CHAPTER 1. INTRODUCTION

1.1 PROJECT OVERVIEW

SAFEEYE (Real-time Weapon Detection and Alerting System) aims to enhance security by detecting guns and knives in live or offline video streams and images using Machine Learning and Python. The system leverages computer vision techniques, particularly YOLOv8n for real-time detection and VGG16 as a baseline model for reference accuracy, to identify weapons and distinguish them from everyday items, minimizing false alarms. By integrating pre-trained object detection models, the system can process video feeds in real time, providing accurate and efficient detection. It can be deployed in environments like public spaces, airports, banks, schools, shopping mall and government buildings to bolster safety.

Upon weapon detection, the system generates alerts, notifying security personnel via alarms or emails, enabling quick responses to potential threats. Key libraries such as OpenCV for video processing, TensorFlow or PyTorch for machine learning, and NumPy for numerical operations are essential in developing and optimizing the system. This solution can be integrated into existing surveillance infrastructure, providing an additional layer of security with automated weapon detection and prompt alerts for timely interventions. The integration of machine learning algorithms, specifically object detection models, enables the system to distinguish between everyday items and potentially harmful objects, ensuring that false alarms are minimized.

1. **Machine Learning Models**: Pre-trained object detection models, such as YOLOv8n for real-time detection and VGG16 as a baseline model, are employed to identify and classify various types of weapons.
2. **Python Libraries and Frameworks**: Libraries such as OpenCV (for video processing), TensorFlow or PyTorch (for machine learning), and NumPy (for numerical operations) are crucial for implementing and optimizing the solution.
3. **Live or Offline Video Processing**: The system processes live or offline video streams from CCTV visuals or videos.
4. **Alert System**: Once a weapon is detected, the system generates an alert, which can be sent to security personnel via alarm sound or email.

1.2 OBJECTIVES

The primary objective of the **SAFEEYE** is to **develop an automated, real-time solution** that can accurately detect the presence of weapons in images or video streams from surveillance cameras, and immediately alert security personnel or authorities about potential threats. This can help to prevent violence, reduce response times, and increase the overall safety of public spaces, institutions, or events.

Specific Objectives:

1. **Weapon Detection**:
   - ❖ Develop a system that can automatically detect weapons (such as guns, knives) from images or video feeds.
   - ❖ Ensure high accuracy in recognizing types of gun and knife from different angles.
2. **Real-Time Processing**:
   - ❖ Design the system to process video streams in real-time, ensuring that weapons can be detected instantly as they appear in the footage.
   - ❖ Optimize the system for low latency performance to ensure that alerts are triggered quickly.
3. **Automated Alerting System**:
   - ❖ Implement an alert mechanism that automatically sends notifications to security personnel or law enforcement when a weapon is detected.
   - ❖ Alerts could be alarm sound or a mail with message ‘Weapon Detected’ and a real-time feed or snapshot from the camera.
4. **Scalability and Integration**:
   - ❖ Ensure that the weapon detection system can be scaled to handle multiple cameras or large environments, such as banks, schools, or campuses.
   - ❖ Integrate the system with existing security infrastructures, such as CCTV surveillance systems or central alarm systems.
5. **Accuracy and Reliability**:
   - ❖ Achieve a high level of accuracy in detecting weapons, minimizing both false positives (non-weapons detected as weapons) and false negatives (actual weapons not detected).

- ❖ Evaluate and fine-tune the system’s performance over time to handle edge cases and environmental challenges.

1.3 SCOPE

Here is the scope of the **SAFEEYE** using Python and Machine Learning:

1. **Real-Time Weapon Detection**: Utilizes machine learning models (YOLOv8n for real-time detection and VGG16 as a baseline model) for weapon detection in video streams. Identifies weapons such as fire arms and knives. Processes live videos or image feeds from CCTV cameras, webcams, and uploaded videos. Sends immediate alerts via alarm sound and email notifications to security personnel with relevant details. Provides real-time feedback by overlaying bounding boxes and weapon labels on the video stream for verification. Minimizes false positives and false negatives to improve detection accuracy.
2. **Integration with Existing Security Infrastructure**: Compatible with CCTV surveillance at entrances of banks, colleges, and schools. Designed to monitor specific high-risk entry points through a simple and efficient integration process. Operates efficiently under different environmental conditions, handling challenges like lighting variations and partial object occlusion. Easily adaptable to existing security workflows and infrastructure without the need for complex cloud setups or multi-camera systems.
3. **System Optimization and Security**: Optimized for performance using lightweight models suitable for real-time deployment. Ensures quick processing with minimal latency and high accuracy. Detection models can be periodically updated with new training data to maintain accuracy as new weapon types and patterns emerge. Focuses on enhancing security and public safety with proactive, automated weapon detection while ensuring simple and secure system operations.
4. **User-Friendly Web Interface**: Provides a simple and secure web platform where authorized users can log in and submit images, videos, or live feeds for weapon detection. Ensures ease of use through an intuitive interface built with HTML, CSS, and JavaScript, allowing quick uploads and live monitoring without technical knowledge. Displays real-time detection results with bounding boxes and labels, enhancing user confidence in system operations.

5. **Alert and Response Mechanism**: Incorporates an automatic alerting system that generates immediate alarm sounds on detection and sends email notifications to predefined authorities. Enables rapid decision-making and emergency response by providing timely and accurate information about the detected threat. Strengthens the first line of defence at sensitive entry points like banks, schools, and colleges by ensuring security teams are informed without delay.

CHAPTER 2. REQUIREMENTS/SURVEYS

2.1 PROBLEM SCENARIO

2.1.1 Description of the Issue:

In today's world, security is a growing concern, especially in crowded or high risk environments like airports, public events, schools, and shopping malls. The presence of weapons in these areas can pose serious threats to public safety and lead to devastating consequences. Traditional security measures, such as manual bag checks or human surveillance, are often slow, inefficient, or prone to human error. These methods are also not scalable, especially in crowded or large areas, which makes it difficult to detect weapons in real-time and respond to potential threats promptly.

The core problem is the inability to quickly and accurately detect weapons in a live environment, leading to delayed responses and increased risk to public safety. For instance, human security personnel may miss dangerous objects, or surveillance systems may not be able to process video feeds in real-time to identify threats. Additionally, manual monitoring of CCTV footage or scanning of bags can be resource intensive and time consuming.

Challenges include**:**

- **Detection Accuracy**: Ensuring that the system can distinguish weapons from nonthreatening objects without false positives.
- **Real-time Performance**: Developing a system that can identify weapons quickly enough to alert authorities in time to prevent violence.
- **Scalability**: Implementing the system in largescale environments with multiple cameras or entry points.
- **Privacy Concerns**: Balancing security measures with the need to respect individuals privacy rights.

The problem scenario emphasizes the need for an automated, scalable, and efficient system to detect weapons in public spaces and trigger immediate alerts, ensuring that security teams can respond swiftly to potential threats.

2.1.2 Context and Background

The primary goal of the project is to develop a system that uses **machine learning (ML)** and **computer vision** techniques to automatically detect weapons in images or video streams from surveillance cameras. By integrating this detection system with an alerting mechanism, the project aims to provide a proactive security solution capable of detecting weapons in real-time and notifying the relevant authorities or security teams [2].

Key Technologies:

Machine Learning (ML): Machine Learning, particularly deep learning, plays a crucial role in image classification and object detection for this project. Models are used to identify weapons (e.g., guns, knives) from visual data. Convolutional Neural Networks (CNNs) are utilized for processing images, with object detection models like YOLO and VGG16 employed to detect and classify weapons accurately.

Alerting System: Upon detecting a weapon, the system automatically triggers notifications, such as sending emails or activating alarm sounds, to alert security personnel and enable quick response to potential threats.

Python and Libraries: Python is used for implementing machine learning and computer vision tasks, utilizing libraries like TensorFlow, Keras, PyTorch, and OpenCV for model development and image processing. Django is used to build web interfaces that display video feeds, detection results, and alert notifications in real-time.

Real World Applications**:**

- **Public Safety**: Enhancing security in crowded areas like airports, shopping malls, and public events.
- **School and Campus Security**: Monitoring students, staff, and visitors for concealed weapons, preventing incidents before they occur.
- **Automated Surveillance**: Integrating with existing CCTV networks for realtime weapon detection, reducing reliance on human monitoring.
- **Border and Airport Security**: Detecting weapons in scanned images of luggage or on individuals walking through security checkpoints.

Key Challenges Addressed:

- ❖ **Speed**: The system needs to detect weapons in real-time, which is crucial in preventing potential threats before they escalate.
- ❖ **Accuracy**: The system must minimize false positives (wrongly identifying nonthreatening objects as weapons) and false negatives (failing to identify an actual weapon).
- ❖ **Scalability**: The solution must be able to handle large amounts of video data from multiple cameras without significant performance issues.
- ❖ **User Privacy**: The system should be designed to respect privacy rights by only focusing on detecting weapons, without storing unnecessary personal data or violating ethical guidelines.

2.2 HARDWARE REQUIREMENTS

SAFEEYE system requires a computer or server with sufficient processing power to handle deep learning model inference in real-time. The hardware should include a **high-performance CPU** (preferably with multiple cores) and a **dedicated GPU** (such as NVIDIA with CUDA support) to accelerate the training and inference of the YOLOv8 Nano model. For real-time video processing, a **webcam** or compatible camera is necessary for capturing live video feeds. Adequate **RAM** (at least 16GB) and **storage** (preferably SSD) are required to handle large datasets and a **stable internet connection** ensure smooth system performance.

Table 2.1 Hardware Requirements

Processor	Min. Intel i5 or Higher
RAM	8GB Or More
Hard Disk	300 GB or above
Monitor	Any standard display
Webcam	Laptop Integrated (Development/Testing)/) External HD or IP Camera (Real-Time Deployment)
Internet	High-speed connection for alert notifications and email services

2.3 SOFTWARE REQUIREMENTS

SAFEEYE relies on several key software tools for development and deployment. **Google Colab** was used for training the deep learning model with **YOLOv8 Nano**, utilizing libraries such as **TensorFlow**, **PyTorch**, and **OpenCV**. **Jupyter Notebook** is employed for implementing the live video detection functionality on a local system. The web application is built using the **Python Django framework**, with **SQLite** as the database. Additional software tools include **OpenCV** for real-time video processing, and email libraries for sending automated detection alerts.

Table 2.1 Software Requirements

Front End Tools	Html, CSS, JavaScript.
Back End Tool	Python, Django, SQLite Database
Operating System	Microsoft Windows
Machine Learning	Python, VGG16, YOLOv8 Nano (Ultralytics), OpenCV
Mail Service	SMTP Configuration for email alerts
Web Server	Django(Localhost)

2.4 TECHNOLOGY/FRAMEWORK

2.4.1 FRONT-END TOOLS: HTML, CSS, and JAVASCRIPT

- **HTML (HyperText Markup Language):**

1. **Purpose:** HTML is the standard language used to create the structure and layout of web pages. It provides the basic framework for displaying text, images, forms, and other elements on the front-end of the Weapon Detection platform.
2. **Usage in Weapon Detection Project:** HTML is used to build the structure of web pages, including the homepage, user dashboard, admin panel, weapon detection result

display. It ensures proper content rendering in browsers, facilitating seamless user interaction.

❖ **CSS (Cascading Style Sheets):**

1. **Purpose:** CSS is responsible for styling and enhancing the visual presentation of HTML content. It controls the layout, design, colour schemes, typography, and overall appearance of web pages.
2. **Usage in Weapon Detection Project:** CSS ensures the responsive and aesthetic design of the system. It makes the interface visually appealing, user-friendly, and adaptable across various devices like desktops and mobile phones. CSS is used to style elements such as buttons, result cards, navigation menus, and alert messages.

❖ **JAVASCRIPT:**

1. **Purpose:** JavaScript is a scripting language that enables dynamic and interactive behaviour on web pages, such as real-time data display, form validation, and event handling.
2. **Usage in Weapon Detection Project:** JavaScript is used to enhance interactivity within the system. It handles webcam video stream control for real-time weapon detection, dynamically displays detection results, triggers alarm sounds, and manages live updates of user inputs without reloading pages.

2.4.2 BACK-END TOOLS: PYTHON, DJANGO, and SQLITE3

❖ **PYTHON:**

1. **Purpose:** Python is a versatile, high-level programming language widely used for developing web applications, backend services, and integrating machine learning models.
2. **Usage in Weapon Detection Project:** Python is used to implement the back-end logic of the system, including processing image frames, integrating the YOLO or VGG16 deep learning models for weapon detection, handling data operations, and managing server-side functionalities like email notifications.

- **DJANGO (Backend Framework):**

1. **Purpose:** Django is a powerful, high-level Python web framework that promotes rapid development, clean design, and secure applications.
2. **Usage in Weapon Detection Project:** Django handles all back-end web operations. It processes requests from the front-end, connects with the machine learning model for predictions, interacts with the database, and returns detection results. It also manages user authentication, admin functionalities, recent incident tracking, and data storage.

- **SQLITE3 (Backend Database):**

1. **Purpose:** SQLite3 is a lightweight, serverless, self-contained SQL database engine commonly used for local and embedded applications.
2. **Usage in Weapon Detection Project:** SQLite3 is used to store and manage system data, including registered user details, recent weapon incident logs, and contact information. It enables efficient data handling during both development and deployment phases.

2.4.3 WEB SERVER: DJANGO DEVELOPMENT SERVER

- **Django Development Server:**

1. **Purpose:** Django's built-in development server is a lightweight web server provided by Django for testing and developing web applications locally.
2. **Usage in Weapon Detection Project:** The Django development server is used to deploy and run the Weapon Detection system during development and testing phases. It handles HTTP requests from users, serves web pages, processes detection requests, and communicates between the front-end and back-end logic. In a production environment, this can be replaced with more scalable servers like Gunicorn or Apache with mod_wsgi.

2.5 FUNCTIONAL REQUIREMENTS

2.5.1 User Registration and Login

- Users should be able to register and log in securely to access the platform.

- No admin approval is required for user registration, users can register independently.

2.5.2 Admin Authentication and Dashboard

- Admin should be able to log in with secure credentials.
- Admin dashboard should allow managing users (view and delete), access profile.

2.5.3 Real-Time Weapon Detection from Webcam

- The system should allow users to open a live webcam stream.
- The integrated YOLOv8 Nano model should process the live video feed frame-by-frame.
- If a weapon (gun or knife) is detected in the video, the system should display detection results with accuracy.

2.5.4 Play Alarm on Detection

- An audible alarm should play immediately if weapon is detected in the video feed.

