UNIVERSITY OF NAIROBI

FACULTY OF ENGINEERING

DEPARTMENT OF GEOSPATIAL AND SPACE TECHNOLOGY

WEB MAPPING OF IMPROVISED EXPLOSIVE DEVICE (IED) INCIDENTS IN KENYA

HESBON MAKAMBI OCHEGO

F56/37688/2020

A Project Report submitted to the Department of Geospatial and Space Technology in partial fulfillment of the requirements for the award of the Degree of Master of Science in Geographic Information Systems of the University of Nairobi.

AUGUST 2022

DECLARATION

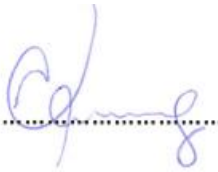
I, **Hesbon Makambi Ochego**, hereby declare that this project is my original work. To the best of my knowledge, the work presented here has not been presented for a degree in any other Institution of Higher Learning.

DATE…. **04 August 2022**

Hesbon Makambi Ochego

This project has been submitted for examination with my approval as university supervisor.

SUPERVISOR:

Dr. C. M. MWANGE: SIGNATURE  DATE…….**04th August, 2022**

DEDICATION

I dedicate this work to God for His grace, and my family for the continued support and prayers.

ACKNOWLEDGEMENT

I acknowledge my supervisor, Dr. C. M. Mwange for his support throughout this project. His guidance was invaluable. I also appreciate the support of Mr. Japheth Gichana for his input on the development of the REST API. Finally, I acknowledge the support of UNMAS through Maj. (Rtd) Raymond Kemei, and ACLED for providing essential IED incidents datasets used in this project.

ABSTRACT

This project reviewed existing Improvised Explosive Device (IED) Incidents data from different sources, cleaned it up and used the data to develop a reliable IED incidents geodatabase for Kenya. This database was then used to develop an interactive web-map that renders the incident data objects to users through the web. The web deployment enhances access of the data by stakeholders, and also promotes corroborative building, and enhancing of the database by allowing users to contribute data and also point out observations upon exploring the data hosted and served through the web Mapping System.

During the project conception, user needs assessment was conducted, and the user requirements used to design the geodatabase, and inform the elements and functionalities of the envisaged IED Web Mapping System (MapIED). Database Entity Relation Diagrams (ERD) were generated using LucidChart software, and the webpage Graphic User Interface (GUI) prototype developed using Figma software. This was used to get user feedback and further enhance the design.

The MapIED System was then developed using Python Django Application to develop the backend of the system and the REST Application Programming Interface (API). On the other hand, Angular application (Open source software by Google Inc.) was used to develop the frontend of the system. The result was a robust Web Mapping System, that in addition to similar systems (e.g. Armed Conflict Location & Events Data - ACLED) is more specific (to IED mapping) and detailed, more corroborative (through user incident reporting functionality), and captures other valuable incident information such as media/incident images and technical reports (by Explosive Ordinance Teams) which are not available in other such mapping systems.

This is a valuable system that upon deployment, has the potential to provide stakeholders with valuable IED incident information, including trends, emplacement patterns, and device constructs. This information is critical is assessing the adversary's Tactics, Techniques and Procedures (TTPs) and responding with appropriate Counter-IED measures to address the IED threat in Kenya.

It is recommended that at the onset, the system be deployed within a closed network, within the existing Multi-Agency Security framework and then progressively expanded to take in other stakeholders. This is to allow for development of appropriate access security protocols for its deployment outside the mainstream security sector, so as to safeguard the sensitive IED data.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF PROGRAM CODES

ABBREVIATIONS AND ACRONYMS

ACLED	Armed Conflict Location & Events Data
AMISOM	Africa Mission in Somalia
API	Application Programming Interface
EOD	Explosive Ordinance Disposal
IED	Improvised Explosive Device
JSON	JavaScript Object Notation
RCMRD	Regional Centre for Mapping of Resources for Development
REST	Representational State Transfer
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
TTP	Tactics Techniques and Procedures
UNMAS	United Nations Mine Action Services

CHAPTER 1: INTRODUCTION

1.1 Background

Improvised Explosive Devices (IEDs) are a weapon of choice for terror groups across the world as they persistently seek to launch asymmetric operations against better equipped conventional security forces (UNMAS, 2020). IEDs are generally cheap and easy to manufacture (using locally available/improvised materials), but their effect in terms of casualties, damage to equipment and infrastructure, impeding movement and psychological impact is huge.

Globally, 15,764 casualties were caused by landmines, explosive remnants of war and IEDs in 2019, while the same claimed 10,102 casualties in 2020. Out of these, IEDS were responsible for 57% and 56 % of the casualties in the respective years (United Nations, 2021).

Regionally, Al-Shabaab and other affiliated militant groups have been using IEDs to target security forces and civilians both in Somalia and neighboring AMISOM troop contributing countries including Kenya. Since Kenya Defence Forces (KDF) crossed into Somalia in 2011 to combat Al Shabaab militants to date, there have been sustained retaliatory IED attacks within Kenya, particularly in the border counties of Mandera, Wajir, Garissa and Lamu. The graph below depicts the IED trends (both incidents and casualties) between 2017 and 2019.

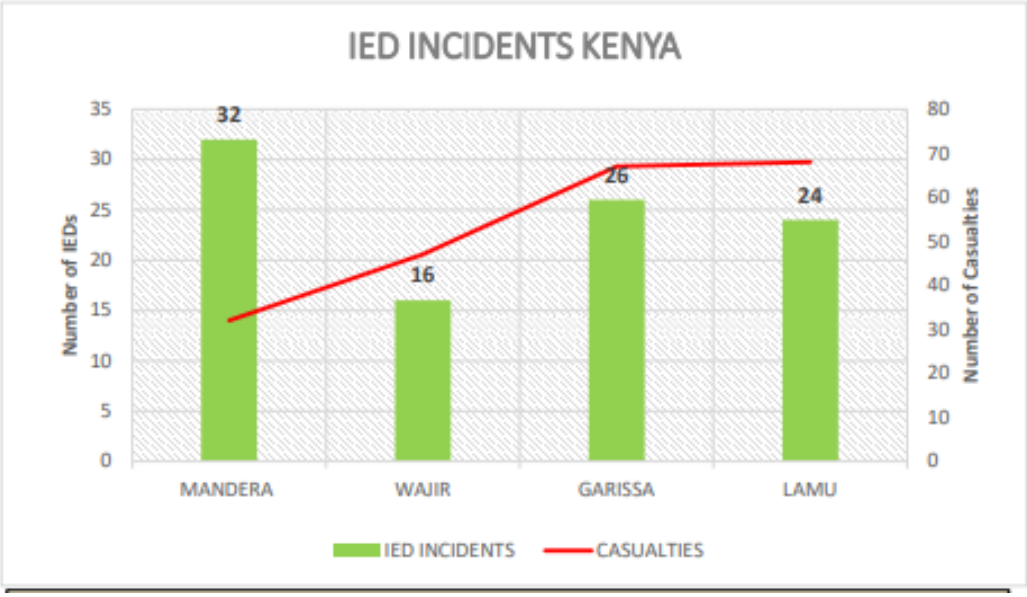


Figure 1:1 IED Incidents in Kenya (2017-2019) – Source UNMAS Report, 2020

From Figure 1, there were a total of 98 IED incidents between 2017 and 2019, which caused 207 casualties. IED incidents also occasion loss of equipment (mostly vehicles) with significant psychological effect on both troops and civilian population. There have been further attacks to date with both security troops and civilian casualties.

To effectively Counter the threat of IEDs, a good understanding of the IED System is critical. The IED System in this case entails various elements involved in the funding, supplying, planning, assembling, transporting, emplacing, triggering and exploiting IEDs (Michaud, 2013). Analyzing previous IED incidents is critical in providing useful insights towards understanding the IED system and thus crafting appropriate Counter-IED strategies, and adapting own Tactics Techniques and Procedures (TTPs) to mitigate against the threat.

An accurate geospatial database of IED incidents will be critical in facilitating necessary analysis to facilitate understanding of the adversary's IED TTPs which then inform counter measures to be adapted by the operational environment stakeholders to safeguard against IED attacks. Such a database would be more useful if it is complete (capturing all incidents) and accessible/sharable to the stakeholders.

This study seeks to therefore build an accurate IED Incidents Geospatial database for the Country. The database in to be served in a web-page to the stakeholders to make it more accessible, interactive and corroborative.

1.2 Problem Statement

Counter-IED strategies are pegged on three main pillars of preparing the force (through training, incorporating lessons learned and environmental situation awareness), attacking the network (use of intelligence to identify personnel/adversary IED experts, exploit the logistics chain and the devices used at strategic and tactical levels) and defeating the device through detecting, neutralizing and mitigating IEDS (EWS, 2021).

The three pillars require a good understanding of the adversary's IED Tactics, Techniques and Procedures (TTPs), particularly on the key aspects of IED emplacement (locations and layouts), IED constructs and initiation. These are best gleaned from understanding and exploiting the patterns of previous IED incidents. It is on this basis that own TTPs can be adapted appropriately to counter IED threat.

Counter-IED efforts have been hampered by lack of reliable and standardized IED data in Kenya which can be used to analyse IED location patterns to support the three C-IED pillars. Currently, different security agencies keep own data in a fragmented, unstandardized and in most cases incomplete. Using such data for requisite IED pattern analysis is thus a challenge. At the same time there are sharing challenges due to agencies data release regulations/protocols.

1.3 Objectives

The main objective of the project was to develop a web-map for IED incidents in Kenya based on an accurate, reliable and standardized IED incidents database, that is more easily accessible to the stakeholders. The project sought to address the following specific objectives:

1. Review and standardize available IED data.
2. Build a geospatial IED database for the IED that meets stakeholders' requirements.
3. Build an interactive web-mapping system for IED incidents in Kenya.

1.4 Justification for the Study

The project aimed to provide a reliable geospatial database with standardized IED data that meets stakeholders' requirements to facilitate IED trends and pattern analysis necessary to inform Counter-IED efforts and strategies in Kenya. Information from the resulting IED Mapping System will facilitate development and adaption of own operational TTPs that are aimed at reducing exposure to IED threat and thus save lives and reduce logistics damage and disruption occasioned by IED attacks.

Presentation of the IED data in a web-based map will also facilitate easy access to, and interaction with the IED data by various stakeholders. The stakeholders that are intended to be the beneficiaries of the database include the following:

1. The Government Security Agencies
2. The County governments
3. Non-Governmental Organizations operating in IED hotspots
4. The United Nations Mine Action Services
5. The Academia interested in studying IEDs

1.5 Scope of work

This project entailed mapping of IED incidents within Kenyan territory, which have taken place since 2012 (after Kenya crossed into Somalia in "Operation Linda Nchi") to date. This include both incidents along the Kenyan-Somalia border and those in the hinterland. The project does not include incidents that have occurred across the border in Somalia, even if they impacted on Kenyan Security forces.

The project involved creating an accurate and reliable IED database based on standardized data. Additionally, the project developed both the backend and front end applications, and an Application Programming Interface (API) to facilitate communication between the two. Users access the required data through the User interface that comprise the frontend. The data is hosted in a Postgres/PostGIS database accessible through the frontend. For access control, owing to the sensitivity of the data involved, deployment of the system was done on a server accessible through a closed network, and only created users can access and interact with the data.

The System was also configured to facilitate submission of incident reports or forwarding information (identified gaps or inconsistencies) on particular incidents, whenever errors are noted by users through the user dashboard (frontend).

CHAPTER 2: LITERATURE REVIEW

2.1 Improvised explosive Device (IED)

The US Homeland Security defines an IED a homemade bomb and/or destructive device that is used to destroy, incapacitate, harass, or distract. On its part, United Nations Department of Peace Keeping Operations defines an IED as a device placed or fabricated in an improvised manner incorporating explosive material, destructive, lethal, noxious, incendiary, pyrotechnic materials or chemicals designed to destroy, disfigure, distract or harass which may incorporate military stores, but are normally devised from non-military components.

