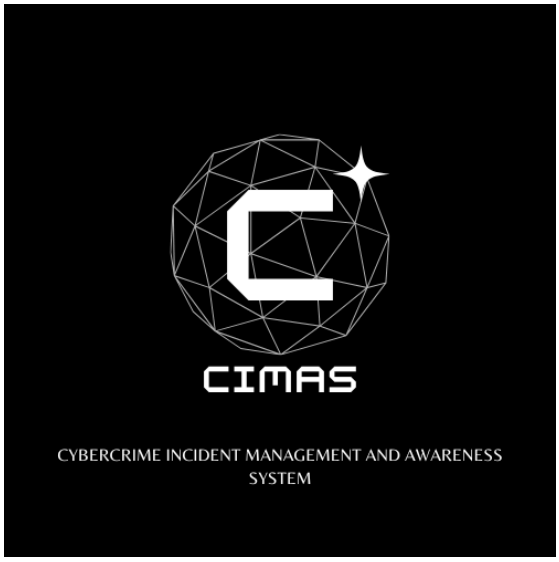
Database Management Systems
Minor Project Report
on

CIMAS- Cybercrime Incident Management and Awareness System



Prepared by:
Group VII
Adithyan A.S, 4
Adwin T Sunil, 5
Evaan Antony Philip, 23

October 23, 2025

Table of Contents

1. Introduction

In the modern digital landscape, cybercrimes have become a significant global challenge affecting individuals, organizations, and governments alike. With the increasing dependence on online services for communication, transactions, and data sharing, the potential for malicious exploitation has grown exponentially.

Cybercrimes such as:

- Identity theft and phishing,
- Ransomware and data breaches,
- Online fraud and financial scams, and
- Harassment and social engineering attacks

have become increasingly frequent and sophisticated. Managing these incidents demands a reliable, secure, and centralized platform that ensures efficient coordination between victims, investigators, and authorities.

However, **traditional reporting mechanisms** are often:

- Fragmented and slow,
- Lacking transparency,
- Poorly integrated with investigative workflows, and
- Unavailable to the general public for awareness or education.

To address these challenges, the **Cybercrime Incident Management & Awareness System (CIMAS)** has been developed. CIMAS serves as a web-based platform that:

- Enables victims to **report incidents** securely and **submit evidence** easily.
- Allows investigators and administrators to **track, assign, and manage cases** efficiently.
- Incorporates a public **Awareness Hub** to promote digital literacy and online safety.
- Utilizes modern technologies like **React**, **Django**, and **PostgreSQL** for scalability, performance, and security.

By combining efficient case management with cybersecurity education, CIMAS enhances both the **response** to and **prevention** of cybercrimes, contributing to a safer digital ecosystem.

2. Objectives

2.1 Problem Statements

The rapid increase in cybercrime incidents has exposed several weaknesses in existing systems for reporting and managing such cases. Current challenges include:

- Lack of a **centralized platform** for reporting and tracking incidents.
- **Inefficient coordination** between victims, investigators, and administrators.
- Limited ability to **submit and handle digital evidence securely.**
- Absence of **real-time case updates** and **status tracking** for victims.
- **Low public awareness** about cybersecurity best practices and preventive measures.

Due to these limitations:

- Victims experience frustration and delays in getting help.
- Investigators face challenges in managing workloads and accessing case data.
- Authorities lack analytical insights into **crime patterns** and **high-risk areas.**

2.2 Primary Objectives

The primary objective of the **Cybercrime Incident Management & Awareness System (CIMAS)** is to create a secure, efficient, and user-friendly platform for reporting, tracking, and managing cybercrime incidents while promoting cybersecurity awareness among the public.

The system focuses on bridging the gap between victims, investigators, and administrators through digital integration and transparent workflows.

Main Objectives

- **To provide a centralized platform** where victims can securely report cybercrime incidents and upload supporting evidence.
- **To enable investigators and administrators** to efficiently manage, assign, and track cases through a role-based system.
- **To ensure data security** by implementing password hashing, encrypted evidence storage, and restricted access controls.
- **To incorporate real-time case tracking** and notification features to keep users informed about case progress.
- **To develop an Awareness Hub** that educates users about safe online practices, digital hygiene, and the latest cybersecurity threats.
- **To offer crime mapping and analytics** that help identify high-risk areas and visualize cybercrime trends.

- **To enhance collaboration** between victims, law enforcement, and cyber investigators through a structured and transparent workflow.

2.3 Future-Oriented Objectives

- Integrate **AI-based crime pattern detection** for predictive analysis and smarter investigations.
- Develop a **mobile application** for easier reporting and access to awareness resources.
- Introduce **chatbot assistance** for faster complaint filing and user support.
- Implement **blockchain-based evidence verification** to ensure data integrity and prevent tampering.

3. Methodology

The methodology adopted for this project follows a systematic approach to the design, development, and deployment of the **Cybercrime Incident Management & Awareness System (CIMAS)**. The process is divided into structured phases to ensure clarity, scalability, and measurable outcomes. The project follows the **Agile Software Development Lifecycle (SDLC)**, allowing iterative improvements and regular testing.

3.1 Research and Requirement Analysis

- Conducted a detailed study of existing cybercrime reporting and management systems to evaluate their limitations and challenges.
- Identified key pain points such as lack of centralized reporting, inefficient tracking mechanisms, and poor evidence handling.
- Analyzed user needs for three primary stakeholders:
    - **Victims:** Simple and secure way to report incidents and upload evidence.
    - **Investigators:** Efficient case management, task assignment, and progress tracking.
    - **Administrators:** User management, oversight, and system monitoring.

- Defined functional requirements:
    - Role-based access control (RBAC).
    - Secure evidence submission and storage.
    - Case tracking with notifications.
    - Awareness Hub for cybersecurity education.
    - Crime mapping and analytics.

- Defined non-functional requirements including performance, reliability, scalability, and security.

3.2 Software Requirements

The project required specific software components to ensure a robust and secure web application environment.

Operating System:

- Windows / Linux / macOS (cross-platform development and deployment).

Core Software Components:

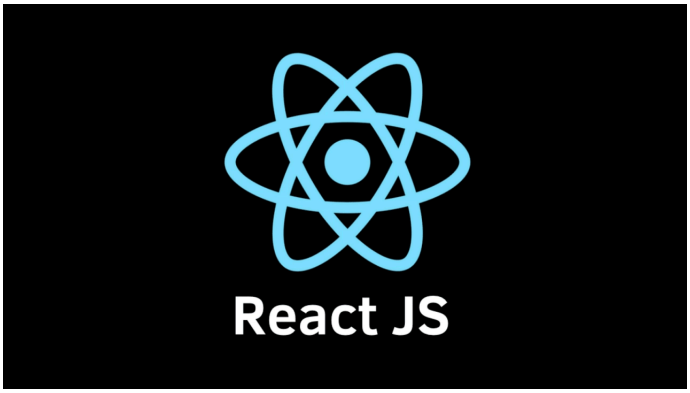
- **Frontend:** React, JavaScript, HTML, CSS – for responsive and dynamic user interface.
- **Backend:** Django Framework (Python) – for secure and scalable server-side operations.
- **Database:** PostgreSQL – for structured and reliable data storage.
- **Authentication:** Django's built-in authentication with password hashing and session management.

Supporting Tools:

- Git & GitHub – version control and collaborative development.
- Postman – API testing.
- Figma / Lucidchart – interface design and architecture visualization.
- Render / Vercel – optional deployment for demonstration purposes.

3.3 Overview of Core Technologies

React

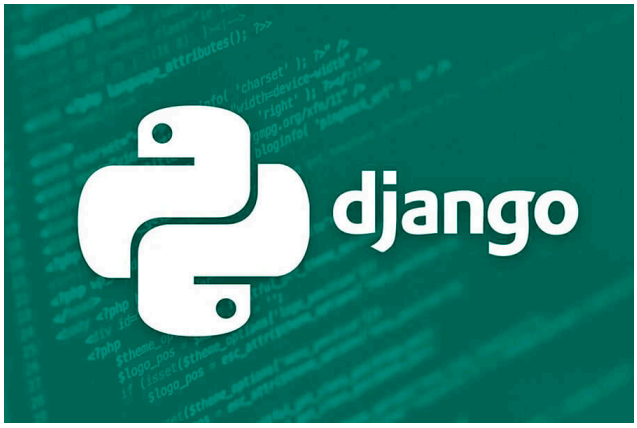


React is a JavaScript library used for building interactive and responsive user interfaces.

Core Features of React:

- Component-based architecture for modular development.
- Virtual DOM for fast rendering.
- Easy integration with APIs for dynamic content.
- Excellent scalability and performance for large-scale web apps.

Django



Django is a high-level Python web framework that promotes rapid development and security.

Core Features of Django:

- Model-View-Template (MVT) architecture for clean code organization.
- Built-in user authentication and permission handling.
- Robust ORM for database operations.
- High-level security features like CSRF protection, SQL injection prevention, and password hashing.

PostgreSQL

PostgreSQL is a powerful, open-source relational database used to store structured data securely.

Core Features of PostgreSQL:

- ACID-compliant transactions ensuring data integrity.
- Support for JSON and relational queries.
- High performance with indexing and query optimization.
- Strong security and role-based access control for sensitive data.

3.4 System Design

- Designed a **three-tier architecture** with clear separation of concerns:
    - **Frontend (Presentation Layer):** Built using React for user interaction.
    - **Backend (Application Layer):** Implemented with Django for logic, routing, and authentication.
    - **Database Layer:** PostgreSQL for storing reports, evidence, and user data.

- Defined workflows for each user role:
    - **Victim:** Register → Report Incident → Upload Evidence → Track Case.
    - **Investigator:** Review Assigned Cases → Analyze Evidence → Update Status.
    - **Administrator:** Manage Users → Assign Investigators → Monitor Overall System.

- Integrated an **Awareness Hub** for general users to learn about safe online practices.
- Designed **crime mapping and analytics** to visualize regional cybercrime trends.

3.5 Environment Setup

- Installed and configured the development environment with **Django**, **React**, and **PostgreSQL**.
- Created Django models for user roles, incidents, evidence, and awareness content.
- Implemented RESTful APIs to connect the frontend and backend.
- Configured PostgreSQL database for structured data storage with encryption and access control.
- Established secure data handling for file uploads and storage using Django's built-in security features.

3.6 Module Development

Developed and integrated the following core modules:

- **Incident Reporting Module:**
  Allows victims to file detailed complaints with supporting documents, images, or videos.
- **Evidence Management Module:**
  Ensures secure storage and restricted access to sensitive case materials.
- **Case Management Module:**
  Enables investigators and administrators to assign, update, and track case progress.
- **Notification System:**
  Provides real-time updates to victims about their case status.