2.5.5 Email Alert System for Weapon Detection

- The system should automatically send an alert email to the user upon weapon (gun or knife) detection also trigger an alarm sound.
- The email should include mail with message ‘Weapon Detected’ and a real-time feed or snapshot from the camera.

2.5.6 Edit Profile and Change Password

- Users and admin should have options to edit their profile information.
- Password change functionality should be provided.

2.6 NON-FUNCTIONAL REQUIREMENTS

2.6.1 Performance Requirements

- Weapon detection results should be displayed within 1–2 seconds of video frame processing.

- The system should be capable of handling real-time webcam video feeds without noticeable lag.
- Email alerts should be sent within 5 seconds after detection confirmation.

2.6.2 Reliability and Availability

- The system must be reliable with minimal downtime.
- Real-time detection should be active whenever a user initiates webcam detection.
- The database should maintain consistency and accuracy of stored records.

2.6.3 Usability Requirements

- The user interface must be simple, clean, and easy to navigate for both technical and non-technical users.
- Error messages and status updates should be clear and informative.
- The system should support responsive design for desktop and mobile devices.

2.6.4 Security Requirements

- User and admin authentication must be enforced through username and password.
- Passwords should be securely stored (hashed) in the database.
- Only authenticated users and admins should access their respective functionalities.
- The system should prevent unauthorized access to critical operations like weapon station management and incident logs.

2.6.5 Scalability

- The application should be designed in a modular fashion to easily scale for additional cameras, users, or weapon detection models in the future.
- The backend database should support expansion without performance degradation.

2.6.6 Portability

- The system should be deployable on both Windows and Linux operating systems.
- It should run on any standard machine with required dependencies like Python, Django, and OpenCV.

2.6.7 Maintainability

- The code base should be modular and well-documented to facilitate easy updates and debugging.
- Database management should be simple with options to back up and restore data.

2.7 PRODUCT FUNCTIONS

The **Weapon Detection using Deep Learning** system, developed as part of our academic project at Kerala Technological University (KTU), serves several essential functions aimed at enhancing public safety and emergency response efficiency. Utilizing the Scrum framework, the application is designed to address critical challenges in weapon threat detection through a user-friendly interface and intelligent, real-time monitoring features. By leveraging deep learning techniques, particularly the YOLOv8 Nano model, the system enables users to detect weapons from live webcam video streams, receive instant alerts, and access emergency services details swiftly [7]. The platform ensures improved responsiveness, minimizes potential harm, and contributes to saving lives by integrating advanced AI-driven weapon detection into accessible, community-focused safety solutions.

1. **Admin Functions:**

- Edit admin profile and change password.
- View registered users and remove them if needed.
- Create and update user profile.

2. **User Functions:**

- Create and update user profile.
- Access live webcam-based weapon detection feature.
- View detection results with accuracy percentage.
- Receive email alerts when a weapon is detected via live video.
- View nearest security force stations based on state, district, and area.
- Edit profile and change password.

3. ANALYSIS

3.1 FEASIBILITY STUDY

Conducting feasibility studies ensures that the **SAFEEYE** is technically, economically, operationally, legally, and socially viable. This includes:

3.1.1 Technical Feasibility

Assessing if the current machine learning models, like **YOLOv8 Nano** and **VGG16**, can effectively detect weapons (such as guns, knives) under various environmental conditions. Ensuring that real-time processing requirements can be met without significant delays or performance bottlenecks, especially for live webcam feeds or uploaded video streams. Examining the availability and quality of weapon detection datasets to train and fine-tune the models for high accuracy.

3.1.2 Economic Feasibility

Estimating the cost of implementation, including hardware (cameras, servers), software (machine learning model development, web application setup, email alert systems), and ongoing maintenance (model updates, security patches, database management). Highlighting reduced risk of security incidents, increased public safety, and operational cost savings compared to traditional manual surveillance methods.

3.1.3 Operational Feasibility

Determining if users (security personnel) can effectively use the system's web interface and respond to weapon detection alerts promptly. Assessing how easily the system can integrate with existing surveillance infrastructure and if it can scale to handle multiple video streams across different locations without impacting performance.

3.1.4 Legal and Ethical Feasibility

Evaluating potential privacy concerns associated with real-time monitoring and recording of public or private spaces, and ensuring full compliance with data protection laws (such as GDPR or local privacy regulations). Addressing ethical considerations regarding the responsible use of AI in surveillance to prevent misuse or wrongful profiling.

3.1.5 Social Feasibility

Ensuring that the system strikes a balance between enhancing security and respecting individual privacy. Gauging the public’s acceptance of AI-driven weapon detection solutions, and ensuring transparency about the system’s purpose to prevent fear of misuse or excessive surveillance.

3.2 PROBLEM STORY

In today's world, ensuring public safety has become an ever-growing concern, especially in crowded public spaces like airports, shopping malls, schools, and public events. Traditional security measures such as human surveillance, manual bag checks, and metal detectors are often slow, intrusive, and inefficient. These methods rely heavily on human judgment and are prone to errors, leading to missed threats or unnecessary delays in response times [6].

Imagine a crowded airport or a busy shopping mall where thousands of people pass through every hour. Security personnel can monitor only a fraction of the surveillance footage at any given time, increasing the chances of missing potential threats. A suspicious individual carrying a concealed weapon could easily pass unnoticed until it’s too late. In such high-pressure environments, the risk of a security threat escalating rapidly becomes a serious concern.

This is where **Weapon Detection and Alerting using ML and Python** can make a real difference. By leveraging advanced machine learning (ML) algorithms and computer vision techniques, the system automatically detects weapons in real-time through surveillance cameras. Using powerful models like YOLOv8 Nano and VGG16, it can accurately identify guns, knives, or other dangerous objects, and trigger instant alerts to security personnel or law enforcement. This automated, intelligent approach enables quicker, more reliable detection, helping prevent harm and ensuring a faster, more efficient emergency response.

3.3 USE CASE STORY

3.1.1. Primary Actors

Primary actors in the **Real-Time Weapon Detection and Alerting System Using ML and Python**:

1. **Admin**
   Role: The Admin manages the overall backend system — controls user accounts, emergency contact information, incident handling, and maintains system security and performance.

Responsibilities:

- ❖ Register and perform login.
- ❖ Manage users (view, delete).
- ❖ Manage system data inputs and datasets.
- ❖ Handle personal profile management.

2. **User**
   Role:The User interacts with the system to detect weapons, view results, get alerts, and access emergency services.

Responsibilities:

- ❖ Perform login and profile management.
- ❖ Input data for live detection (webcam).
- ❖ Receive real-time alert notifications.
- ❖ View emergency contact details.
- ❖ View weapon detection results.

3.3.2 Use Cases

Use Case 1: **Login** (Admin/User)

Goal: Secure authentication for accessing system features.
Scenario:

- ❖ Admin/User logs in with valid credentials.
- ❖ System authenticates and redirects to respective dashboard.

Outcome: Authorized access to system functionalities.

Use Case 2: **Input Data** (User)

Goal: Allow users to input real-time data for weapon detection.
Scenario:

- ❖ User initiates webcam capture for real-time monitoring.
- ❖ Data (frames) is sent to the detection model.

Outcome: System starts analyzing input for weapon presence.

Use Case 3: **Profile Management** (Admin/User)

Goal: Manage personal information securely.
Scenario:

- ❖ Admin/User updates name, email, password, etc.
- ❖ System saves updated profile details.

Outcome: Updated, secure personal account information.

Use Case 4: **User Management** (Admin)

Goal: Manage user accounts and access rights.
Scenario:

- ❖ Admin views or deletes user accounts.
- ❖ Admin monitors user activity logs if needed.

Outcome: Controlled and secure user database.

Use Case 5: **View Emergency Contact Details** (Admin/User)

Goal: Provide access to emergency service contacts.

Scenario:

- ❖ Admin manages weapon force station information.
- ❖ User views nearby emergency contact details when needed.

Outcome: Faster access to help in case of emergencies.

Use Case 6: **Receive Alert Notification** (Admin/User)

Goal: Ensure immediate response during a weapon detection event.

Scenario:

- ❖ Upon weapon detection, system sends an instant email or dashboard notification to Admin/User.
- ❖ Alert contains incident time, location, and type

Outcome: Timely awareness and rapid response to potential threats.

Use Case 7: **View Result** (User)

Goal: Allow users to see detection outcomes.

Scenario:

- ❖ System displays weapon detection results live on the dashboard.
- ❖ Results include prediction type (weapon/non-weapon) and confidence score.

Outcome: Users are informed in real-time about any weapon detection.

3.4 USE CASE DIAGRAM

Creating UML use case diagrams is a vital method for identifying and outlining the requirements of a system or software program under development. It focuses on the expected behavior of the system (what) rather than detailing the method of implementation (how). Use cases can be represented both textually and visually through use case diagrams. This modeling technique is especially effective for designing systems from an end-user perspective, ensuring that all externally visible behaviors are clearly specified in user-friendly terms. By facilitating communication in this way, use case diagrams help bridge the gap between technical requirements and user expectations.

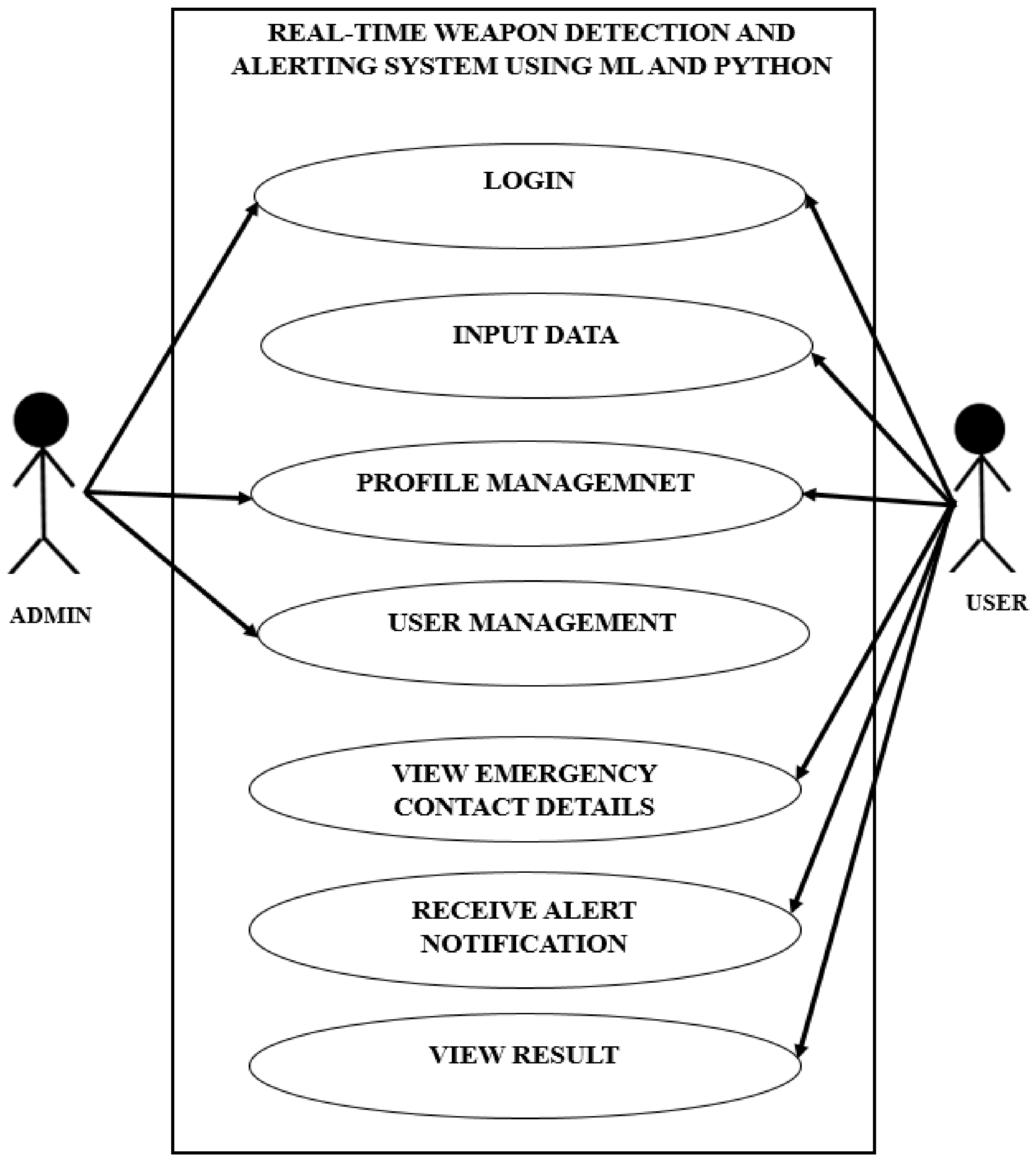

REAL-TIME WEAPON DETECTION AND ALERTING SYSTEM USING ML AND PYTHON
LOGIN
INPUT DATA
PROFILE MANAGEMNET
USER MANAGEMENT
VIEW EMERGENCY CONTACT DETAILS
RECEIVE ALERT NOTIFICATION
VIEW RESULT
ADMIN
USER

Fig 3.1: Use Case Diagram

3.5 USER STORIES

User stories are short, simple descriptions of a feature or functionality from the perspective of the end user. They help development teams understand what the user needs and why it is important. Each user story typically follows the format "As a [user], I want [goal] so that [reason]." In project planning, user stories are essential for breaking down complex features into manageable tasks, prioritizing work, and ensuring that the final product delivers real value to users.

Table 3.1: User Story

USER STORY ID	PRIORITY	AS A	I WANT TO	SO THAT I CAN	FINAL STORY
1.1	High	Team	Plan the project and gather requirements	Ensure the project is well-defined and on track	As a Team, I want to plan the project and gather requirements so that the project is well-defined and on track.
1.2	High	Developer	Collect data	Ensure the model is trained with relevant data	As a Developer, I want to collect data so that the model is trained with relevant data.
1.3	Medium	Developer	Input data into the system	Begin the weapon detection process	As a Developer, I want to input data into the system so that I can begin the weapon detection process.
1.4	High	Developer	Select a model	Choose the most suitable model for detection	As a Developer, I want to select a model so that I can choose the most suitable model for detection.
1.5	High	Developer	Develop a baseline model	Establish a baseline for comparison	As a Developer, I want to develop a baseline model so that I can establish a baseline for comparison.