From the definitions above, it is implied that an IED can range from a simple device to a very complex IED, which may at times even be combined with other components or elements to defeat attempts by Explosive Ordinance Disposal (EOD) experts or Counter IED teams to neutralize them. Other IEDs can also have enhancements incorporated in their assembly to make them more destructive. The construct of an IED, since it is improvised, can therefore only limited to the imagination and ingenuity of the maker.

An IED by design has five main components. These include the initiator, a switch mechanism, the main charge (explosive substance), power source and a casing/container (Homeland Security, 2013). Different actors will assemble these elements in a variety of ways, depending on training and experience.

2.2 IED emplacement

Once an IED has been assembled, it is emplaced at a suitable location where it can effectively "get" its target. A number of factors will influence the emplacement; these include the prevailing tactical situation, the target of interest, access or local support and available time. The devices can be placed under the ground surface (dug in), on the ground surface and even above the ground surface. Most of the IEDs targeting security forces are placed along the Main Supply Routes (MSR) where the forces are likely to pass. Such locations in most cases have some tactical aspects that make them a preferred choice by the emplacers. It is thus common practice to have IEDs emplaced at specific locations multiple times.

2.3 Approaches to Countering IEDs

Effective Counter-IED approach is based on three activity pillars as highlighted in Figure 2.1.

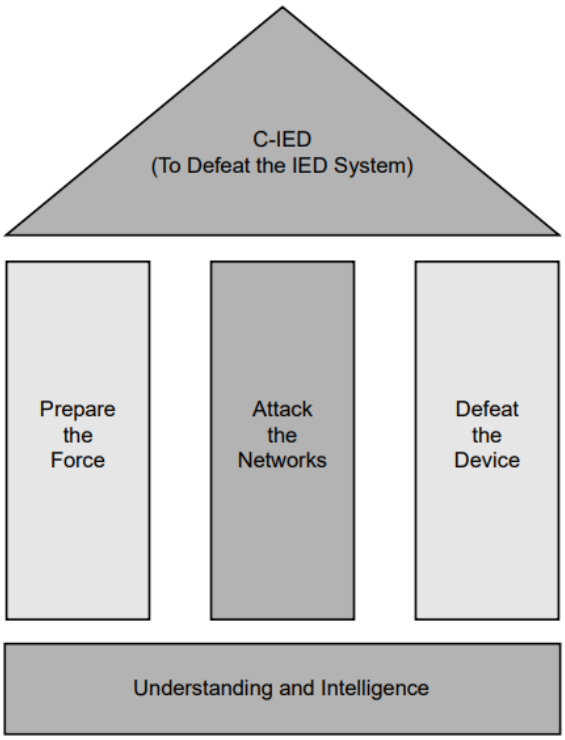


Figure 2.1: Three activity Pillars of Counter-IED Approach. Source NATO, 2011

The IED System comprise of personnel, resources and activities that support the execution of an IED event. It entails all elements involved in various facilitating activities such as funding, supplying, planning, assembling, transportation, emplacement, triggering and exploiting IEDs (Michaud, 2013).

At the bottom of the pillars is understanding the IED System. It is critical that for Counter-IED approaches to be effective, there has to be a good understanding of the IED System. Although this is not an easy, as there is no single model on how it operates, its structure can be predicted based on IED emplacement activities. Further, information on the constructs of the IED devices can also provide crucial insights towards understanding the IED system. Such information can be gleaned from exploitation of information from previous IED incidents. Intelligence on the IED System is the foundation of the three Counter-IED pillars of preparing the force, defeating the device and attacking the networks, highlighted in Figure 2.1.

The preparing the force pillar above entails measures of readying the force to operate in an environment where there is an IED system. It basically entails a good situation awareness of the operational environment. It is usually achieved through training at all levels (individual and collective), and is anchored of the TPPs of the adversary IED system. Exploitation of information on IED incidents from the operational environment, and existing intelligence on the IED system are key in facilitating the prepare pillar.

Defeating the Device on the other hand involves measures, both proactive and reactive, aimed at detecting and neutralizing IEDs before impact. IED detection systems detect both the IED emplacement and other nodes of the IED System such as infiltration routes and IED caches awaiting assembly or emplacement. Information on the current IED trends/ The IED situation in the operating environment is a key facilitator of this pillar. This can be achieved through exploitation of the IED incidents data.

Lastly, the third pillar of attacking the network entails intelligence driven offensive and proactive activities aimed at disrupting the network of the adversary IED System. This is considered the most effective strategy as it focuses on activities that precede an IED event.

2.4 IED Incidents Mapping

An incident map is a key intelligence analysis tool. It facilitates the establishment of patterns and trends in events that would not otherwise be obvious without spatial visualization. At the same time, a spatial database of the incidents, when appropriately designed to include requisite fields that are essential requirements in facilitating full situational awareness can be valuable source of information for incidents analysis, now and in the future. Critical aspects of an incident information include the who, what, when, where, why and how facets, often referred to as 5W +H of an incident. These should be adequately addressed in any incident mapping system.

For the case of IED incidents, a spatial database will be particular important in providing information on the IED emplacement patterns, where the adversary emplaces the IEDs, support systems (e.g. local support, especially when the IEDs are placed near settlements), timing of IED incidents/when, IED constructs/design (where images of the devices are available for analysis), etc. Such information is very critical in supporting the three pillars of the Counter-IED strategy.

2.5 The Web-Mapping Approach

A web Map is an interactive display of geographic information, in the form of a web page, that that can be used to represent geospatial information and facilitate queries. Web-mapping enables information to be shared, visualized, and edited in the browser. The most profound upside of a web map is accessibility: the map can be accessed by any user, using any device that has an internet browser and is connected to the internet (Dorman, 2021).

Web-mapping is therefore a very powerful tool for representing information meant to be accessed by many users/ stakeholders who are spread-out in terms of geographic location. In addition to accessibility, web maps are also a powerful geospatial data visualization tool, which can also facilitate collaboration in building geospatial databases.

In the case of IED incidents data, which would be required by various stakeholders to make decisions to support their Counter-IED strategies, a web map is the most appropriate way of serving the information. This can also facilitate data reviews by stakeholders and correction of observed errors, thus adding to the integrity of the data rendered to customers.

2.6 Web-Mapping Technologies

2.6.1 Hyper Text Markup Language (HTML)

This is the core of every web page on the internet today regardless of its complexity. It is a data format used for encoding the structure and the contents of a web page. This is achieved through different HTML tags which help structure a web page into elements such as headings, paragraphs, images, links, etc. HTML is stored in plain text with .html extension.

2.6.2 Cascading Style Sheets (CSS)

This is a design language that is used to style a web page in order to make it look presentable, well formatted and laid out. Adding well-thought CSS styles improves the attractiveness a web page and make it appealing to the users. Otherwise, with HTML alone the web pages created will share the same default single-column arrangement, default colours, default fonts, etc. In addition, CSS can also be used to make a webpage responsive so it can be viewed in large, medium and small screens.

2.6.3 JavaScript

JavaScript is a programming language that is used to control interactive behaviour in web pages. Together with HTML and CSS, they form the three core technologies of the web and are therefore the fundamental technologies of the web (Dorman, 2021). Through various functions and classes, JavaScript allows for modification of the web page content differently with respect to a user's actions. Thus, different users can view different parts of a web page depending on how it has been programmed with JavaScript. As a programming language, JavaScript can also be used to perform arithmetic calculations and display results to the end users of a web application. Although JavaScript can be used for both client-end and server-end programming, it is primarily a client-end programming language.

Most mapping and visualization libraries that are open-source are written in JavaScript. Leaflet, and Open Layers are examples of such. Also, a number of available commercial web mapping services use JavaScript API for building web maps with their tools. These include Google Maps JavaScript API, Mapbox GL: JS and ArcGIS JavaScript API.

2.6.4 Leaflet

Leaflet is an open-source and light-weight JavaScript library for developing responsive and interactive maps. Leaflet allows creation of base maps, overlaying different types of spatial data on the base maps and configuring different map controls to allow end users to easily interact with maps. Some of the common map controls include zoom, click, pan, scale etc. Leaflet is supported across multiple commonly used browsers such as Chrome, Firefox, Safari and Internet Explorer among others.

2.6.5 GeoJSON

GeoJSON is a JSON (JavaScript Object Notation) based data exchange format designed to represent spatial data together with their attribute information. It presents collective information about the geographic features, their spatial extents and their properties. The GeoJSON format supports a variety of geographic feature types including points, polylines, polygons, multi-points, multi-lines, and multi-polygons. These features and their properties are contained in Feature Collection objects. This is the standard data format used by the Leaflet library described above.

2.6.6 Databases & APIs

A database is a modern technology that was developed to replace traditional file systems. Databases are an organized collection of structured information stored electronically in a computer. In the context of web applications, databases are used to store structured information about the users of a web page, spatial data, payment information and much more. On the other hand, an API is a piece of software that is used by another piece of software to communicate with each other. This communication is typically between a server and a client. In the context of a web mapping application, an API is the software that allows data stored in the database to be relayed to a web page for display or data entered by a user on the web page to be relayed to the database for storage. There are different types of APIs, that include REST, SOAP, RPC, and GraphQL APIs which have different architectures of implementation.

2.6.7 Open-Source Web Mapping Application development frameworks

There exist open-source development frameworks and platforms that can be used to more easily and quickly develop server-end and client-end applications. Two common frameworks are the Django and Angular frameworks that are used for Server and Client-end development respectively.

2.6.7.1 Django Framework

Django is an open-source web framework based on the Python programming language that is used for developing web applications. Django provides a set of tools and libraries that make it easier to develop web applications. It also provides the Django REST Framework which is a python package built on top of Django for developing RESTful APIs that are used to relay data to and from the database (backend) and the client (frontend). Django follows the models-templates-views architecture. It has a comprehensive documentation that guides through the development process, with tutorials that can be used to learn how to build web applications using the framework.

2.6.7.2 Angular Framework

Angular is a typescript based open-source web framework developed by Google for developing single-page web applications. This framework provides a standard structure for developing the frontend components of a web application in a consistent and maintainable manner. It consists of different building blocks that include components, services, templates, directives and many more, all of which perform different functions for a web application to work as required**.** The

framework's documentation is a very important reference, with tutorials on how to use Angular framework and development platform.

2.7 A Focus on some existing Web-Mapping efforts

In taking advantage of the web-mapping capabilities, there have been projects mapping incident data or spatial phenomenon through web maps. One such is the COVID 19 incidents web map by John Hopkins University Corona Virus Resource Centre at https://coronavirus.jhu.edu/map.html, which provides real-time updated data on COVID 19 incidents across the world. Such information can therefore be accessed by millions of people across the world at the same time, provided they are connected to the internet.

Another such web-mapping project is the Armed Conflict Location & Event Data Project (ACLED) that is available at https://acleddata.com/dashboard#/dashboard. ACLED is a disaggregated data collection, analysis, and crisis mapping project that collects the dates, actors, locations, fatalities, and types of all reported political violence and protest events around the world and maps the same on a web-page. The web map facilitates geographic visualization of the reported incidents, facilitates queries, filters, provides graphical summaries of the data and offers tools for downloading the data in csv format.

The ACLED incident data is a very comprehensive data on armed conflict events across the globe. The web map facilitates visualization of armed conflict events in time and space, and at the same time has an analytics component that depicts trends through statistical graphs.

The ACLED Web Mapping application actually provides data on explosion/IED incidents across the world, including Kenya as one of the armed conflict events. The source of the data, however is mainly open source reporting, and a number on incidents, that are not reported through social media end up missing from the ACLED database. It is important to note that for IEDs, there are a number of cases that constitute "find" incidents – where IED devices are detected and disabled by Explosive Ordinance experts before impact. Most of these do not find their way to the media, and would therefore not be captured by ACLED.

An IED incidents web map will therefore need to be more complete and accurate to capture all incidents for analysis. It therefore needs casting the net widely to exploit multiple sources to ensure that all incidents are captured in the system and mapped.