- **Awareness Hub:**
  Publishes educational articles, cybersecurity tips, and alerts to promote public safety.
- **Crime Mapping and Analytics:**
  Visualizes reported incidents on an interactive map for better situational awareness.

3.7 Security Implementation

- Implemented **Password Hashing** for secure authentication.
- Enforced **Role-Based Access Control (RBAC)** for user differentiation.
- Utilized **Input Validation** and **Safe File Handling** to prevent data corruption or exploitation.
- Configured **Database Encryption** and restricted access for evidence files.
- Maintained **Audit Logs** to track administrative and investigative activities for accountability.

3.8 Testing and Validation

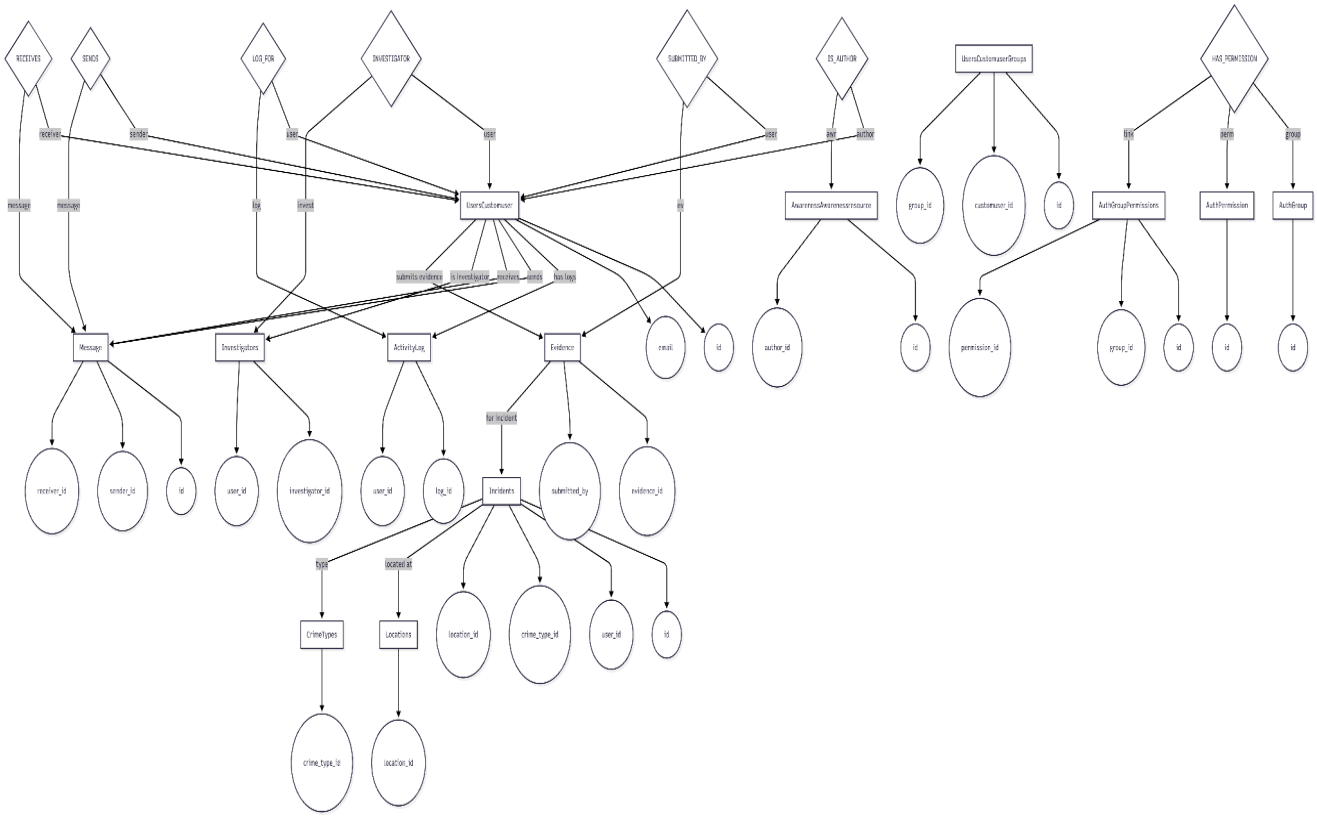
- **Unit Testing:** Verified individual modules such as incident reporting, evidence upload, and authentication.
- **Integration Testing:** Ensured seamless communication between frontend, backend, and database layers.
- **System Testing:** Validated overall functionality and user workflows.
- **Security Testing:** Checked for vulnerabilities like SQL injection, XSS, and insecure file uploads.
- **Performance Testing:** Assessed response times and scalability under multiple concurrent users.

3.9 Deployment and Future Enhancements

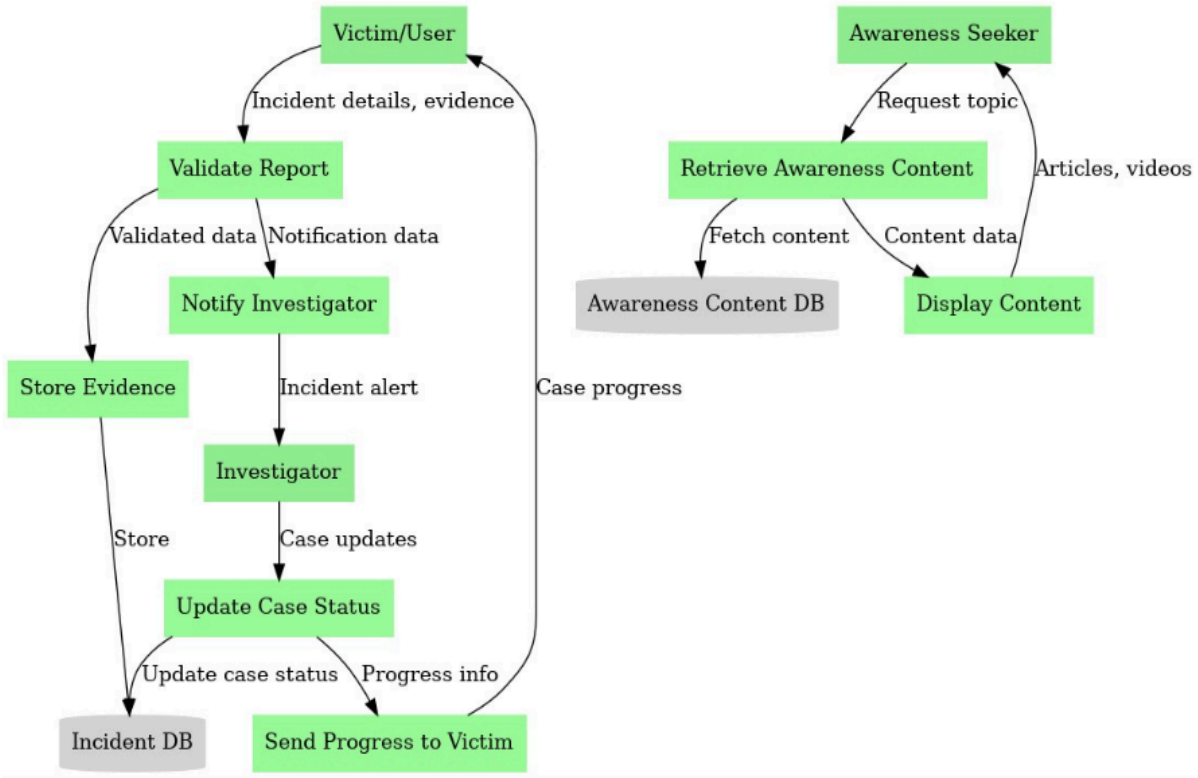
- Deployed the web-based system in a secure environment for demonstration and evaluation.
- Planned for **future enhancements**, including:
    - AI-based crime pattern detection and prediction.
    - Chatbot integration for faster complaint registration.
    - Mobile application development for on-the-go reporting.
    - Blockchain-based evidence verification for data integrity assurance.

4. Entity- Relationship Diagram and Data Flow Diagrams

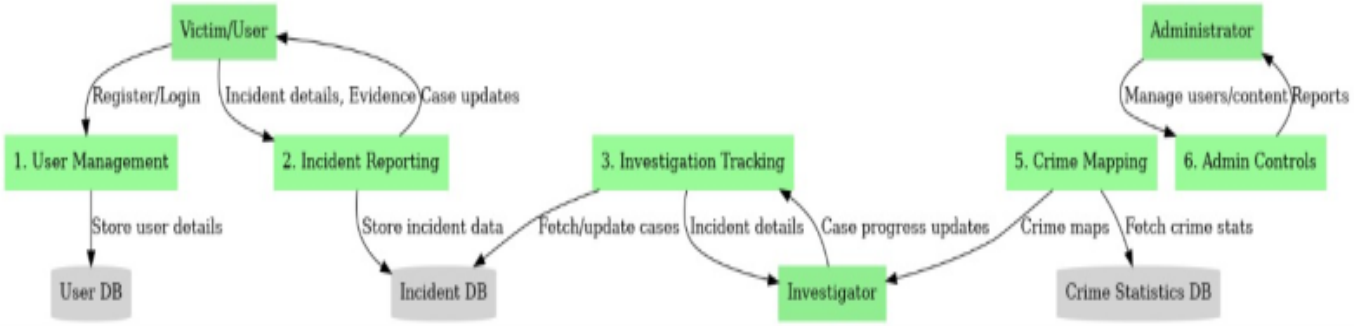
4.1 Entity- Relationship Diagram

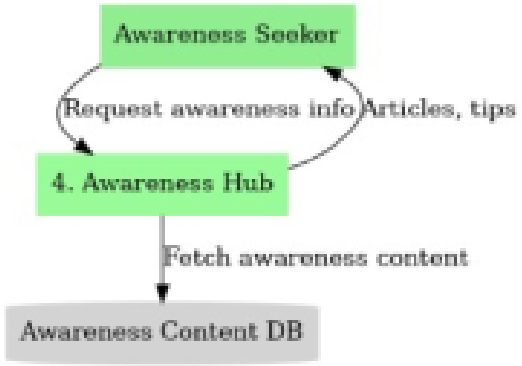


4.2 Data Flow Diagram- Level 0



4.3 Data Flow Diagram- Level 1





5. Tables

EMAIL	FIRST NAME	LAST NAME	ROLE	STAFF STATUS	SUPERUSER STATUS
victim2@example.com	Adam	Brown	Victim/Reporter	✗	✗
admin.panel@system.internal	Admin	Panel	Admin	✗	✗
victim14@example.com	Alexander	Huang	Victim/Reporter	✗	✗
investigator19@example.com	Alison	Kelly	Investigator	✓	✗
investigator29@example.com	Alyssa	Schmidt	Investigator	✓	✗
investigator25@example.com	Amanda	Dillon	Investigator	✓	✗
investigator6@example.com	Amy	Johnson	Investigator	✓	✗
victim7@example.com	Andrew	Vasquez	Victim/Reporter	✗	✗
victim17@example.com	Brett	Glover	Victim/Reporter	✗	✗
admin4@example.com	Chloe	Oconnor	Admin	✓	✗
investigator12@example.com	Christopher	White	Investigator	✓	✗