USER STORY ID	PRIORITY	AS A	I WANT TO	SO THAT I CAN	FINAL STORY
1.6	High	Developer	Implement an advanced model	Improve the accuracy and detection capabilities	As a Developer, I want to implement an advanced model so that I can improve the accuracy and detection capabilities.
1.7	High	Developer	Train and evaluate the model	Assess the model’s readiness for deployment	As a Developer, I want to train and evaluate the model so that I can assess its readiness for deployment.
1.8	High	Developer	Set up the detection system	Enable real-time weapon detection	As a Developer, I want to set up the detection system so that I can enable real-time weapon detection.
1.9	Medium	Developer	Test and evaluate the system	Ensure the system functions as expected	As a Developer, I want to test and evaluate the system so that I can ensure it functions as expected.
1.10	High	Developer	Deploy the system	Make the system available for use	As a Developer, I want to deploy the system so that I can make it available for use.
2.1	High	User	Log in to the system	Access the system and its features	As a User, I want to log in to the system so that I can access the system and its features.
2.2	High	User	Input data into the system	Start the weapon detection process	As a User, I want to input data into the system so that I can start the weapon detection process.
2.3	High	User	View the results of the detection	Take appropriate actions based on the results	As a User, I want to view the results of the detection so that I can take appropriate actions based on the results.

3.6 TECHNICAL DOCUMENTATION

3.6.1 Project Overview

The SAFEEYE (**Weapon Detection and Alerting using Machine Learning and Python)** project aims to develop an intelligent security system that automatically detects weapon in realtime through video surveillance, leveraging machine learning algorithms. Using computer vision techniques and deep learning models, such as Convolutional Neural Networks (CNNs), the system analyzes live video to identify weapons, such as fire arms and knives. Upon detection, the system generates instant alerts, notifying security personnel via various channels (alarm sound, email, etc.), enabling rapid response to potential threats. Built with Python libraries like OpenCV for video processing and TensorFlow or PyTorch for model development, the solution provides an efficient and scalable method for enhancing public safety in environments like airports, schools, or government buildings. The project focuses on high accuracy, low false positives, ease of use, and integration with existing security infrastructures.

3.6.2 System Architecture

The system follows a modular design where video feeds from surveillance cameras are processed by a deep learning model [3], which then triggers alerts when weapon is detected. **The architecture consists of**:

- ❖ **Input Layer:** Capturing images and videos from cameras.
- ❖ **Processing Layer:** CNN models analysing visual data.
- ❖ **Output Layer:** Real-time alerts and notifications sent to response teams.

Guidelines for Implementation:

- ❖ Utilize pre-trained CNNs for faster deployment.
- ❖ Implement real-time data processing using GPU acceleration.
- ❖ Optimize detection accuracy through extensive model training.

System Components:

1. Data Collection

Collect image and video feeds from CCTV cameras or other source.

2. Object Detection

Use a deep learning based object detection algorithm (e.g., YOLO, VGG16)

Detect objects in video frames and extract bounding box coordinates

3. Weapon (Gun and Knife) Classification

Use a machine learning based classification algorithm (YOLO, VGG16)

Classify detected objects as weapons or non-weapons

4. Alerting

Send alerts to authorities or security personnel when a weapon is detected

Use communication protocols such as email or alarm sound.

5. Data Storage

Store login details (userid, password) in a database or file system

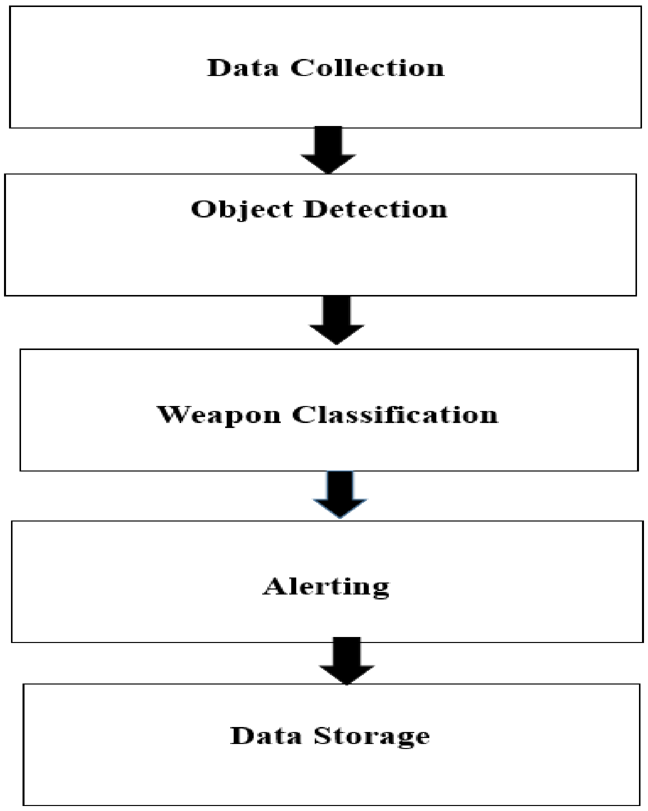
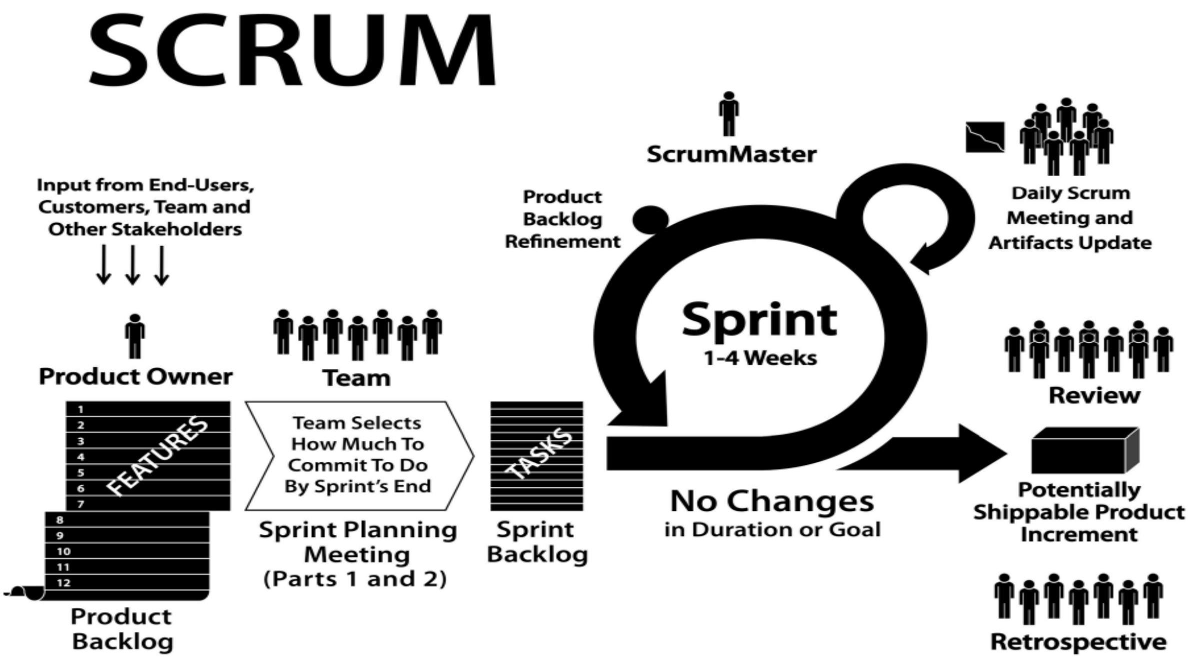


Fig 3.2: System Architecture

CHAPTER 4. PROJECT PLANNING

Project planning in the Software Development Life Cycle (SDLC) using Scrum methodology focuses on iterative progress through well-defined sprints. During this phase, the project scope, goals, and timelines are defined in collaboration with stakeholders. Scrum emphasizes breaking down the project into smaller, manageable tasks in a product backlog, with priorities set for each sprint. Resources are allocated, and risks are identified, ensuring that the project progresses smoothly within the set time and budget constraints. This approach ensures flexibility, continuous feedback, and a focus on delivering high-value features incrementally.


SCRUM
ScrumMaster
Input from End-Users, Customers, Team and Other Stakeholders
Product Backlog Refinement
Daily Scrum Meeting and Artifacts Update
Sprint
1-4 Weeks
Product Owner
Team
Review
FEATURES
Team Selects How Much To Commit To Do By Sprint's End
TASKS
No Changes
in Duration or Goal
Potentially Shippable Product Increment
Sprint Planning Meeting (Parts 1 and 2)
Sprint Backlog
Product Backlog
Retrospective

[Source: ResearchGate]

Fig 4.1: Scrum Methodology

4.1 PRODUCT BACKLOG

A product backlog is a prioritized list of tasks, features, enhancements, and fixes that are needed for the development of a product. It serves as a dynamic and evolving document, created and maintained by the product owner in collaboration with the development team and stakeholders. Each item in the backlog is known as a "backlog item" and can range from user stories to technical tasks. The backlog is continuously refined and updated, with items reordered based on priority, business value, and feedback from previous sprints. This ensures that the most important and valuable features are worked on first, enabling the team to deliver incremental value throughout the project's lifecycle. The product backlog is a central element in Scrum, providing transparency and guiding the development process.

Table 4.1: Product Backlog

FID	**Feature**	**Priority**	**Story Points**	**Rank**	**Estimation (hrs)**	**User Story**
1	Project Planning and Gather Requirements	High	13	1	30	As a Team, we want to plan the project and gather requirements so that the project is well-defined and on track.
2	Data Collection	High	17	2	45	As a Developer, I want to collect data so that the model is trained with relevant data.
3	Input Data	Medium	8	4	15	As a Developer, I want to input data into the system so that I can begin the weapon detection process.
4	Model Selection	High	10	5	25	As a Developer, I want to select a model so that I can choose the most suitable model for detection.
5	Baseline Model Development	Medium	13	6	30	As a Developer, I want to develop a baseline model so that I can establish a baseline for comparison.
6	Advanced Model Implementation	High	20	7	45	As a Developer, I want to implement an advanced model so that I can improve the accuracy and detection capabilities.

FID	**Feature**	**Priority**	**Story Points**	**Rank**	**Estimation (hrs)**	**User Story**
7	Model Training and Evaluation	High	19	8	30	As a Developer, I want to train and evaluate the model so that I can assess its readiness for deployment.
8	Detection Setup	Medium	10	9	25	As a Developer, I want to set up the detection system so that I can enable real-time weapon detection.
9	Testing and Evaluation	High	14	10	35	As a Developer, I want to test and evaluate the system so that I can ensure it functions as expected.
10	Deployment	High	15	11	30	As a Developer, I want to deploy the system so that I can make it available for use.
11	Login	High	5	12	10	As a User, I want to log in to the system so that I can access the system and its features.
12	Input Data (UI)	High	8	13	15	As a User, I want to input data into the system so that I can start the weapon detection process.
13	View Result	High	8	14	15	As a User, I want to view the results of the detection so that I can take appropriate actions based on the results.
Total			160		350	

4.2 SPRINT PLANNING DOCUMENTS

Sprint planning documentation is a crucial part of Agile project management, where detailed plans are created for each sprint before the work begins. It outlines the sprint goals, selected backlog items, priorities, story points, estimated hours, and clear objectives for the sprint. This documentation ensures that the entire team has a shared understanding of what needs to be accomplished during the sprint and helps manage workload effectively. Proper sprint planning improves team coordination, clarifies expectations, and sets a clear roadmap for achieving project milestones within the given timeline [3].

4.2.1 Sprint Planning

The sprint planning for the project is organized as follows:

Sprint 1 (Week 1 & Week 2): Project Planning

- **Sprint Goal**: Define the project scope, set timelines, assign tasks, and allocate resources.
- **Selected Backlog Item(s)**: Task ID 1
- **Priority**: High
- **Estimated Hours**: 30
- **Objectives**:
    1. Define clear project deliverables.
    2. Create a work breakdown structure.
    3. Assign team members based on expertise.
    4. Develop a high-level project schedule.

Sprint 2 (Week 3 & Week 4): Requirement Analysis

- **Sprint Goal**: Gather functional and non-functional requirements and refine project goals.
- **Selected Backlog Item(s)**: Task ID 2
- **Priority**: High
- **Estimated Hours**: 35
- **Objectives**:

1. Conduct requirement gathering sessions.
2. Create user stories for different system users.
3. Document functional and technical requirements.

Sprint 3 (Week 5 & Week 6): Data Collection & Preprocessing

- **Sprint Goal**: Collect and preprocess data for model training.
- **Selected Backlog Item(s)**: Task ID 3
- **Priority**: High
- **Estimated Hours**: 45
- **Objectives**:

  1. Collect weapon/weapon-related images.
  2. Clean and normalize datasets.
  3. Handle missing and inconsistent data.

Sprint 4 (Week 7 & Week 8): Model Selection & Baseline Development

- **Sprint Goal**: Select appropriate machine learning models and create baseline models.
- **Selected Backlog Item(s)**: Task ID 4, 5
- **Priority**: Medium
- **Estimated Hours**: 60
- **Objectives**:

  1. Evaluate different CNN models.
  2. Build a simple baseline model for benchmarking.
  3. Test initial model performance.

Sprint 5 (Week 9 & Week 10): Advanced Model Implementation & Training

- **Sprint Goal**: Implement advanced models and fine-tune them for best performance.
- **Selected Backlog Item(s)**: Task ID 6, 7
- **Priority**: High
- **Estimated Hours**: 70

- **Objectives**:

    1. Implement optimized models.
    2. Perform hyperparameter tuning.
    3. Train the models for higher accuracy.

Sprint 6 (Week 11 & Week 12): System Evaluation & UI Development

- **Sprint Goal**: Evaluate the model's performance and develop a user interface for interaction.
- **Selected Backlog Item(s)**: Task ID 8, 9
- **Priority**: Medium
- **Estimated Hours**: 55
- **Objectives**:

    1. Evaluate the detection model.
    2. Develop the system's frontend dashboard.
    3. Conduct usability testing.

Sprint 7 (Week 13 & Week 14): Integration, Deployment & Performance Monitoring

- **Sprint Goal**: Integrate system components, deploy the solution, and monitor performance.
- **Selected Backlog Item(s)**: Task ID 10, 11, 12, 13
- **Priority**: High
- **Estimated Hours**: 55
- **Objectives**:

    1. Integrate machine learning model with the web application.
    2. Deploy the system on a server or local system and see the result.
    3. Monitor system performance and refine based on feedback.

4.2.2 Sprint Planner

A **Sprint Planner** is a Scrum tool used by the development team to define, prioritize, and schedule tasks for the upcoming sprint, ensuring clear goals and efficient workload distribution.