CHAPTER 3: MATERIALS AND METHODS

3.1 Datasets used in the project

The following datasets were used for the IED incidents web-mapping project (MapIED System development).

1. IED Incidents data sourced from UNMAS, and ACLED and Kenyan Security Agencies.
2. The Kenya map base layer and image – World imagery and world topographic map respectively.
3. Thematic layers sourced from RCMRD Geoportal.

3.2. System Design

Figure 3.1 shows the MapIED System design. This section will focus on the elements this design.

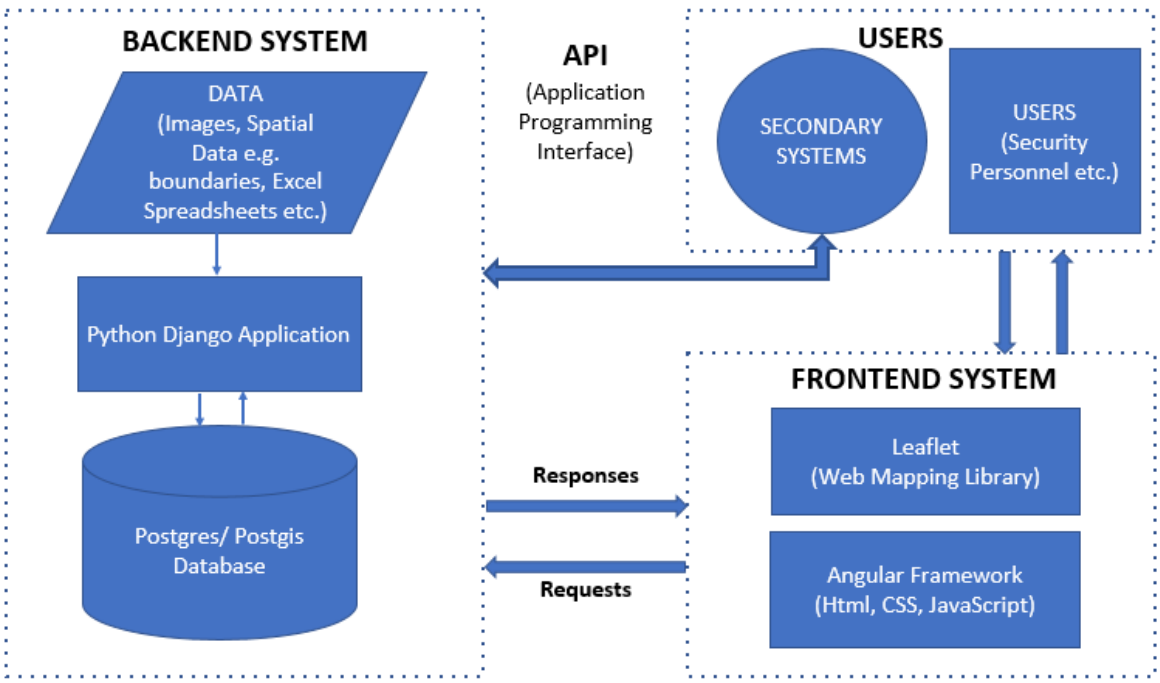


Figure 3.1: The Web Mapping System (MapIED) design

The MapIED System comprise the backend and frontend, with an Application Programing Interface (API) facilitating communication between them. The core of the backend is the Django application that provides the API of the system. The API allows for adding of new records to the database, retrieving existing records and updating records as needed. It is embedded directly into the backend of the application and it can also be used by secondary applications independently to access data from the database.

The database management system (DBMS) comprise the open source Postgres/Postgis database for storing data in form of tables. It also supports SQL for querying, updating, creating and deleting data from the database. Postgis is an extension of the Postgres database that allows the DBMS to store/manage spatial data.

In the frontend of the system is the client-facing web application through which users visualize and interact with the application. It leverages the leaflet mapping library for visualizing spatial data on a map. Angular was used in the frontend development. This is a web framework developed by Google for faster and secure development of web applications. It provides html for defining the structure of the webpage, CSS for styling the web page, and Javascript for adding interactivity to the web page.

The details on implementation of the above design elements are discussed in the methodology part of this chapter.

3.3 Methodology

The implementation of the MapIED Web Mapping System was effected through the following the stages.

3.3.1 Design stage

This stage started with conceptual design, where user requirements were assessed through meeting key stakeholders that included security actors (Kenya Defence Forces and National Police Service operations analysts) and a representative from the United Nations Mine Action Services. This helped to understand the specific user requirements of the envisaged IED Information System to be served through the web. At the same time, the status of available IED incidents data and the existing gaps were clarified through the user engagement meetings. Based these meetings, specific IED incident information identified as critical for the MapIED System include date and time of incident, description of the incident (5W+H), day of the incident, location information (coordinates and place name), category and type of incident, assessed target, casualties (injuries and fatalities), images of the incident, technical reports, and incident information update history.

Based on the information, two design elements were implemented. One, a database design element in form of Entity Relation Diagram (ERD), and the second one, a prototype of the user interface dashboard for the front-end through which IED incidents data is served to users.

The ERD was developed using LucidChart application. It defines the relationship between all the entities of the database. Figure 3.2 depicts the ERD that was developed for the MapIED database:

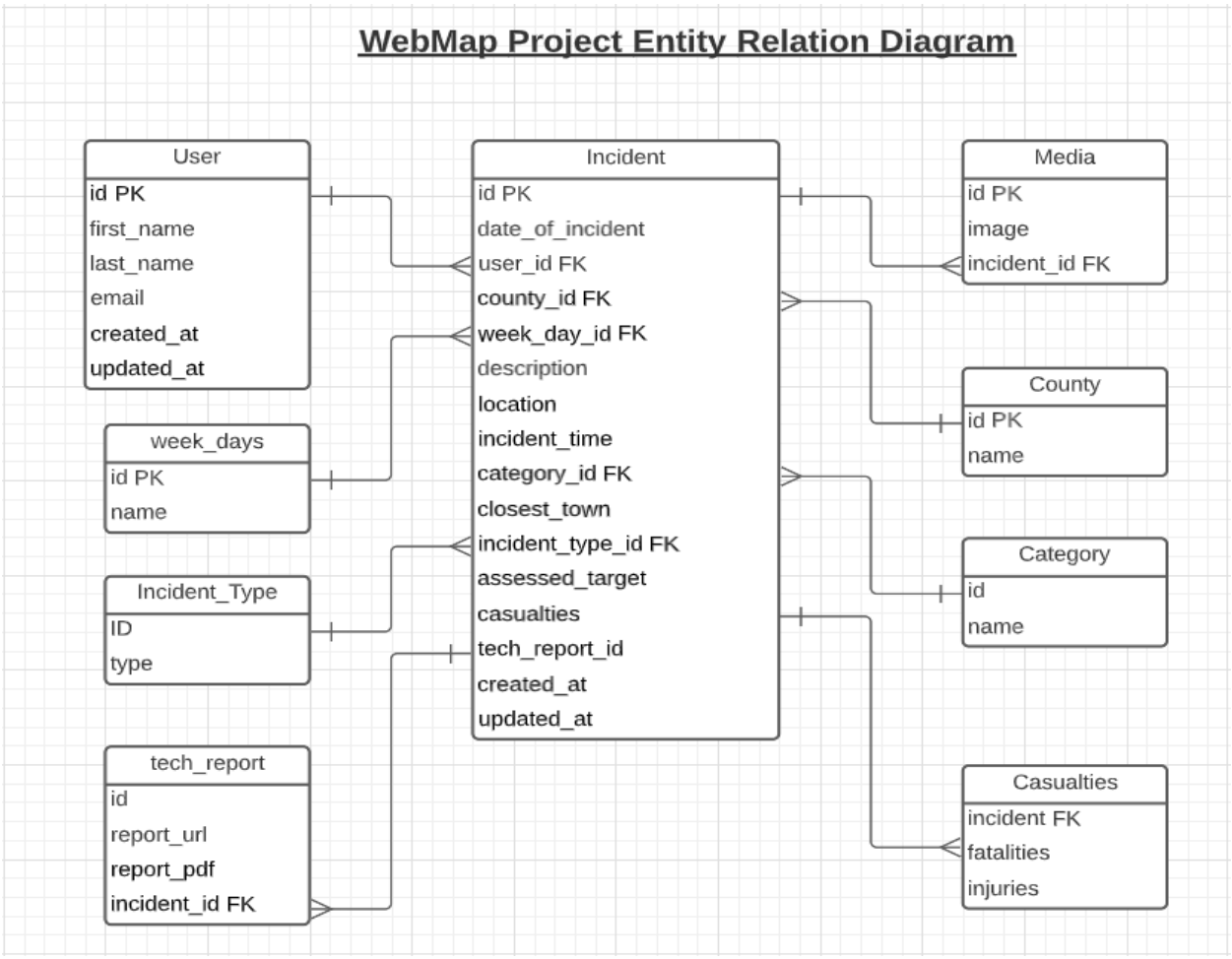


Figure 3.2: The MapIED Database ERD

The prototype for the MapIED dashboard / user interface design was developed using Figma Application. Figma is a web-based graphics editor for prototyping and user interface design. In this case, the design was done to the mockup stage (without interactivity). This was adequate as the users involved in the requirements assessment had a fairly good understanding of the datasets, requirements and web-application interfaces.

The initial mockup was shared with the users for their input. Based this input, the user interface was continuously enhanced, taking into account user requirements to the final dashboard application that is presented in the results section of this report. Figure 3.3 is a snapshot of one of the first mockup prototype for the MapIED application dashboard developed at the design stage. It depicts the key elements of the user dashboard in the client-end of the system, without the interactivity aspect and is a pointer to what elements the dashboard would comprise of.

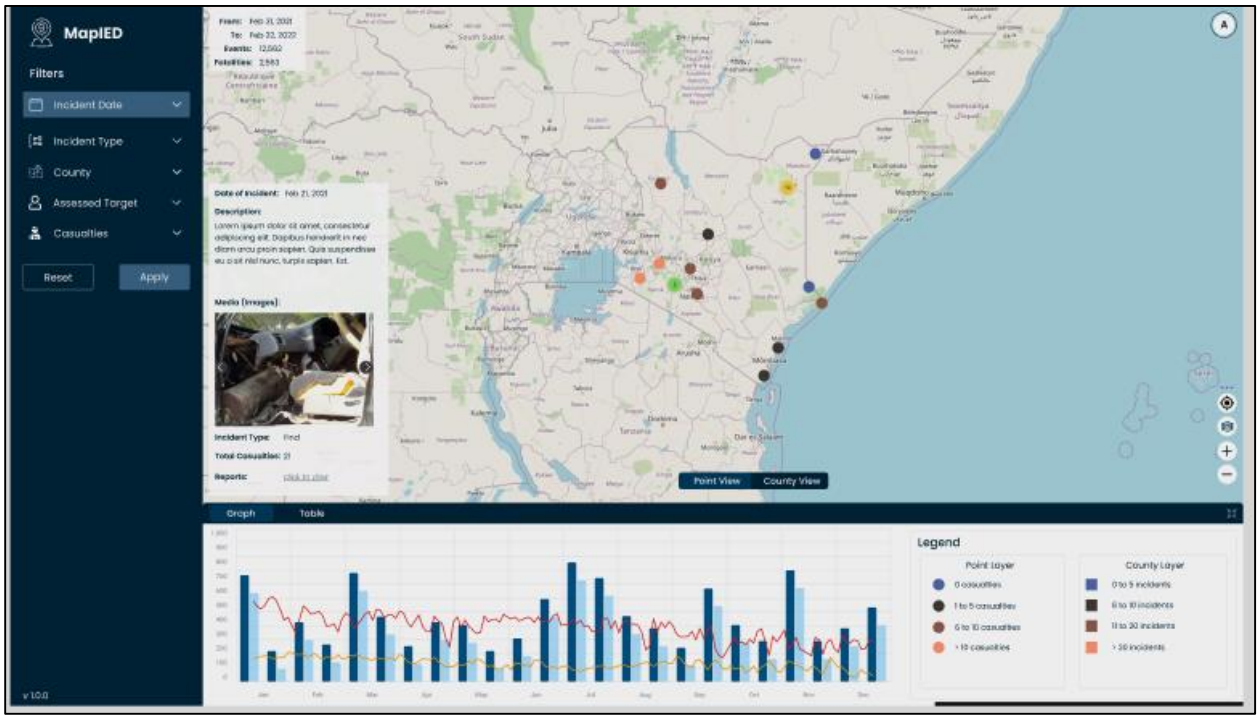


Figure 3.3: The Prototype (mock-up) for MapIED Dashboard

3.3.2 Review and Standardization of the IED Incidents Data

The IED incidents datasets used in this project were sourced from UNMAS, ACLED and incidents data from Kenyan Security Agencies collated under the Kenya Multi-Agency Security framework. These datasets were in form of excel sheets with varied fields, IED incidents Images, and in some cases technical reports on incidents compiled by Counter IED personnel.