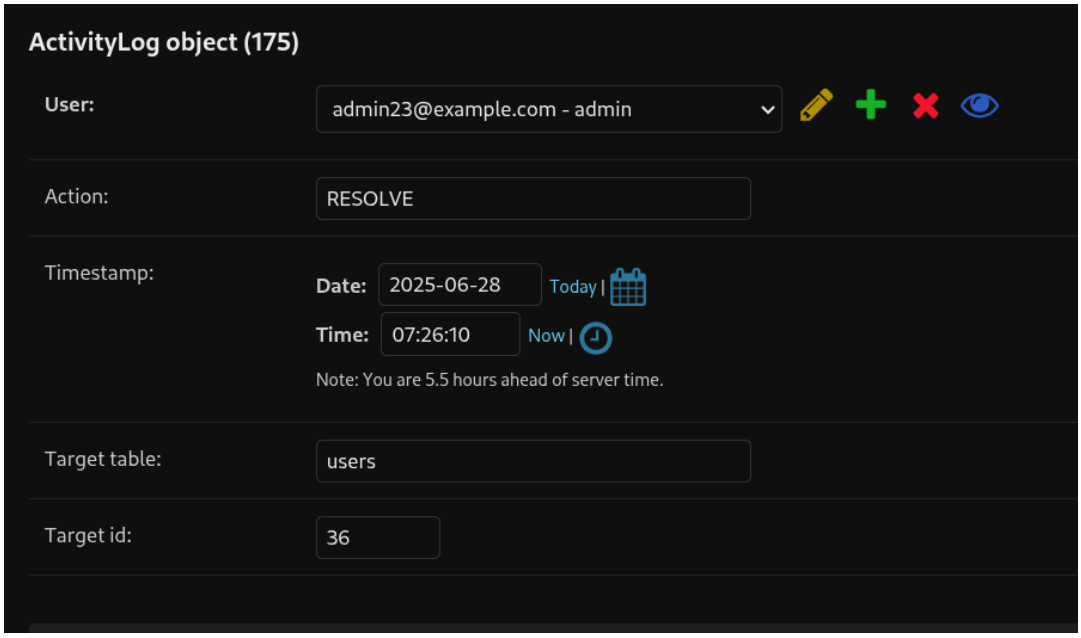
User Management Table: This table manages all individuals accessing and interacting with the system. It enforces **Role-Based Access Control (RBAC)** by storing user emails, names, and their assigned **ROLE** (e.g., **Admin**, **Investigator**, or **Victim/Reporter**). This is foundational to secure operations and the different user workflows

	assigned_at timestamp with time zone	incident_id integer	assigned_to_id bigint	id [PK] bigint	assigned_deadline timestamp with time zone	priority character varying (10)	resolved_at timestamp with time zone
1	2025-05-25 13:12:52.458947+05:30	1	17	1	2025-06-24 13:12:52.458947+05:30	high	[null]
2	2025-05-28 03:12:52.464618+05:30	2	31	2	2025-06-27 03:12:52.464618+05:30	high	2025-06-09 03:12:52.464618+05:30
3	2025-02-20 22:12:52.468146+05:30	3	9	3	2025-03-04 22:12:52.468146+05:30	medium	[null]
4	2025-03-02 02:12:52.471294+05:30	5	4	4	2025-03-26 02:12:52.471294+05:30	high	[null]
5	2025-05-07 00:12:52.472787+05:30	6	5	5	2025-05-23 00:12:52.472787+05:30	high	2025-05-25 00:12:52.472787+05:30
6	2024-11-16 08:12:52.47483+05:30	7	11	6	2024-12-10 08:12:52.47483+05:30	low	2024-12-11 08:12:52.47483+05:30
7	2025-01-26 07:12:52.476685+05:30	8	8	7	2025-02-21 07:12:52.476685+05:30	high	2025-02-09 07:12:52.476685+05:30
8	2024-11-03 03:12:52.477883+05:30	9	27	8	2024-11-21 03:12:52.477883+05:30	high	[null]
9	2025-09-22 16:12:52.486899+05:30	13	4	9	2025-10-06 16:12:52.486899+05:30	medium	2025-10-20 16:12:52.486899+05:30
10	2024-11-10 12:12:52.488811+05:30	14	31	10	2024-11-27 12:12:52.488811+05:30	low	[null]
11	2024-11-07 13:12:52.49094+05:30	15	28	11	2024-11-14 13:12:52.49094+05:30	high	[null]
12	2025-08-27 11:12:52.493165+05:30	16	9	12	2025-09-20 11:12:52.493165+05:30	high	2025-08-30 11:12:52.493165+05:30
13	2025-09-04 13:12:52.496491+05:30	18	17	13	2025-09-23 13:12:52.496491+05:30	medium	2025-09-22 13:12:52.496491+05:30
14	2025-06-19 15:12:52.503306+05:30	21	5	14	2025-07-06 15:12:52.503306+05:30	medium	2025-07-11 15:12:52.503306+05:30
15	2025-08-10 06:12:52.504595+05:30	22	4	15	2025-09-07 06:12:52.504595+05:30	medium	2025-09-09 06:12:52.504595+05:30
16	2025-05-11 09:12:52.50705+05:30	24	28	16	2025-06-07 09:12:52.50705+05:30	medium	[null]
17	2025-09-21 18:12:52.508291+05:30	25	22	17	2025-09-28 18:12:52.508291+05:30	high	2025-10-15 18:12:52.508291+05:30

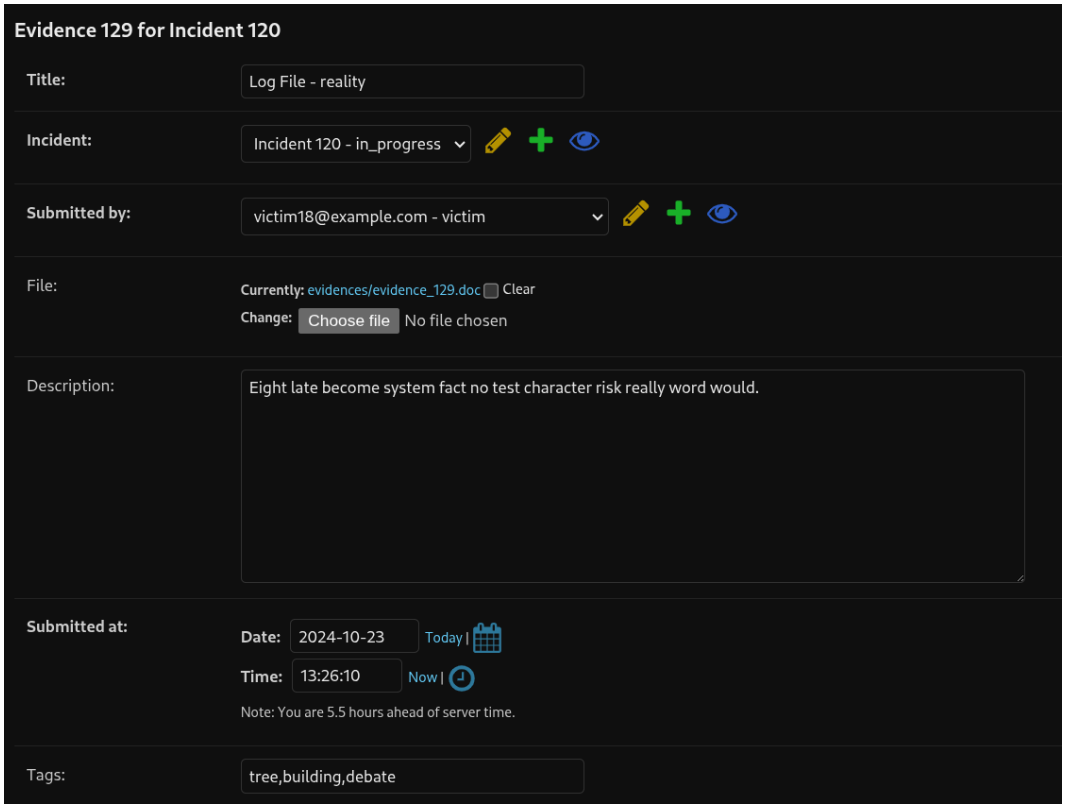
Case Assignment and Tracking Table: This table tracks the lifecycle of every reported cybercrime incident. It links an `incident_id` to the `assigned_to_id` (Investigator), records the **priority**, and monitors key timestamps like `assigned_at`, `assigned_deadline`, and `resolved_at`. This data is crucial for the **Progress Tracking** and **Analytics** features of CIMAS.

id	title	synopsis	content	image	created_at	updated_at	author_id
1	Protecting Yourself from Phishing Attack	Learn how to identify and avoid phishing scams that target your personal informatic	Task old	awareness/image_1.jpg	2025-10-20 00:12:53.069749+05:30	2025-10-20 00:12:53.069766+05:30	28
2	Ransomware Prevention Guide	Best practices to protect your data and systems against ransomware attacks	Who		2025-10-20 00:12:53.073204+05:30	2025-10-20 00:12:53.073215+05:30	3
3	Identity Theft Protection	Comprehensive steps to safeguard your personal information online	Woman	awareness/image_3.jpg	2025-10-20 00:12:53.074587+05:30	2025-10-20 00:12:53.074598+05:30	5
4	Safe Online Shopping Tips	How to shop online securely and avoid e-commerce fraud	Structure	awareness/image_4.jpg	2025-10-20 00:12:53.075917+05:30	2025-10-20 00:12:53.075927+05:30	3
5	Password Security Best Practices	Creating and managing strong passwords to protect your accounts	Able own		2025-10-20 00:12:53.077241+05:30	2025-10-20 00:12:53.077252+05:30	9
6	Social Media Privacy Settings	Protecting your privacy and personal data on social platforms	Place	awareness/image_6.jpg	2025-10-20 00:12:53.079813+05:30	2025-10-20 00:12:53.079841+05:30	20
7	Recognizing Online Scams	Common online scams and warning signs to watch out for	Similar	awareness/image_7.jpg	2025-10-20 00:12:53.082155+05:30	2025-10-20 00:12:53.082166+05:30	19
8	Cyberbullying Awareness	Understanding and combating cyberbullying in digital spaces	Stock	awareness/image_8.jpg	2025-10-20 00:12:53.083829+05:30	2025-10-20 00:12:53.083841+05:30	19
9	Two-Factor Authentication Guide	Enhancing account security with multi-factor authentication	Last new	awareness/image_9.jpg	2025-10-20 00:12:53.085479+05:30	2025-10-20 00:12:53.085491+05:30	30
10	Data Breach Response	What to do when your personal data is compromised	Anything		2025-10-20 00:12:53.0873+05:30	2025-10-20 00:12:53.087317+05:30	22
11	Cryptocurrency Security	Protecting your digital assets from theft and fraud	Down		2025-10-20 00:12:53.089023+05:30	2025-10-20 00:12:53.089037+05:30	12
12	Email Security Tips	How to secure your email and avoid email-based attacks	Few	awareness/image_12.jpg	2025-10-20 00:12:53.090845+05:30	2025-10-20 00:12:53.09086+05:30	9
13	Mobile Device Security	Best practices for securing your smartphone and tablet	For finish	awareness/image_13.jpg	2025-10-20 00:12:53.092829+05:30	2025-10-20 00:12:53.092844+05:30	27
14	Social Engineering Awareness	Understanding manipulation tactics used by cybercriminals	Next risk	awareness/image_14.jpg	2025-10-20 00:12:53.097717+05:30	2025-10-20 00:12:53.097731+05:30	31
15	Safe Banking Online	Protecting your financial information during online transactions	Sing		2025-10-20 00:12:53.099453+05:30	2025-10-20 00:12:53.099467+05:30	22