Table 4.2: Sprint Planner

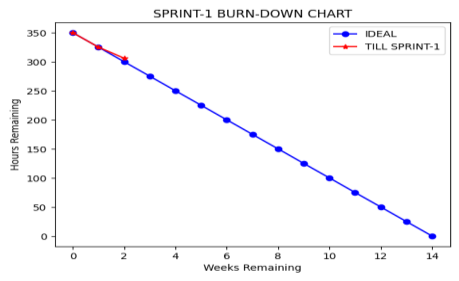
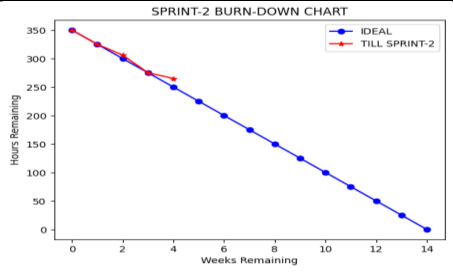
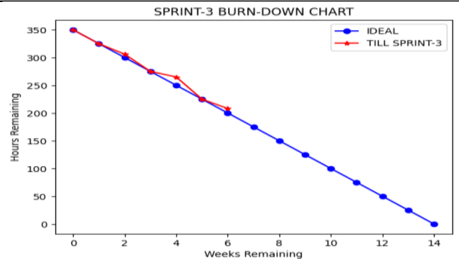
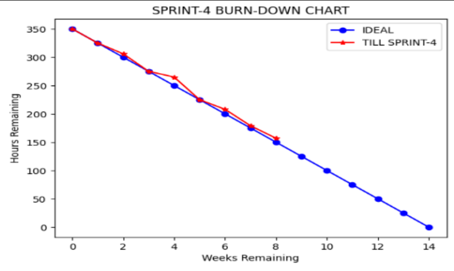
Sprint No	**Goal**	**Task ID(s)**	**Priority**	**Story Points**	**Estimated Hours**	**Objectives**
Sprint 1	Project Planning	1	High	8	30	Define project scope, set timelines, assign tasks, and allocate resources.
Sprint 2	Requirement Analysis	2	High	13	35	Gather functional and non-functional requirements, create user stories, and refine project goals.
Sprint 3	Data Collection & Pre-processing	3	High	23	45	Collect and clean data, handle missing values, and preprocess data for model training.
Sprint 4	Model Selection & Baseline Development	4, 5	Medium	19	60	Select appropriate machine learning models, create baseline models, and evaluate initial performance.
Sprint 5	Advanced Model Implementation & Training	6,7	High	25	70	Implement advanced models, train them, and fine-tune hyper parameters for optimal results.

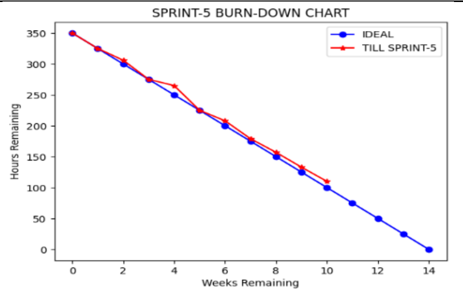
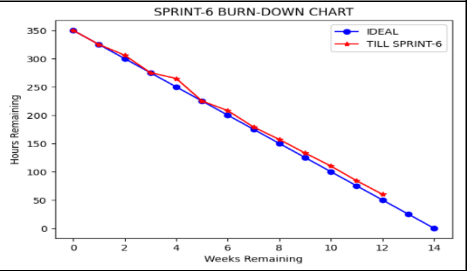
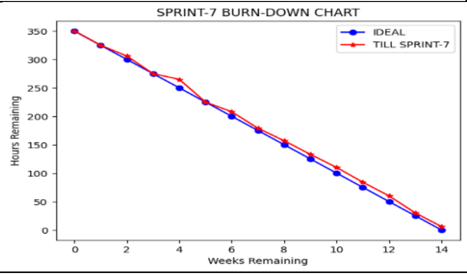
Sprint No	Goal	Task ID(s)	Priority	Story Points	Estimated Hours	Objectives
Sprint 6	System Evaluation & UI Development	8,9	Medium	17	55	Evaluate model performance, develop UI to interact with the system, and test usability.
Sprint 7	Integration, Deployment & Performance Monitoring	10,11,12, 13	High	18	55	Integrate system components, deploy the solution, and monitor performance for improvements.
Total				**104**	**350**	

4.3 SPRINT BACKLOG

Sprint backlog is a list of selected tasks and user stories that a team commits to completing during a specific sprint. It is created during sprint planning and includes all the work needed to achieve the sprint goal.

Table 4.3: Sprint Backlog

FI. D	**Sprint No.**	**User Story**	**Start Date - End Date**	**Task-In Description**	**Status**	**Burn-Down Chart**
1	1	1.1	30/12/24 to 12/01/25	**1.**Develop a project plan	On-going	
1, 2	2	1.1	13/01/25to 26/01/24	**1.**Comprehensive project plan	Complete	
				2.Requirement Gathering and documenting	On-going	
2, 3	3	1.2	27/01/25to 09/02/25	**1.**Requirement Gathering and Documenting	Complete	
				2.Collect, Pre-process and prepare data for analysis	On-going	
4,5	4	1.3, 1.4,1.5	10/02/25 to 23/02/25	**1.**Model Research and Selection	Complete	
				2. Baseline model selection. training, evaluation	On-going	

FI. D	Sprint No.	User Story	Start Date - End Date	Task-In Description	Status	Burn-Down Chart
10, 11	5	1.10, 2.1	24/02/25 to 09/03/25	**1.**Django setup- Project setup, UI design deployment	On-going	
				2.UI Development - ML model integration, testing	On-going	
12, 13	6	2.2, 2.3	10/03/25to 23/03/25	**1.**Deployment and dataset creation	On-going	
				2.Finalize UI	On-going	
6, 7, 8, 9	7	1.6, 1.7, 1.8, 1.9	24/03/25 to 06/04/25	**1.**Advanced model development	Complete	
				2.Model ensemble and evaluation	Complete	

4.4 PROJECT PLANS

The project plan outlines the structured approach for executing the project through clearly defined phases, milestones, and deliverables. It sets the overall timeline, distributes tasks across sprint cycles, assigns responsibilities, and allocates resources effectively to ensure smooth progress. The plan helps track development activities, manage risks, and maintain project scope and quality. Regular reviews and adjustments are incorporated to adapt to changing requirements, ensuring that the project remains on schedule and meets its objectives efficiently.

Sprint 1: Project Planning

This sprint focuses on defining the project scope, setting clear timelines, assigning tasks to team members, and allocating necessary resources. (Task ID 1)

Priority: High, **Estimated Hours:** 30

Sprint 2: Requirement Analysis

During this sprint, the team will gather both functional and non-functional requirements, create user stories, and refine the overall project goals to ensure clarity and alignment. (Task ID 2)

Priority: High, **Estimated Hours:** 35

Sprint 3: Data Collection & Pre-processing

This sprint includes the collection of weapon(gun and knife) related images, cleaning the data, handling missing values, and preparing the dataset properly for model training to ensure quality learning. (Task ID 3)

Priority: High, **Estimated Hours:** 45

Sprint 4: Model Selection & Baseline Development

The goal of this sprint is to select the most suitable machine learning models, build baseline models, and evaluate their initial performance to set a reference for future improvements. (Task IDs 4, 5)

Priority: Medium, **Estimated Hours:** 60

Sprint 5: Advanced Model Implementation & Training

This sprint covers the implementation of advanced models, training them thoroughly, and performing hyper-parameter tuning to optimize the model's accuracy and reliability. (Task IDs 6, 7)

Priority: High, **Estimated Hours:** 70

Sprint 6: System Evaluation & UI Development

Here, the focus is on evaluating the trained model’s performance, developing a user-friendly interface for smooth interaction, and conducting usability testing to enhance the system's effectiveness. (Task IDs 8, 9)

Priority: Medium, **Estimated Hours:** 55

Sprint 7: Integration, Deployment & Performance Monitoring

The final sprint involves integrating all system components, deploying the complete weapon detection system, and continuously monitoring its performance to ensure stability and enable future improvements. (Task IDs 10, 11, 12, 13)

Priority: High, **Estimated Hours:** 55

Resource Allocation:

All sprint tasks are handled by a single team member who manages multiple specialization areas. Backend development tasks such as authentication, data processing, and system integration, frontend tasks including user interface design and usability testing, and machine learning tasks like model development, training, and evaluation are all carried out by the same individual. Resource planning is reviewed and adjusted during each sprint planning meeting to maintain efficiency and ensure on-time delivery.

4.5 IDEAL BURN-DOWN CHART

An ideal burndown chart for a project with 7 sprints over 14 weeks and a total of 350 hours would show a steady, diagonal line from the top-left corner (start with 350 hours of work) to the bottom-right corner (end with 0 hours) [4]. Each sprint would ideally reduce the total remaining work by an equal amount, with approximately 50 hours of work completed per sprint, leading to consistent progress and a clear visual representation of work being "burned down" over time.

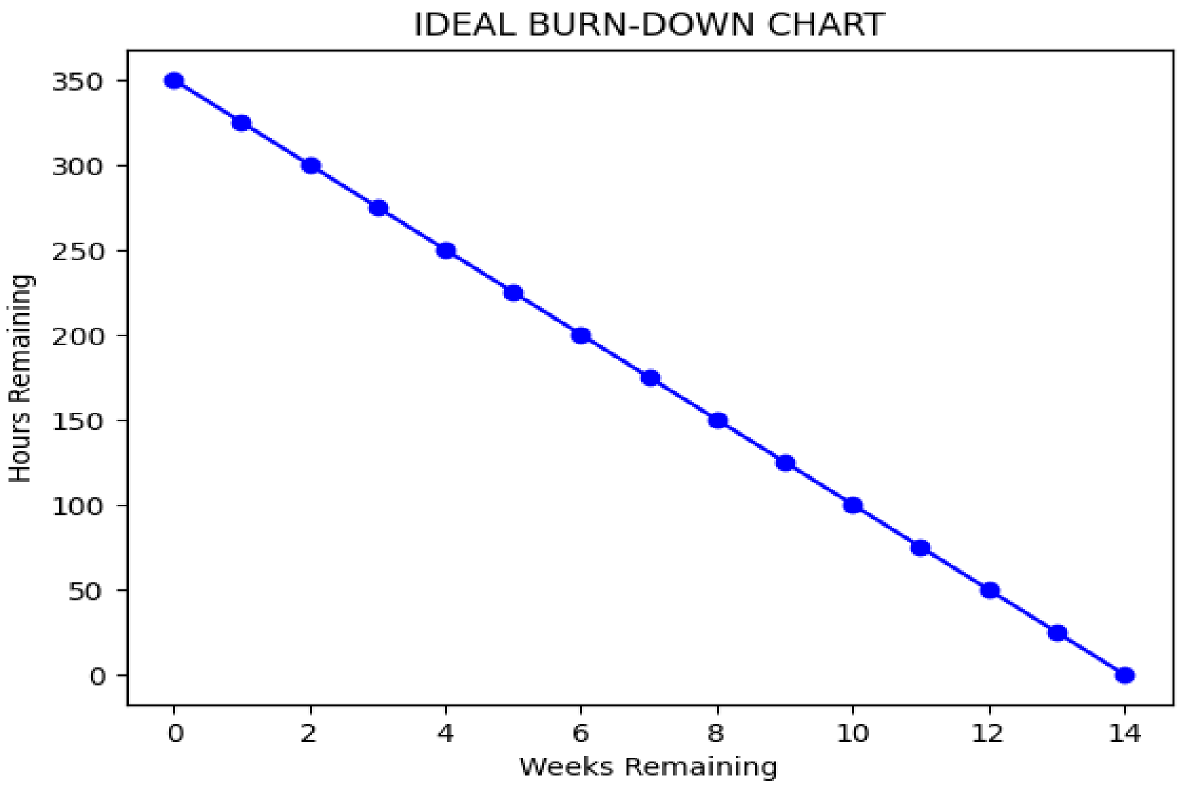


Fig 4.2: Ideal Burn-Down Chart

CHAPTER 5. DESIGNS

Design is a crucial stage in any project, serving as the blueprint that guides the implementation of ideas into functional outcomes. It involves creating detailed plans and models that align with the project's objectives and requirements. In software or system design, this could include crafting user interface designs that enhance user interaction, developing data models for efficient information handling, or illustrating workflows with flow and sequence diagrams. These elements ensure that every component works cohesively and meets user expectations.

Moreover, design emphasizes both functionality and aesthetics. It is about striking the right balance between solving problems efficiently and offering a seamless, engaging experience. For example, flowcharts and activity diagrams help visualize processes, ensuring clarity and minimizing potential issues during implementation. By meticulously designing each aspect, teams can create robust, user-friendly, and innovative solutions that fulfill the intended purpose while leaving room for scalability and improvement.

5.1 UID DESIGN DOCUMENTATION

This system uses image recognition algorithms, object detection models to classify objects and identify weapon such as gun and knife. The UI design is focused on delivering intuitive, user-friendly experience for security personnel.

User Interface Overview

- ❖ **Login:** Secured access for authorized users.
- ❖ **Data Input Page:** Interface enable authorized users to upload static images or access live camera feeds for real-time processing and weapon detection.
- ❖ **Result:** To view the result of weapon detection.
- ❖ **Alert Dashboard:** Display detected weapons and generate email notification or alarm sound.

5.2 DATA DESIGN DOCUMENTATION

Data Design Documentation is a concise blueprint describing how data is structured, managed, and flows within a system, ensuring efficiency, scalability, and security. It covers data models defining entities and attributes, relationships such as foreign key connections, input formats like images or live feeds, processed outputs in JSON, storage designs for file paths or schemas, security measures including encryption and role-based access controls, and workflows integrating data movement across components from input to output. This foundation helps maintain clarity, consistency, and best practices, and the implementation utilizes DB Browser for SQLite for efficient database management.

Database Tables:

1. Weap_Registration Table:

Store basic details of registered users for access control and user identity.

Primary Key: id

Foreign Key: user_id

Table 5.1: Registration Database

Field name	**Data type**	**Description**	**Constraints**
id	integer	Auth User id	Not Null, Unique, Primary Key
user_role	Varchar(20)	User system role	Null
user_id	integer	Unique user Identifier	Null Not, Unique, Foreign Key
address	Varchar(700)	Address of user/admin	Not Null
email	Varchar(254)	Email of the user/admin	Null
phoneno	Varchar(10)	Phone number of user/admin	Not Null
password	Varchar(8)	Password of Auth User	Not Null

2. Auth_User

Manage authorization, authentication, and roles for system users.

