**CRISIS GUARD**

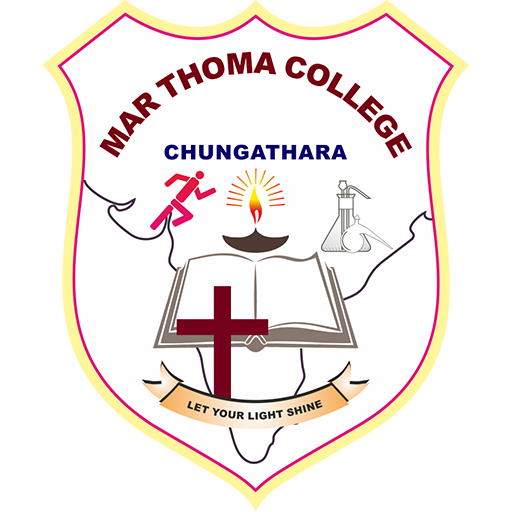
**Project Report**

**Submitted in partial fulfilment of the requirements for the degree of**

**B.Sc. Computer Science**

Submitted by

**Shahan V Saleem (Reg. No. MAAWSCS009)**



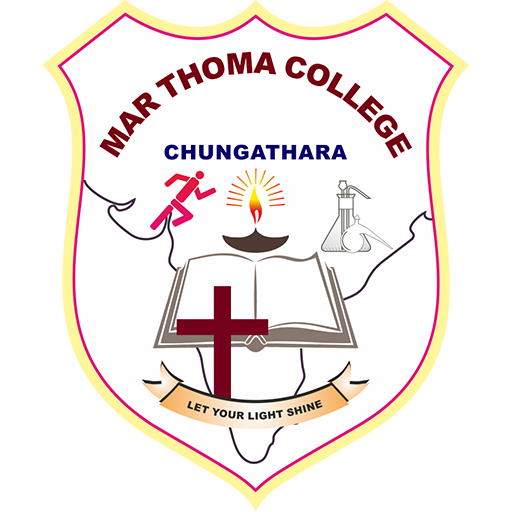
**MAR THOMA COLLEGE, CHUNGATHARA**

**(Affiliated to Calicut University)**

**2022 – 2025**

**MAR THOMA COLLEGE, CHUNGATHARA**

**(Affiliated to Calicut University)**



**DEPARTMENT OF COMPUTER SCIENCE**

***Certificate***

This is to certify that the project report titled “**CRISIS GUARD**” is a record of the project done by **Shahan V Saleem (MAAWSCS009)** submitted to **Mar Thoma College, Chungathara** as a partial fulfilment of the requirements for the award of the degree of **B.Sc. Computer Science** from Calicut University from 2022 to 2025 under the supervision of **Mr. SREEKANTH M**, **Mar Thoma College, Chungathara**

Submitted for the viva voice held on ……….

**Head of Department External Examiner**

**ACKNOWLEDGEMENT**

The success of the project depends upon the effort invested. It's my duty to acknowledge and thank the individuals who have contributed in the successful completion of the project.

First and foremost, I praise the God Almighty for the grace he showered on my studies as well as day to day life activities.

I wish to express my heartfelt gratitude to **Prof. Dr. Rajeev Thomas, Principal, Mar Thoma College Chungathara,** for the encouragement and inspiring guidance throughout the preparation of this project.

Moreover, I am deeply indebted to **Mr. Sreekanth. M, Head of the Department of Computer Science**, for his invaluable guidance

, contribution and encouragement which helped me a lot to complete this project successfully.

I finally give a bunch of thanks to my friends and all my well-wishers who had supported me directly and indirectly throughout the project with their constructive criticisms.

**Shahan V Saleem**

**ABSTRACT**

Crisis Guard is an innovative disaster management system designed to enhance the efficiency and effectiveness of disaster preparedness, response, and recovery efforts. In the face of increasing natural and man-made disasters, the system addresses the critical need for timely communication, coordination, and resource management. By integrating real-time weather prediction, Crisis Guard enables authorities to issue early warnings, implement evacuation plans, and mobilize resources proactively, thereby minimizing loss of life and property damage.

The system supports multiple user roles, including Admin, Camp Coordinators, Volunteers, Public, and Emergency Response Teams, each with tailored functionalities to ensure a seamless and coordinated response. Key features include disaster-prone area mapping, resource tracking, medical support coordination, and emergency request handling. Crisis Guard also emphasizes community involvement, enabling volunteers and the public to contribute effectively during crises.