Awareness Hub Content Table: This table serves as the repository for all educational materials in the **Awareness Hub**. It stores the `title`, `synopsis`, and links to the content for guides on topics like **Ransomware Prevention** and **Social Engineering Awareness**. This table enables the system to provide the public with **cybersecurity best practices.**



Activity Log table: consists of objects that describe an activity log.This log demonstrates the system's ability to track all investigation activities and updates, ensuring **transparency** and meeting the non-functional requirement for **regular audits**



Incidents table: consists of objects that describe an incident.This view confirms the system's capability for **secure evidence submission** by tracking the file name, the submitting user and the **timestamp**, directly supporting the **secure evidence storage** and integrity requirements

6. Code

App.jsx

frontend > src > App.jsx > RoleBasedProfile

```jsx
"use client"

import { BrowserRouter, Routes, Route } from "react-router-dom"
import "./App.css"
import HomeGuest from "./components/Guest/Home"
import DashboardAdmin from "./components/Admin/Dashboard"
import DashboardInvestigator from "./components/Investigator/Dashboard"
import DashboardUser from "./components/Victim/Dashboard"
import Profile from "./components/Profile"
import Signup from "./components/Signup"
import Login from "./components/Login"
import ProtectedRoute from "./components/ProtectedRoute"
import { AuthProvider, useAuth } from "./contexts/AuthContext"
import Navbar from "./components/Navbar"
import Messaging from "./components/Messaging"
import AwarenessList from "./components/AwarenessList"
import AwarenessDetail from "./components/AwarenessDetail"
import AwarenessCreate from "./components/AwarenessCreate"
import PhishingAnalyzer from "./pages/PhishingAnalyzer"

function RoleBasedDashboard() {
  const { user } = useAuth()
  const role = user?.role

  if (role === "admin" || false) {
    return <DashboardAdmin />
  }
  if (role === "investigator" || false) return <DashboardInvestigator />
  else if (role === "user" || role === "victim" || false) {
    return <DashboardUser />
  } else {
    return <HomeGuest />
  }
}
```

```jsx
function RoleBasedProfile() {
  const { user } = useAuth()
  const role = user?.role

  if (role === "admin") {
    return <Profile />
  } else if (role === "user" || role === "victim") {
    return <Profile />
  } else if (role === "investigator") {
    return <Profile />
  }
}

function App() {
  return (
    <AuthProvider>
      <BrowserRouter>
        <Navbar />
        <Routes>
          <Route path="/" element={<RoleBasedDashboard />} />
          <Route path="/messages" element={<Messaging />} />
          <Route
            path="/profile"
            element={
              <ProtectedRoute>
                <RoleBasedProfile />
              </ProtectedRoute>
            }
          />
          <Route path="/signup" element={<Signup />} />
          <Route path="/login" element={<Login />} />
          <Route path="/awareness" element={<AwarenessList />} />
          <Route path="/awareness/:id" element={<AwarenessDetail />} />
          <Route
            path="/awareness/create"
            element={
              <ProtectedRoute>
                <AwarenessCreate />
              </ProtectedRoute>
```

```
                }
              />
              <Route
                path="/phishing"
                element={
                  <ProtectedRoute>
                    <PhishingAnalyzer />
                  </ProtectedRoute>
                }
              />
            </Routes>
          </BrowserRouter>
        </AuthProvider>
      )
    }

    export default App
```

Activity logs>views.py

backend > CIMAS > activity_logs > views.py > ...

```python
from rest_framework.decorators import api_view, permission_classes    Import "rest_framework.decorators" could not be resolved
from rest_framework.permissions import IsAuthenticated    Import "rest_framework.permissions" could not be resolved
from rest_framework.response import Response    Import "rest_framework.response" could not be resolved
from django.shortcuts import get_object_or_404    Import "django.shortcuts" could not be resolved from source
from .models import ActivityLog

def get_user_role(user):
    if getattr(user, "is_superuser", False):
        return "admin"
    return getattr(user, "role", "user")

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_logs(request):
    role = get_user_role(request.user)

    if role == "admin":
        logs = ActivityLog.objects.all()
    elif role == "investigator":
        logs = ActivityLog.objects.filter(target_table="incidents")
    else:
        logs = ActivityLog.objects.filter(user=request.user)

    data = [
        {
            "id": log.id,
            "user": log.user.email if hasattr(log.user, "email") else log.user.username,
            "action": log.action,
            "timestamp": log.timestamp,
            "target_table": log.target_table,
            "target_id": log.target_id,
        }
        for log in logs
    ]
    return Response(data)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_log(request, id):
    log = get_object_or_404(ActivityLog, id=id)
```

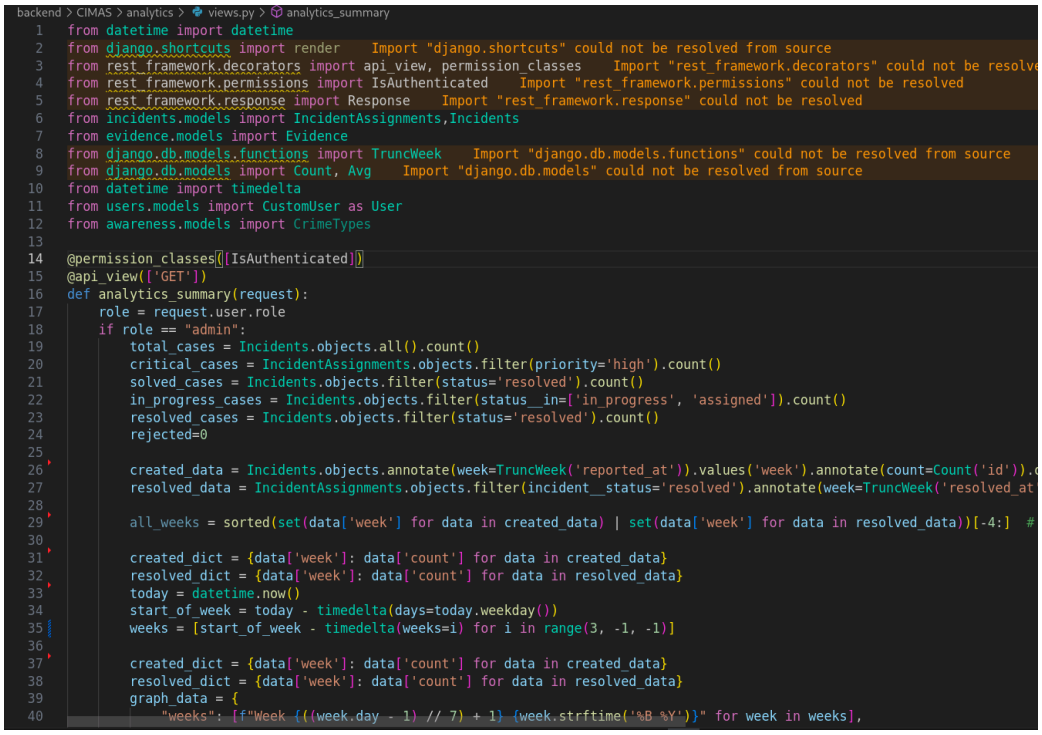
```python
    role = get_user_role(request.user)

    if role == "admin" or log.user == request.user:
        return Response({
            "id": log.id,
            "user": log.user.email if hasattr(log.user, "email") else log.user.username,
            "action": log.action,
            "timestamp": log.timestamp,
            "target_table": log.target_table,
            "target_id": log.target_id,
        })
    return Response({"error": "You do not have permission to view this log."}, status=403)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_user_logs(request, userId):
    role = get_user_role(request.user)
    if role in ["admin", "investigator"] or request.user.id == userId:
        logs = ActivityLog.objects.filter(user_id=userId)
        return Response(list(logs.values()))
    return Response({"error": "Access denied."}, status=403)

@api_view(['GET'])
@permission_classes([IsAuthenticated])
def get_incident_logs(request, incidentId):
    role = get_user_role(request.user)
    if role in ["admin", "investigator"]:
        logs = ActivityLog.objects.filter(target_id=incidentId, target_table="incidents")
        return Response(list(logs.values()))
    return Response({"error": "Access denied."}, status=403)
```