Primary Key: id

Foreign Key: Nil

Table 5.2: Authorized User Database

Field Name	Data Type	Description	Constraints
id	Integer	Unique ID of registration table	Not Null, Unique, PrimaryKey
password	Varchar(8)	Password of admin or user	Not Null
last_login	Varchar(200)	Last login time of admin or user	Null
is_superuser	bool	Indicates if the user is a superuser	Not Null
username	Varchar(50)	Username of admin or user	Not Null, Unique
Last_name	Varchar(50)	Last name of admin or user	Not Null
email	Varchar(254)	Email ID of admin or user	Not Null
first_name	Varchar(50)	First name of admin or user	Not Null
date_joined	Datetime	The date and time the user joined	Not Null
is_staff	bool	Indicates if the user is staff	Not Null
is_active	bool	Indicates if the user is active	Not Null

5.3 ACTIVITY DIAGRAM

An activity diagram is a visual representation used in software and process modeling to illustrate the flow of activities or tasks within a system. It showcases the sequence of actions, decisions, and parallel processes, helping to understand how different components interact and ensuring clarity in complex workflows. With symbols such as ovals for activities, diamonds for decisions, and arrows for transitions, it provides a clear, logical overview of dynamic behaviours. Activity diagrams are instrumental in identifying potential bottlenecks and optimizing processes during system design and analysis [12].

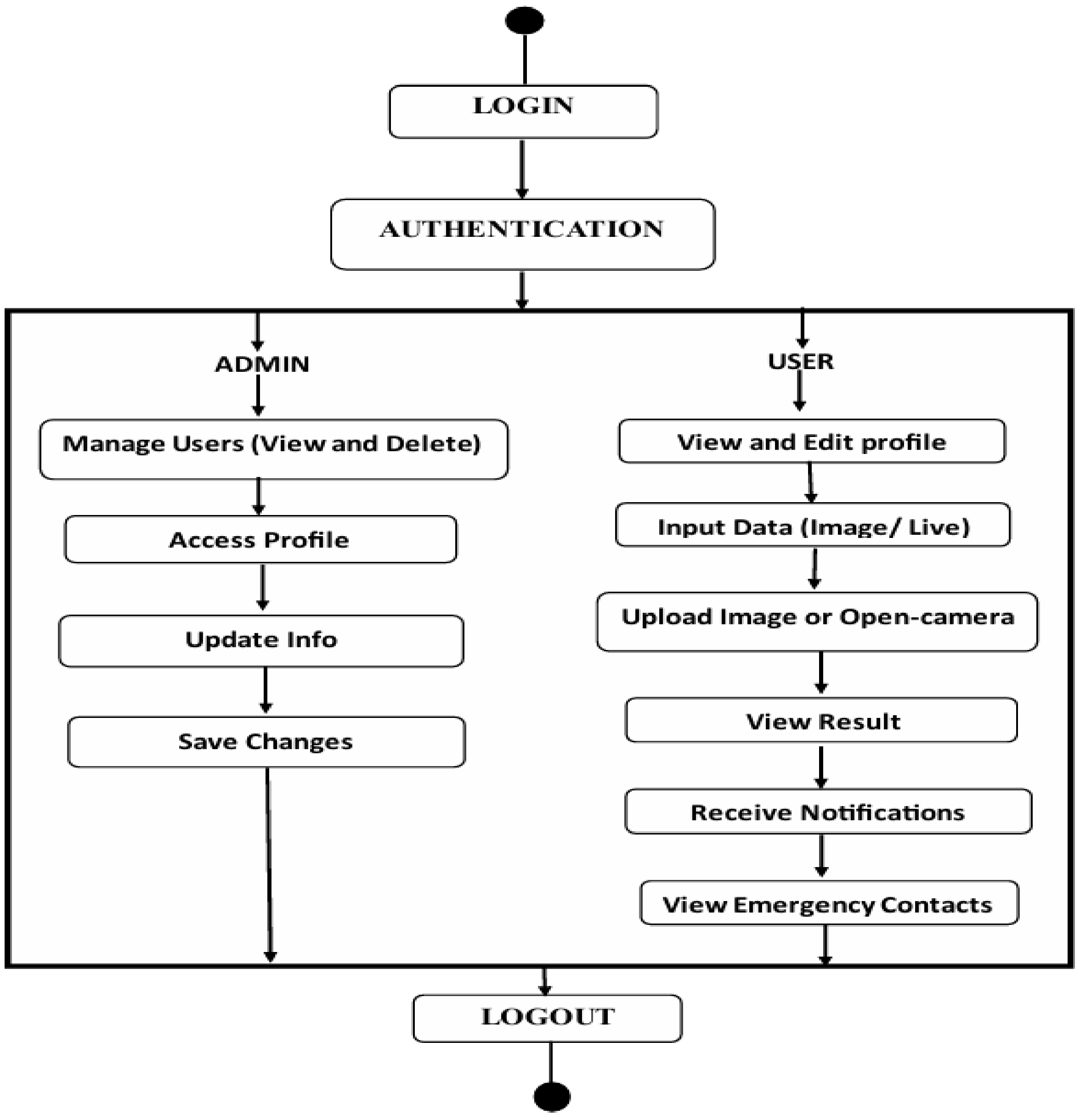


Fig 5.1: Activity Diagram

CHAPTER 6. CODING/ DEVELOPMENT

6.1 INCREMENT DEFINITION

Increment 1: Project Planning and Requirement Gathering

- Task: Define project objectives and gather requirements.
- Description: Establish project scope, gather stakeholder requirements, and set clear objectives for the Weapon Detection project.
- Outcome: Clear project plan and requirements document for the entire system.

Increment 2: Finalizing Requirements and Planning

- Task: Complete project planning and continue requirement documentation.
- Description: Refine and finalize the comprehensive project plan while progressing with the collection and documentation of system requirements.
- Outcome: A fully detailed project plan and partially completed requirement specifications.

Increment 3: Data Preparation and Requirement Completion

- Task: Finalize requirement documentation and begin data collection.
- Description: Complete requirement gathering and initiate data pre-processing for training the weapon detection model.
- Outcome: Completed system requirements and partially prepared dataset for model training.

Increment 4: Model Research and Baseline Model Training

- Task: Research suitable models and begin baseline model training.
- Description: Conduct in-depth research on available deep learning models and initiate training of the selected baseline model.
- Outcome: Selected baseline model with initial training and evaluation metrics.

Increment 5: Web Application Development and Integration

- ❖ Task: Setup Django framework and integrate UI with the machine learning model.
- ❖ Description: Begin setting up the Django environment, design user interfaces, and integrate the weapon detection model with the backend.
- ❖ Outcome: Web application structure established with ongoing ML model integration and UI testing.

Increment 6: Deployment Setup and UI Finalization

- ❖ Task: Prepare for deployment and finalize the user interface.
- ❖ Description: Configure deployment environment and polish front-end components for a seamless user experience.
- ❖ Outcome: Deployment environment initiated and final UI elements near completion.

Increment 7: Advanced Modeling and Evaluation

- ❖ Task: Develop advanced models and perform ensemble evaluations.
- ❖ Description: Build advanced model variants and perform ensemble techniques to optimize weapon detection accuracy.
- ❖ Outcome: Improved detection performance through advanced modeling and final model evaluation.

6.2 CODING STANDARDS USED IN OUR PROJECT

- ❖ Consistency: All code follows a consistent style across the project. This includes uniform naming conventions, consistent formatting, and indentation practices to ensure readability.
- ❖ Indentation: Indentation is done using 4 spaces per level for Python code. For HTML, CSS, and JavaScript, a consistent 2 spaces per indentation level is followed to maintain uniformity across all files.
- ❖ Comments: Meaningful comments are used to explain complex logic, functions, or specific code blocks. Comments should clarify the purpose of code, but not state the obvious. Excessive comments are avoided.

- Descriptive Naming: Variable and function names should be meaningful and follow naming conventions. For example, instead of check Weapon, used detect_Weapon_in_image.
- Readability: Code should be formatted in a way that is easy to read, with no excessively long lines of code. Code blocks should be well-separated and grouped logically.
- Modularization: Functions are broken into smaller, reusable units wherever possible to reduce redundancy and improve clarity.

6.2.1 HTML, CSS, and JavaScript Standards (Front-End):

1. Hyper Text Markup Language (Structure):

- **Structure and Layout**: HTML defines the structure of your web pages—login, upload section, result display, and profile editing are all laid out using semantic HTML.
- **Role-Based UI**: Helps differentiate between Admin and User functionalities (e.g., managing users vs. uploading data).
- **Data Input and Forms**: Enables image/video upload and user authentication through structured input fields.

2. Cascading Style Sheets (Style):

- **User-Friendly Interface**: CSS styles make your interface clean and professional, improving user experience during critical tasks like viewing alerts or uploading images
- **Responsiveness**: Adapts the layout across devices (phones, tablets, PCs), which is crucial for real-time access in surveillance environments
- **Consistency**: Ensures visual consistency across different pages (login, profile, emergency contacts), improving usability and reducing confusion

3. JavaScript (Behaviour):

- **Real-Time Functionality**: Enables features like accessing live camera feed, showing detection results dynamically, and handling real-time notifications.

- **Validation and Logic**: Adds client-side validation for login forms, profile edits, and image uploads—preventing errors before they hit the server.
- **Interactive UI**: Improves user engagement by dynamically loading sections (e.g., emergency contacts or result views) without full page reloads.

4. Python/Django (Back-End):

- **Handles Core Business Logic**: Django acts as the engine that processes image inputs, invokes the trained weapon detection model (like YOLOv8 or your CNN), and makes decisions based on the result.
- **Robust Authentication & User Roles**: Django's built-in user management system makes it easy to implement secure authentication and distinguish between Admin and User workflows (e.g., profile management, alerts).
- **Fast Integration with AI/ML**: Python's compatibility with machine learning libraries (like TensorFlow/Keras or OpenCV) allows you to seamlessly integrate your model into the Django view logic for real-time inference.

5. Database Interaction (SQLite3)

- **Lightweight & Easy to Use**: SQLite3 is file-based, making it ideal for development and small to medium-scale deployments like SafeEye, where data like users, alerts, and incidents need to be stored locally.
- **Stores Detection Logs & Profiles**: It holds crucial data such as user profiles, emergency contact lists, and incident detection logs that can be retrieved and displayed instantly on the UI.
- **Secure and Structured Storage**: Through normalization and parameterization, SQLite ensures that data is organized, reduces redundancy, and prevents security risks like SQL injection attacks.

6.2.2 Coding Practices and Tools:

- **Code Formatting Tools:** The project uses Python's PEP 8 guidelines for consistent code formatting Integrated development environments (IDEs) like PyCharm or Visual Studio Code help maintain clean code with proper indentation, readability, and structure

- ❖ **Server Configuration:** The Django framework is configured to handle requests efficiently, with the server running on a local development environment using SQLite3 for data storage. The configuration ensures fast deployment and smooth interactions between the backend and the deep learning model.
- ❖ External Libraries:
  1. **TensorFlow/Keras**: These libraries are used for building and deploying the YOLOv8 Nano deep learning model, which is responsible for detecting Weapon in real-time video input.
  2. **OpenCV**: Utilized for capturing video feed from the webcam and processing the images to detect Weapon using the trained YOLOv8 model.
  3. **Bootstrap**: Used for creating a responsive front-end, ensuring that the user interface adapts well across different screen sizes and devices.
  4. **Ultralytics**: A critical library for implementing YOLO (You Only Look Once) models, specifically the YOLOv8 Nano model, used for Weapon detection in live webcam video streams.
  5. **Playsound**: A library used to play sound alerts when a Weapon is detected in the video stream, notifying the user immediately of a potential Weapon hazard.
- ❖ Cross-Browser Compatibility: The UI components are tested across major web browsers (eg, Chrome, Firefox, Safari) to ensure consistent behaviour, appearance, functionality.

6.2.3 Security Standards:

- ❖ Input Validation: Strict input validation is applied to all user inputs (eg, registration forms, Weapon station searches). This helps prevent malicious data from entering the system and ensures that only valid inputs are processed Inputs like images or video files for Weapon detection are also validated before being passed to the deep learning model.
- ❖ Session Management: Secure handling of user sessions ensures that user data, including login information and predictions, is safely stored during interaction with the system Django’s session management ensures the integrity and confidentiality of user sessions.
- ❖ Authentication and Authorization:
  1. Secure Login Mechanisms: Passwords are encrypted using Django’s built-in

hashing mechanisms, ensuring that user credentials are stored securely.

2. Role-Based Access Control: The system uses role-based access control (RBAC) to differentiate access between Admins and Users Admins have access to manage Weapon stations, view incident records, and user profiles, while regular users can access Weapon station locations and perform Weapon detection.