Built on a robust technology stack comprising Python-Django, Flutter, and MySQL, Crisis Guard offers a user-friendly interface and scalable architecture. By automating traditional manual processes and leveraging real-time data, the system ensures faster response times, better resource allocation, and improved disaster recovery outcomes. Crisis Guard represents a significant step forward in modernizing disaster management, making communities more resilient in the face of adversity.

**MODULES**

* **ADMIN**
  + Add and Manage Camp
  + Add and Manage Camp
  + Add and Manage Coordinator
  + Send Guidelines
  + Verify and Manage Emergency Team
  + Send Notifications
  + View Donations
  + Reply to Complaints
* **Camp Coordinator**
  + Add and Manage Camp Stock
  + Add and Manage Members
  + Register Missing Assets
  + Add and Manage Camp Needs
  + Add / Verify and Manage Volunteers
  + Group Chat with Volunteers
  + Assign Task to Volunteers
  + View Notifications
  + Edit Profile
* **Emergency Team**
  + Registration
  + View Emergency Requests
  + View Camps
  + View Notifications
  + Landslide Prediction
* **Volunteer**
  + View Camp Needs
  + Collect Donated Goods
  + View Assigned Tasks
  + Manage Missing Assets
  + Group Chat
  + View Notification
  + Edit Profile
* **Public**
  + Registration
  + View Needs
  + Donate Goods
  + Volunteer Registration
  + Pay Donations
  + Send Emergency Requests
  + Send Complaints
  + View Notifications
  + Edit Profile

**CONTENTS**

1. INTRODUCTION
   1. OVERVIEW OF THIS SYSTEM
   2. OBJECTIVE
2. SYSTEM ANALYSIS
   1. INTRODUCTION
   2. EXISTING SYSTEM
      1. DISADVANTAGES OF EXISTING SYSTEM
   3. PROPOSED SYSTEM
      1. ADVANTAGES OF PROPOSED SYSTEM
   4. FEASIBILITY STUDY
      1. INTRODUCTION
      2. ECONOMIC FEASIBILITY
      3. TECHNICAL FEASIBILITY
      4. OPERATIONAL FEASIBILITY
   5. ANALYSIS MODELLING
      1. DATA FLOW DIAGRAM
      2. ER DIAGRAM
   6. SYSTEM SPECIFICATION
      1. HARDWARE REQUIREMENTS
      2. SOFTWARE REQUIREMENTS
      3. SOFTWARE DESCRIPTION
3. SYSTEM DESIGN
   1. INTRODUCTION
   2. INPUT DESIGN
   3. OUTPUT DESIGN
   4. DATABASE DESIGN
      1. NORMALIZATION
      2. TABLE DESIGN
   5. MODULE DESCRIPTION
4. SYSYEM TESTING AND IMPLEMENTATION
   1. SYSTEM TESTING
      1. TYPES OF TESTING
      2. TESTING STRATEGY
   2. SYSTEM IMPLEMENTATION
5. SYSTEM MAINTENANCE AND SECURITY
   1. SYSTEM MAINTENANCE
   2. SYSTEM SECURITY
6. CONCLUSION
7. APPENDIX
   1. DATA FLOW DIAGRAM
   2. ER DIAGRAM
   3. CODE
   4. SCREENSHORT
8. FUTURE SCOPE
9. BIBLIOGRAPHY

INTRODUCTION

* 1. **OVERVIEW OF THIS SYSTEM**

Crisis Guard represents a groundbreaking solution in the field of disaster management, transforming the way communities prepare for, respond to, and recover from disasters. By integrating real-time data analysis, advanced weather prediction, and active community involvement, Crisis Guard redefines disaster response mechanisms. The platform empowers users—ranging from government authorities to local volunteers—to collaborate seamlessly, ensuring a swift and coordinated response during emergencies.

At its core, Crisis Guard enables real-time disaster monitoring and resource allocation, allowing stakeholders to identify vulnerable areas, track resource availability, and respond to emergencies proactively. The system supports multiple user roles, including Admin, Camp Coordinators, Volunteers, Public, and Emergency Response Teams, each contributing to a unified disaster management ecosystem. Whether it’s issuing early warnings, managing relief camps, or coordinating volunteer efforts, Crisis Guard ensures that every action is timely, efficient, and impactful.