Activity logs>urls.py

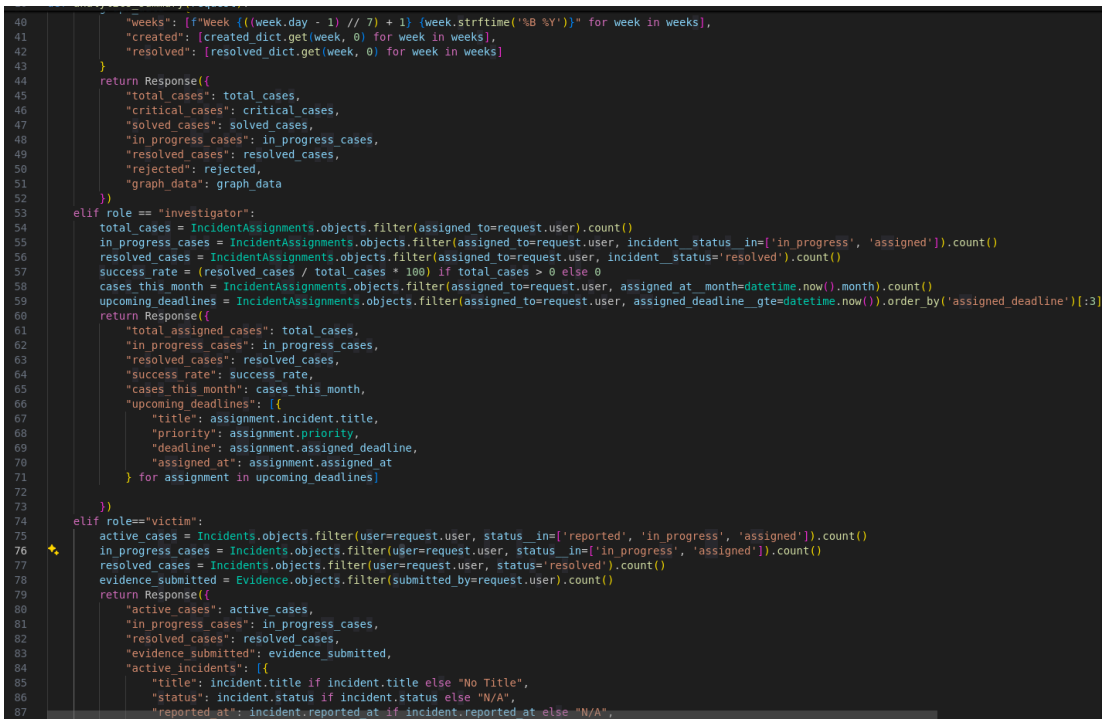
backend > CIMAS > activity_logs > urls.py > ...

```python
from django.urls import path    Import "django.urls" could not be resolved from source
from . import views

urlpatterns = [
    path('api/logs', views.get_logs, name='get_logs'),
    path('api/logs/<int:id>', views.get_log, name='get_log'),
    path('api/logs/user/<int:userId>', views.get_user_logs, name='get_user_logs'),
    path('api/logs/incidents/<int:incidentId>', views.get_incident_logs, name='get_incident_logs'),
]
```

Analytics>views.py



```
backend > CIMAS > analytics > views.py > analytics_summary
from datetime import datetime
from django.shortcuts import render    Import "django.shortcuts" could not be resolved from source
from rest_framework.decorators import api_view, permission_classes    Import "rest_framework.decorators" could not be resolve
from rest_framework.permissions import IsAuthenticated    Import "rest_framework.permissions" could not be resolved
from rest_framework.response import Response    Import "rest_framework.response" could not be resolved
from incidents.models import IncidentAssignments,Incidents
from evidence.models import Evidence
from django.db.models.functions import TruncWeek    Import "django.db.models.functions" could not be resolved from source
from django.db.models import Count, Avg    Import "django.db.models" could not be resolved from source
from datetime import timedelta
from users.models import CustomUser as User
from awareness.models import CrimeTypes

@permission_classes([IsAuthenticated])
@api_view(['GET'])
def analytics_summary(request):
    role = request.user.role
    if role == "admin":
        total_cases = Incidents.objects.all().count()
        critical_cases = IncidentAssignments.objects.filter(priority='high').count()
        solved_cases = Incidents.objects.filter(status='resolved').count()
        in_progress_cases = Incidents.objects.filter(status__in=['in_progress', 'assigned']).count()
        resolved_cases = Incidents.objects.filter(status='resolved').count()
        rejected=0

        created_data = Incidents.objects.annotate(week=TruncWeek('reported_at')).values('week').annotate(count=Count('id')).
        resolved_data = IncidentAssignments.objects.filter(incident__status='resolved').annotate(week=TruncWeek('resolved_at

        all_weeks = sorted(set(data['week'] for data in created_data) | set(data['week'] for data in resolved_data))[-4:]  #

        created_dict = {data['week']: data['count'] for data in created_data}
        resolved_dict = {data['week']: data['count'] for data in resolved_data}
        today = datetime.now()
        start_of_week = today - timedelta(days=today.weekday())
        weeks = [start_of_week - timedelta(weeks=i) for i in range(3, -1, -1)]

        created_dict = {data['week']: data['count'] for data in created_data}
        resolved_dict = {data['week']: data['count'] for data in resolved_data}
        graph_data = {
            "weeks": [f"Week {((week.day - 1) // 7) + 1} {week.strftime('%B %Y')}" for week in weeks],
```



```
            "weeks": [f"Week {((week.day - 1) // 7) + 1} {week.strftime('%B %Y')}" for week in weeks],
            "created": [created_dict.get(week, 0) for week in weeks],
            "resolved": [resolved_dict.get(week, 0) for week in weeks]
        }
        return Response({
            "total_cases": total_cases,
            "critical_cases": critical_cases,
            "solved_cases": solved_cases,
            "in_progress_cases": in_progress_cases,
            "resolved_cases": resolved_cases,
            "rejected": rejected,
            "graph_data": graph_data
        })
    elif role == "investigator":
        total_cases = IncidentAssignments.objects.filter(assigned_to=request.user).count()
        in_progress_cases = IncidentAssignments.objects.filter(assigned_to=request.user, incident__status__in=['in_progress', 'assigned']).count()
        resolved_cases = IncidentAssignments.objects.filter(assigned_to=request.user, incident__status='resolved').count()
        success_rate = (resolved_cases / total_cases * 100) if total_cases > 0 else 0
        cases_this_month = IncidentAssignments.objects.filter(assigned_to=request.user, assigned_at__month=datetime.now().month).count()
        upcoming_deadlines = IncidentAssignments.objects.filter(assigned_to=request.user, assigned_deadline__gte=datetime.now()).order_by('assigned_deadline')[:3]
        return Response({
            "total_assigned_cases": total_cases,
            "in_progress_cases": in_progress_cases,
            "resolved_cases": resolved_cases,
            "success_rate": success_rate,
            "cases_this_month": cases_this_month,
            "upcoming_deadlines": [{
                "title": assignment.incident.title,
                "priority": assignment.priority,
                "deadline": assignment.assigned_deadline,
                "assigned_at": assignment.assigned_at
            } for assignment in upcoming_deadlines]

        })
    elif role=="victim":
        active_cases = Incidents.objects.filter(user=request.user, status__in=['reported', 'in_progress', 'assigned']).count()
        in_progress_cases = Incidents.objects.filter(user=request.user, status__in=['in_progress', 'assigned']).count()
        resolved_cases = Incidents.objects.filter(user=request.user, status='resolved').count()
        evidence_submitted = Evidence.objects.filter(submitted_by=request.user).count()
        return Response({
            "active_cases": active_cases,
            "in_progress_cases": in_progress_cases,
            "resolved_cases": resolved_cases,
            "evidence_submitted": evidence_submitted,
            "active_incidents": [{
                "title": incident.title if incident.title else "No Title",
                "status": incident.status if incident.status else "N/A",
                "reported_at": incident.reported_at if incident.reported_at else "N/A",
```

```
                "reported_at": incident.reported_at if incident.reported_at else "N/A",
                "assigned_investigator": incident.assignment.assigned_to.first_name + " " + incident.assignment.assigned_to.last_name if hasattr(incident, 'assign
                "priority": incident.assignment.priority if hasattr(incident, 'assignment') else "N/A",
                "progress": "50%" if incident.status == "in_progress" else "75%" if incident.status == "assigned" else "0%"
            } for incident in Incidents.objects.filter(user=request.user, status__in=['reported', 'in_progress', 'assigned']).order_by('-reported_at')[:3]]
        })
    return Response({"error": "You do not have permission to view this."}, status=403)

@permission_classes([IsAuthenticated])
@api_view(['GET'])
def analytics_detailed(request):
    if request.user.role != "admin":
        return Response({"error": "You do not have permission to view this."}, status=403)
    case_solved = Incidents.objects.filter(status='resolved').count()
    new_users = User.objects.filter(date_joined__month=datetime.now().month).count()
    avg_resolution_time = IncidentAssignments.objects.filter(incident__status='resolved').annotate(
        resolution_time=Count('resolved_at') - Count('assigned_at')
    ).aggregate(Avg('resolution_time'))['resolution_time__avg']

    last_month = datetime.now().month - 1 if datetime.now().month > 1 else 12
    last_month_year = datetime.now().year if datetime.now().month > 1 else datetime.now().year - 1

    last_month_case_solved = Incidents.objects.filter(status='resolved', reported_at__month=last_month, reported_at__year=last_month_year).count()
    last_month_new_users = User.objects.filter(date_joined__month=last_month, date_joined__year=last_month_year).count()
    last_month_avg_resolution_time = IncidentAssignments.objects.filter(
        incident__status='resolved',
        resolved_at__month=last_month,
        resolved_at__year=last_month_year
    ).annotate(
        resolution_time=Count('resolved_at') - Count('assigned_at')
    ).aggregate(Avg('resolution_time'))['resolution_time__avg']

    case_solved_change = case_solved - last_month_case_solved
    new_users_change = new_users - last_month_new_users
    avg_resolution_time_change = avg_resolution_time - last_month_avg_resolution_time if avg_resolution_time and last_month_avg_resolution_time else 0

    total_cases = Incidents.objects.count()
    efficincy = (case_solved / total_cases * 100) if total_cases > 0 else 0

    last_month = datetime.now().month - 1 if datetime.now().month > 1 else 12
    last_month_year = datetime.now().year if datetime.now().month > 1 else datetime.now().year - 1
    last_month_cases_solved = Incidents.objects.filter(status='resolved', reported_at__month=last_month, reported_at__year=last_month_year).count()
    last_month_total_cases = Incidents.objects.filter(reported_at__month=last_month, reported_at__year=last_month_year).count()
    last_month_efficiency = (last_month_cases_solved / last_month_total_cases * 100) if last_month_total_cases > 0 else 0

    efficincy_change = efficincy - last_month_efficiency
```

```
    created_data = Incidents.objects.annotate(month=TruncWeek('reported_at', kind='month')).values('month').annotate(count=Count('id')).order_by('month')
    resolved_data = IncidentAssignments.objects.filter(incident__status='resolved').annotate(month=TruncWeek('resolved_at', kind='month')).values('month').annotat


    all_months = sorted(set(data['month'] for data in created_data) | set(data['month'] for data in resolved_data))[-6:]  # Limit to the last 6 months


    created_dict = {data['month']: data['count'] for data in created_data}
    resolved_dict = {data['month']: data['count'] for data in resolved_data}
    graph_data = {
        "months": [month.strftime('%B %Y') for month in all_months],
        "created": [created_dict.get(month, 0) for month in all_months],
        "resolved": [resolved_dict.get(month, 0) for month in all_months]
    }

    category_data = Incidents.objects.values('crime_type__crime_type_name').annotate(count=Count('id')).order_by('-count')
    category_dict = {data['crime_type__crime_type_name']: data['count'] for data in category_data}
    category_graph_data = {
        "categories": list(category_dict.keys()),
        "counts": list(category_dict.values())
    }

    time_taken_data = IncidentAssignments.objects.filter(incident__status__in=['in_progress', 'assigned']).annotate(
        time_taken=Count('assigned_at') - Count('incident__reported_at')
    ).values('priority', 'time_taken').annotate(count=Count('id')).order_by('priority', 'time_taken')


    priority_dict = {}
    for data in time_taken_data:
        priority = data['priority']
        time_taken = data['time_taken']
        count = data['count']
        if priority not in priority_dict:
            priority_dict[priority] = {}
        priority_dict[priority][time_taken] = count


    time_taken_graph_data = {
        "priorities": list(priority_dict.keys()),
        "time_taken": {
            priority: list(priority_dict[priority].keys()) for priority in priority_dict
        },
        "counts": {
            priority: list(priority_dict[priority].values()) for priority in priority_dict
        }
    }

    hotspot_data = Incidents.objects.values('location__city').annotate(count=Count('id')).order_by('-count')
```

```
        }
    }

    hotspot_data = Incidents.objects.values('location__city').annotate(count=Count('id')).order_by('-count')
    hotspot_graph_data = {
        "cities": [data['location__city'] for data in hotspot_data],
        "counts": [data['count'] for data in hotspot_data]
    }

    return Response({
        "case_solved": case_solved,
        "case_solved_change": case_solved_change,
        "new_users": new_users,
        "new_users_change": new_users_change,
        "avg_resolution_time": avg_resolution_time,
        "avg_resolution_time_change": avg_resolution_time_change,
        "efficiency": efficincy,
        "efficiency_change": efficincy_change,
        "incident_trend_graph": graph_data,
        "category_graph": category_graph_data,
        "time_taken_graph": time_taken_graph_data,
        "hotspot_graph": hotspot_graph_data
    })

def analytics_trends(request):
    pass

def analytics_hotspots(request):
    pass

def analytics_categories(request):
    pass
```