Upon evaluation of the data against the required fields of the schema generated from user requirements assessment, it was noted that the UNMAS data was more complete in detailed in relation to the required fields. It was therefore used as the base data, and other datasets used to fill gaps identified, and also corroborate the incident information provided. The data was also examined and edited to correct errors identified.

The images/media files, and technical reports were also found to have varied indexing, for different sources of data. To standardize the media files, they were indexed according to the dates on the IED incidents they represent. The result of this process was a standardized (with respect to fields, and information/details format) excel table of IED incidents data, and indexed (by date of incident) images and technical reports of the incidents.

The fields of the standardized IED incidents table include Incident ID, Incident Date, Description (5W+H), Incident Time, Day of the Week, Location (Longitude, Latitude), County, Nearest Town, Incident Category, Incident Type, Assessed Target, Casualties (injuries, deaths), Technical Report ID, Date Created and Date Updated. This is the standardized data that was imported into the MapIED System through the data import feature (see Fig. 4.3)

3.3.3 Backend Development

The backend of this application is responsible for managing all the data used in the system. It handles operations such creation, updating and deleting of data in the database. It also handles the user management module of the application. The backend was developed using Django through the steps highlighted in the following subsections.

3.3.3.1 Creating a Django Project

The MapIED project was created using the `*django-admin projectname*` command. This created three essential project files. The first is the *manage.py file***.** This is a command-line utility file for running Django related commands for different purposes such as opening a development server, running database migrations, etc. Another file is the *settings.py file* which contains all the configurations for the django project, including database configuration, all installed applications, **e**mail configuration, and **t**emplate configuration. The third file is the *urls.py file***.** This file handles all the routing configurations for the project, which enable data to be relayed from the backend to the frontend of the system.

3.3.3.2 Creating Django Applications

An application contains python files with related functionality. In the MapIED Web Mapping System, three applications were created using the command `*python manage.py startapp appname*`. These applications include Users, Incidents and Submissions applications.

3.3.3.2.1 Users Application

This application manages users in the MapIED system. It allows users to sign up to the system, login and access data stored in the database. The application serves to safeguard security and integrity of the system. The user's application contains three main files. These include *models.py file***,** the *serializers.py file* and *the views.py* file**.**

Models.py file defines the user table and the fields within the table for storing user data in the database. It also contains any constraints in the fields that have to be met when adding user data to the database. The *serializers.py file* on the other hand is used to convert the user data stored in the database to JSON format which can then be sent to the frontend. Lastly, the *views.py* file contains the logic for processing a client's request and returning a response. Snippets of the scripts for these *models*, *serializers* and *views* are shown in Codes 3.1, 3.2 and 3.3 respectively.

```python
from django.contrib.auth.models import models
from django.contrib.auth.models import AbstractBaseUser, PermissionsMixin
from .managers import CustomUserManager


class CustomUser(AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(max_length=255, unique=True)
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    created_at = models.DateField(auto_now_add=True)
    updated_at = models.DateField(auto_now=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)

    objects = CustomUserManager()

    USERNAME_FIELD = "email"
    REQUIRED_FIELDS = ["first_name", "last_name"]

    def get_full_name(self):
        return " ".join([self.first_name, self.last_name])

    def get_short_name(self):
        return self.first_name

    def __str__(self):
        return self.get_full_name()
```

Code 3.1: Models script of the User application

```
from webbrowser import get
from djoser.serializers import UserCreateSerializer
from django.contrib.auth import get_user_model

User = get_user_model()


class UserCreateSerializer(UserCreateSerializer):
    class Meta(UserCreateSerializer):
        model = User
        fields = ("id", "email", "first_name", "last_name", "password")
```

Code 3.2: Serializers script of the User application

```
from rest_framework_simplejwt.serializers import TokenObtainPairSerializer
from rest_framework_simplejwt.views import TokenObtainPairView


class CustomTokenObtainPairSerializer(TokenObtainPairSerializer):
    def validate(self, attrs):
        ## This data variable will contain refresh and access tokens
        data = super().validate(attrs)
        ## You can add more User model's attributes like username,email etc.
in the data dictionary like this.
        data["first_name"] = self.user.first_name
        return data


class CustomTokenObtainPairView(TokenObtainPairView):
    serializer_class = CustomTokenObtainPairSerializer
```

Code 3.3: Views script of the User application

3.3.3.2.2 Incidents Application

The Incidents Application was created to hold the python files for managing IED incidents data. The main elements of this application just like in the Users application, include models, serializers and views. These elements are described in the next sub-sections.

3.3.3.2.1.1 Models.py file

This file defines the tables for storing all data related to IED incidents. The tables in turn contain the fields for storing specific aspects of the IED incidents. Key elements here include: importing important packages (See Code 3.4); creating County, Category and Incident Type models (see Code 3.5); creating the main Incident model (see Code 3.6); creating Media model for storing

images associated with incidents and storing number of incidents per month and Technical Reports model for holding technical reports that are written by CIED and EOD experts on the IED incidents (see Code 3.7).

```
from django.contrib.gis.db import models
from django.contrib.auth import get_user_model
from django.utils.translation import gettext_lazy as _


def upload_to(instance, filename):
    return "incidents/{}-{}-{}".format(
        instance.incident.date_of_incident, instance.incident.incident_time,
filename
    )


DAYS_OF_WEEK = (
    ("Monday", "Monday"),
    ("Tuesday", "Tuesday"),
    ("Wednesday", "Wednesday"),
    ("Thursday", "Thursday"),
    ("Friday", "Friday"),
    ("Saturday", "Saturday"),
    ("Sunday", "Sunday"),
    ("Not Provided", "Not Provided"),
)
```

Code 3.4: Importing important packages

```
class County(models.Model):
    name = models.CharField(max_length=50, blank=False, null=False)
    poly = models.MultiPolygonField(srid=4326)
    class Meta:
        verbose_name_plural = "counties"
    def __str__(self):
        return self.name
class Category(models.Model):
    name = models.CharField(max_length=100)

    class Meta:
        verbose_name_plural = "categories"
    def __str__(self):
        return self.name
class IncidentType(models.Model):
    incident_type = models.CharField(max_length=100)

    def __str__(self):
        return self.incident_type
```

Code 3.5: Creating County, Category and Incident Type models

```
class Incident(models.Model):
    description = models.TextField(blank=False, null=False)
    date_of_incident = models.DateField(auto_now=False, auto_now_add=False,
blank=False)
    location = models.PointField(srid=4326)
    closest_town = models.CharField(max_length=50, blank=True, null=True)
    week_day = models.CharField(max_length=50, choices=DAYS_OF_WEEK)
    created_by = models.ForeignKey(
        get_user_model(), on_delete=models.PROTECT,
related_name="%(class)s_createdby"
    )
    modified_by = models.ForeignKey(
        get_user_model(),
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name="%(class)s_modifiedby",
    )
    county = models.ForeignKey(
        County, related_name="incidents", on_delete=models.PROTECT,
blank=False
    )
    assessed_target = models.CharField(max_length=100, blank=True)
    incident_time = models.TimeField(
        auto_now=False, auto_now_add=False, blank=True, null=True
    )
    incident_type = models.ForeignKey(IncidentType, on_delete=models.PROTECT)
    category = models.ForeignKey(Category, on_delete=models.PROTECT)
    fatalities = models.IntegerField(default=0)
    injuries = models.IntegerField(default=0)
    created_at = models.DateField(auto_now_add=True)
    updated_at = models.DateField(auto_now=True)

    def __str__(self):
        return str(self.date_of_incident) + " " + str(self.closest_town)
```

Code 3.6: Creating the main incident model

```
class Media(models.Model):
    incident = models.ForeignKey(
        Incident,
        blank=False,
        null=False,
        on_delete=models.PROTECT,
        related_name="incident_images",
    )
    image = models.ImageField(
        _("Image"), upload_to=upload_to, default="incidents/default.jpg"
    )

    def __str__(self):
        return self.image.name


class TechReport(models.Model):
    incident = models.ForeignKey(
        Incident, blank=False, null=False, on_delete=models.PROTECT
    )
    report_pdf = models.FileField(blank=True, null=True)
```

Code 3.7: Creating the Media and technical Reports models

3.3.3.2.1.2 Serializers.py file

This file converts all the incidents data stored in the database to GeoJSON format that can be sent to the frontend via the REST API. Code 3.8 is a snippet of the serializers script in the Incidents Application.

```
from rest_framework_gis.serializers import GeoFeatureModelSerializer
from rest_framework import serializers
from .models import Incident, Media, County, GroupIncidents


class IncidentSerializer(GeoFeatureModelSerializer):
    """
    Serializes data stored in the incident model
    """

    incidenttype_name =
serializers.ReadOnlyField(source="incident_type.incident_type")
    county_name = serializers.ReadOnlyField(source="county.name")
    incident_images = serializers.StringRelatedField(many=True)
    casualties = serializers.SerializerMethodField()
```

```
    def get_casualties(self, incident: Incident):
        """Returns sum of injuries and fatalities (casualties)"""
        return incident.injuries + incident.fatalities

    class Meta:
        model = Incident
        geo_field = "location"
        fields = (
            "description",
            "date_of_incident",
            "closest_town",
            "week_day",
            "created_by",
            "modified_by",
            "county",
            "county_name",
            "assessed_target",
            "incident_time",
            "incidenttype_name",
            "incident_type",
            "category",
            "casualties",
            "fatalities",
            "injuries",
            "created_at",
            "updated_at",
            "incident_images",
        )


class CountySerializer(GeoFeatureModelSerializer):
    """
    Serializes data stored in the county model
    """
    incidents = serializers.StringRelatedField(many=True, read_only=True)

    class Meta:
        model = County
        geo_field = "poly"
        fields = (
            "id",
            "name",
            "poly",
            "incidents",
        )
```

```python
class IncidentsGroupMonthSerializer(serializers.ModelSerializer):
    """Serializes aggregated incidents per month"""

    class Meta:
        model = GroupIncidents
        fields = ("month", "count")

class MediaSerialiser(serializers.ModelSerializer):
    """
    Serializes images in the media model
    """
    class Meta:
        model = Media
        fields = "__all__"

    def get_photo_url(self, obj):
        request = self.context.get("request")
        photo_url = obj.fingerprint.url
        return request.build_absolute_uri(photo_url)
```

Code 3.8: Serializers Script of the Incidents Application

3.3.3.2.1.3 Viewss.py file

This file processes all API requests for all incidents data and returns the data depending on the request. Some of the aspects in the script include importing requisite python packages (see Code 3.9), returning all incidents (Code 3.10), filter by date and County (Code3.11), aggregate incidents by month (Code 3.12), to provide "county view" by returning all or selected counties (Code 3.13) and accessing media files (Code 3.14).

```python
from django.db.models import Count
from django.db.models.functions import TruncMonth
from django.shortcuts import render
from datetime import datetime, timedelta
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from rest_framework.response import Response
from rest_framework.decorators import action
from .models import Incident, Media, County
from .serializers import (
    IncidentSerializer,
    MediaSerialiser,
    CountySerializer,
    IncidentsGroupMonthSerializer,
)
```

