alerts

Field	Description
Use Case ID	UC-Notification
Use Case Name	Real-Time Alerts & Notifications
Primary Actor	System
Secondary Actors	Admin, Teacher, Student
Preconditions	Notification service active
Postconditions	Alert delivered to recipient
Trigger	New violation, fine, or report update
Main Success Scenario	1. Event occurs. 2. System identifies recipients. 3. Sends push notification.
Alternate Flows	A1. Device offline → retry queue.
Special Requirements	Firebase/FCM or custom push service
Frequency	High

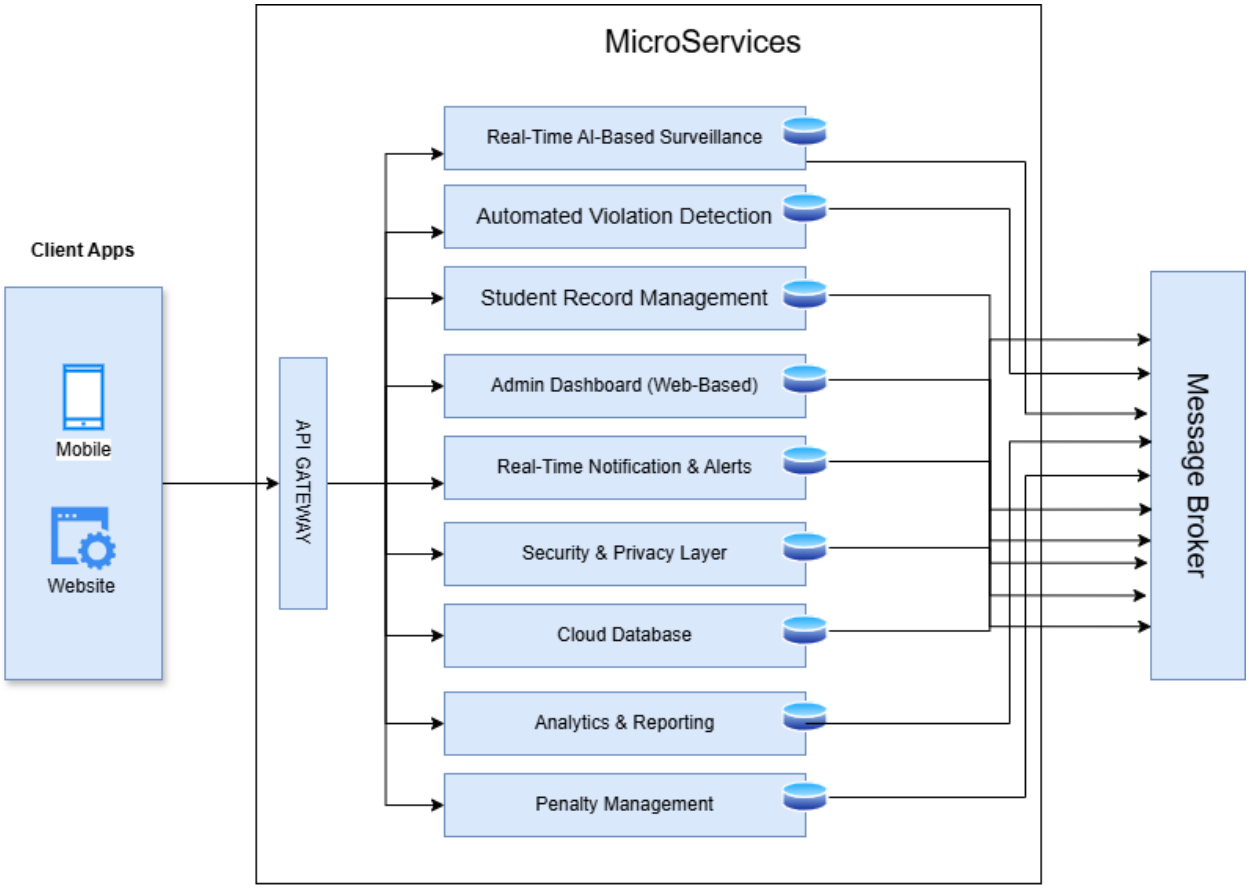
Chapter 4

System Design

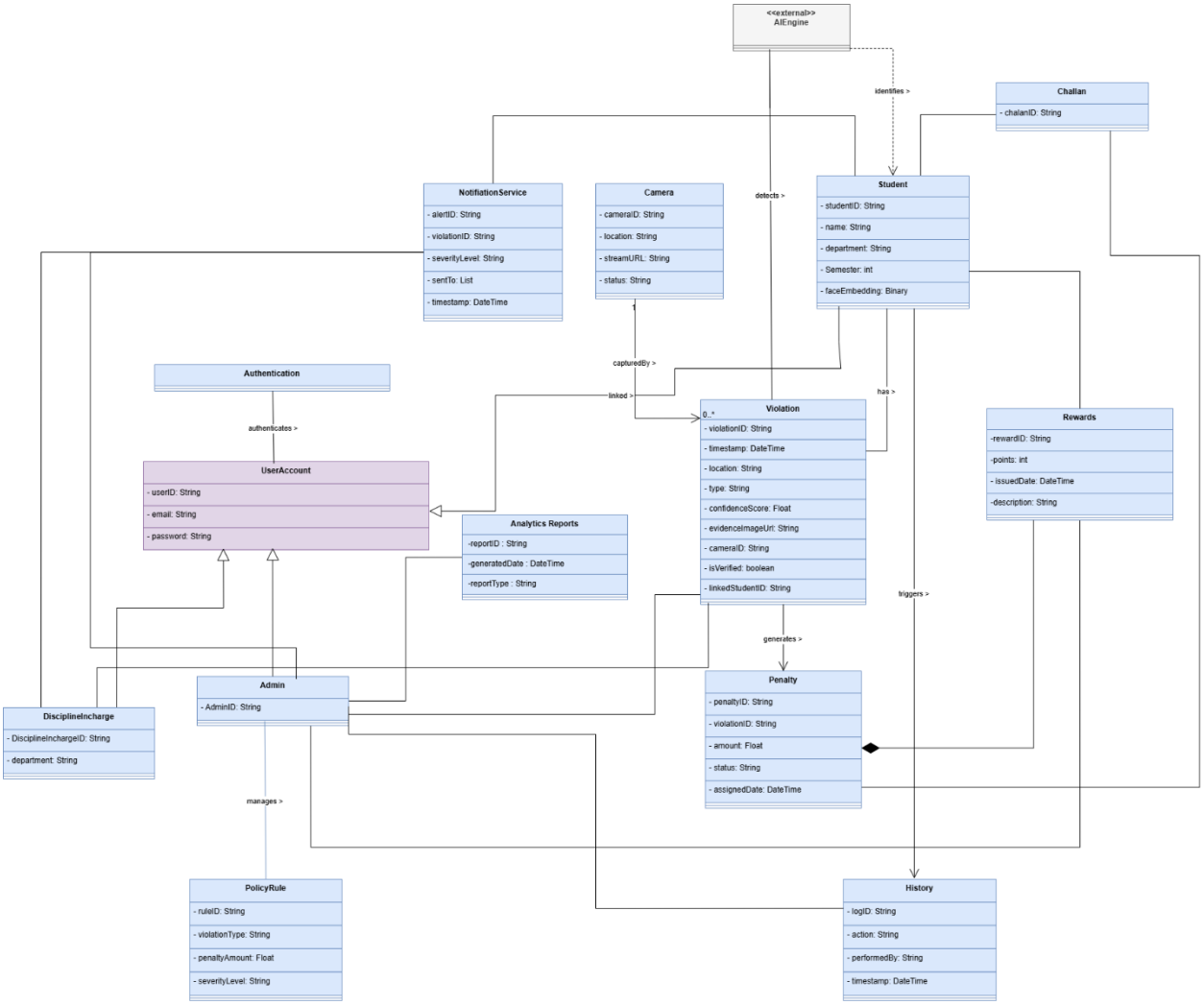
Chapter 4: System Design

This chapter focuses on the architectural and structural design of the HawkEye system. It explains how the system's components, modules, and data structures are organized to meet the requirements identified earlier. System design transforms analysis into a blueprint for implementation, including architectural layers, domain concepts, database structure, data flow, component distribution, and interaction behaviors. This chapter ensures that the system is scalable, secure, and maintainable by following industry-standard design practices and modeling techniques.