What sets Crisis Guard apart is its emphasis on community-driven disaster management. By involving the public in reporting emergencies, donating resources, and volunteering, the platform fosters a sense of collective responsibility. This decentralized approach not only accelerates response times but also ensures that no community is left behind during crises. Additionally, Crisis Guard’s landslide prediction module leverages advanced algorithms to assess risks and provide actionable insights, further enhancing its ability to mitigate disasters.

Crisis Guard is more than just a disaster management tool—it’s a dynamic platform that evolves with every disaster. By continuously learning from real-world data and user feedback, the system improves its accuracy and effectiveness over time. This commitment to innovation and collaboration sets a new standard in disaster management, promising a future where communities are better prepared, more resilient, and capable of facing disasters with confidence.

* 1. **OBJECTIVE**

The primary goal of Crisis Guard is to revolutionize disaster management by leveraging technology to enhance preparedness, response, and recovery efforts.

The specific objectives of the system are as follows:

* Real-Time Disaster Monitoring: Develop a system capable of monitoring disaster-prone areas in real-time, enabling early detection and timely warnings for events such as floods, landslides, and extreme weather conditions.
* Resource Allocation and Management: Create a centralized platform to track and manage resources such as food, medical supplies, and emergency kits, ensuring optimal distribution during disasters.
* Community Engagement: Foster active participation from the public by enabling them to report emergencies, donate resources, and volunteer for relief efforts, ensuring a collective response to disasters.
* Emergency Response Coordination: Design a system that facilitates seamless communication and coordination between Admin, Camp Coordinators, Volunteers, Public, and Emergency Response Teams, ensuring a unified and efficient response.
* Landslide Prediction: Implement advanced algorithms to predict landslide risks, providing actionable insights for disaster mitigation and evacuation planning.
* User-Friendly Interface: Develop an intuitive and accessible platform that can be easily used by individuals with varying levels of technical expertise, ensuring widespread adoption.
* Scalability and Adaptability: Build a scalable system that can be adapted to different disaster scenarios and geographic regions, making it a versatile tool for global disaster management.
* Data-Driven Decision Making: Utilize real-time data and analytics to support informed decision-making by authorities, ensuring a proactive and effective response to disasters.
* Continuous Improvement: Incorporate a feedback mechanism to allow users to report issues, suggest improvements, and validate system outputs, ensuring the platform evolves and improves over time.

By achieving these objectives, Crisis Guard aims to save lives, reduce property damage, and build resilient communities capable of facing disasters with confidence.

SYSTEM ANALYSIS

* 1. **INTRODUCTION**

The Crisis Guard system is a cutting-edge solution designed to enhance disaster preparedness, response, and recovery through the integration of real-time data analysis, community involvement, and advanced prediction algorithms. By leveraging modern technologies, the system automates the process of disaster monitoring, resource allocation, and emergency coordination, ensuring a swift and effective response to crises. With features like real-time weather prediction, landslide risk assessment, and community-driven reporting, Crisis Guard empowers stakeholders—ranging from government authorities to local volunteers—to collaborate seamlessly during disasters. The system’s user-friendly interface and scalable architecture make it a versatile tool for managing various disaster scenarios, ultimately contributing to safer and more resilient communities.

The various tasks in the system analysis include the following:

1. Requirement Elicitation: Gather and document the needs of stakeholders, including government authorities, volunteers, and the public.
2. System Modelling: Design the system architecture, workflows, and user roles to ensure seamless functionality.
3. Data Collection and Analysis: Collect real-time data from weather sensors, user reports, and other sources to support decision-making.
4. Algorithm Selection and Design: Develop algorithms for weather prediction, landslide risk assessment, and resource allocation.
5. User Interface Design: Create an intuitive and accessible interface for users with varying levels of technical expertise.
6. Performance Evaluation: Test the system’s accuracy, speed, and reliability in real-world disaster scenarios.
7. Privacy and Security Assessment: Implement robust data privacy and security measures to protect user information.
8. Integration and Deployment: Ensure seamless integration with existing disaster management systems and deploy the platform for widespread use.
   1. **EXISTING SYSTEM**

The existing disaster management systems rely heavily on manual processes and fragmented communication, leading to delays and inefficiencies during emergencies. These systems often involve paper-based records, limited real-time data integration, and minimal community involvement. While some platforms allow users to report emergencies, they lack advanced features like real-time weather prediction, landslide risk assessment, and automated resource allocation. As a result, the response to disasters is often slow, uncoordinated, and inadequate, leaving communities vulnerable to significant loss of life and property damage.