Code 3.9:Importing requisite python packages for Incident Application views

```
class IncidentView(viewsets.ModelViewSet):
    permission_classes = [IsAuthenticated]
    queryset = Incident.objects.all().order_by("date_of_incident")
    serializer_class = IncidentSerializer

    @action(detail=False)
    def getAllIncidents(self, request):
        """Returns all Incidents in the database"""
        all_incidents = self.get_queryset()
        serializer = IncidentSerializer(data=all_incidents, many=True)
        serializer.is_valid()
        return Response(serializer.data)
```

Code 3.10:Returns all incidents in the Incidents Application

```
@action(detail=False)
def getDatedIncidents(self, request):
    """Returns filtered incidents to specific counties and within specified start and end dates"""
    date_format = "%Y-%m-%d"
    from_date = self.request.query_params.get("from_date")
    to_date = self.request.query_params.get("to_date")
    from_date = datetime.strptime(from_date, date_format)
    to_date = datetime.strptime(to_date, date_format)
    counties = self.request.query_params.get("counties").split(",")
    incident_types = self.request.query_params.get("incident_types").split(",")
    queryset = self.queryset.filter(incident_type__in=incident_types)
    queryset = queryset.filter(county__in=counties)
    queryset = queryset.filter(date_of_incident__range=[from_date, to_date])
    serializer = IncidentSerializer(data=queryset, many=True)
    serializer.is_valid()
    return Response(serializer.data)
```

Code 3.11:Filters Incidents by date, County

```python
@action(detail=False)
def getAggregateIncidents(self, request):
    """Returns aggregated incident data per month"""
    queryset = (
        Incident.objects.annotate(month=TruncMonth("date_of_incident"))
        .values("month")
        .annotate(count=Count("id"))
        .values("month", "count")
        .order_by("month")
    )
    serializer_class = IncidentsGroupMonthSerializer(data=queryset, many=True)
    serializer_class.is_valid()
    return Response(serializer_class.data)
```

Code 3.12: To aggregate Incidents by month

```python
class CountyView(viewsets.ModelViewSet):
    permission_classes = [IsAuthenticated]
    queryset = County.objects.all().order_by("id")
    serializer_class = CountySerializer

    @action(detail=False)
    def getAllCounties(self, request):
        """Returns all counties in the database"""
        all_counties = self.get_queryset()
        serializer = CountySerializer(data=all_counties, many=True)
        serializer.is_valid()
        return Response(serializer.data)

    @action(detail=False)
    def getFilteredCounties(self, request):
        """Returns only selected counties as per counties query param"""
        all_counties = self.get_queryset()
        counties = self.request.query_params.get("selected_counties").split(",")
        queryset = all_counties.filter(id__in=counties)
        serializer = CountySerializer(data=queryset, many=True)
        serializer.is_valid()
        return Response(serializer.data)
```

Code 3.13: To provide the "County View" by returning all or selected counties

```
class MediaView(viewsets.ModelViewSet):
    permission_classes = [IsAuthenticated]

    def get(self, request, format=None):
        """Provides url for accessing incident images"""
        queryset = Media.objects.all()
        serializer = MediaSerialiser(queryset, context={"request": request},
many=True)
        return Response(serializer.data)
```

Code 3.14: To to access incident images/ media files

3.3.3.2.3 Submissions Application

This Application was developed to facilitate contribution of IED incidents reporting/information by stakeholders that will use the MapIED System. To safeguard the integrity of data that is input into the system, information/ reports contributed through the Submission application will be available for review by the system administrator (in the backend) who will then create incidents based on the information, and other reports/information available after verification. A number of submissions can therefore be made by different users on the same incident, from which the administrator can corroborate the information provided and create an incident based on the standard incident format.

This application can also be used by a user who notes an error on a particular incident in the system, or who has additional information on a particular incident. In addition to incident information, the application also gives provision for adding images of the incident being reported. It provides a form, with fields drawn from the schema of the standardized incidents database. A user fills relevant details in the form.

The structure of the database for storing this data is as defined in the *models.py* file (see Code 3.15). Separately, the model for submitting images for the incidents reported is defined in Code 3.16.

```python
class IncidentSubmission(models.Model):
    description = models.TextField(blank=False, null=False)
    date_of_incident = models.DateField(auto_now=False, auto_now_add=False, blank=False)
    location = models.PointField(srid=4326, blank=True, null=True)
    closest_town = models.CharField(max_length=50, blank=True, null=True)
    week_day = models.CharField(max_length=50, choices=DAYS_OF_WEEK)
    created_by = models.ForeignKey(
        get_user_model(), on_delete=models.PROTECT, related_name="%(class)s_createdby"
    )
    modified_by = models.ForeignKey(
        get_user_model(),
        null=True,
        blank=True,
        on_delete=models.SET_NULL,
        related_name="%(class)s_modifiedby",
    )
    county = models.ForeignKey(County, related_name='submission', on_delete=models.PROTECT, blank=False)
    assessed_target = models.CharField(max_length=100, blank=True)
    incident_time = models.TimeField(
        auto_now=False, auto_now_add=False, blank=True, null=True
    )
    incident_type = models.ForeignKey(IncidentType, on_delete=models.PROTECT)
    category = models.ForeignKey(Category, on_delete=models.PROTECT)
    fatalities = models.IntegerField(default=0)
    injuries = models.IntegerField(default=0)
    created_at = models.DateField(auto_now_add=True)
    updated_at = models.DateField(auto_now=True)
    additional_details = models.TextField(blank=True, null=True)

    def __str__(self):
        return str(self.date_of_incident) + " " + str(self.closest_town)
```

Code 3.15: Model for Incident Report Submission

```python
class SubmissionImages(models.Model):
    incident_submission = models.ForeignKey(
        IncidentSubmission,
        blank=False,
        null=False,
        on_delete=models.PROTECT,
        related_name="submission_images",
    )
    image = models.ImageField(
        _("Image"), upload_to=upload_to, default="submissions/default.jpg"
    )

    def __str__(self):
        return self.image.name
```

Code 3.16: Model for Submitting Images in the Submission Application

3.3.4 Frontend Development

The frontend includes all the web pages that the user interacts with. It fetches data from the backend through the Rest API and visualizes it for the user to interact with it. The frontend for the MapIED application was developed using Angular Frontend Framework and Leaflet 2D web mapping library.

Angular is a typescript based open-source web framework developed by Google for developing single-page applications. It provides a standard structure for developing the frontend components of a web application in a consistent and maintainable manner. The building blocks for the Angular application include components, services and models.

3.3.4.1 Components

These are the basic building block for an angular application. It contains classes, methods and properties for building the logic for your application and binding it to the templates. In the MapIED application, 5 components were developed i.e. register, activate, login, dashboard and submissions components. The register, activate and login components (see Codes 3.17, 3.18.3.19 respectively) manage user registration, user account activation and user login respectively. The dashboard component on the other hand handles all MapIED dashboard operations, display and styling (see Code 3.20 for an excerpt of the dashboard component) while the submission component handles Incident Submissions by users.

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from '@angular/forms';
import { AuthService } from 'src/app/services/auth.service';
import { NotificationService } from 'src/app/services/notification.service';
import { IUser } from 'src/app/_models/user';

@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrls: ['./register.component.scss'],
})
export class RegisterComponent implements OnInit {
  error: any = null;
  activate: boolean = false;
  newUser: IUser = {
    first_name: '',
    last_name: '',
    email: '',
    password: '',
    re_password: '',
  };
```

```
  constructor(
    private authService: AuthService,
    private notificationService: NotificationService
  ) {}
  ngOnInit(): void {}

  onRegistrationSuccess(successResponse: any) {
    this.notificationService.showSuccessToast(
      'Registration was successful!',
      null
    );
  }
  onSubmit(form: NgForm) {
    const post_data = { ...this.newUser };
    if (!form.valid) {
      return;
    }
    this.authService.registerUser(post_data).subscribe(
      (result) => {
        console.log(result);
        this.activate = true;
        this.onRegistrationSuccess(result);
      },
      (errorMessage) => {
        this.error = errorMessage;
        this.notificationService.showErrorToast(this.error, null);
      }
    );
    form.reset();
  }
}
```

Code 3.17: Registration Component for User Interface

```
import { ActivatedRoute, Router } from '@angular/router';
import { AuthService } from 'src/app/services/auth.service';

@Component({
  selector: 'app-activate',
  templateUrl: './activate.component.html',
  styleUrls: ['./activate.component.scss'],
})
export class ActivateComponent implements OnInit {
  constructor(
    private authService: AuthService,
    private route: ActivatedRoute,
    private router: Router
  ) {}

  ngOnInit(): void {}

  activateAccount() {
    let uid = this.route.snapshot.params['uid'];
    let token = this.route.snapshot.params['token'];
    this.authService.activateUser(uid, token).subscribe(
      () => {
        console.log('You have successfully activated your account');
        this.router.navigate(['login']);
      },
      (error) => {
        console.log(error);
      }
    );
  }
}
```

Code 3.18: Activation Component for User Interface

```
import { Router } from '@angular/router';
import { AuthService } from 'src/app/services/auth.service';
import { NotificationService } from 'src/app/services/notification.service';
@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss'],
})
export class LoginComponent implements OnInit {
  user = {
    email: '',
    password: '',
  };
  constructor(
    private authService: AuthService,
    private router: Router,
    private notificationService: NotificationService
  ) {}
  ngOnInit(): void {}
  onLoginSuccess(result: any) {
    localStorage.setItem('access_token', result.access);
    localStorage.setItem('refresh_token', result.refresh);
    this.notificationService.showSuccessToast('Login Successful', null);
    console.log(result);
  }
  onSubmit(form: NgForm) {
    const post_data = { ...this.user };
    if (!form.valid) {
      return;
    }
    this.authService.loginUser(post_data).subscribe(
      (result) => {
        this.onLoginSuccess(result);
        this.router.navigate(['dashboard']);
      },
      (error) => {
        console.log(error);
      }
    );
  }
}
```

Code 3.19: Login Component for User Interface

```
getMap() {
    /** Creates a leaflet map and adds controls to the map */
    this.myMap = L.map('map').setView([-0.0236, 37.9062], 6);
    this.myMap.zoomControl.remove();
    var osm = L.tileLayer(
      'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png',
      {
        maxZoom: 19,
        attribution:
          '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
      }
    );
    var CartoDb_Positron = L.tileLayer(
      'https://{s}.basemaps.cartocdn.com/light_all/{z}/{x}/{y}{r}.png',
      {
        attribution:
          '© <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors',
      }
    ).addTo(this.myMap);
    var satellite = L.tileLayer(
      'https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServ
er/tile/{z}/{y}/{x}',
      {
        attribution:
          'Tiles &copy; Esri &mdash; Source: Esri, i-cubed, USDA, USGS, AEX,
GeoEye, Getmapping, Aerogrid, IGN, IGP, UPR-EGP, and the GIS User Community',
      }
    );
    var basemaps = {
      Carto: CartoDb_Positron,
      OSM: osm,
      Satellite: satellite,
    };
    L.control.scale().addTo(this.myMap);
    L.control
      .zoom({
        position: 'bottomright',
      })
      .addTo(this.myMap);
```