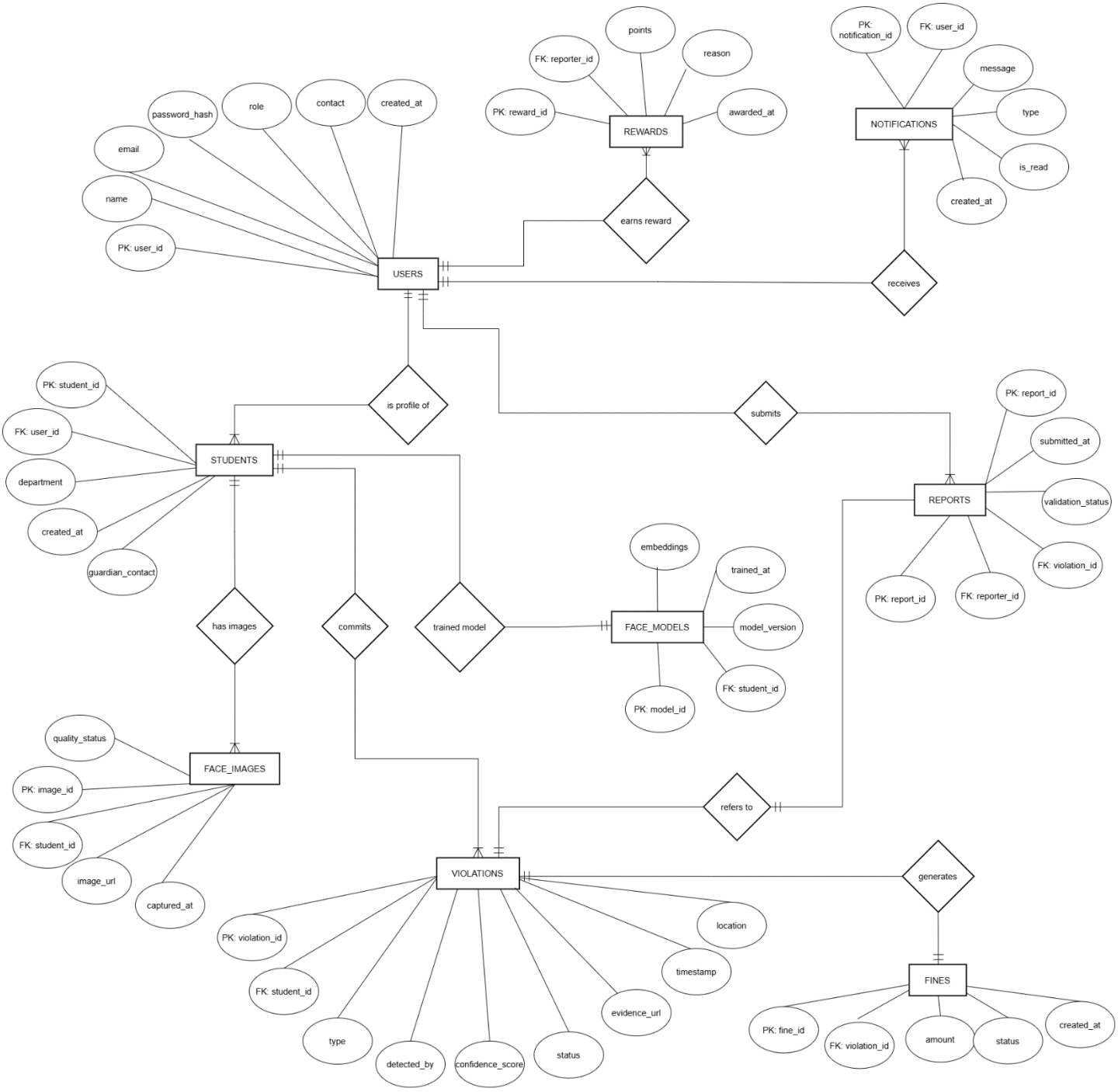
4.1. Architecture Diagram



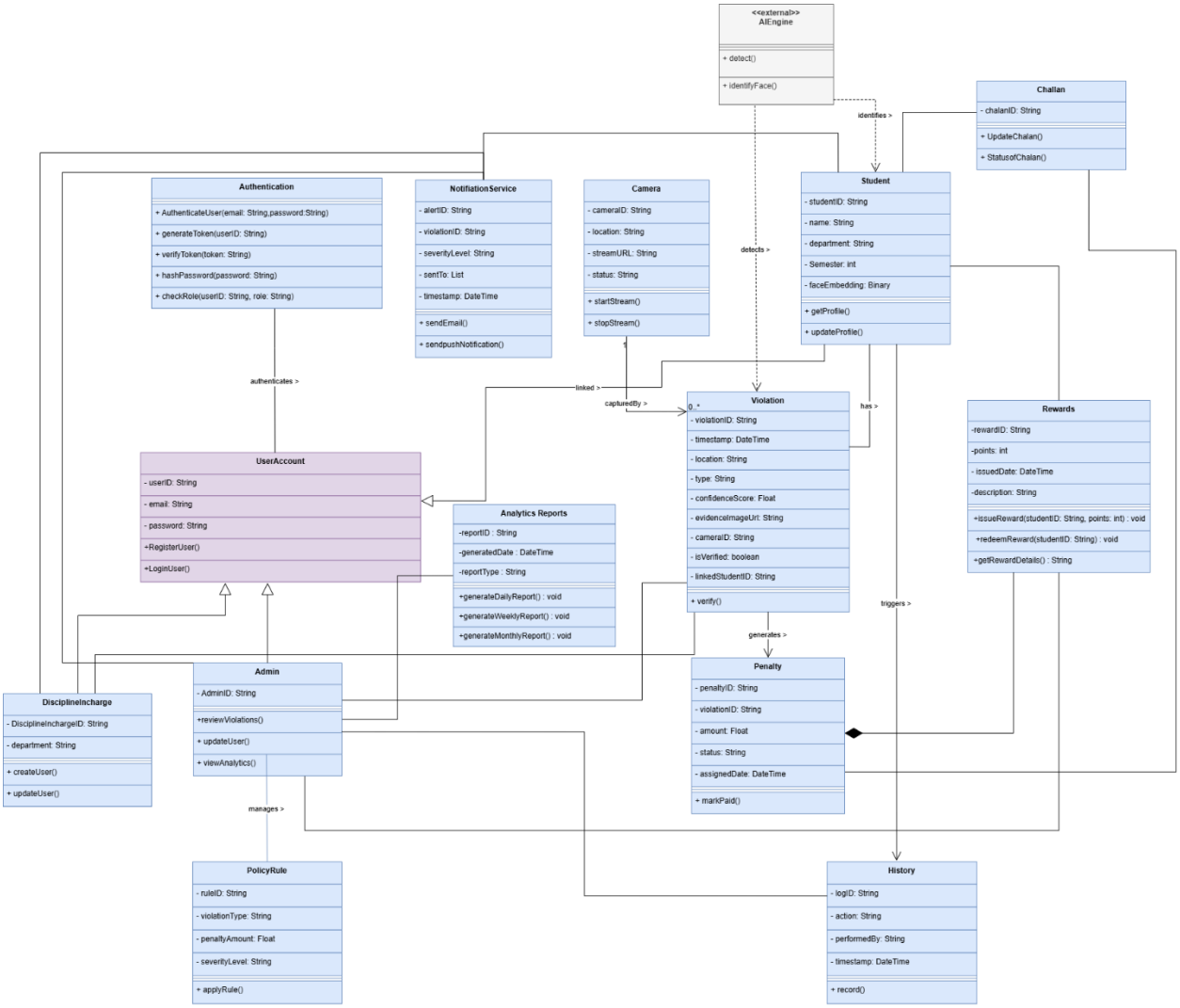
4.2. Domain Model



4.3. Entity Relationship Diagram

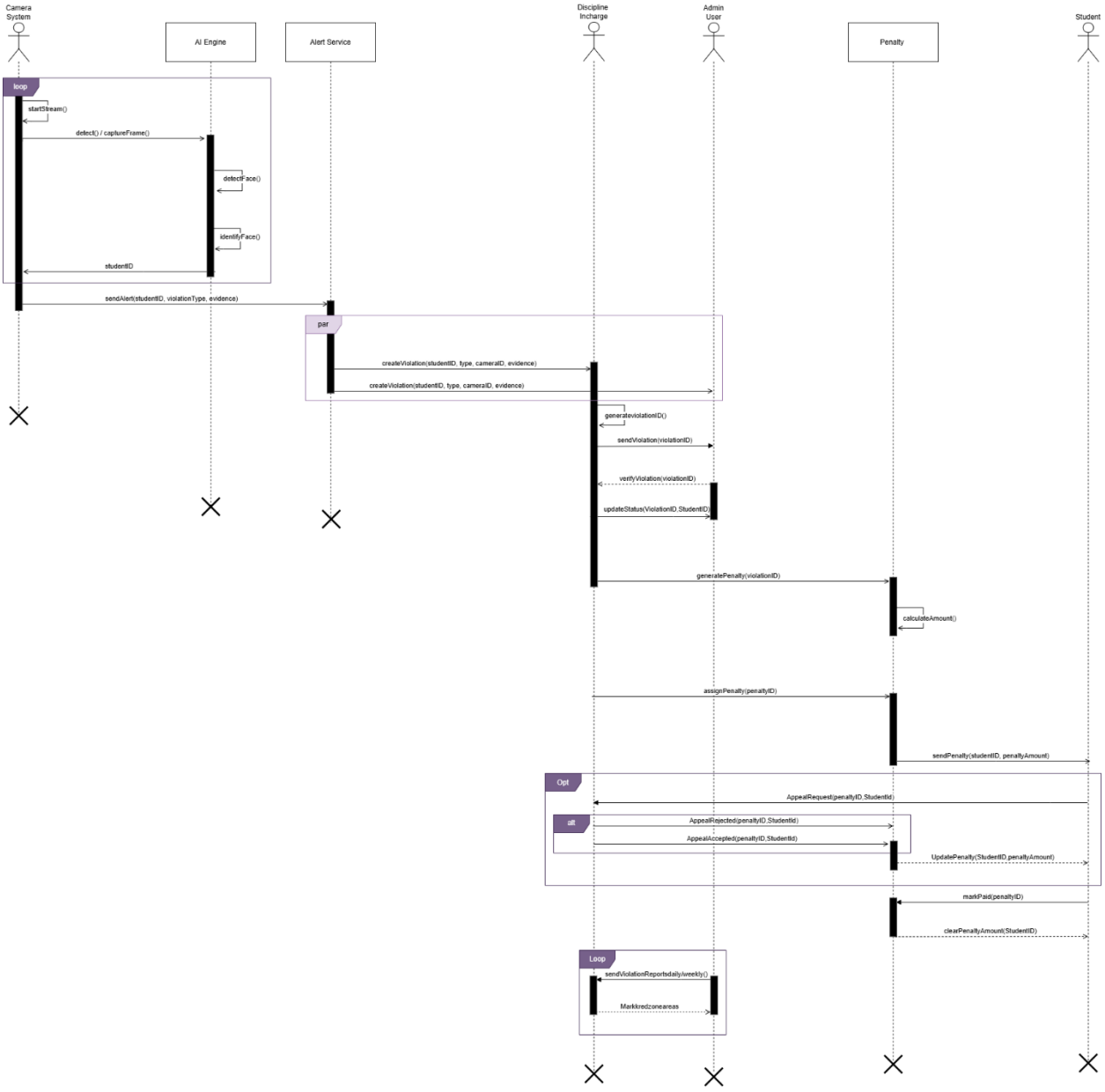