6.3 CODE SAMPLES

```
{% load static %}

<html lang="en">

<head>
  <meta charset="utf-8">
  <meta content="width=device-width, initial-scale=10" name="viewport">
  <title>User Home</title>
  <meta name="description" content="Real-Time Weapon Detection and Alerting System for
enhanced security">
  <meta name="keywords" content="weapon detection, security, AI, real-time, surveillance,
safety">


  <link href="{% static 'sta/assets/img/tit_logojpg' %}" rel="icon">
  <link href="selogopng" rel="apple-touch-icon">

<link href="{% static 'sta/assets/css/maincss' %}" rel="stylesheet">
<style>
    dropdown-menu {
      display: none;
      position: absolute;
      background: white;
      padding: 10px;
      list-style: none;
      box-shadow: 0px 4px 6px rgba(0,0,0,01);
    }

    dropdown:hover dropdown-menu {
      display: block;
    }
    dropdown-menu li {
      padding: 5px 10px;
    }
      /* Hero Section */
     hero {
    position: relative;
    background-color: #111;
    color: #fff;
    padding: 100px 0;
    text-align: center;
    overflow: hidden;
}

  hero img {
    width: 100%;
    height: auto;
    position: absolute;
    top: 0;
    left: 0;
    z-index: -1;
    opacity: 05;
}

  hero h2 {
    font-size: 3rem;
    font-weight: 700;
    color: #fff;
    margin-bottom: 20px;
    text-transform: uppercase;
}

  hero p {
    font-size: 12rem;
    color: #ddd;
    margin-bottom: 30px;
}

 btn-get-started {
    padding: 12px 30px;
    background-color: #007bff;
    color: #fff;
    font-size: 11rem;
```

```
   text-decoration: none;
   border-radius: 5px;
   transition: background-color 03s ease;
}

btn-get-started:hover {
   background-color: #0056b3;
}

@media (max-width: 768px) {
hero h2 {
       font-size: 2rem;
   }

hero p {
       font-size: 1rem;
   }

btn-get-started {
       font-size: 1rem;
   }
}


image-container {
  display: block; /* Ensure it takes up the full width */
  text-align: center; /* Center align the content inside */
  margin-top: 20px;
  padding: 10px;
  margin-left: 600px;
  }

uploaded-image {
  width: 100%;
  max-width: 200px;
  margin-top: 300px;
  height: 180px;
  border-radius: 12px;
  border: 2px solid #ff4d4d;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 015);
  transition: transform 03s ease, box-shadow 03s ease;
  }
uploaded-image:hover {
  transform: scale(103);
  box-shadow: 0 6px 16px rgba(0, 0, 0, 025);
  margin-left: 50px;
  }
prediction-box {
   text-align: center;
   font-size: 18px;
   background: linear-gradient(135deg, #fff, #f8f8f8); /* Subtle gradient for depth */
   padding: 20px;
   margin-top: 100px;
   border-radius: 12px;
   border-left: 4px solid #ff4d4d; /* Red accent on the left */
   box-shadow: 0 4px 12px rgba(0, 0, 0, 015);
   color: #333;
   max-width: 350px; /* Match the image width */
   margin-left: auto;
   margin-right: 05px;
}

prediction-box p {
   margin: 8px 0; /* Better spacing between lines */
   font-family: 'Arial', sans-serif; /* Consistent typography */
}

prediction-box p strong {
   color: #ff4d4d; /* Red color for labels to match fire theme */
   font-weight: 700; /* Bold labels */
}

prediction-box p span {
   color: #555; /* Slightly lighter color for values */
}
</style>

</head>

<body class="index-page">

   <div style="" align="center">

               {% if messages %}

                   {% for msg in messages %}
                   {{ msg }}

                 {% endfor %}

               {% endif %}

           </div>

  <header id="header" class="header d-flex align-items-center fixed-top">
   <div class="container-fluid container-xl position-relative d-flex align-items-center justify-
content-between">

      <a href="indexhtml" class="logo d-flex align-items-center">
        <!-- Uncomment the line below if you also wish to use an image logo -->
        <img src="{% static 'sta\assets\img\weapon_logopng' %}" alt="">
        <h1 class="sitename">SAFEEYE</h1>
      </a>

      <nav id="navmenu" class="navmenu">
        <ul>
          <li><a href="#home" class="active">Home</a></li>
          <li><a href="{% url 'user_profile' %}">Profile</a></li>
          <li><a href="{% url 'user_profile' %}">Live Detection</a></li>
          <li><a href="{% url 'user_profile' %}">Emergency Contacts</a></li>
          <li><a href="{% url 'home' %}">Logout</a></li>
          <li><a href="#contact">Contact</a></li>
        </ul>
        <i class="mobile-nav-toggle d-xl-none bi bi-list"></i>
      </nav>

   </div>
  </header>
  <main class="main">


   <section id="hero" class="hero section dark-background">
   <!-- Background Image -->

        <img src="{% static 'sta\assets\img\tit_logojpg' %}" alt="WELCOME TO
HOMEPAGE" >
```

```
    <div class="container text-center" data-aos="fade-up" data-aos-delay="100">
        <div class="row justify-content-center">
            <div class="col-lg-8">
                <div>
                <h2>WELCOME TO HOMEPAGE</h2>
                <p>Experience the power of weapon detection, delivering immediate alerts to keep
your environment secure</p>
                </div>
<div>
  <article>
      <footer>
          <ul class="nospace inline pushright">
              <li>
                  <form enctype="multipart/form-data" method="post" style="display: inline;">
                      {% csrf_token %}
                      <input accept="image/*" id="image-upload" name="image"
                             onchange="thisformsubmit();" style="display: none;" type="file">
                      <label class="btn-get-started" for="image-upload">Image</label>
                  </form>
              </li>
          </ul>
      </footer>
{% if image_data %}

                    <div class="image-container" style="text-align: center;">
                        <img alt="Uploaded Image" class="uploaded-image"
src="data:image/jpeg;base64,{{ image_data }}" >

                    </div>
                    <div>
<!--                        <label><a href="#" id="open-cam" class="btn">LIVE</a></label>-->
                    </div>

<!--                    <div class="image-container" style="display: flex; justify-content: center;
align-items: center; height: 100px;">-->
<!--                        <img alt="Uploaded Image" class="uploaded-image"
src="data:image/jpeg;base64,{{ image_data }}">-->
<!--                    </div>-->
                    <div class="prediction-box">
                        <p><strong>Confidence:</strong>
<span>{{ confidence|floatformat:2 }}%</span></p>
                        <p><strong>Classification:</strong> <span>{{ label }}</span></p>
                    </div>
                    {% endif %}
                </article>
                </div>
</div>
        </div>
    </div>
        <div class="webcam-container" id="webcam-container">
        <img id="webcam-feed" class="webcam-feed" src="" alt="Webcam Feed">
        <div class="btn-container">
          <button id="stop-cam" class="btn" style="display: none;">Stop Camera</button>
        </div>
      </div>
</section>
<section id="about" class="about section">

    <!-- Section Title -->
    <div class="container section-title" data-aos="fade-up">
        <h2>About</h2>
        <p><span>Learn More</span> <span class="description-title">About
Us<br></span></p>
    </div><!-- End Section Title -->

    <div class="container">

        <div class="row gy-5">

            <div class="content col-xl-5 d-flex flex-column" data-aos="fade-up" data-aos-
delay="100">
                <h3>Weapon (Knife and Gun) Detection and Alerting System</h3>
                <p>
                    Our advanced weapon detection system identifies potential threats, including
knives and guns, in real-time It instantly alerts authorities to ensure swift action for public
safety
                </p>
                <a href="#about" class="about-btn align-self-center align-self-xl-start">
                    <span>Learn More</span>
                    <i class="bi bi-chevron-right"></i>
                </a>
            </div>
<div class="container section-title" data-aos="fade-up">
        <h2>About</h2>
        <p><span>Learn More</span> <span class="description-title">About
Us<br></span></p>
```

```
  </div><!-- End Section Title -->

  <div class="container">

      <div class="row gy-5">

          <div class="content col-xl-5 d-flex flex-column" data-aos="fade-up" data-aos-
delay="100">
            <h3>Weapon (Knife and Gun) Detection and Alerting System</h3>
            <p>
              Our advanced weapon detection system identifies potential threats, including
knives and guns, in real-time It instantly alerts authorities to ensure swift action for public
safety
            </p>
            <a href="#about" class="about-btn align-self-center align-self-xl-start">
              <span>Learn More</span>
              <i class="bi bi-chevron-right"></i>
            </a>
          </div><!-- End content -->

          <div class="col-xl-7" data-aos="fade-up" data-aos-delay="200">
            <div class="row gy-4">

              <div class="col-md-6 icon-box position-relative">
                <i class="bi bi-gun"></i> <!-- Icon for gun detection -->
                <h4><a href="" class="stretched-link">Gun Detection</a></h4>
                <p>Our system instantly detects firearms, alerting authorities immediately to
prevent potential threats</p>
              </div><!-- Icon-Box -->

              <div class="col-md-6 icon-box position-relative">
                <i class="bi bi-knife"></i> <!-- Icon for knife detection -->
                <h4><a href="" class="stretched-link">Knife Detection</a></h4>
                <p>The system identifies knives, triggering real-time alerts to ensure
safety</p>
              </div><!-- Icon-Box -->

              <div class="col-md-6 icon-box position-relative">
                <i class="bi bi-bell"></i> <!-- Icon for alert system -->
                <h4><a href="" class="stretched-link">Instant Alerts</a></h4>
                <p>Immediate notifications sent to security or law enforcement agencies to
respond swiftly to threats</p>
              </div><!-- Icon-Box -->
          <div class="col-md-6 icon-box position-relative">
            <i class="bi bi-shield-lock"></i> <!-- Icon for security -->
            <h4><a href="" class="stretched-link">Enhanced Security</a></h4>
            <p>Ensures public safety by detecting weapons early and enabling proactive
measures to protect the community</p>
          </div><!-- Icon-Box -->

        </div><!-- End row -->
      </div><!-- End col-xl-7 -->

    </div><!-- End row -->

  </div><!-- End container -->

</section>
</main>
<script>
documentaddEventListener('DOMContentLoaded', function () {
  const openCamBtn = documentgetElementById('open-cam');
  const stopCamBtn = documentgetElementById('stop-cam');
  const webcamContainer = documentgetElementById('webcam-container');
  const webcamFeed = documentgetElementById('webcam-feed');
  const alarm = documentgetElementById('alarm');
  let weaponDetected = false;

  // Open camera
  openCamBtnaddEventListener('click', function (e) {
    epreventDefault();
    webcamFeedsrc = "{% url 'open_camera' %}";
    webcamContainerstyledisplay = 'flex';
    openCamBtnstyledisplay = 'none';
    stopCamBtnstyledisplay = 'inline-block';
  });

  // Stop camera
  stopCamBtnaddEventListener('click', function () {
    fetch("{% url 'stop_camera' %}")
      then(response => responsejson())
      then(data => {
        webcamFeedsrc = '';
        webcamContainerstyledisplay = 'none';
        openCamBtnstyledisplay = 'inline-block';
```

```
            stopCamBtnstyledisplay = 'none';
            alarmpause();
            alarmcurrentTime = 0; // Reset alarm
            weaponDetected = false;
        });
    });

    // Polling function for weapon detection alert
    function pollWeaponAlert() {
        if (webcamContainerstyledisplay === 'flex') {
            fetch('/weapon-alert-status/')
                then(response => responsejson())
                then(data => {
                    if (dataweapon_detected && !weaponDetected) {
                        weaponDetected = true;
                        alarmplay()catch(err => {
                            consolewarn("Autoplay error:", err);
                        });
                    }
                })
                catch(error => {
                    consoleerror('Error polling weapon alert status:', error);
                });
        }
    }

    // Start polling every 3 seconds
    setInterval(pollWeaponAlert, 3000);
});

    function previewMedia(event) {
        const previewContainer = documentgetElementById('mediaPreview');
        const file = eventtargetfiles[0];
        previewContainerinnerHTML = ''; // Clear previous preview

        if (file) {
            const fileType = filetypesplit('/')[0];

            if (fileType === 'image') {
                const imagePreview = documentcreateElement('img');
                imagePreviewsrc = URLcreateObjectURL(file);
                imagePreviewclassListadd('image-preview');
                previewContainerappendChild(imagePreview);
```

```
          } else if (fileType === 'video') {
             const videoPreview = documentcreateElement('video');
             videoPreviewcontrols = true;
             videoPreviewsrc = URLcreateObjectURL(file);
             videoPreviewclassListadd('video-preview');
             previewContainerappendChild(videoPreview);
          }
       }
    }
</script>

</body>

</html>
```

CHAPTER 7. TESTING

Software testing is the systematic process of verifying and validating a system’s performance and functionalities to ensure it meets design expectations and operational requirements. It involves uncovering potential errors, evaluating system attributes in diverse scenarios, and ensuring quality standards are met. In the case of weapon detection systems, testing ensures accuracy in identifying objects like guns and knives, reliable real-time alert mechanisms, and seamless integration of features. This process plays a pivotal role in maintaining robustness and security within the system.

7.1 TESTING PROCESSS

The goal of testing is to detect errors across various phases, including coding, design, and system integration. Testing helps confirm that the system adheres to its specified functionalities while addressing requirements and environmental challenges. Software testing forms a cornerstone of quality assurance and is essential for detecting vulnerabilities and improving system reliability [9].

7.1.1 Unit Testing

Unit testing focuses on verifying individual components of the system in isolation. For the weapon detection alert system, it targets the following:

- ❖ **Detection Model Performance**: Validate the accuracy of the object detection model (eg, YOLOv8) in identifying weapons such as guns and knives using simulated images.
- ❖ **Real-Time Alert Mechanism:** Test live notifications, such as alarm sounds and email triggers, ensuring proper functioning.
- ❖ **User Interface:** Check that forms and inputs—like uploading images or accessing live feeds—operate smoothly and handle validations effectively [8]

7.1.2 Validation Checks

Ensuring correctness and security in user inputs is critical to avoid system errors and maintain operational integrity.

- ❖ **Form Validation:** Test forms for uploading images or accessing live feeds, ensuring data completeness and accuracy. Reject invalid formats such as unsupported image types or incorrect email addresses.

- ❖ **Live Feed Validation:** Confirm that camera feeds are correctly displayed and processed without disruptions.
- ❖ **Error Handling:** Verify that error messages are displayed appropriately for invalid inputs or system issues, ensuring clarity for end users.

7.1.3 Integration Testing

Integration testing examines how the system’s modules interact, ensuring seamless coordination between components such as detection algorithms, interfaces, and alert mechanisms.

1. Key Testing Objectives:

- ❖ Validate data flow between the detection model, user interface, and real-time alerts.
- ❖ Confirm smooth integration between the backend (eg, Django) and frontend components.
- ❖ Test the interaction between live webcam feeds and the detection model.

2. Critical Modules to Test:

- ❖ Detection Model: Verify the integration of YOLOv8 with the backend, ensuring accurate weapon identification during live scenarios.
- ❖ Alert System: Check that alarms and email notifications are reliably triggered upon detection.
- ❖ Admin Panel: Ensure admin functionalities, such as profile access and registered users viewing and deletion.
- ❖ User Dashboard: Validate the user interface for uploading images, tracking alerts, and accessing results.

3. Test Cases:

- ❖ Simulate weapon detection scenarios under different conditions, such as varying lighting or occlusions.
- ❖ Assess the system’s ability to process and react to webcam feeds in real time.
- ❖ Confirm the reliability of notifications, including sound alarms and emails with detailed detection reports.
- ❖ Test UI responsiveness, ensuring a user-friendly experience across devices.

4. Integration Testing Approaches:

- Top-Down: Test high-level features first, then progressively verify lower-level functionalities.
- Bottom-Up: Start by evaluating the detection model and build up to the user-facing components.
- End-to-End: Conduct system-wide testing to confirm all modules interact effectively and deliver expected results.

7.1.4 Testing Tools

The following tools support the testing process:

- **PyTest**: Unit testing for individual components like detection algorithms and alert triggers.
- **Postman:** Validate API interactions between frontend and backend systems.
- **Selenium:** Test user interface workflows and ensure smooth interactions.
- **Jupyter Notebook:** For testing the integration of the YOLOv8 Nano model within the back-end logic.
- **OpenCV**: Verify real-time processing of webcam feeds by the detection model.
- **Email Testing APIs**: Ensure notifications are correctly formatted and delivered to specified recipients.