```
    L.control
      .layers(basemaps, {}, { collapsed: true, position: 'bottomright' })
      .addTo(this.myMap);
  }

  addMapMarkers(geoData: any) {
    let markerGroup = L.markerClusterGroup();
    let oneMarker = L.geoJSON(geoData, {
      pointToLayer: function (feature: any, latlng: any) {
        return L.circleMarker(latlng, {
          radius: 10,
          fillOpacity: 1,
          color: 'black',
          fillColor: getColor(
            feature.properties.fatalities,
            feature.properties.injuries
          ),
          weight: 1,
        });
      },
    });
    oneMarker.addTo(markerGroup);
    markerGroup.addTo(this.myMap);
    this.addMapSidebarContent(markerGroup);
  }
```

Code 3.20: Excerpt of the Dashboard Component

3.3.4.2 Services

Services in angular are files that contain the logic for fetching data from the backend through the REST API. In the MapIED application there are two main services that were created: authentication service for fetching user data from the database (see Code 3.21) and the incident service (see Code 3.22) for fetching IED incidents from the backend and rendering it to the user through dashboard component for displaying using Leaflet.

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { IUser } from '../_models/user';
import { catchError, Observable, throwError } from 'rxjs';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root',
})
export class AuthService {
```

```
  constructor(private httpClient: HttpClient) {}

  registerUser(user: IUser): Observable<IUser> {
    return this.httpClient
      .post<IUser>(environment.REGISTRATION_URL, user, {
        responseType: 'json',
      })
      .pipe(
        catchError((errorRes) => {
          let errorMessage = 'An unknown error occurred!';
          switch (errorRes.error.email[0]) {
            case 'custom user with this email already exists.':
              errorMessage = 'This email already exists';
          }
          return throwError(errorMessage);
        })
      );
  }

  activateUser(uid: string, token: string): Observable<any> {
    let activation_data = { uid: uid, token: token };
    return this.httpClient.post(environment.ACTIVATION_URL, activation_data);
  }

  loginUser(user: any) {
    return this.httpClient.post(environment.LOGIN_URL, user, {
      headers: new HttpHeaders({
        'Content-Type': 'application/json',
      }),
    });
  }
}
```

Code 3.21: The Authentication Service

```
import { Injectable } from '@angular/core';
import { environment } from 'src/environments/environment';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/internal/Observable';

@Injectable({
  providedIn: 'root',
})
export class IncidentService {
  constructor(private httpClient: HttpClient) {}
```

```
  getAllIncidents(): Observable<any> {
    return this.httpClient.get(environment.ALL_INCIDENTS_URL);
  }

  getFilteredIncidents(
    start: string,
    end: string,
    counties: string
  ): Observable<any> {
    const searchParams = new URLSearchParams({
      from_date: start,
      to_date: end,
      counties: counties,
    });

    const finalUrl =
      environment.DATE_FILTERED_INCIDENTS_URL + `?${searchParams}`;
    return this.httpClient.get(finalUrl);
  }

  getFilteredCounties(county_id: string): Observable<any> {
    const filterParams = new URLSearchParams({
      selected_counties: county_id,
    });
    const finalUrl = environment.FILTERED_COUNTIES_URL + `?${filterParams}`;
    return this.httpClient.get(finalUrl);
  }

  getCountiesData(): Observable<any> {
    return this.httpClient.get(environment.COUNTIES_URL);
  }
}
```

Code 3. 22: The Incidents Service for fetching Incidents data

3.3.4.3 Models

Models in angular are interfaces for defining the fields that are expected from the backend so as to ensure consistency. Code 3.23 shows the models for the MapIED frontend.

```typescript
export interface ISubmission {
    description: string;
    date_of_incident: string;
    closest_town: string;
    week_day: string;
    county: any;
    assessed_target: string;
    incident_time: string;
    incident_type: any;
    category: any;
    fatalities: number;
    injuries: number;
    additional_details: string;
    location: any;
}

```

Code 3.23: Models for the MapIED frontend

3.3.5 Testing the REST API

The REST API was tested to ensure that data was being relayed properly from the backend to the frontend. This was done using Postman. Postman is a desktop API client tool that enables developers to test, share, and document APIs. This was done by creating HTTP requests and sending GET, PUT or POST requests to test them. Figure 3.4 is a snippet of the tests in Postman.

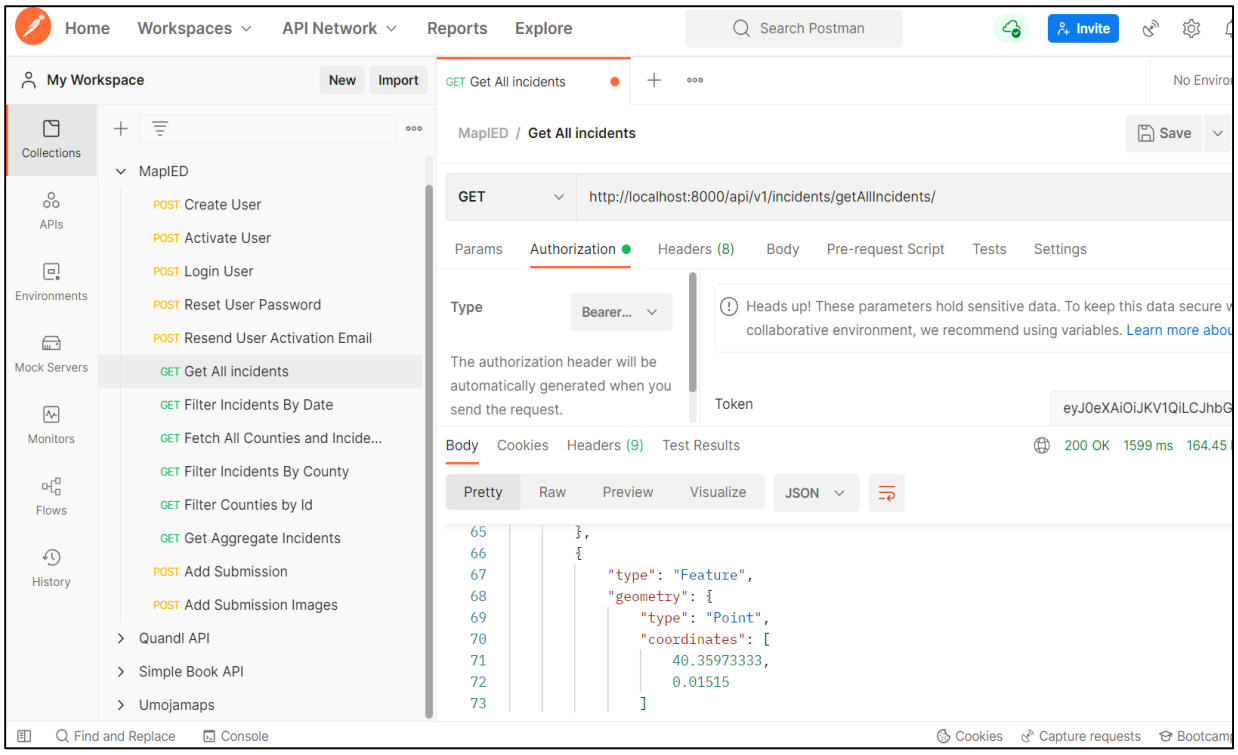


Figure 3.4: Testing the REST API with Postman

CHAPTER 4: RESULTS AND DISCUSSION

4.1 The MapIED System

The project delivered a web-based IED Incidents Mapping System (MapIED), with applications that provide functionalities for capturing, exploring, interacting with, editing, analyzing, visualizing and managing IED incidents data. Using UNMAS IED incidents data as the base data, ACLED data and Kenyan Security Agencies IED Incidents data were used to address inconsistencies and fill gaps in the base data. The result was a reliable, standardized and more complete IED incidents database for Kenya. The MapIED System developed has various elements that facilitate capture, interactivity, editing, storage, exploration, analysis, visualization and management of the IED incidents information.

4.1.1 Elements of the Map IED System

The key Components of the MapIED syatem include User Management, Database Management, Users Dashboard and Incidents Submission. Each of these elements is discussed separately.

4.1.1.1 User Management

This constitutes the User registration/Signup page and the login page. The Signup page facilitates the registration of new users into the system. Once a user registers, he is created in the system and can access the system through the user dashboard. The administrator can however, deactivate/delete a user account from the system.

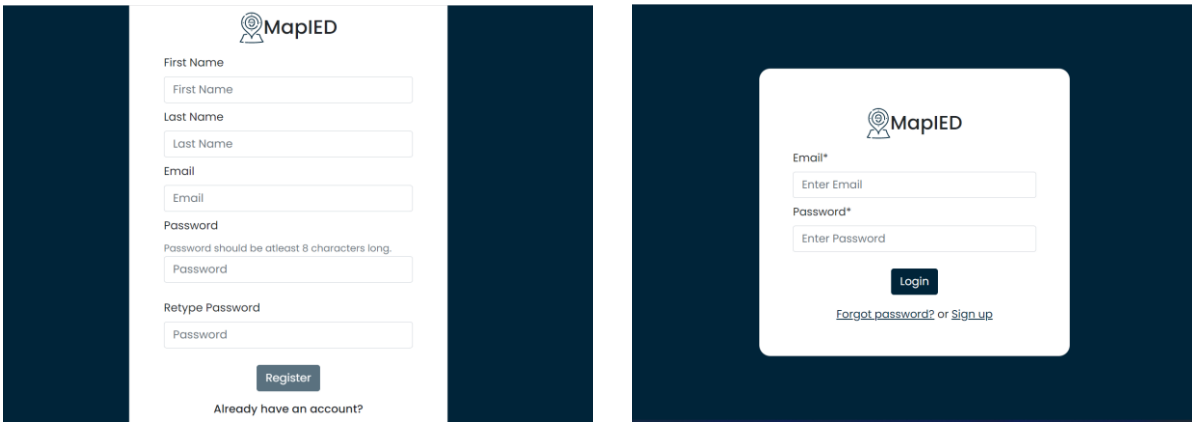


Figure 4.1: MaPIED SignUp and Login Pages

4.1.1.2 Database Management

The MapIED Admin Dashboard enables the System Administrator to manage users, add incidents data, edit data and review submissions by users.