4.4. Class Diagram

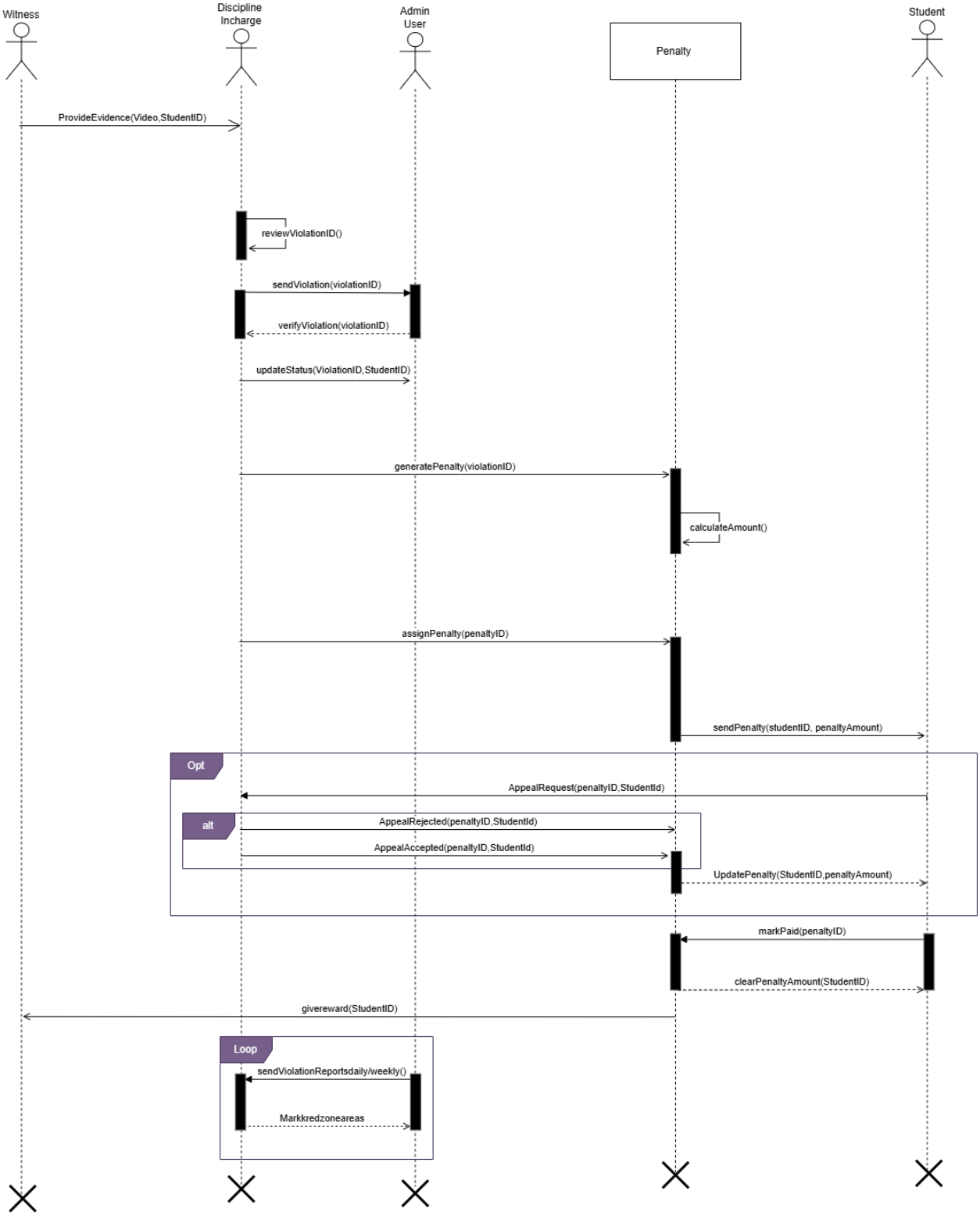


4.5. Sequence / Collaboration Diagram

AI Engine Automatic:



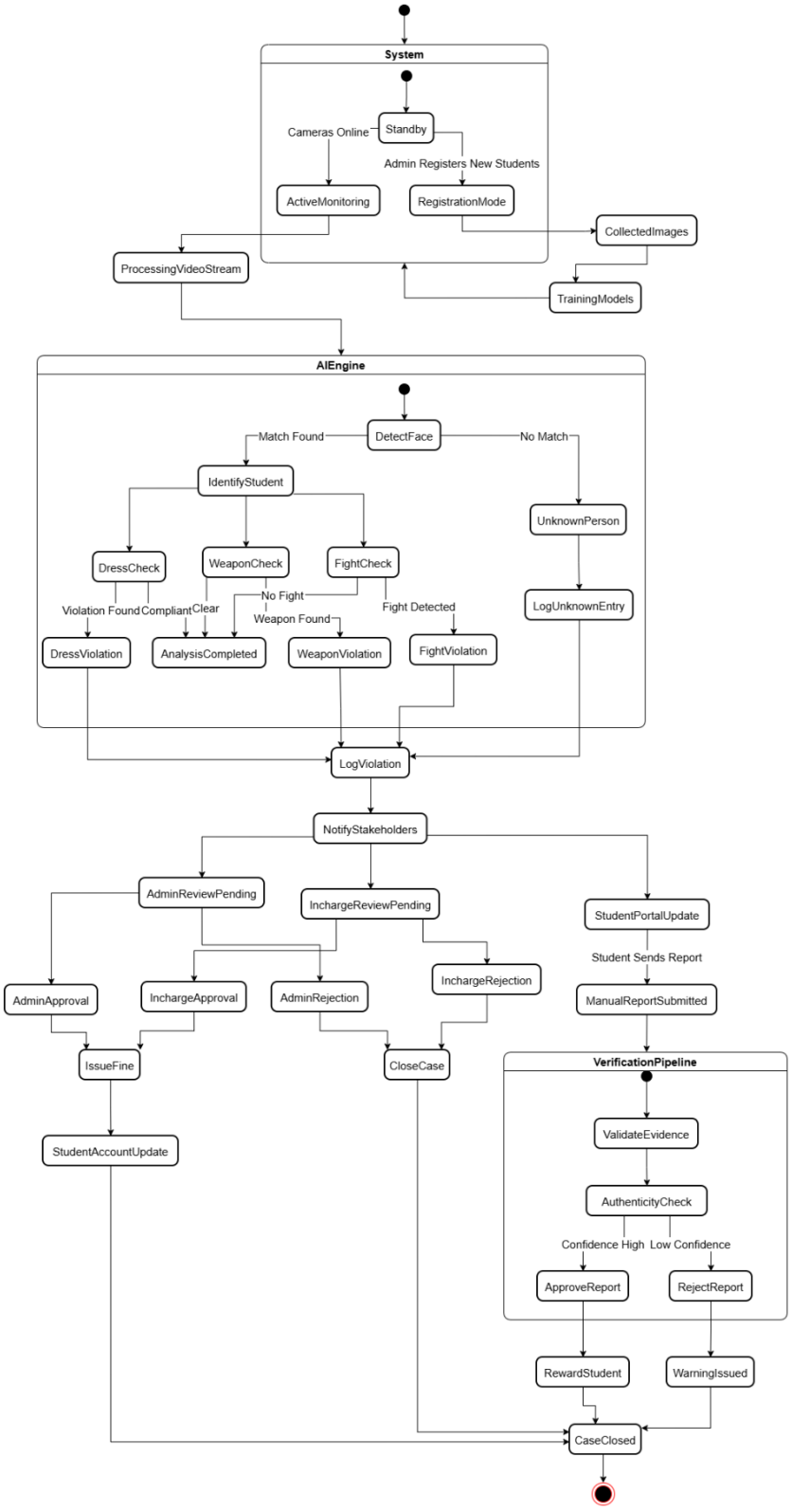
Manual-Reporting:

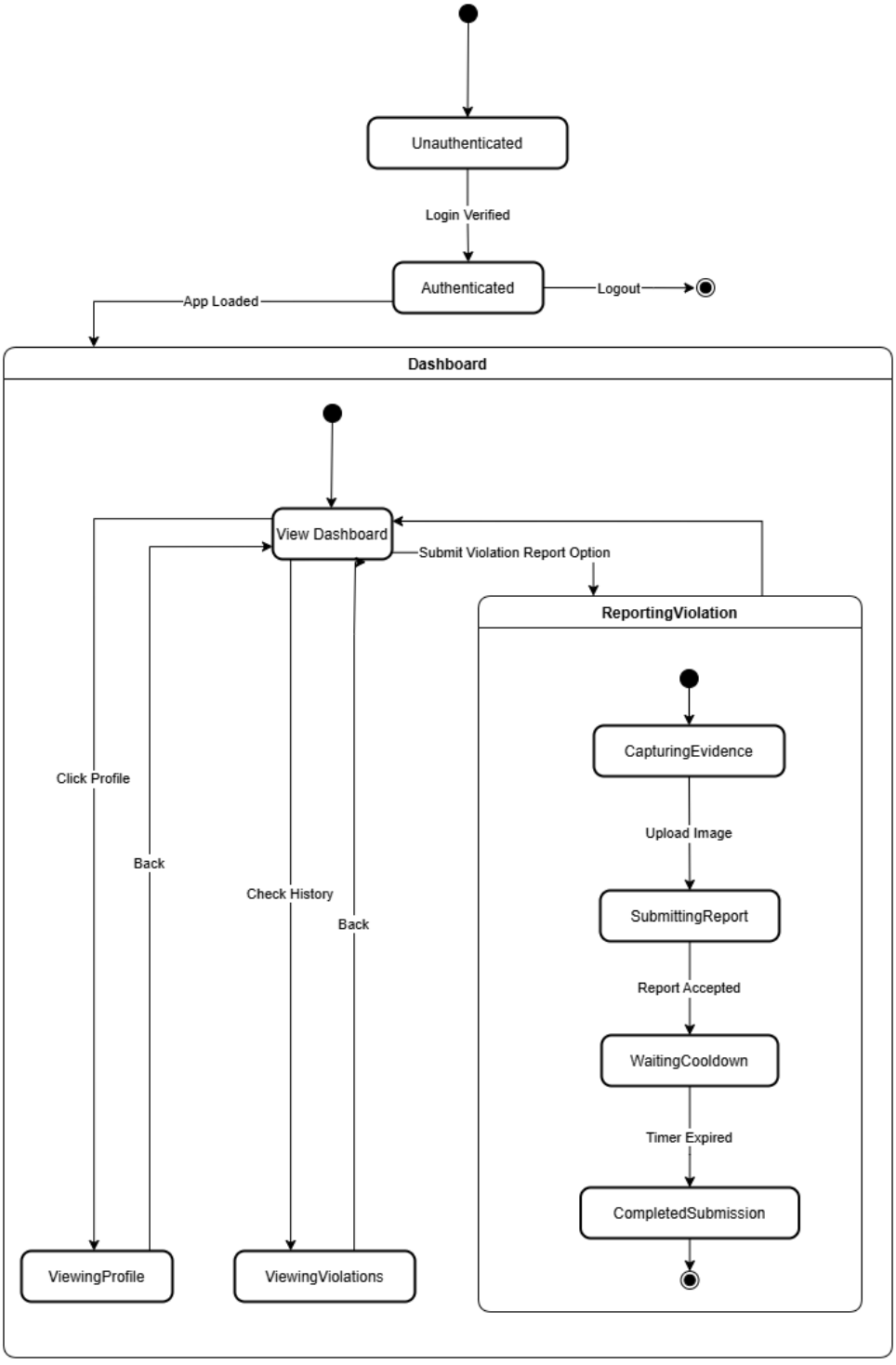


4.6. Activity Diagram

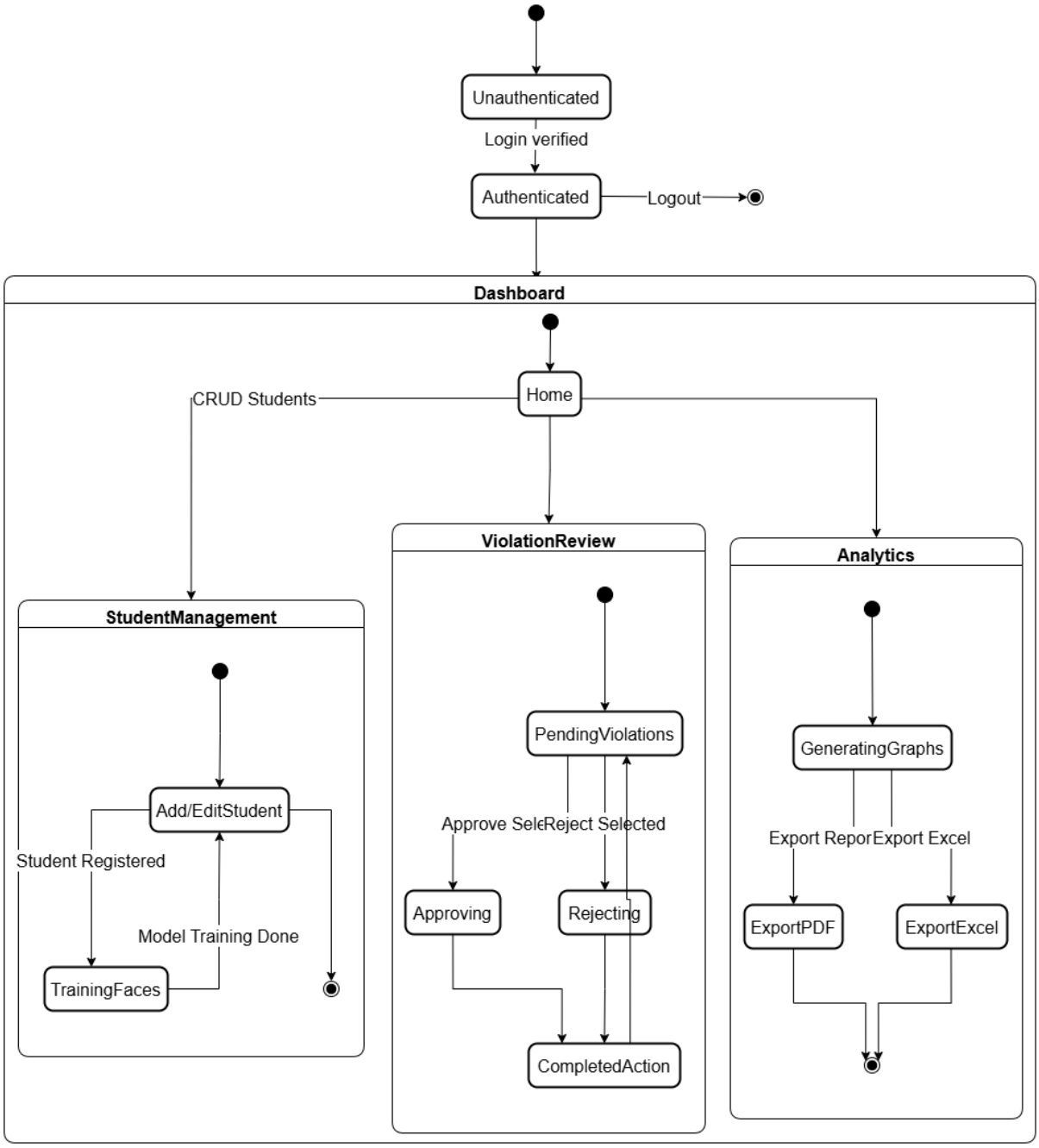
4.7. State Transition Diagram

System

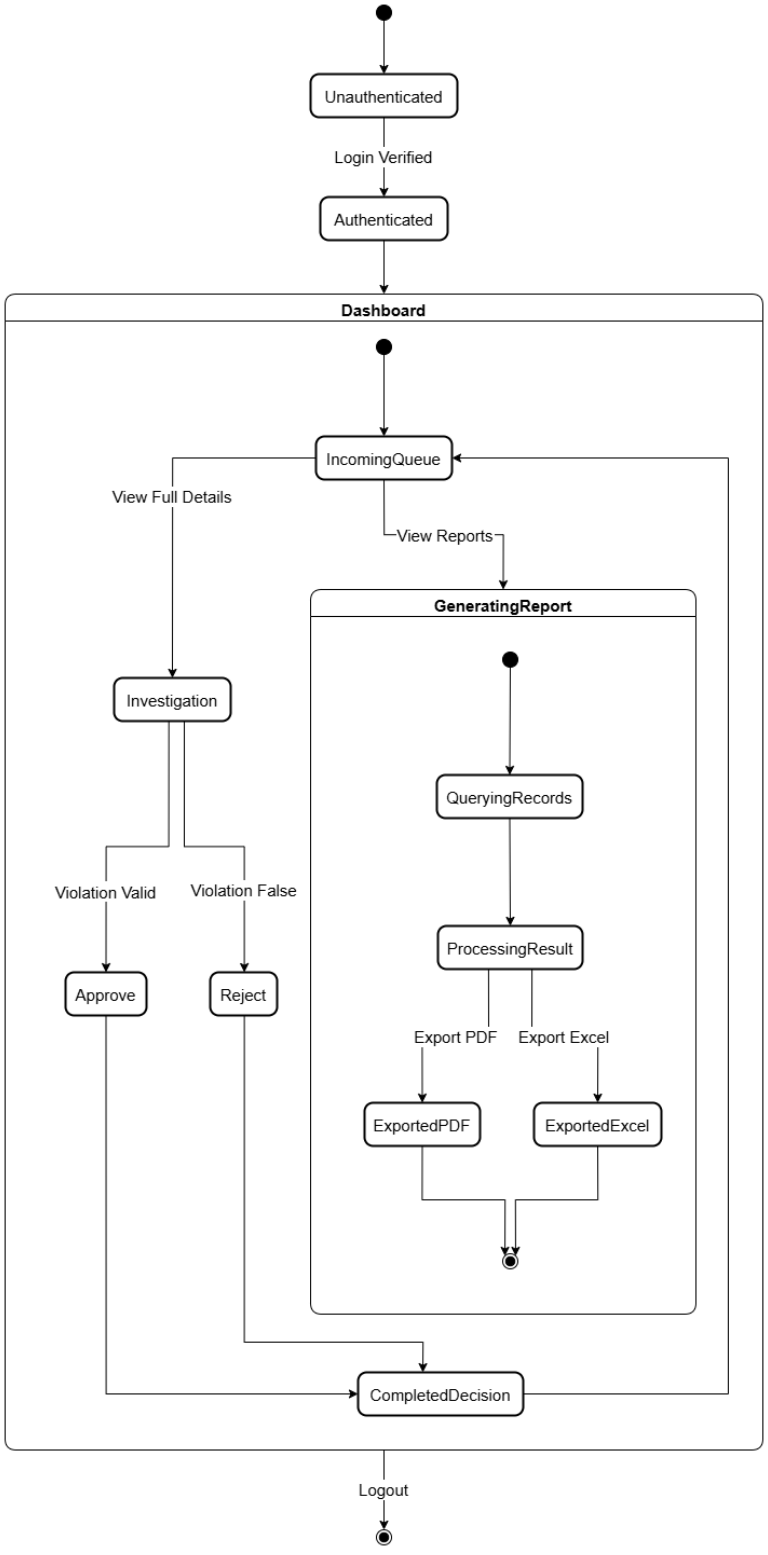



Unauthenticated
Login Verified
Authenticated
Logout
App Loaded
Dashboard
View Dashboard
Submit Violation Report Option
ReportingViolation
CapturingEvidence
Upload Image
SubmittingReport
Report Accepted
WaitingCooldown
Timer Expired
CompletedSubmission
Click Profile
Back
Check History
Back
ViewingProfile
ViewingViolations