Analytics>urls.py

backend > CIMAS > analytics > urls.py > ...

```
1  from django.urls import path    Import "django.urls" could not be resolved from source
2  from . import views
3
4  urlpatterns = [
5      path('api/analytics/summary', views.analytics_summary, name='analytics_summary'),
6      path('api/analytics/detailed', views.analytics_detailed, name='analytics_detailed'),
7      path('api/analytics/trends', views.analytics_trends, name='analytics_trends'),
8      path('api/analytics/hotspots', views.analytics_hotspots, name='analytics_hotspots'),
9      path('api/analytics/categories', views.analytics_categories, name='analytics_categories'),
10 ]
11
```

Awareness>views.py

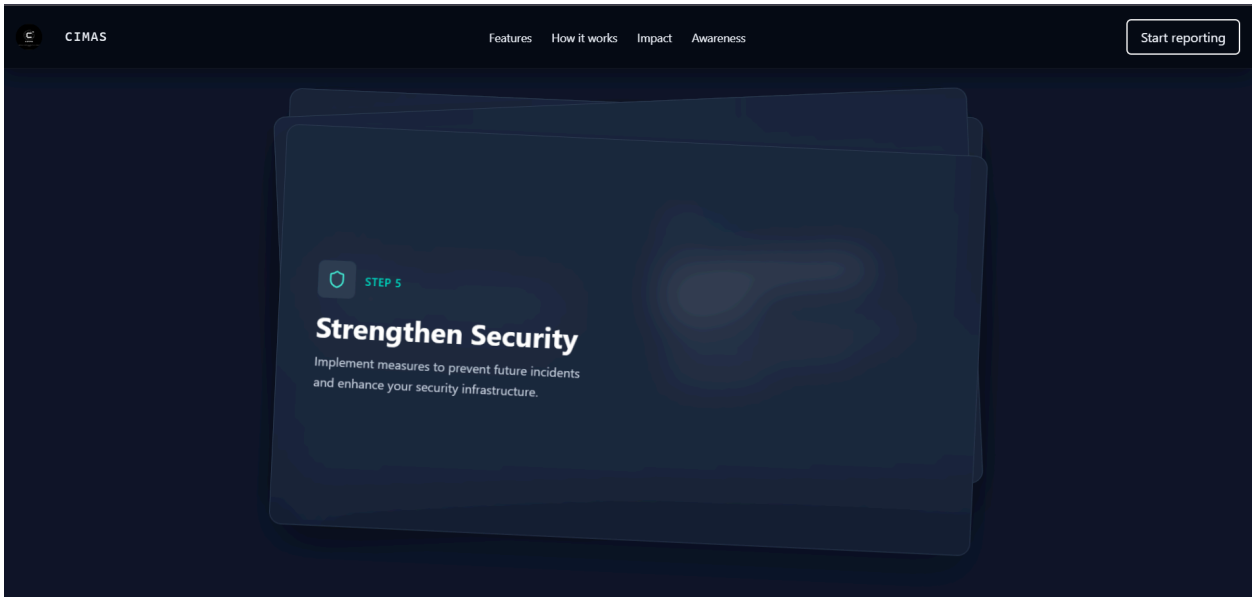
```python
class FlairListView(generics.ListAPIView):
    queryset = Flair.objects.all().order_by("name")
    serializer_class = FlairSerializer
    permission_classes = [permissions.AllowAny]


class AwarenessResourceListCreateView(generics.ListCreateAPIView):
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]

    def get_serializer_class(self):
        if self.request.method == "GET":
            return AwarenessListSerializer
        return AwarenessDetailSerializer

    def get_queryset(self):
        queryset = AwarenessResource.objects.all().order_by("-created_at")

        flair_param = self.request.query_params.get("flair")

        if flair_param:
            queryset = queryset.filter(
                flair__name__iexact=flair_param
            ) | queryset.filter(flair_id=flair_param)

        return queryset.distinct()

    def perform_create(self, serializer):
        serializer.save(author=self.request.user)


class AwarenessResourceDetailView(generics.RetrieveUpdateDestroyAPIView):
    queryset = AwarenessResource.objects.all()
    serializer_class = AwarenessDetailSerializer
    permission_classes = [permissions.IsAuthenticatedOrReadOnly]

    def perform_update(self, serializer):
        if self.request.user == serializer.instance.author or self.request.user.is_staff:
            serializer.save()
        else:
            raise PermissionDenied("You are not allowed to update this resource.")

    def perform_destroy(self, instance):
        if self.request.user == instance.author or self.request.user.is_staff:
            instance.delete()
        else:
            raise PermissionDenied("You are not allowed to delete this resource.")
```

Awareness>urls.py

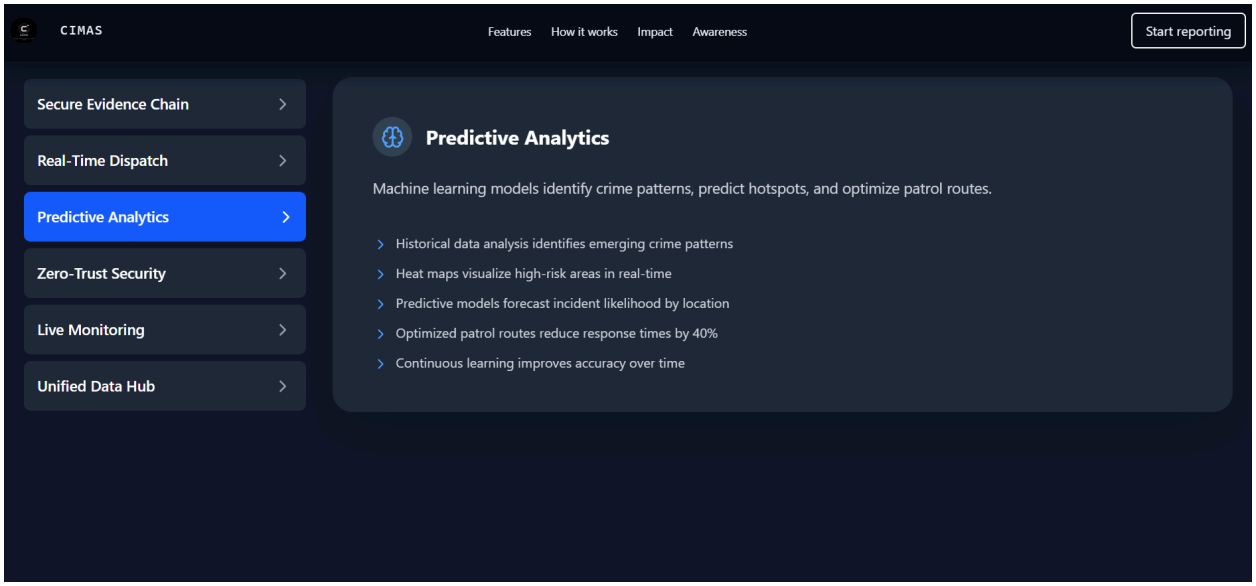
```python
from django.urls import path    Import "django.urls" could not be resolved from source
from .views import (
    AwarenessResourceListCreateView,
    AwarenessResourceDetailView,
    FlairListView,
)

urlpatterns = [

    path("api/awareness/resources/", AwarenessResourceListCreateView.as_view(), name="resource_list_create"),


    path("api/awareness/resources/<int:pk>/", AwarenessResourceDetailView.as_view(), name="resource_detail"),


    path("api/awareness/flairs/", FlairListView.as_view(), name="flair_list"),
]
```

7. Website Screenshots

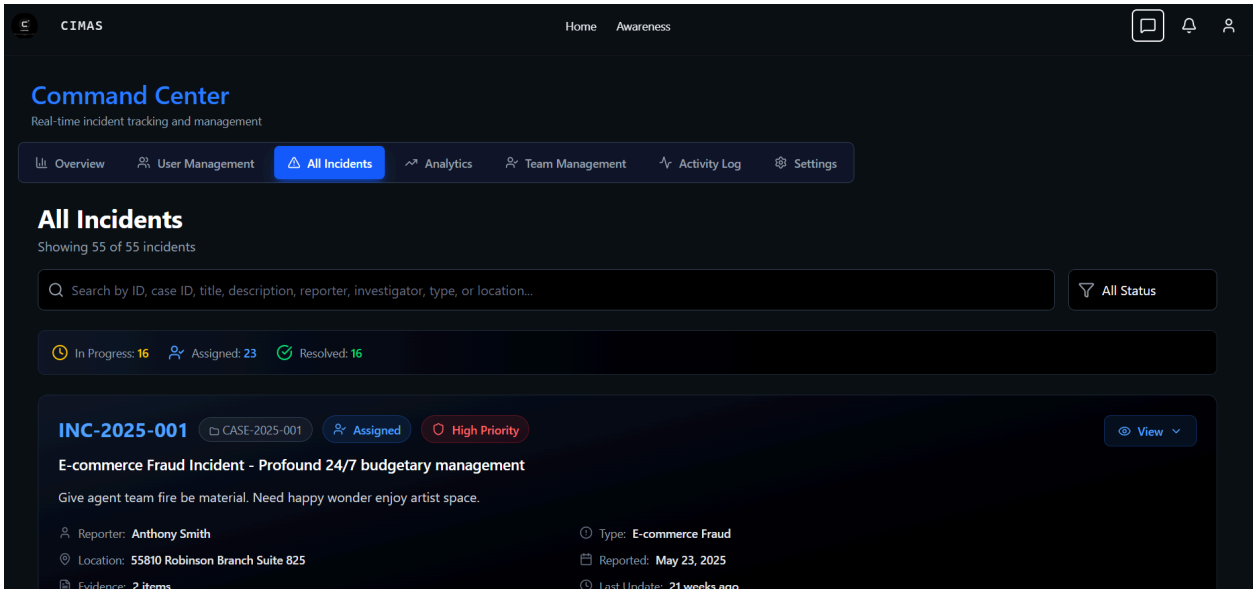
CIMAS Landing Page: The main entry point, introducing the **Intelligent Crime Incident Management** system, highlighting its real-time reporting, AI-powered analytics, and secure evidence management.