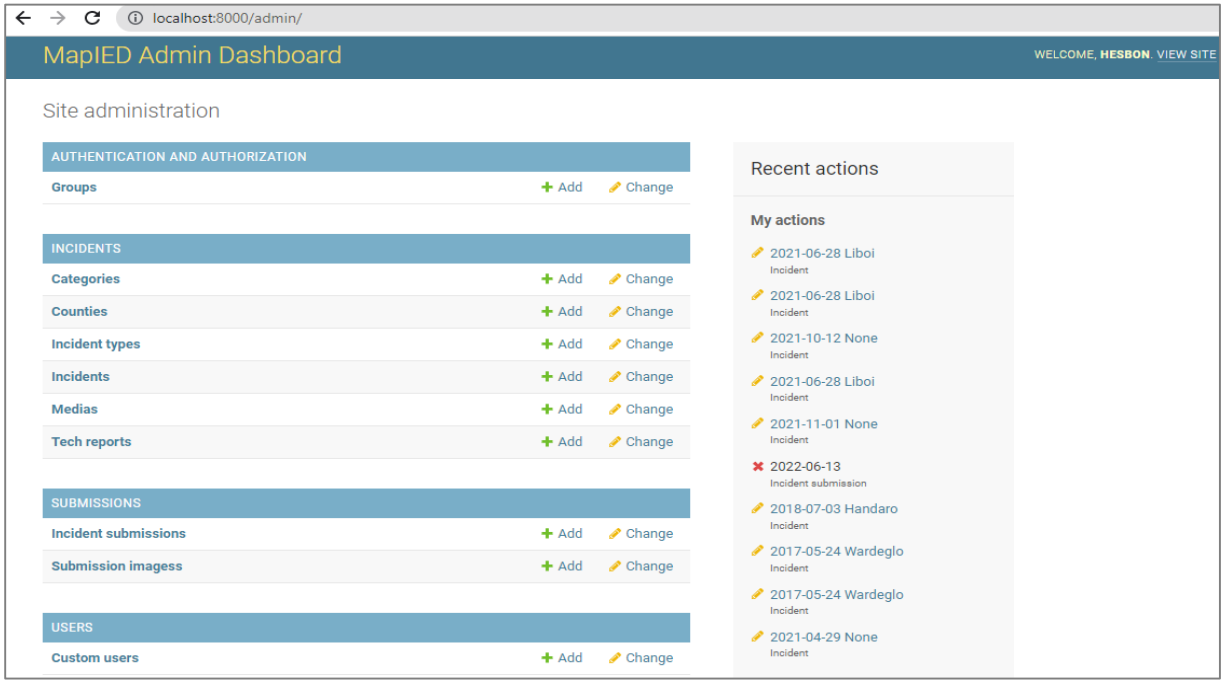


Figure 4.2: MapIEd Admin Dashboard Home Page

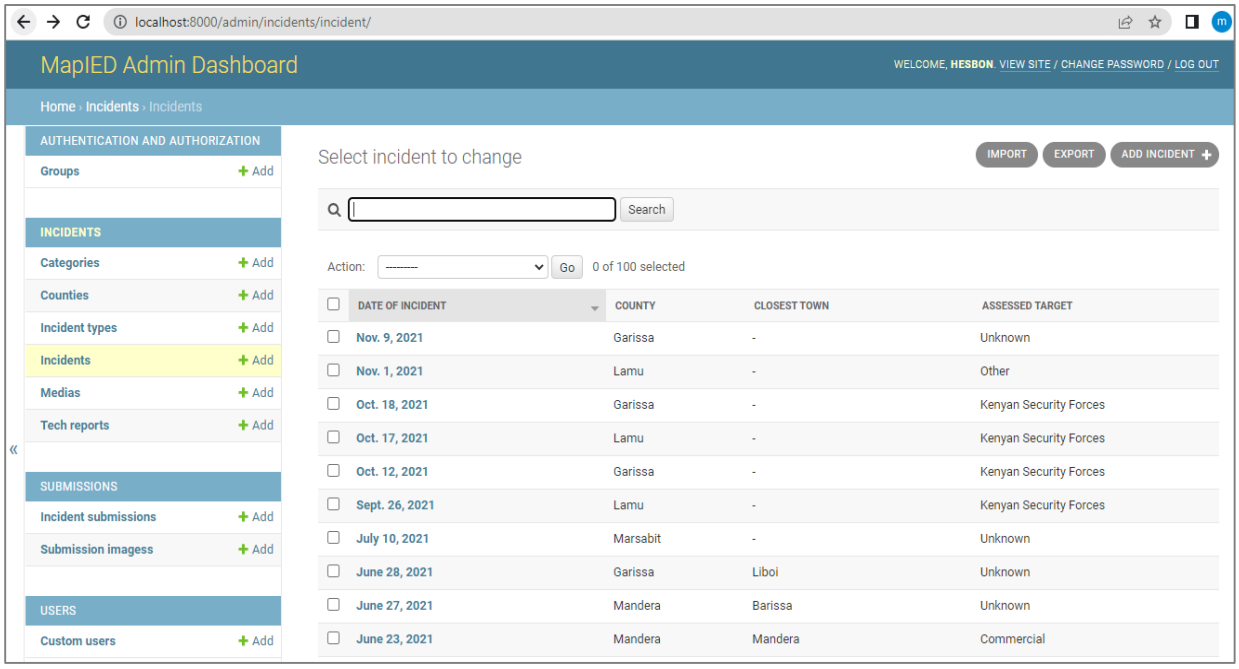


Figure 4.3: MapIEd Admin Incidents Page

4.1.1.3 Users Dashboard

The user dashboard is the graphical user interface through which users interact with the system. It displays IED incidents map and offers tools for navigating through the map, filtering incidents based on date, type, county, assessed target and number of casualties, toggling between base maps, switching views, reporting IED incidents and managing displays, including the incident analytics (trend graphs). Snippets of the dashboard configurations are shown in figures 4.4,4.5,4.6 and 4.7.

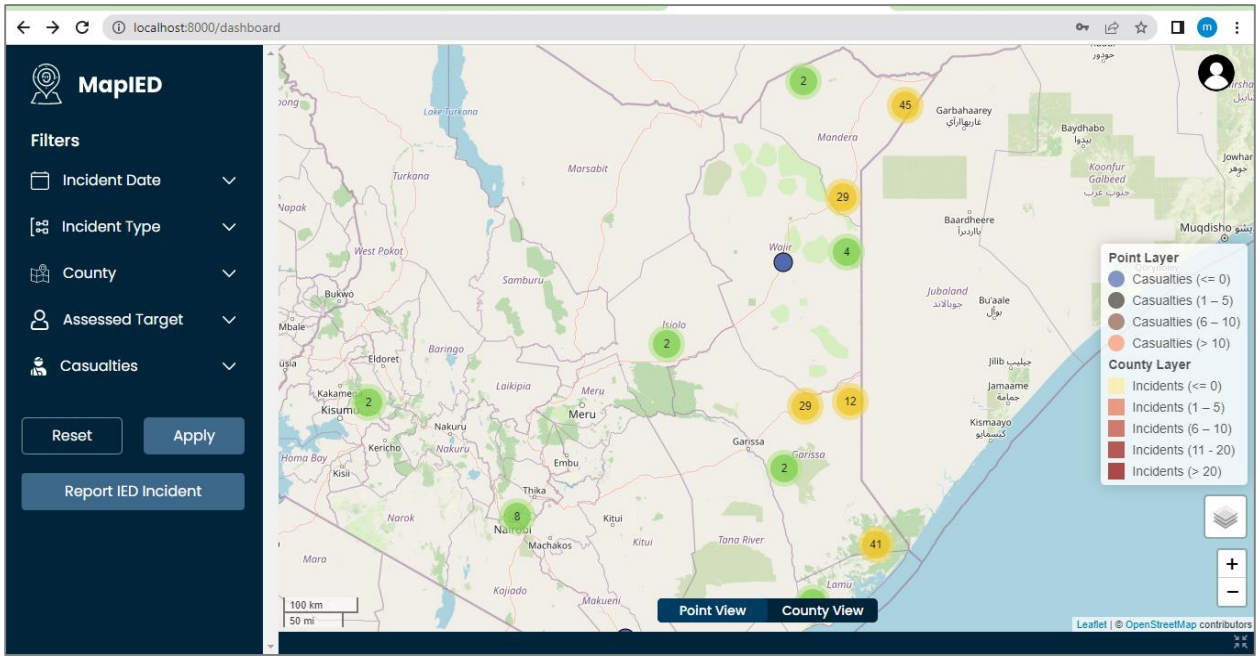


Figure 4.4: Users Dashboard – Point View

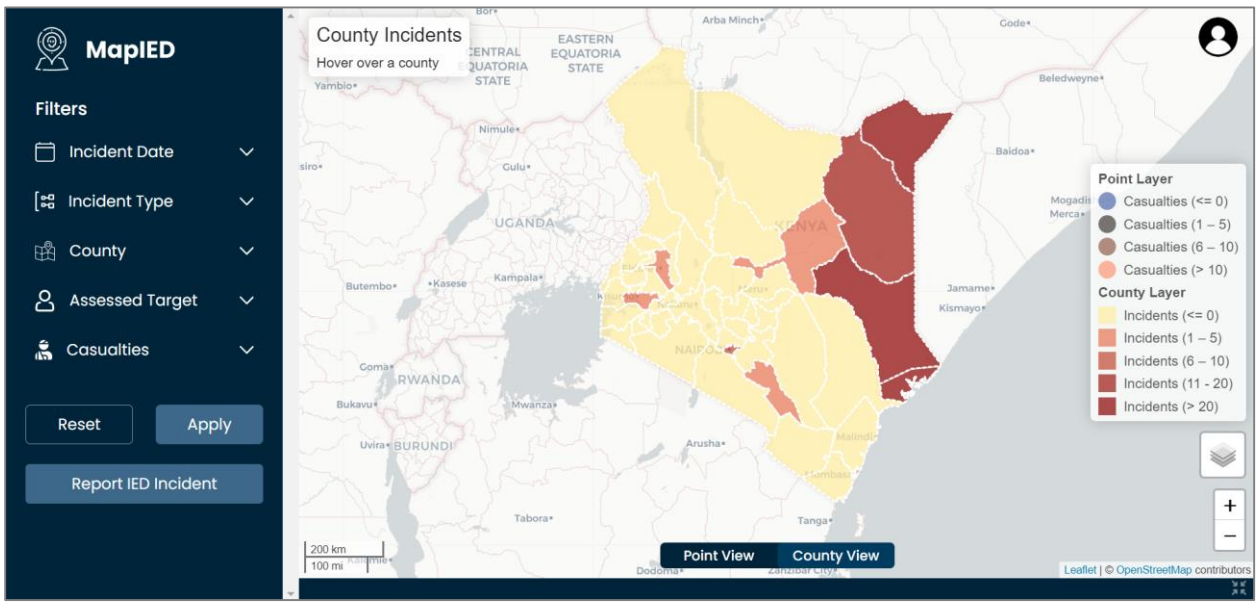


Figure 4.5: Users Dashboard – County View

When an incident is selected, a detailed description of the incident pops up, and images of the incident, and a technical/expert report (if available) are displayed. When the page is viewed at full extent, the graphical analysis of the incident data is displayed.