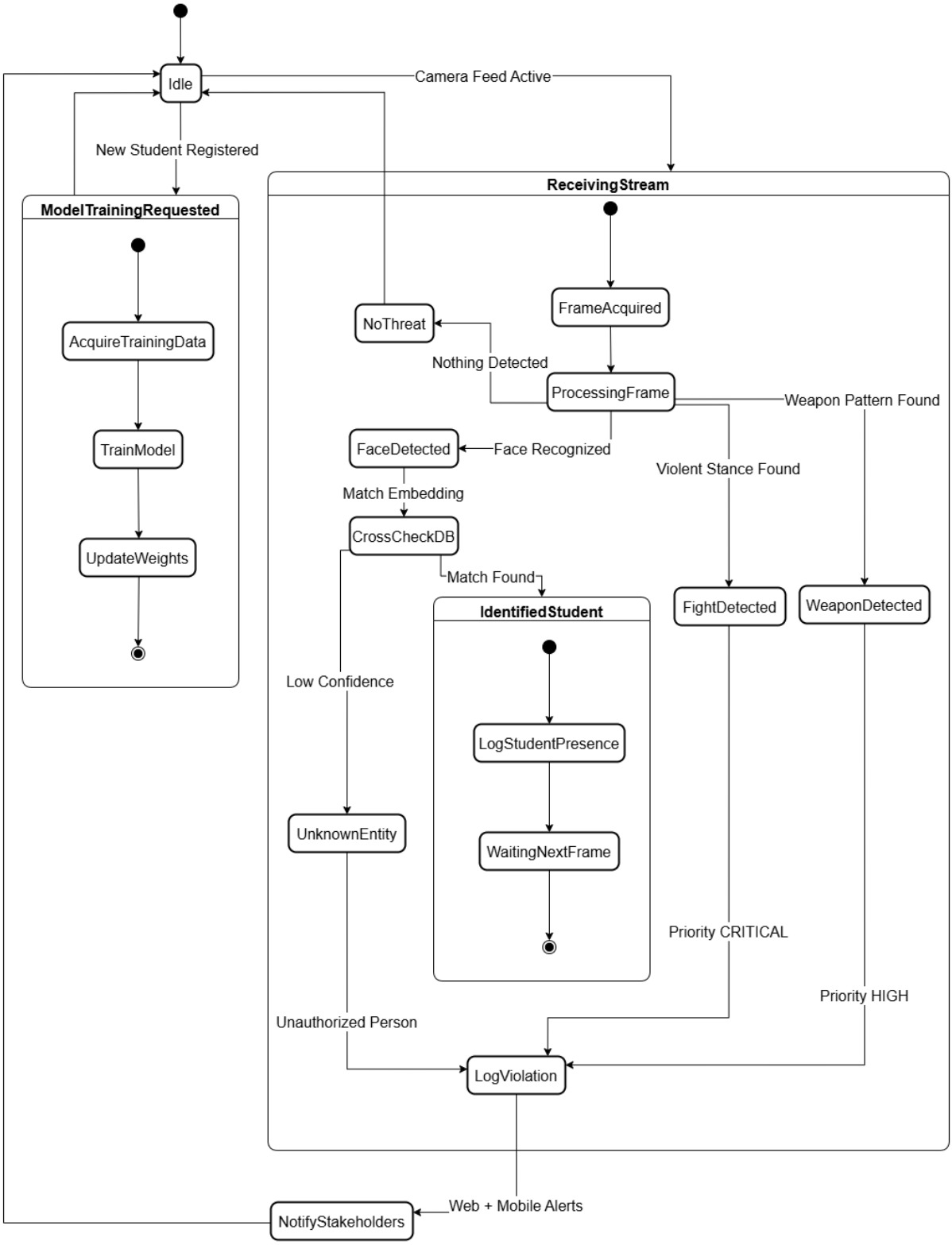
Student


Unauthenticated
Login verified
Authenticated
Logout
Dashboard
Home
CRUD Students
StudentManagement
Add/EditStudent
Student Registered
Model Training Done
TrainingFaces
ViolationReview
PendingViolations
Approve Sele
Reject Selected
Approving
Rejecting
CompletedAction
Analytics
GeneratingGraphs
Export Repor
Export Excel
ExportPDF
ExportExcel

Admin

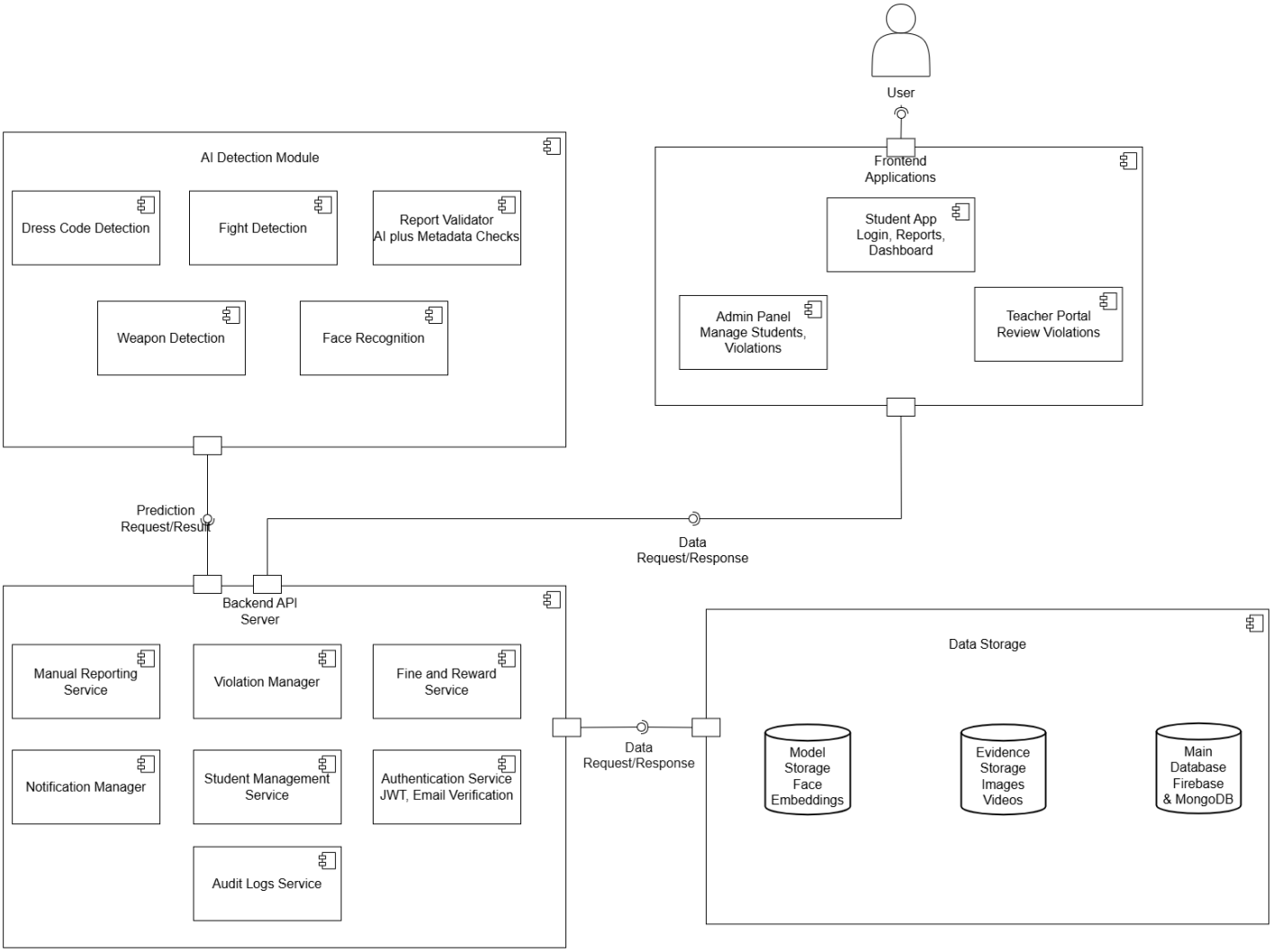


Discipline Incharge

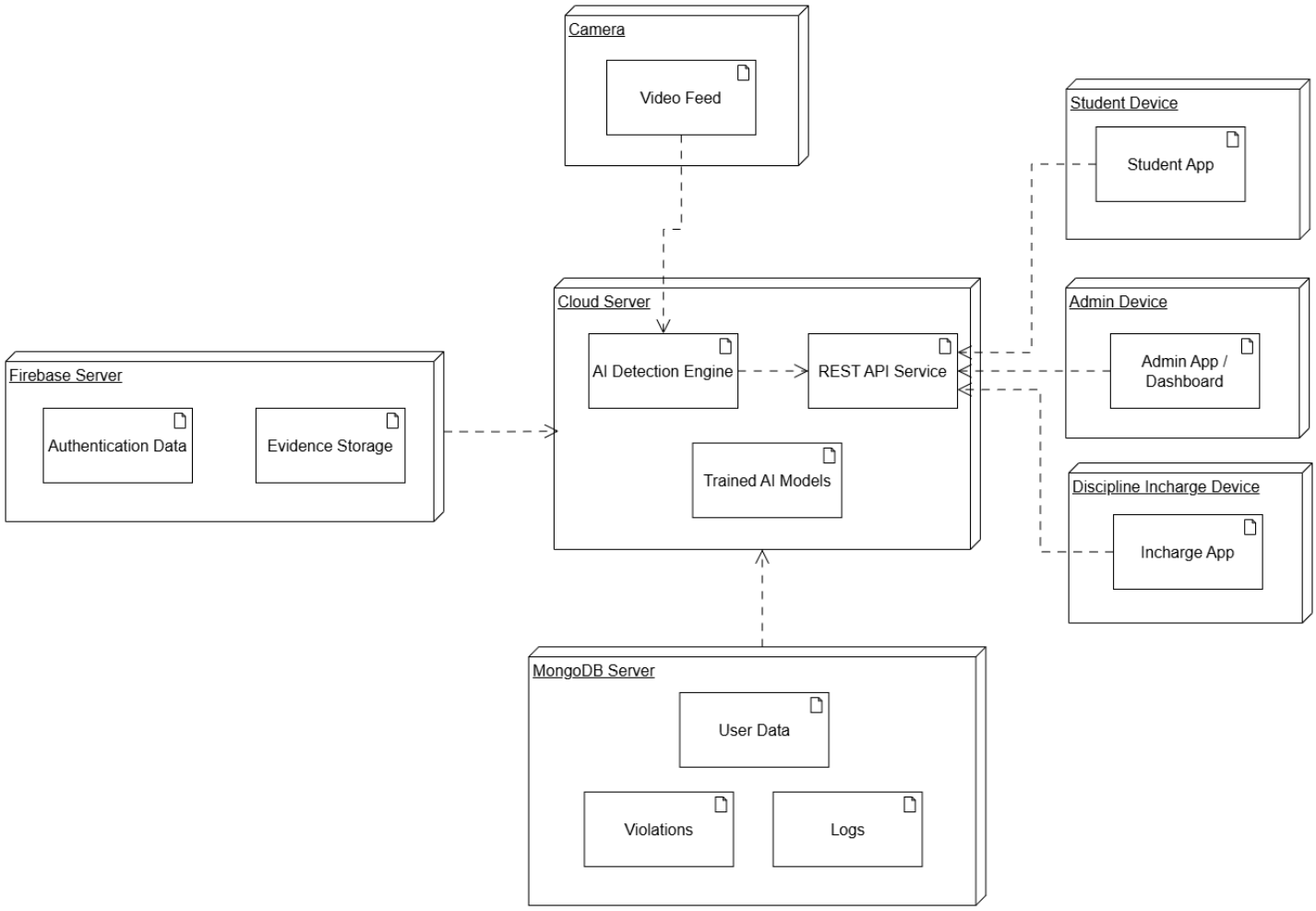

Idle
Camera Feed Active
New Student Registered
ModelTrainingRequested
AcquireTrainingData
TrainModel
UpdateWeights
ReceivingStream
FrameAcquired
NoThreat
Nothing Detected
ProcessingFrame
Weapon Pattern Found
FaceDetected
Face Recognized
Violent Stance Found
Match Embedding
CrossCheckDB
Match Found
IdentifiedStudent
FightDetected
WeaponDetected
Low Confidence
LogStudentPresence
UnknownEntity
WaitingNextFrame
Priority CRITICAL
Priority HIGH
Unauthorized Person
LogViolation
Web + Mobile Alerts
NotifyStakeholders