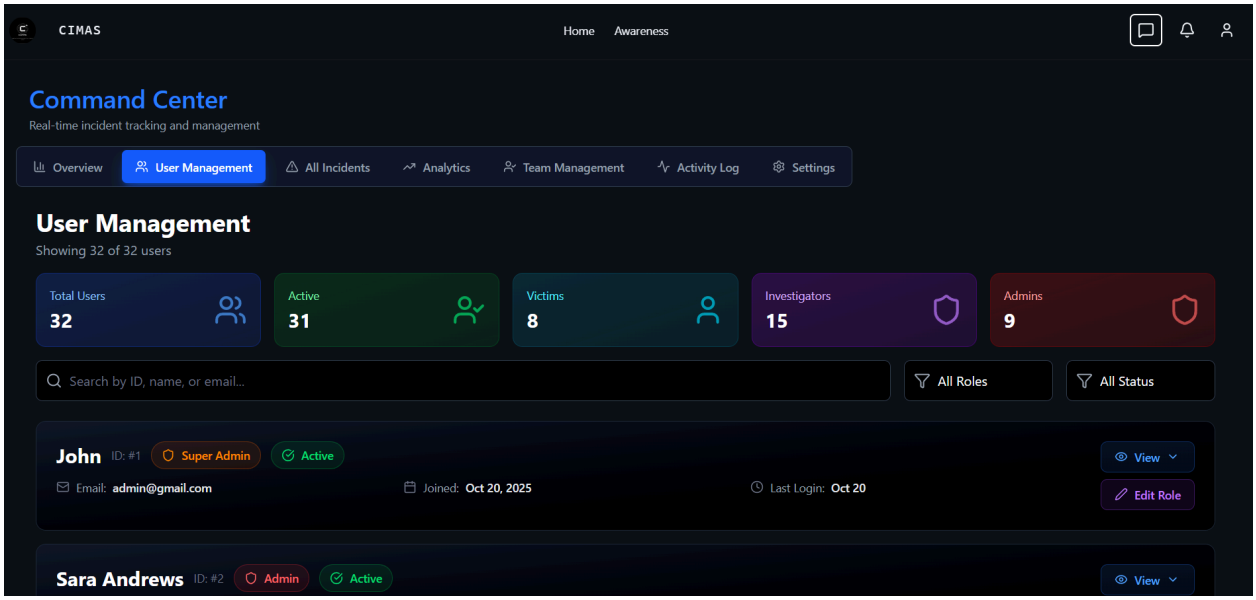
Part of the Landing page, instructs the user the steps to follow



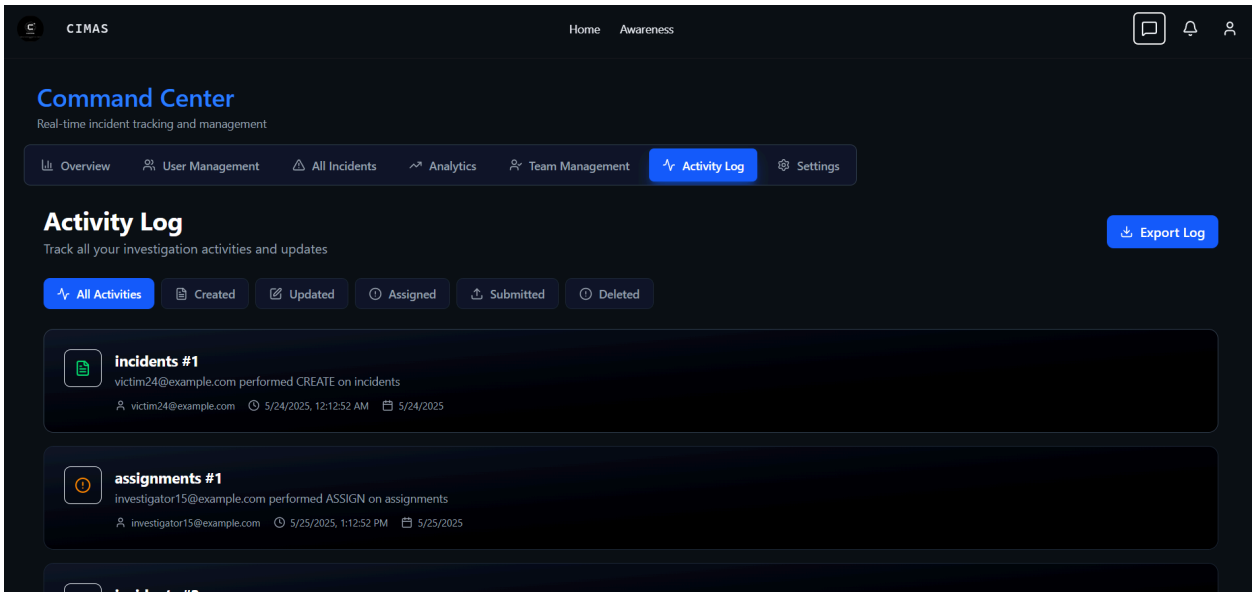
Part of the landing page, describes the features of the system



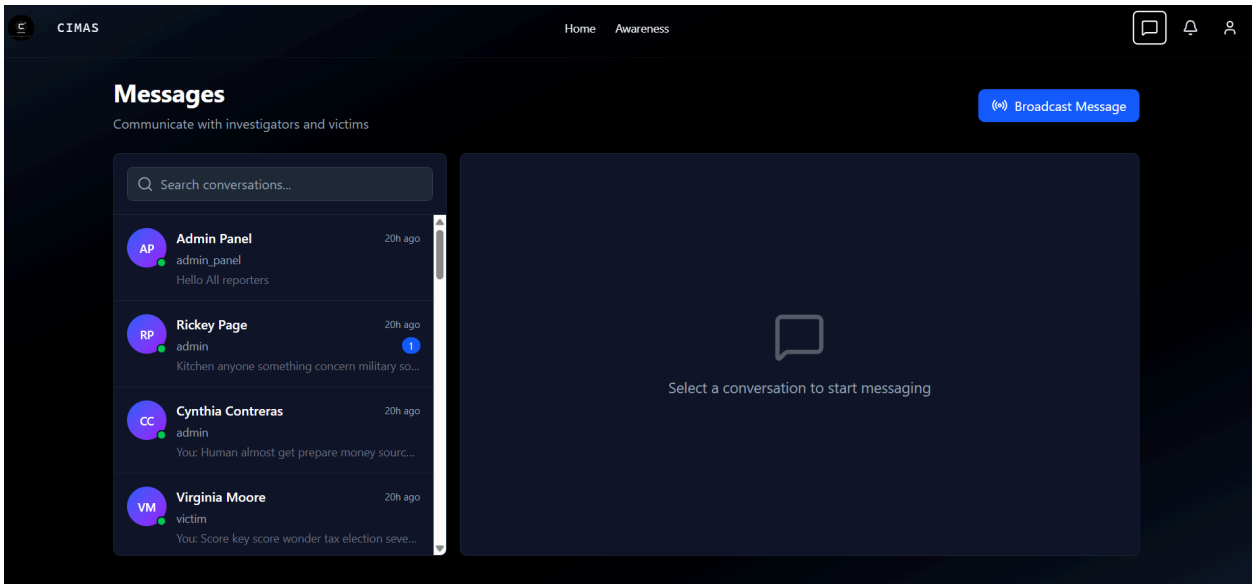
Command Center - All Incidents View. The investigator/administrator interface for managing incidents, demonstrating features like **case tracking** (In Progress, Assigned, Resolved), **priority tagging** (High Priority), and detailed **case status management**



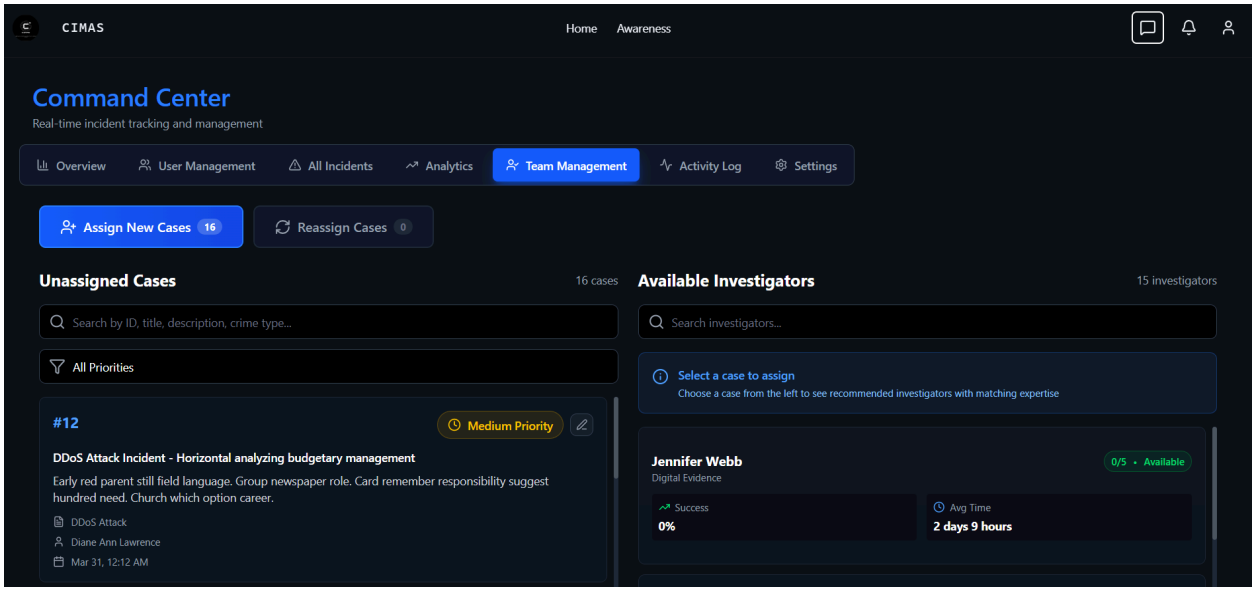
Command Center - User Management. The Administrator's view for managing system users, which tracks **Victims**, **Investigators**, and **Admins**.