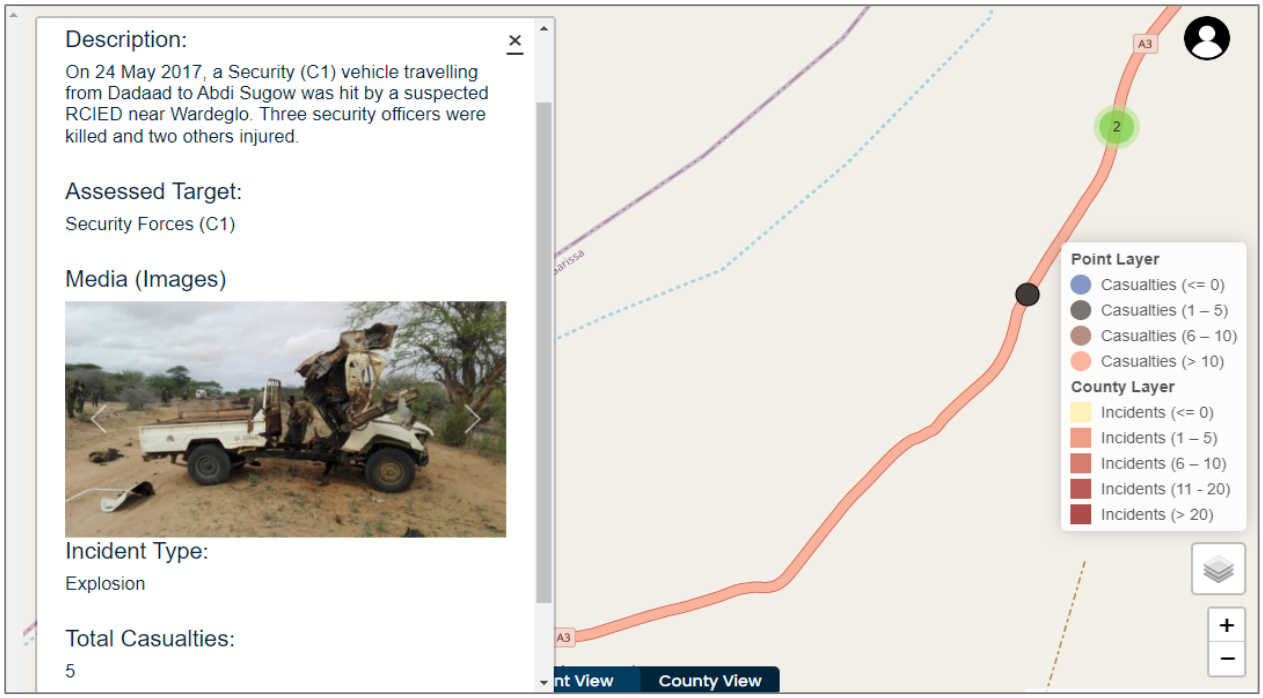


Figure 4.6: Display of incident information

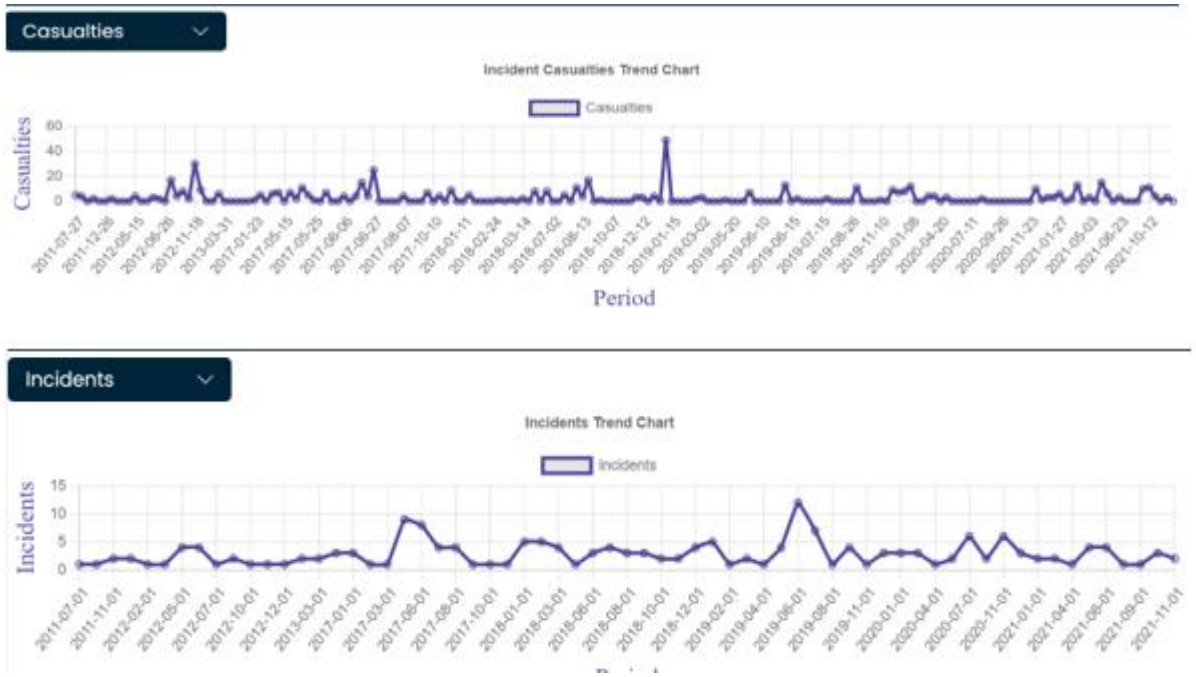


Figure 4.7: Incident Analysis display

4.1.1.3.1 Incidents Submission

This is one of the user dashboard functionalities. When the *Report Incident* button is clicked, it opens the IED incident submission dashboard that provides a form in which all required fields (and additional information) are filled and incident report submitted. A notification message is displayed following a submission (*report submitted successfully!)*

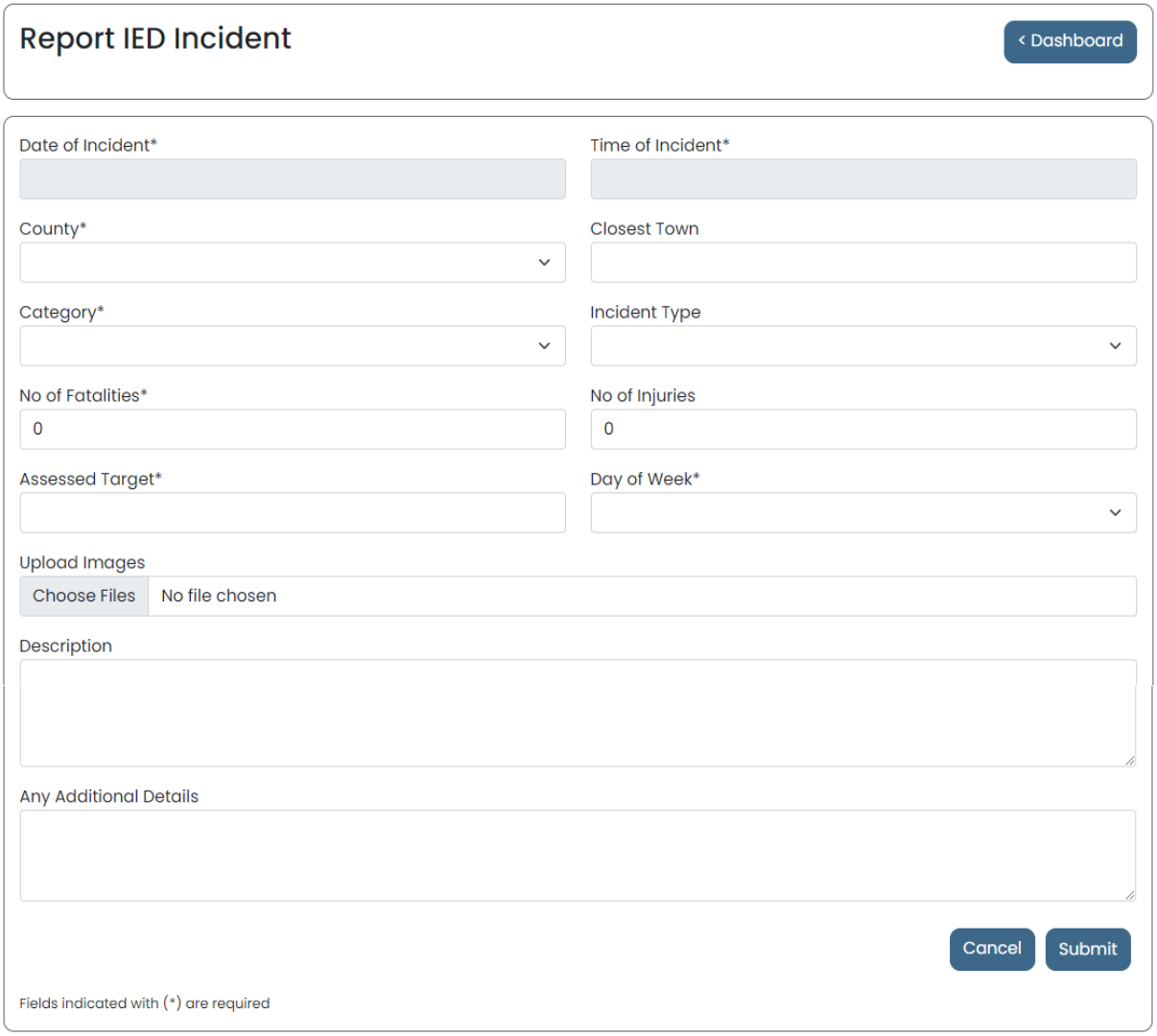


Figure 4.8: IED incident submission dashboard

4.2 Discussion of the Results

The results of this project demonstrate that a reliable IED Incidents Web-Mapping System that meets user needs is practical, particularly when a comprehensive user needs assessment is done, which then informs the system design. The spatial patterns of the IED incidents (locations where IEDs are emplaced or IEDs and IED materials found) is a very critical element of the MapIED System. This, upon analysis provides more insights into IED event activities, that together with further analysis with population demographics can generate valuable intelligence to facilitate Counter-IED efforts.

This system, though similar to the ACLED Web-Mapping System (available at https://acleddata.com/dashboard#/dashboard) that maps armed conflict events across the world, it is unique in the following aspects:

1. It is specific to a particular event type (IEDs), and therefore provides more comprehensive information on the same. While ACLED data has IED/Explosions as one its events, the level of detail is less that the developed MapIED System. This therefore offers capacity for more in-depth IED incidents analysis.
2. The MapIED System has a media capability that facilitates capture and display images of incidents. This provides opportunity for more analysis (especially technical analysis) on the IEDs' constructs/assembly, which generates critical information on the adversary's Tactics, Techniques and Procedures (TTPs) which facilitate own Counter-IED efforts.
3. The MapIED System provides has an Incident Submission dashboard that facilitates users to report IED incidents directly through the system. These reports are reviewed by the system administrator before being added to the system. The same can be used to observe inconsistencies/errors in the MapIED System data, which facilitates review and correction by admin. This is a unique feature that facilitates cleaning of the data, while maintaining integrity of the database report submissions do not POST to the database).
4. The provision to post technical reports (when available) of incidents is also critical to understanding of the adversary TTPs, and facilitates further analysis by other technical teams.

CHAPTER 5: CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The main objective of this project was to develop a Web Map for IED incidents in Kenya based on accurate, reliable and standardized IED data that was to be sourced from different security stakeholders. The project managed review available IED incident data, which was cleaned up/edited to remove errors, and standardized (based on fields and format) before it was used to create a Postgres/Postgis database in Django (python application). The project managed to deliver a MapIED System that hosts IED incidents database and serves incidents data objects through an interactive user interface on the web. The project particularly introduced critical capabilities that include contribution to the database by stakeholders through provision for users to report IED incidents, capability to accept media/incidents images and technical/expert IED incident reports. This is in addition to the usual visualization capability of web maps. The MapIED System is therefore a robust IED incidents Web-Mapping System that if deployed well, has the potential to significantly contribute to Counter IED efforts in Kenya.

5.2 Recommendations

The system can be used to serve different Counter-IED stakeholders that include security agencies, UNMAS, NGOs in IED hotspots and academia interested in Counter-IED studies. However, it is recommended that the system be deployed first on pilot basis, in a closed network at a smaller scale, preferably starting with security agencies before it can be rolled out to the full spectrum of its intended users in a wider web. This will provide opportunity to identify and address any issues that may not have been realized through the automated tests and data reviews performed during system development. Deployment in a closed network also helps to safeguard the project data, which is quite sensitive. This is as regulation protocols on management of the sensitive data outside the mainstream security agencies are formulated and implemented.

5.2.1 Recommendation for further research/study

It is further recommended that based on the spatial patterns of IED emplacement locations depicted by the MapIED system, further analysis of the IED prone locations need to be done. This is particularly in respect to terrain analysis to determine terrain factors that influence the choice of attack locations. This will generate useful information that can be utilizes in IED incidents

prediction systems to more accurately predict where IED incidents are likely to occur in future. Also, demographic analysis to establish the human factors' influence on the overall IED Incident Support System. These were aspects that though important, could not be explored in the scope of this project owing to the time required to comprehensively address them, which could not be met within the project timelines.

REFERENCES

Django Software Foundation (2022). Django Documentation. https://docs.djangoproject.com/en/4.0/ (Last accessed on 20 June, 2022)

EWS Consultancy (2022). Counter Threat and C-IED Capability Development and Training. https://solutions-ew.com/counter-threat-and-c-ied-capability-development-and-training-2/ (Accessed on 08 Jan 2022)

Google Inc. (2010-2022). Angular Documentation. https://devdocs.io/angular/ . (Last accessed on 20 June, 2022)

Michael Dorman, (2021). Introduction to Web Mapping. CRC Press. https://web-mapping.surge.sh/ (Accessed on 20 February, 2022)

Michaud, Y. (2013). The Silver Bullet. The Comprehensive Approach Towards Counter Improvised Explosive Devices. https://www.cfc.forces.gc.ca/259/290/299/286/michaud_y.pdf (Accessed on 19 February, 2022)

NATO (2011). Allied Joint Doctrine for Countering – Improvised Explosive Devices. https://www.dtra.mil/Portals/61/Documents/Missions/NATO%20AJP-3.15(A)%20ALLIED%20C-IED%20MAR%2011.pdf?ver=2017-03-10-134619-480 (Accessed on 20 February, 2022)

UNMAS (2020). Somali Counter IED Annual Report, 2020. https://reliefweb.int/report/somalia/unmas-somalia-annual-report-2020-explosive-hazard-analysis-report (Accessed on 20 February 2022)

UNITED NATIONS (2021). Assistance in Mine Action: Report of the Secretary General. United Nations Digital Library. https://digitallibrary.un.org/record/3938660?ln=en (Accessed on 19 February, 2022)

United Nations (2013). DPKO-DFS Guidelines on Improvised Explosive Device (IED) Threat Mitigation in Mission Settings. https://www.unmas.org/sites/default/files/documents/ied_threat_mitigation_guidelines.pdf (Accessed on 20 February, 2022)

US Department of Homeland Security (2013). IEDs fact sheet. https://www.dhs.gov/xlibrary/assets/prep_ied_fact_sheet.pdf (Accessed on 19 February, 2022)