AI Engine

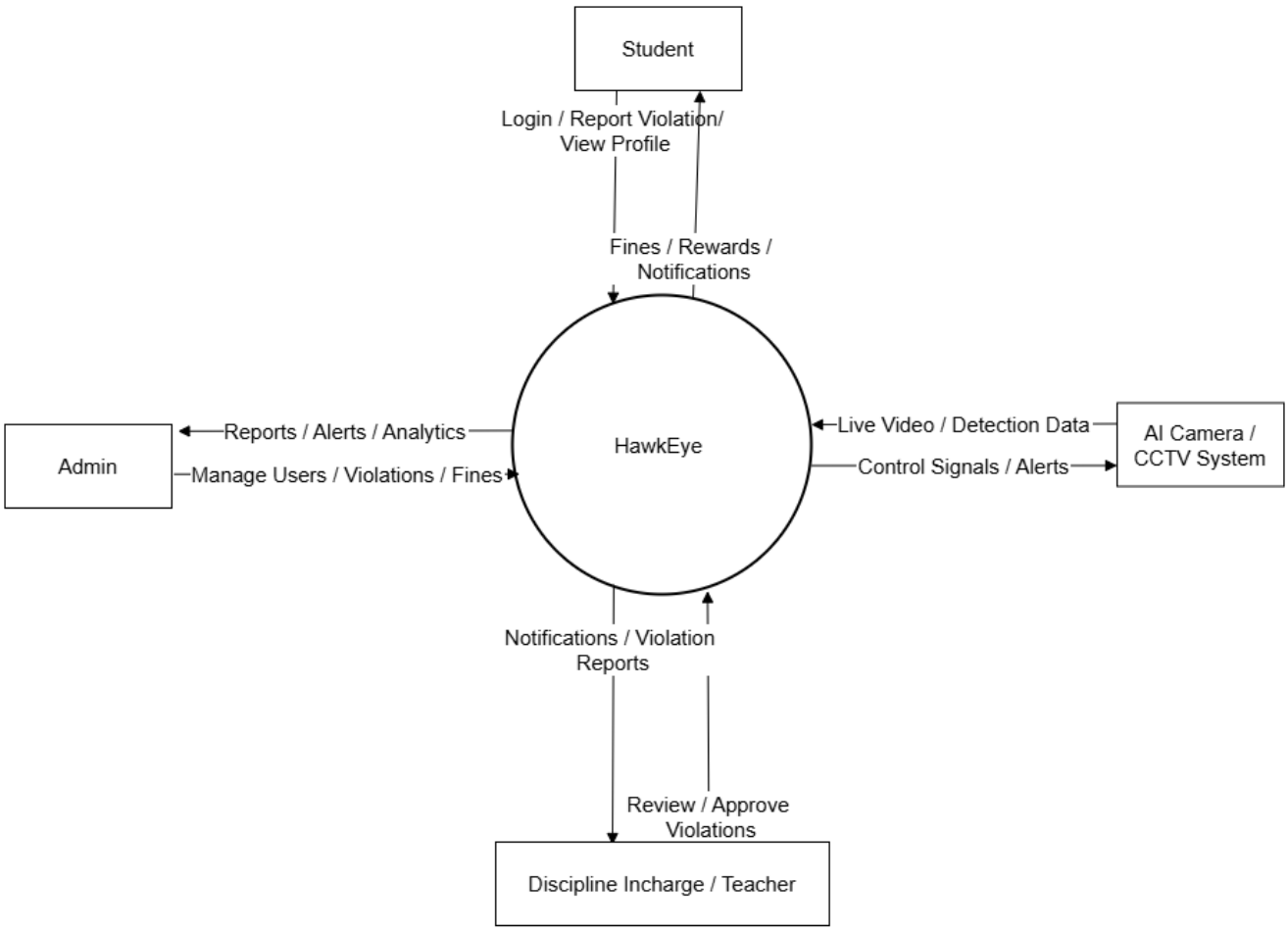
4.8. Component Diagram



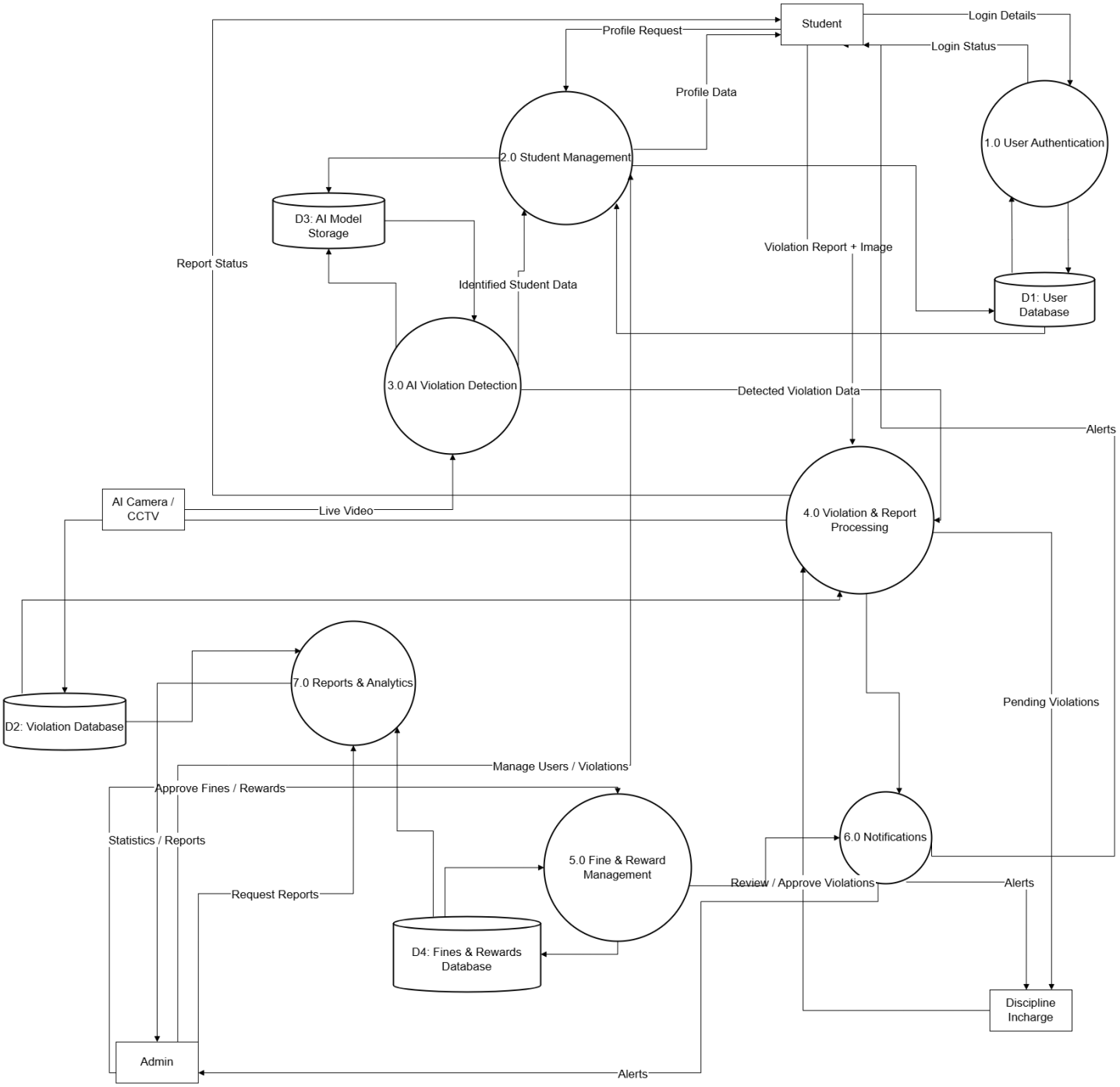
4.9. Deployment Diagram



4.10. Data Flow diagram



DFD 0


Student
Login Details
Profile Request
Login Status
Profile Data
2.0 Student Management
1.0 User Authentication
D3: AI Model Storage
Violation Report + Image
Report Status
Identified Student Data
D1: User Database
3.0 AI Violation Detection
Detected Violation Data
Alerts
AI Camera / CCTV
Live Video
4.0 Violation & Report Processing
7.0 Reports & Analytics
D2: Violation Database
Pending Violations
Manage Users / Violations
Approve Fines / Rewards
Statistics / Reports
5.0 Fine & Reward Management
6.0 Notifications
Request Reports
Review / Approve Violations
Alerts
D4: Fines & Rewards Database
Discipline Incharge
Admin
Alerts

DFD 1

Chapter 5
Implementation

Chapter 5: Implementation

This chapter explains the complete implementation of *HawkEye*, an AI-based Campus Discipline Monitoring System. It covers the system's internal workflows, major components, integration of AI modules, and the development of web and mobile interfaces. The chapter also highlights the deployment environment, tools and libraries used, and the coding standards and version control strategies followed throughout development. The purpose of this chapter is to demonstrate how the conceptual design of HawkEye was transformed into a fully functioning, AI-powered, multi-platform campus monitoring solution.