7.1.5 Expected Outcomes

- **High Detection Accuracy**: The system must identify weapons with minimal errors, even under challenging conditions.
- **Reliable Alert Mechanisms**: Alarms and notifications should activate immediately upon detection events.
- **Smooth Integration**: The interaction between the detection model, UI, and backend should remain stable and consistent.
- **Enhanced Usability:** Admin and user dashboards should offer a seamless experience, allowing quick access to data and tools.

7.1.6 Reporting

Defects, test case outcomes, and execution status are logged systematically to document progress, identify areas for improvement, and ensure quality assurance. Tools such as JIRA or GitHub Issues can be employed to track findings and resolutions effectively.

7.2 TEST PLAN DERIVED

The test plan derived for the project includes a series of sprint-based test cases that validate core functionalities such as user authentication, weapon detection, emergency contact management, and real-time alerts. Each test case was designed to ensure system reliability, usability, and accurate detection outcomes. All planned tests passed successfully, confirming system readiness for deployment.

Table 71: Test Plan

Sprint	Test Case	Expected Outcome	Status
Sprint 1	Project setup and planning validation	Project scope, timelines, and task assignments are clearly defined	Passed
Sprint 2	Verify completeness of gathered requirements	Functional and non-functional requirements are well-documented	Passed
Sprint 3	Load and validate raw data	Data is cleaned, missing values handled, and ready for model training	Passed
Sprint 4	Train baseline weapon detection model	Model trains successfully with baseline metrics established	Passed
Sprint 5	Train advanced CNN model and tune hyper-parameters	Advanced model achieves improved accuracy and performance	Passed
Sprint 6	UI elements function and model output is integrated properly	User interface displays inputs, results, and handles errors correctly	Passed
Sprint 7	Integration testing and system-wide functional validation	Full system runs end-to-end with accurate alerts and performance monitoring	Passed

7.3 RESULTS FROM TESTING

Testing results validated the **reliability, accuracy, and usability** of the weapon detection system:

- **100% coverage** of all core modules including user authentication, weapon detection (image and live), emergency alert notifications, and contact management.
- **Responsive user interface** across desktop and mobile platforms, ensuring accessibility for all users.
- **Model inference and database operations** completed within optimal response times, providing real-time feedback and efficient performance.
- **All bugs and issues** identified during the testing process were resolved within their respective sprint cycles.
- **Final UAT (User Acceptance Testing)** confirmed the system's **readiness for deployment** and real-world usage, with no critical issues remaining.

CHAPTER 8. DEPLOYMENT STRATEGY

8.1 VERSION CONTROL AND GIT

In academic projects like those at APJA Kerala Technological University (APJA-KTU), utilizing. Scrum as a development framework, version control systems such as Git are indispensable tools. These systems ensure that code integrity is maintained while enabling seamless collaboration among team members. By leveraging Git, development teams can manage code changes systematically, foster teamwork, and ensure smooth progress throughout the project lifecycle [10].

Importance of Version Control and Git

- **Effective Collaboration**: Git allows multiple developers to work simultaneously on a single project without interfering with each other's contributions. Each team member can create separate branches for feature development or bug fixes, supporting independent work. This branching mechanism ensures efficient integration of individual efforts, which is particularly valuable for Scrum-based workflows
- **Comprehensive Change Tracking**: Git maintains a detailed record of all modifications, capturing who made a change, what was altered, and when it occurred. This level of traceability helps teams understand the evolution of the project and facilitates problem-solving by identifying the source of issues quickly.
- **Safe Rollbacks**: Mistakes or bugs introduced in new updates can be quickly resolved by reverting to a prior, stable version of the codebase. This feature is particularly useful in academic projects with strict deadlines, as it minimizes time lost in debugging.
- **Code Review and Quality Improvement**: Git supports collaborative coding by enabling code reviews through pull requests. Before code merges into the main branch, team members can review, suggest improvements, and ensure adherence to best practices. This fosters a learning environment and enhances the overall quality of the project.
- **Data Security and Recovery**: Storing all project files in a centralized Git repository ensures that the codebase is protected against local machine failures. In case of data loss or hardware issues, the entire project can be recovered quickly from the repository.

- **Efficient Workflow Management**: By linking Git commits to Scrum tasks or user stories, teams can efficiently track development progress and ensure alignment with project objectives. This integration enhances Sprint planning and execution by providing a clear picture of task completion.

Platform in Use: Git repositories for the project are managed on GitHub, which provides an accessible interface for code storage, collaboration, and version control.

8.2 RELEASE NOTES

Release notes summarize the features delivered in each major version of the project, highlight the changes introduced, and document any known issues or limitations.

8.2.1 Version 1.0 – Final Academic Submission

• Real-time weapon detection using YOLOv8 Nano integrated with OpenCV
• Live video stream analysis using system webcam or external camera
• Automated email alert system using SMTP configuration
• User authentication module (login, registration, session management)
• Role-based access control (Admin/User)
• Fully responsive web interface using HTML, CSS, and JavaScript
• Integration of Django framework with SQLite database
• Modular structure for future integration of email alerts
• Project trained and tested using Jupyter Notebook, Google Colab and deployed locally via Django server

8.2.2 Known Issues

• Higher-than-expected **false positive rate** in crowded or low-light environments
• **Limited dataset availability** for diverse weapon types and environmental conditions
• **Insufficient system specifications** caused slower model inference during real-time detection
• Current system is optimized for local deployment — **cloud-based scalability is a future goal**

8.3.3 Repository Details

- **Platform**: GitHub
- **Repository Name**: Weapon-Detection-Knife-and-Gun-and-Alerting-System
- **Visibility**: Private (Academic submission)
- **Link address**: https://github.com/Aiswarya1419/Weapon-Detection-Knife-and-Gun-and-Alerting-System.

CHAPTER 9. SUMMARY

9.1 RETROSPECTIVE NOTES

1. **Project Scope and Management**

❖ **Lesson Learned**: Early scope definition is essential for project success.

❖ **Observation**: Delays occurred due to ambiguous requirements and insufficient time allocated for planning. Refining the scope during development helped the team focus on critical features like weapon detection algorithms and alert systems.

❖ **Action**: Invest more time in defining project scope, objectives, and timelines at the start to prevent future inefficiencies.

2. Team Coordination and Workflow

❖ **Lesson Learned**: Strong team coordination drives effective development.t

❖ **Observation**: Initial misalignments in task distribution impacted workflow Introducing collaboration tools and periodic check-ins helped resolve these issues, fostering better progress.

❖ **Action:** Incorporate consistent team updates and collaboration mechanisms to streamline workflow in future projects.

3. Requirements Analysis

❖ **Lesson Learned**: Clear requirements prevent development bottlenecks.

❖ **Observation**: Lack of clarity in user expectations initially caused challenges in deploying real-time detection features. Refining requirements improved overall development and system integration.

❖ **Action**: Collaborate closely with stakeholders to establish comprehensive and precise requirements before development begins.

4. Technology Proficiency

❖ **Lesson Learned**: Mastery of tools and technologies saves time.

❖ **Observation**: Learning curves with frameworks like YOLOv8 and alarm notification libraries slowed progress. Hands-on training sessions significantly enhanced team efficiency.

- **Action**: Prioritize team proficiency in tools and frameworks used, offering workshops or training as needed before starting the project.

5. **Agile Approach**

- **Lesson Learned**: Agile methods ensure adaptability in evolving projects.
- **Observation**: Quick adjustments were made to refine model performance and improve detection reliability using iterative Agile practices These ensured the project adapted to new challenges effectively.
- **Action**: Continue embracing Agile principles, promoting flexibility in response to changes and improvements.

6. **Optimization Strategies**

- **Lesson Learned**: Early performance optimization avoids bottlenecks later.
- **Observation**: Real-time detection systems experienced delays due to high computational demands. Database and algorithm optimizations helped improve response times and system reliability.
- **Action**: Implement profiling and optimization strategies early in future projects to ensure smooth performance under high loads.

7. **Model Refinement**

- **Lesson Learned**: Iterative refinement maintains accuracy and efficiency.
- **Observation:** Edge cases, such as varied lighting conditions, posed challenges in detection accuracy. Additional training data and model adjustments improved system reliability.
- **Action**: Plan for ongoing model improvements, leveraging real-world data to refine accuracy continuously.

8. User Interface Enhancements

- **Lesson Learned**: Intuitive UI design enhances usability and user adoption.
- **Observation**: Feedback revealed usability issues in the initial design Iterative improvements resulted in a user-friendly interface suitable for emergency situations.
- **Action**: Focus on user-centric design principles and gather regular feedback during development to create seamless interfaces.

9. Emergency Service Integration

- **Lesson Learned**: Integration boosts practical utility.
- **Observation**: Incorporating emergency service notifications greatly improved response times and system effectiveness.
- **Action**: Make emergency service integration a priority in future systems to ensure comprehensive responses during crises.

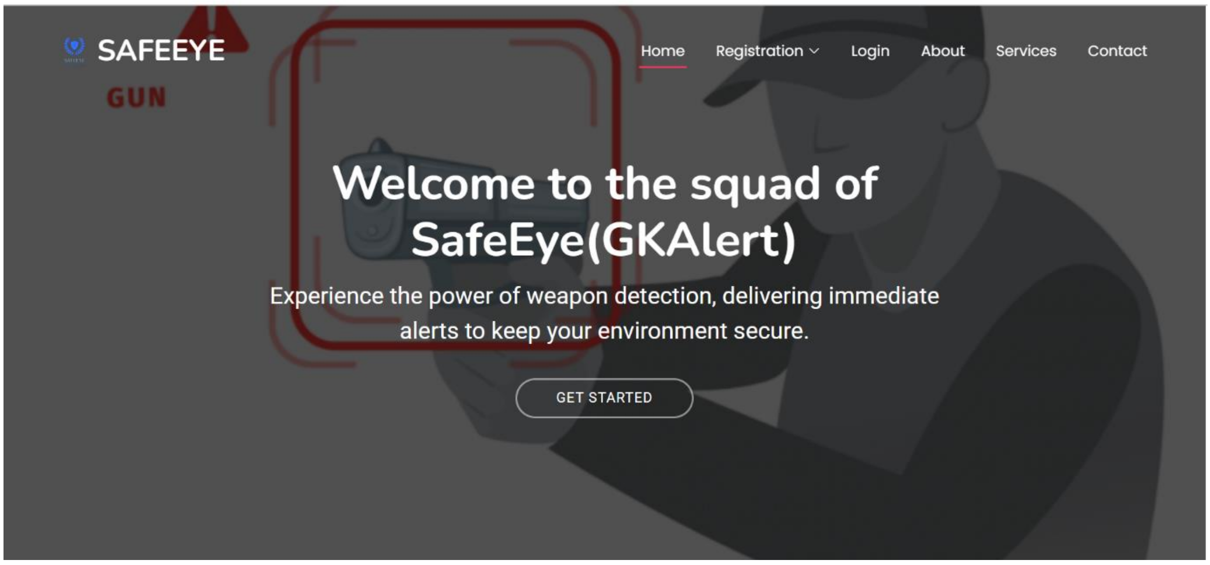
10. Testing and Reliability

- **Lesson Learned**: Rigorous testing safeguards system performance in real-world scenarios.
- **Observation**: Testing revealed gaps in detection under specific conditions, leading to targeted fixes before deployment Enhanced testing strategies ensured robust performance.
- **Action**: Strengthen testing methodologies, focusing on diverse conditions and edge cases, and implement continuous quality assurance processes [11].

APPENDICES

SCREENSHOTS OF USER INTERFACE

1. Home Page



Welcome to the squad of SafeEye(GKAlert)

Experience the power of weapon detection, delivering immediate alerts to keep your environment secure.

Fig 1: Home page

2. Admin Registration

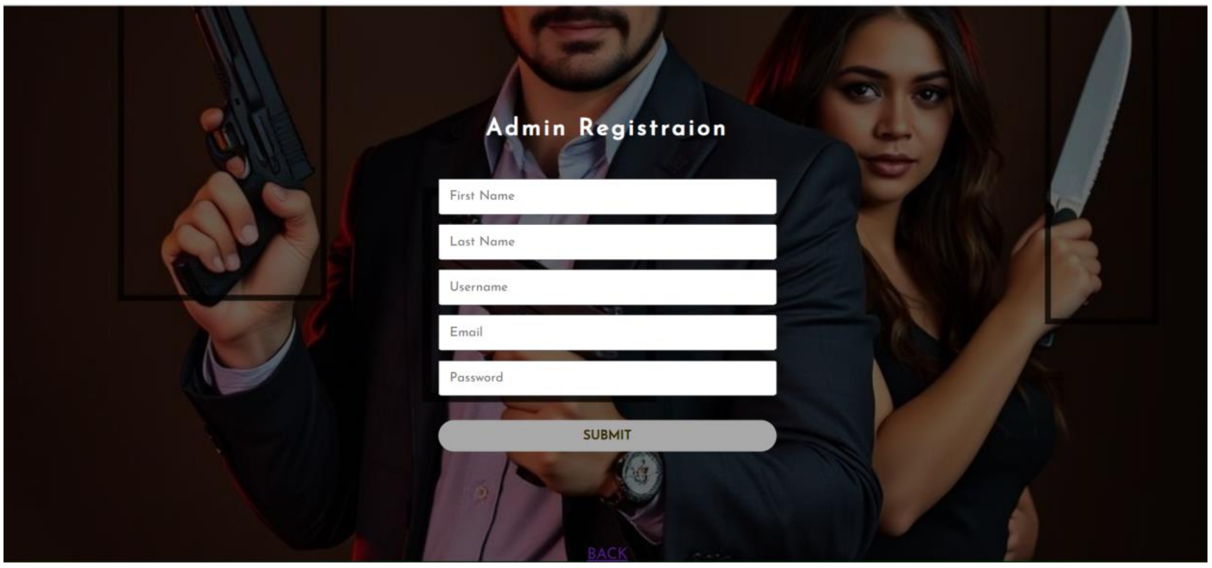
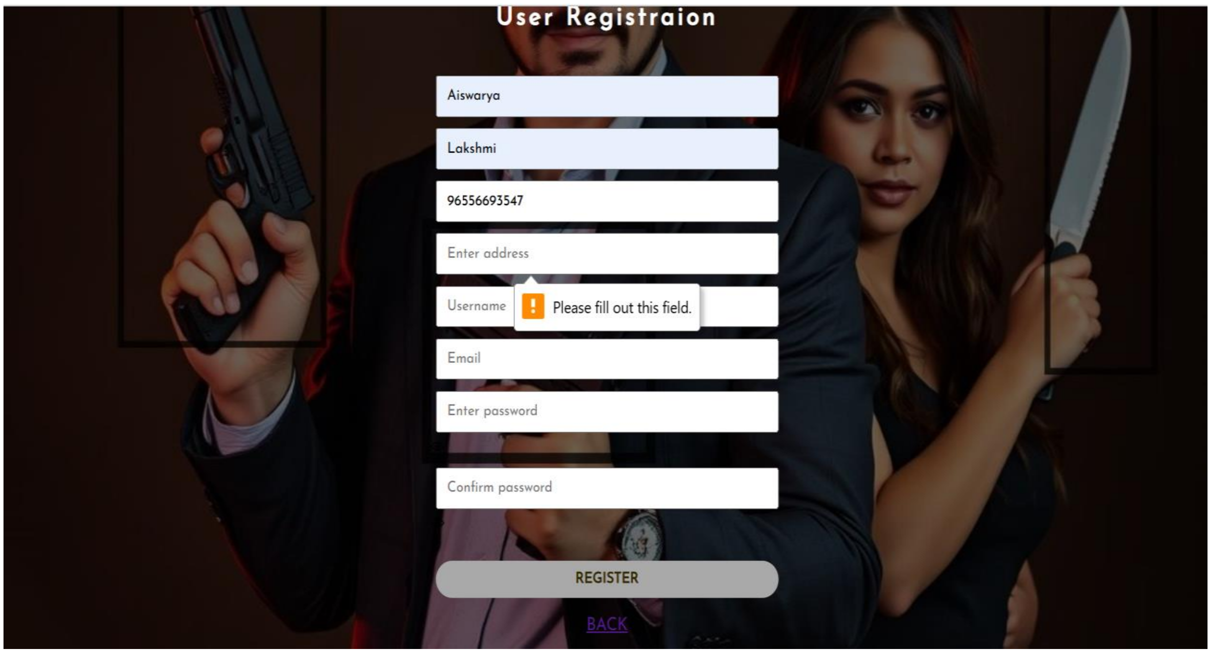