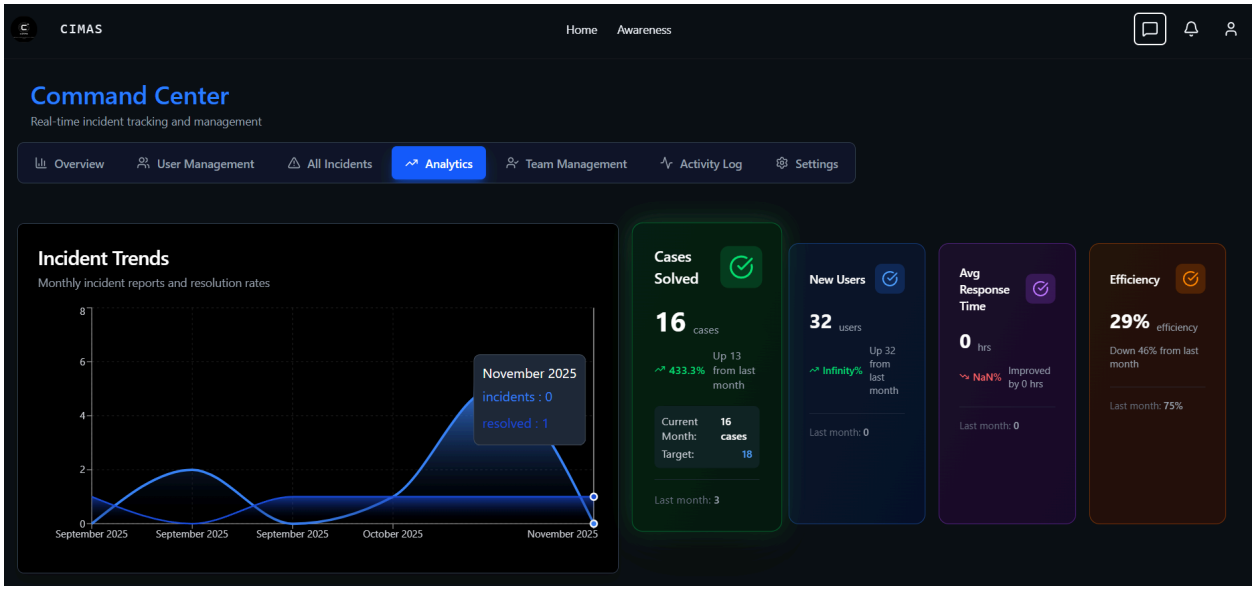
Command Center - Activity Log. An essential component for system reliability and auditability, this log tracks all investigation activities and updates, including **CREATE incidents** and **ASSIGN** actions, supporting the requirement for **secure and transparent case management**.



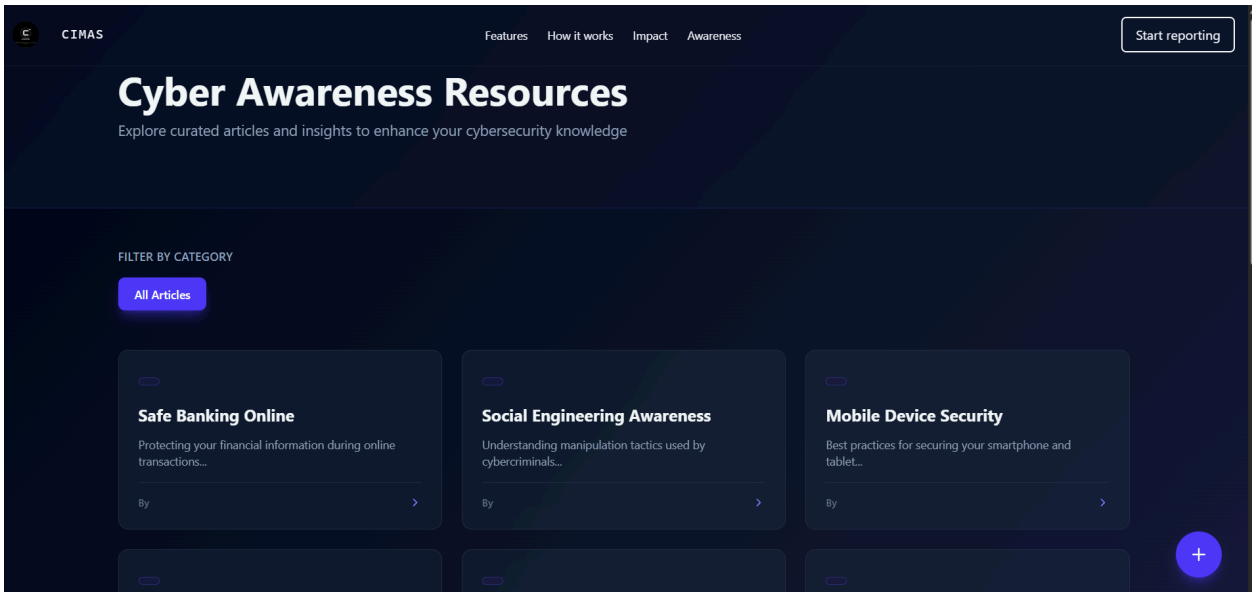
Command Center - Internal Messaging. The messaging feature for direct communication between **Admins**, **Investigators**, and **Victims**, supporting efficient **case management** and facilitating timely notifications



Command Center - Team Management and Case Assignment. The **Administrator's** interface for **Case Assignment** showing **Unassigned Cases** and **Available Investigators**. This view enables efficient **investigator assignment** based on case priority and investigator expertise, supporting timely response.

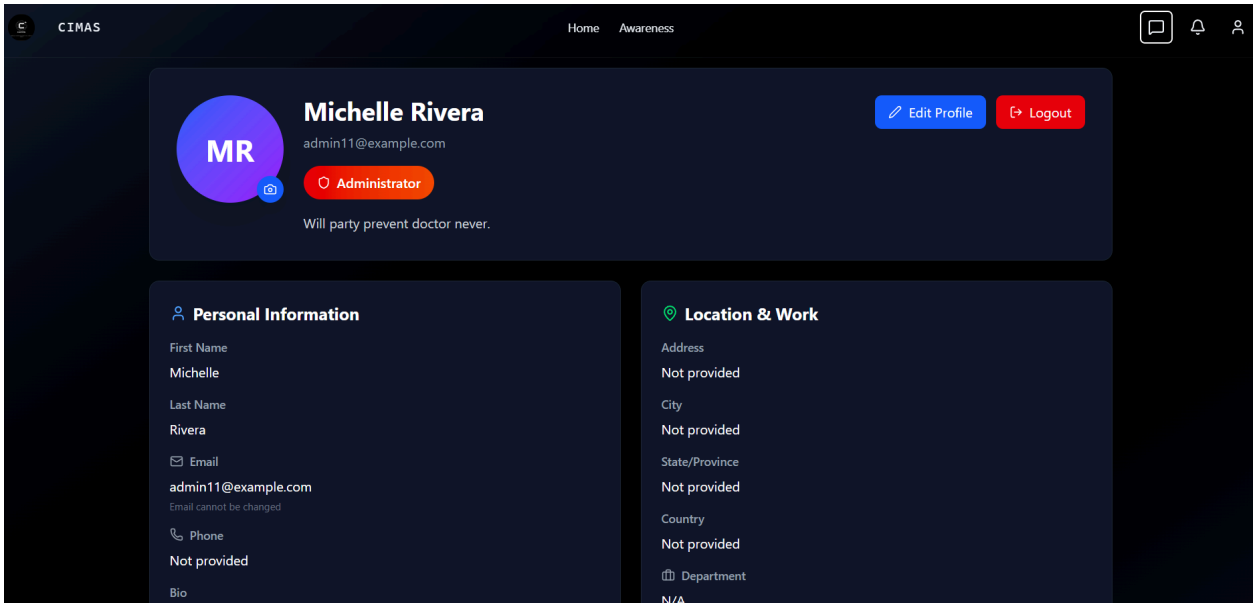


Command Center - Analytics Dashboard. The interface for viewing **Incident Trends** and key performance metrics (KPIs) like **Cases Solved**, **New Users**, and **Efficiency**.



Awareness Hub - Resource Categories. The landing page of the **Awareness Hub**, offering **curated articles** on Cyber Security to guide the public on safe online practices.

View of a solitary article



User Profile View. An example of the **Administrator** user profile, illustrating the system's interface for managing user details, which is integral to maintaining **Role-Based Access Control (RBAC)** and secure operations

8. Conclusion

The **Cybercrime Incident Management & Awareness System (CIMAS)** successfully demonstrates how technology can be leveraged to streamline cybercrime reporting, enhance investigation efficiency, and promote cybersecurity awareness among the public. The project bridges the communication gap between victims, investigators, and administrators through a secure and user-friendly web-based platform.

By integrating features such as **incident reporting with evidence upload**, **role-based dashboards**, **crime mapping**, and an **awareness hub**, the system provides a comprehensive approach to tackling cybercrime from both preventive and investigative perspectives.

The backend, powered by **Django REST Framework**, ensures data integrity, authentication, and secure evidence handling, while the **React frontend** delivers an accessible and responsive user interface. Furthermore, the **MySQL database** efficiently manages structured data, enabling smooth retrieval and analysis.

Key takeaways from this project include:

- Demonstrating the practical application of **web technologies** for digital crime management.
- Enhancing **transparency and accountability** in cybercrime investigation workflows.
- Empowering users through **awareness programs and educational resources** on safe online practices.
- Laying the foundation for **future advancements**, such as AI-based threat prediction and blockchain-based evidence verification.

In conclusion, CIMAS is not just a reporting tool but a **complete ecosystem for cyber safety and digital empowerment**. With further development and integration into law enforcement networks, this system has the potential to become a vital framework for combating cybercrime in the modern digital landscape.

9. References

Ministry of Home Affairs, Government of India. (2024). *National Cybercrime Reporting Portal.* Retrieved from https://cybercrime.gov.in

Django Software Foundation. (2024). *Django Documentation – The Web Framework for Perfectionists with Deadlines.* Retrieved from https://docs.djangoproject.com

Meta Platforms Inc. (2024). *React Official Documentation.* Retrieved from https://react.dev

PostgreSQL Global Development Group. (2024). *PostgreSQL Documentation.* Retrieved from https://www.postgresql.org/docs/

Open Web Application Security Project (OWASP). (2024). *Web Application Security Guidelines and Best Practices.* Retrieved from https://owasp.org

Singh, R., & Mehta, P. (2023). *Design and Implementation of a Web-Based Cybercrime Reporting System. International Journal of Emerging Technologies in Engineering Research (IJETER),* 11(6), 45–52.