5.1. Important Flow Control/Pseudo codes

System Login & User Workflow

```
BEGIN System
    DISPLAY LoginScreen
    INPUT userID, password
    userRole = AUTHENTICATE(userID, password)
    IF userRole = INVALID THEN
        DISPLAY "Invalid Credentials"
        EXIT
    ENDIF
    LOAD Dashboard(userRole)
    LOOP
        DISPLAY menu based on userRole
        IF userRole = "Student" THEN
            OPTIONS:
                - View Violations
                - Report Violation
                - View Rewards
                - Logout
        ELSE IF userRole = "Admin" THEN
            OPTIONS:
                - Manage Users
                - View All Violations
                - View Analytics
                - Logout
        ELSE IF userRole = "Discipline Incharge" THEN
            OPTIONS:
```

```
                - Review AI Violations
                - Review Student Reports
                - Generate Reports
                - Logout
        ENDIF
        INPUT choice
        EXECUTE choice
        IF choice = Logout THEN
            BREAK
        ENDIF
    END LOOP
END
```

AI Detection Workflow

```
FUNCTION ProcessCameraFeed(frame)

    studentID = FacialRecognition(frame)

    IF studentID = "Unknown" THEN
        IGNORE frame
        RETURN
    ENDIF

    // Check Dress Code
    IF CheckDressCode(frame, studentID) = "Violation" THEN
        CALL LogViolation("Dress Code", studentID, frame)
    ENDIF

    // Check for Weapons
    IF DetectWeapons(frame) = TRUE THEN
        CALL LogViolation("Weapon Possession", studentID, frame)
    ENDIF

    // Check for Fights
    IF DetectFight(frameSequence) = TRUE THEN
        CALL LogViolation("Fight Detected", studentID, frame)
    ENDIF

END FUNCTION
```

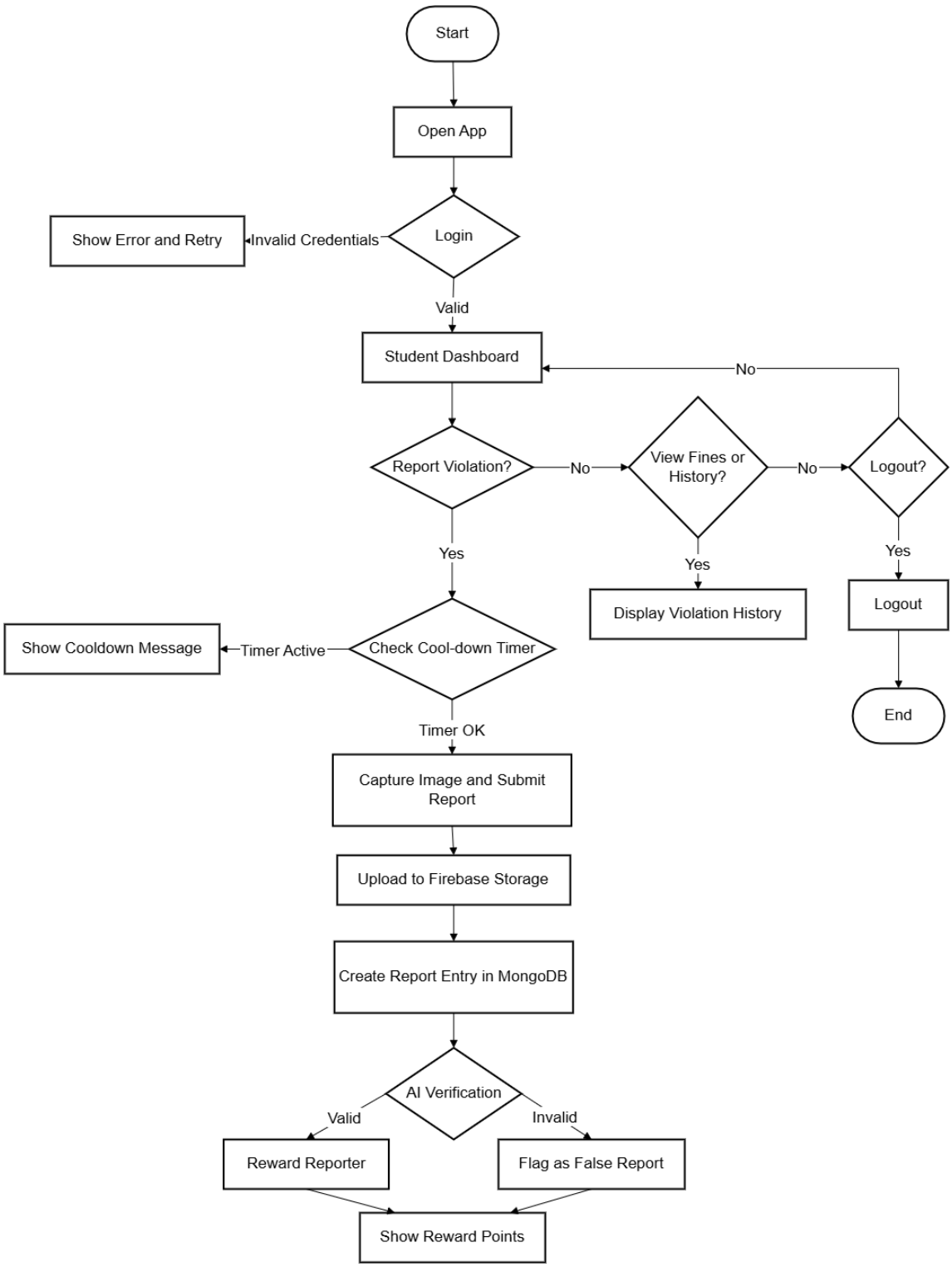
Violation Handling (Covers Reporting, AI Verification, Fines, Rewards, Notifications)

```
FUNCTION HandleViolation(type, studentID, evidenceImage)

    evidenceURL = UPLOAD_TO_FIREBASE(evidenceImage)

    CREATE violationRecord:
        - studentID
        - type
        - evidence = evidenceURL
        - timestamp = NOW

    SAVE violationRecord TO MongoDB

    // Automatically assign fine
    fineAmount = GET_FINE(type)
    SAVE FineRecord(studentID, fineAmount, type)

    // Send notifications
    SEND_NOTIFICATION(Admin, "New Violation Detected")
    SEND_NOTIFICATION(Incharge, "Violation: " + type)
    SEND_NOTIFICATION(studentID, "A violation has been logged")

END FUNCTION
```

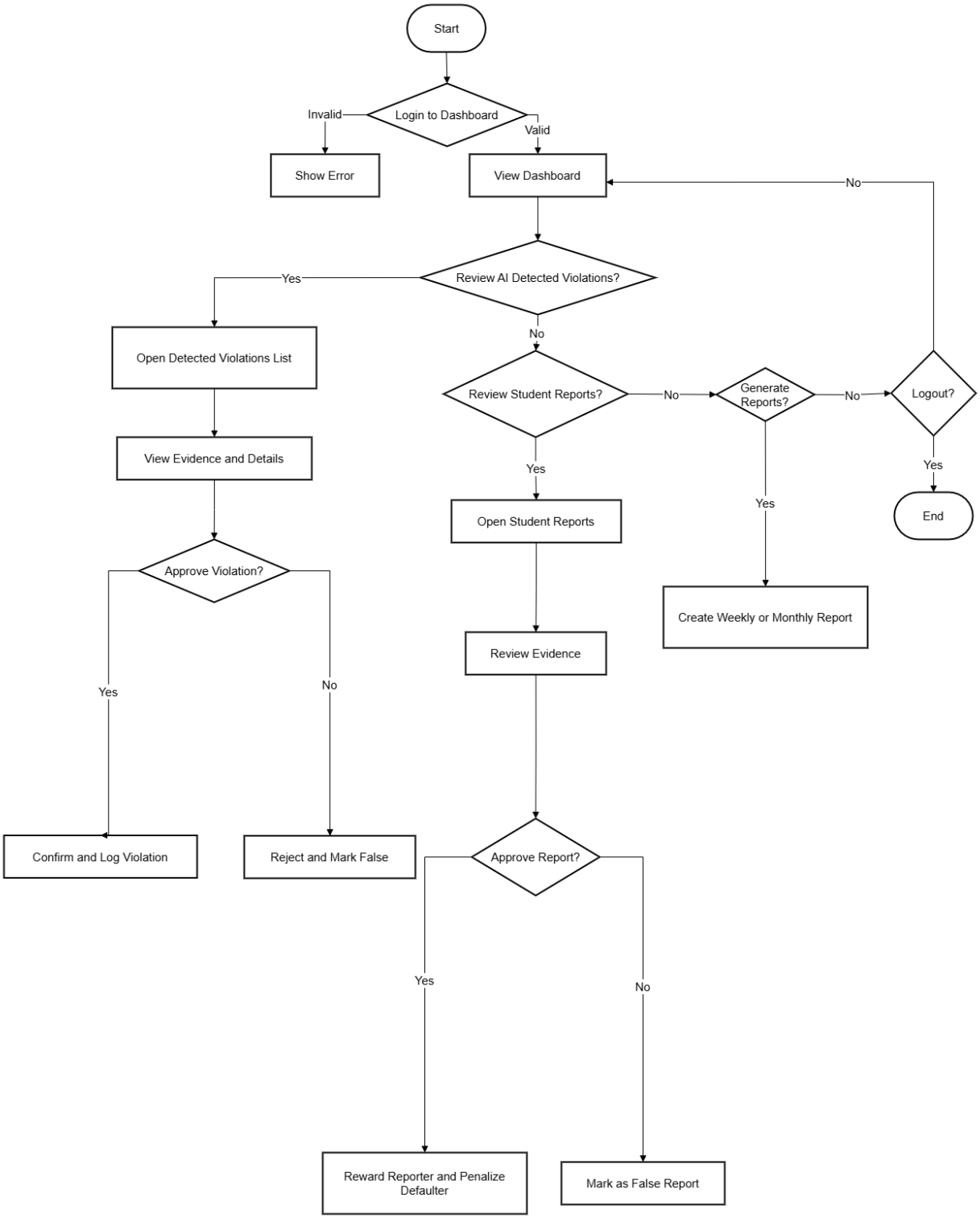
Student manual reporting:

```
FUNCTION StudentReport(studentID, description, image)

    evidenceURL = UPLOAD_TO_FIREBASE(image)

    verification = AI_VERIFY(image)

    IF verification = "Valid" THEN
        defaulterID = IDENTIFY_DEFUALTER(image)
        CALL HandleViolation("Manual Report", defaulterID, image)
        AWARD_POINTS(studentID)
    ELSE
        FLAG_FOR_FALSE_REPORT(studentID)
    ENDIF

END FUNCTION
```

Flow Charts:

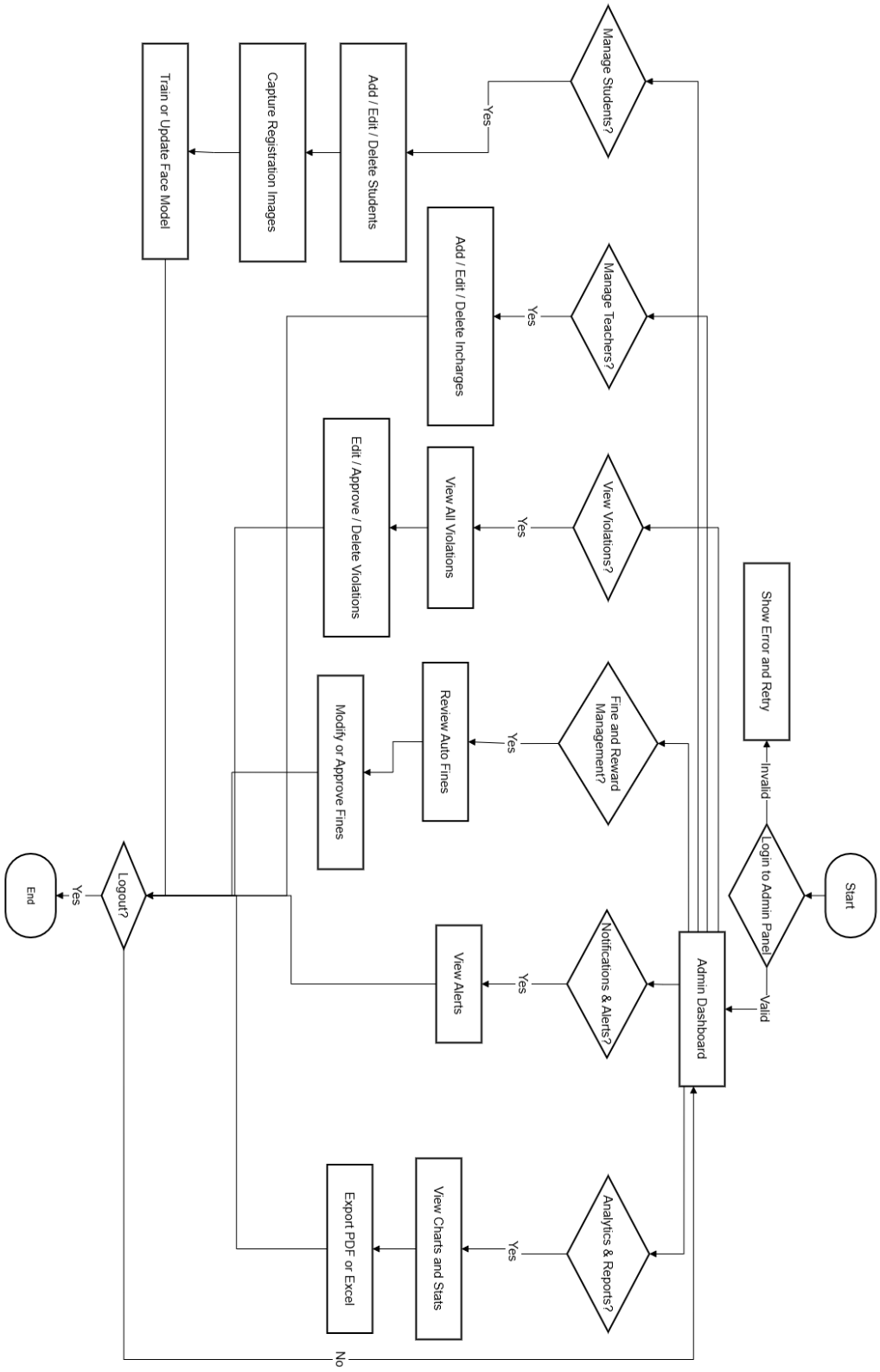


Student



Discipline Incharge

Admin


Start
Login to Admin Panel
Invalid
Show Error and Retry
Valid
Admin Dashboard
Manage Students?
Yes
Add / Edit / Delete Students
Capture Registration Images
Train or Update Face Model
Manage Teachers?
Yes
Add / Edit / Delete Incharges
View Violations?
Yes
View All Violations
Edit / Approve / Delete Violations
Fine and Reward Management?
Yes
Review Auto Fines
Modify or Approve Fines
Notifications & Alerts?
Yes
View Alerts
Analytics & Reports?
Yes
View Charts and Stats
Export PDF or Excel
Logout?
Yes
End
No

5.2. Components, Libraries, Web Services and stubs

HawkEye consists of multiple interconnected components divided into Web, Mobile, Backend, and AI modules. The MERN stack is used for dashboards, while Flutter powers the student mobile app. MongoDB and Firebase provide secure and scalable storage for user data, images, logs, and notifications.

Major Components

- **Admin Dashboard (React.js)**
  View, filter, and manage violations; manage users; view analytics.
- **Discipline Incharge Dashboard**
  Validate AI-detected events and review student reports.
- **Student Mobile App (Flutter)**
  Report violations, view personal fines, access rewards, and chat with incharge.
- **Backend Server (Node.js + Express)**
  API endpoints for CRUD, model integration, authentication, and logging.
- **AI Modules (Python / TensorFlow / OpenCV / YOLOv8)**
  Real-time face recognition, weapon detection, and fight detection.

Libraries Used

- **Front-end**: React, Redux, Material UI, Flutter SDK
- **Backend**: Express.js, JWT, Firebase SDK, Multer
- **AI**: TensorFlow, Keras, OpenCV, YOLOv8, PyTorch
- **Database**: MongoDB Atlas, Firebase Firestore, Firebase Storage
- **Communication**: Socket.IO for real-time chat and live updates

Web Services

- Live camera feed processing endpoints
- Facial model training & update service
- Violation logging REST API
- Notification service using Firebase Cloud Messaging (FCM)

Stubs Used

During early development, offline stubs were used for:

- Simulated violation detection
- Dummy facial recognition responses
- Mock admin data
- Fake notification triggers

5.3. Deployment Environment

HawkEye is deployed in a hybrid cloud environment ensuring performance, reliability, and continuous availability.

Server Environment

- **Node.js Backend Server**
  Hosted on cloud (AWS / Render / Railway)
- **MongoDB Atlas**
  Cloud-hosted database for all student records, logs, and violation history.
- **Firebase Cloud**
  Used for:
  - Storage (images, evidence)
  - Authentication
- **AI Processing Server**
  GPU-enabled machine for running YOLO, Facial Recognition, Fight Detection.

Client Environment

- **Web Dashboards**: Chrome/Edge compatible
- **Mobile App**: Android/iOS (Flutter-based)
- **Cameras / Webcams**: Used as live video input devices

Deployment Considerations

- Secure API gateways
- JWT-based role authentication
- Load balancing for high activity periods
- Background workers for continuous video inference

5.4. Tools and Techniques

Development Tools

- Visual Studio Code (MERN development)
- Visual Studio Code (Flutter development)
- Jupyter Notebook / PyCharm (AI Model training)
- Postman (API testing)
- Git & GitHub (version control)
- Firebase Console (notifications & storage)

Techniques Used

- Modular coding
- Microservice-inspired backend
- RESTful API architecture
- Continuous Integration (CI)

5.5. Best Practices / Coding Standards

To maintain a clean and robust codebase, the following practices were implemented:

Frontend Standards

- Component-based structure in React
- Separation of UI and business logic
- Reusable Flutter widgets
- Clean state management

Backend Standards

- Layered architecture (routes → controllers → services → models)
- Use of environment variables (.env)
- Centralized error handling

AI Coding Standards

- Modules for each model
- Versioned datasets for reproducibility
- Continuous model retraining when new students are registered

General Coding Practices

- Meaningful variable names
- Consistent indentation
- API documentation
- Git-branching standards (dev, main, feature branches)

5.6. Version Control

Git and GitHub were used as the primary version control system for all components — Web, Mobile, Backend, and AI.

Version Control Strategy

- **Main Branch**: Stable, deployable version
- **Development Branch**: Active development
- **Feature Branches**: Dedicated branches for AI modules, UI features, etc.
- **Pull Requests**: Used for merging and review
- **Commit Messages**: Semantic structure (fix:, feat:, update:, refactor:)

Repository Organization

- /frontend-admin (React Dashboard)
- /frontend-incharge (React Dashboard)
- /flutter-student-app (Mobile App)
- /backend-node (Node.js APIs)
- /ai-models (YOLO, Facial Recognition, Fight Detection)

This structure ensured seamless collaboration, rollback capability, and clear tracking of changes across the entire project.

Appendices

Appendix A: Information / Promotional Material

[Paragraph Text 12 pt, Calibri, 1.5 Line Spacing, Justified]

[*Between 4 to 8 lines describe what is this appendix all about*]

A.1. Broacher

A.2. Flyer

A.3. Standee

A.4. Banner

A.5. First Level heading [16 pt, Calibri, Bold, Left aligned]

[Paragraph Text 12 pt, Calibri, 1.5 Line Spacing, Justified]

A.5.1. Second level heading [14 pt, Calibri, Bold, Left aligned]

[Paragraph Text 12 pt, Calibri, 1.5 Line Spacing, Justified]

A.1.1.1. Third level heading [12 pt, Calibri, Bold, Left aligned]

[Paragraph Text 12 pt, Calibri, 1.5 Line Spacing, Justified]

Appendix [no.]: Appendix Title

[Paragraph Text 12 pt, Calibri, 1.5 Line Spacing, Justified]

[*Between 4 to 8 lines describe what is this chapter all about*]

A.1. First Level heading [16 pt, Calibri, Bold, Left aligned]

[Paragraph Text 12 pt, Calibri, 1.5 Line Spacing, Justified]

A.1.1. Second level heading [14 pt, Calibri, Bold, Left aligned]

[Paragraph Text 12 pt, Calibri, 1.5 Line Spacing, Justified]

A.1.1.2. Third level heading [12 pt, Calibri, Bold, Left aligned]

[Paragraph Text 12 pt, Calibri, 1.5 Line Spacing, Justified]

Reference and Bibliography

Reference and Bibliography

[1] M. Sher, M. Rehman, "*Title of the Paper*" Conference name/Journal Name, Edition, Volume, Issue, ISBN/ISSN, PP, Publisher/City-Country, Year.

[2] ……