Fig 2: Admin Registration

3. User Registration


User Registraion

Aiswarya

Lakshmi

96556693547

Enter address

Username

Please fill out this field.

Email

Enter password

Confirm password

Fig 3: User Registration

4. Login Page

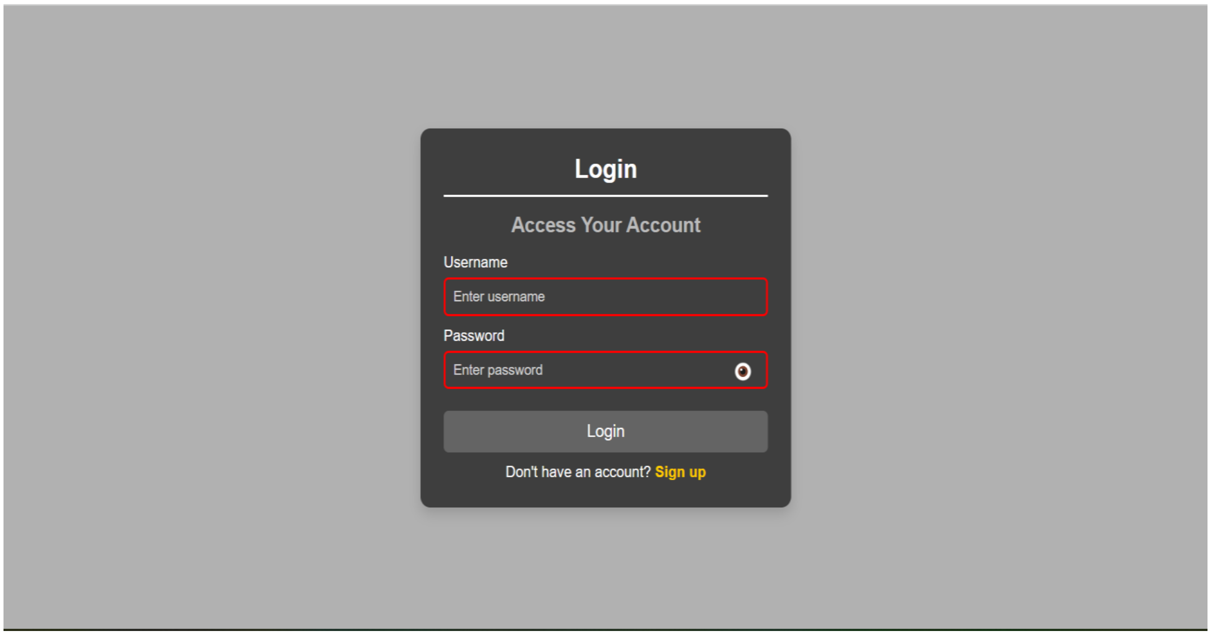
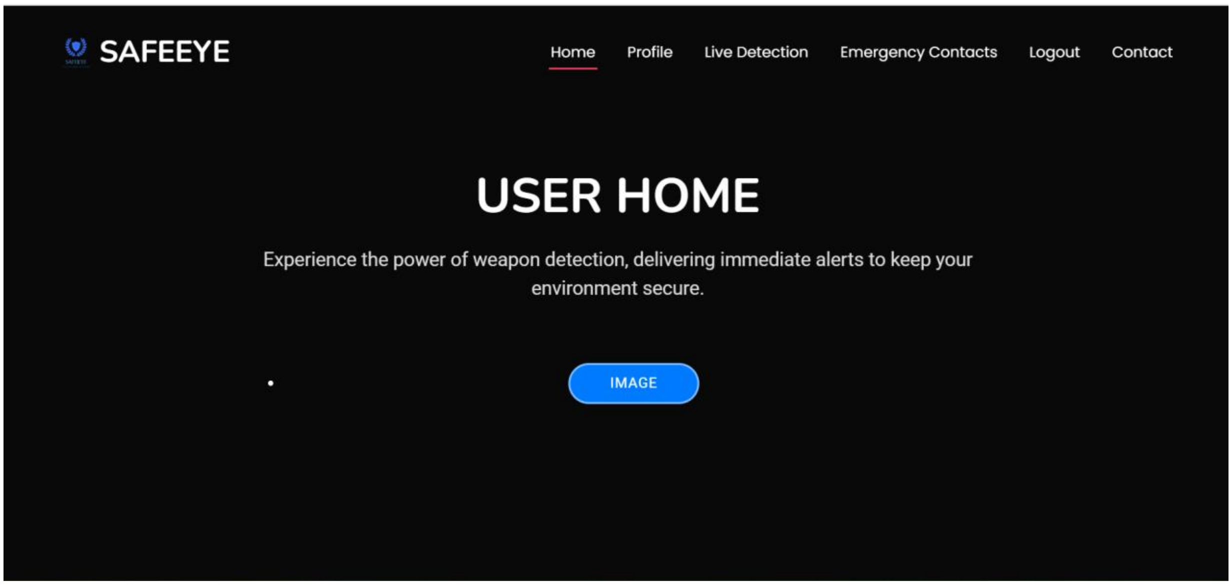


Fig 4: Login Page

5. User Home

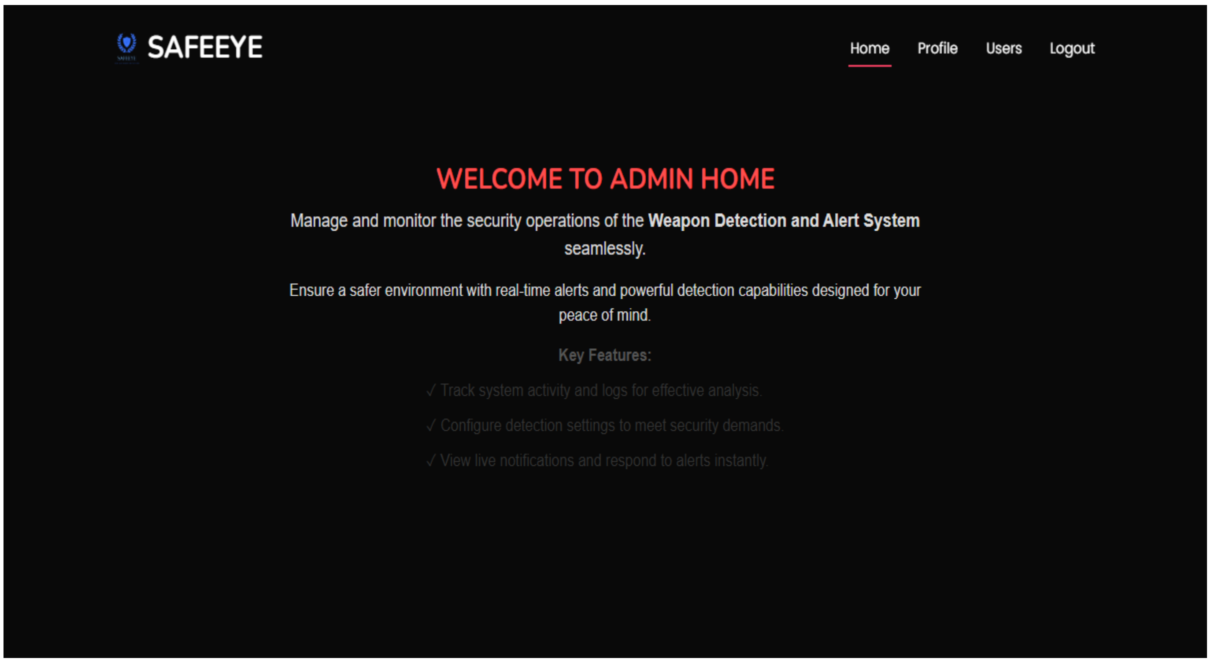


USER HOME

Experience the power of weapon detection, delivering immediate alerts to keep your environment secure.

Fig 5: User Home

6. Admin Home



WELCOME TO ADMIN HOME

Manage and monitor the security operations of the **Weapon Detection and Alert System** seamlessly.

Ensure a safer environment with real-time alerts and powerful detection capabilities designed for your peace of mind.

Key Features:

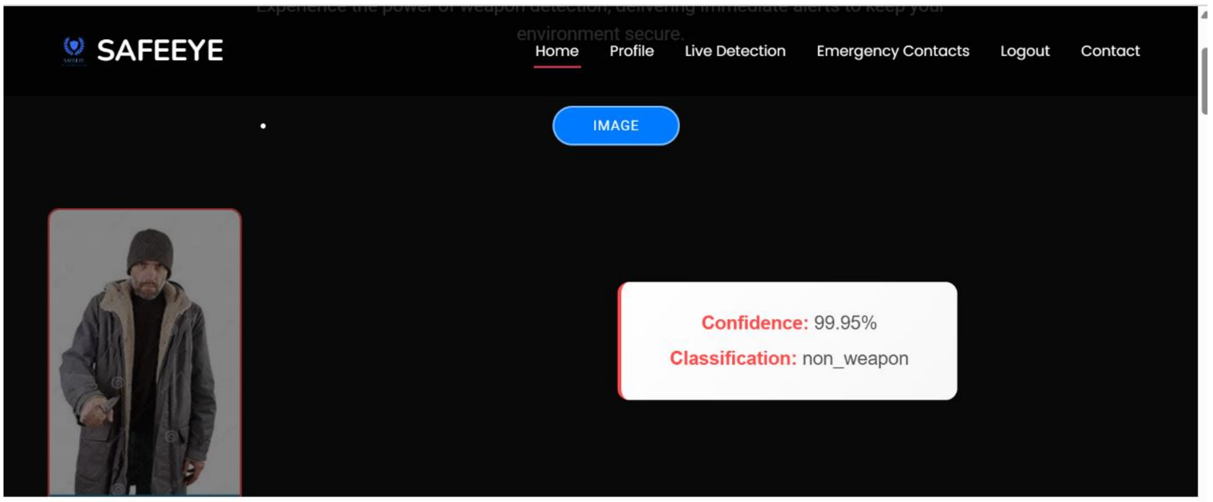
✓ Track system activity and logs for effective analysis.

✓ Configure detection settings to meet security demands.

✓ View live notifications and respond to alerts instantly.

Fig 6: Admin Home

7 .Weapon Detection



environment secure.

Confidence: 99.95%

Classification: non_weapon

Fig 7: Weapon Detection

REFERENCES

1. **Alberto Castillo Lamas, Francisco Herrera Triguero,** *Weapons detection for security and video surveillance,* Soft Computing and Intelligent Information Systems, Available: https://sci2s.ugr.es/weapons-detection
2. **IBM** (22 Sep 2021), *What is machine Learning,* Available:https://www.ibm.com/topics/machinelearning?src_trk=em66f6c45983f3f5.70131270346653526
3. **Gupta, Pavinder Yadav**, *A comprehensive study towards high-level approaches for weapon detection using classical machine learning and deep learning methods,* Science Direct,Available:https://www.sciencedirect.com/science/article/pii/S0957417422017286
4. **geeksforgeeks** , *Overview of Burndown Chart in Agile,* Available:https://www.geeksforgeeks.org/overview-of-burndown-chart-in-agile/
5. **Snehil Sanyal**, **Weapon Detection Dataset**, Kaggle , Available:https://www.kaggle.com/datasets/snehilsanyal/weapon-detection-test
6. **Team VOLT** (August 13, 2024) **, Comparing Traditional vs Modern Weapons Detection Systems**", Volt, Available:
7. **Pavinder Yadav, Nidhi Gupta, Pawan Kumar Sharma** *(*February 2023), *A comprehensive study towards high-level approaches for weapon detection using classical machine learning and deep learning methods*, Science Direct, Volume 212, Available: https://resources.volt.ai/blog/weapons-detection-systems
8. **Amazon** (February 2025), *What is Unit Testing*, AWS, Available:https://aws.amazon.com/what-is/unit-testing/
9. **Sairam Uppugundla** (June 3, 2024), *Software Testing Projects For Final Year With Source Code,* codegnan, Available: https://codegnan.com/software-testing-projects
10. **Rajeshwar Yadav, Gourinath Banda** (02 Nov 2023), *A Lightweight Deep Learning-based Weapon Detection Model for Mobile Robots* , ACM, Available:https://dl.acm.org/doi/10.1145/3610419.3610489
11. **Geeksforgeeks** (03 Jan 2025), *Activity Diagrams – Unified Modeling Language (UML),* Available: https://www.geeksforgeeks.org/unified-modeling-language-uml-introduction/