* + 1. **DISADVANTAGES OF EXISTING SYSTEM**
* Limited Real-Time Data Integration: Existing systems lack real-time data analysis, making it difficult to issue timely warnings and coordinate responses.
* Fragmented Communication: Poor communication between stakeholders leads to delays and inefficiencies during emergencies.
* Minimal Community Involvement: Limited opportunities for public participation reduce the effectiveness of disaster response efforts.
* Manual Processes: Reliance on manual record-keeping and resource allocation slows down response times and increases the risk of errors.
* Lack of Advanced Features: Absence of features like landslide prediction and automated resource management limits the system’s ability to handle complex disaster scenarios.
  1. **PROPOSED SYSTEM**

The Crisis Guard system aims to revolutionize disaster management by automating processes, integrating real-time data, and fostering community involvement. The proposed system leverages advanced algorithms for weather prediction and landslide risk assessment, enabling authorities to issue early warnings and implement evacuation plans proactively. It also includes features like real-time resource tracking, emergency request handling, and community-driven reporting, ensuring a coordinated and efficient response to disasters. By empowering stakeholders—including Admin, Camp Coordinators, Volunteers, Public, and Emergency Response Teams—Crisis Guard creates a unified platform for disaster management, making communities more resilient and better prepared to face future challenges.

* + 1. **ADVANTAGES OF PROPOSED SYSTEM**
* Real-Time Monitoring: Enables real-time tracking of disaster-prone areas and weather conditions for timely warnings.
* Efficient Resource Allocation: Automates resource tracking and distribution, ensuring optimal use of supplies during emergencies.
* Community Involvement: Encourages public participation in reporting emergencies, donating resources, and volunteering for relief efforts.
* Advanced Prediction Algorithms: Uses machine learning and AI to predict landslides and assess risks, providing actionable insights for disaster mitigation.
* User-Friendly Interface: Offers an intuitive and accessible platform for users with varying levels of technical expertise.
  1. **FEASIBILITY STUDY**
     1. **INTRODUCTION**

Feasibility is defined as the practical extent to which a project can be performed successfully. The objective of a feasibility study is to establish the reasons for developing the software, ensuring it is acceptable to users, adaptable to changes, and conforms to established standards. A feasibility study allows developers to foresee the future of the project and its usefulness. Key aspects of the feasibility study include:

* Determining whether a new system is required or not.
* Identifying the potential and drawbacks of the existing system.
* Understanding user requirements and the benefits expected from the new system.
* Exploring various alternatives available for system development.
* Defining the objectives and components of the proposed system.
* Ensuring the proposed system meets the end needs of the users.
* Assessing the technical, economic, and operational feasibility of the proposed system.

The feasibility study for Crisis Guard considers the following types of feasibility:

1. Economic Feasibility
2. Technical Feasibility
3. Operational Feasibility
   * 1. **ECONOMIC FEASIBILITY**

Economic feasibility determines whether the proposed system is capable of generating financial gains or cost savings for an organization. It involves analysing the costs incurred during software development, including hardware, software, and human resources, and comparing them to the expected benefits.

For Crisis Guard, the economic feasibility is justified as follows:

* The system reduces operational costs by automating manual processes, such as resource allocation and emergency response coordination.
* It minimizes losses during disasters by enabling timely warnings and efficient resource management.
* The long-term benefits, such as improved community resilience and reduced property damage, outweigh the initial development costs.
* The system’s scalability ensures it can be adapted to various regions, making it a cost-effective solution for disaster management.

Thus, Crisis Guard is economically feasible as it provides significant long-term benefits while keeping development and operational costs manageable.

* + 1. **TECHNICAL FEASIBILITY**

Technical feasibility assesses whether the current resources (hardware, software, and technology) are sufficient to develop and implement the proposed system within the allocated time and budget. It also evaluates whether the proposed system can be integrated with existing technologies.

For Crisis Guard, the technical feasibility is analysed as follows:

* The system is built using widely available technologies such as Python-Django, Flutter, and MySQL, which are cost-effective and scalable.
* The hardware requirements (e.g., standard computers, mobile devices, and servers) are minimal and easily accessible.
* The system can be deployed on existing infrastructure, reducing the need for additional investments.
* The use of real-time data analysis and advanced algorithms ensures the system is technically robust and capable of handling complex disaster scenarios.

Given the availability of resources and technologies, Crisis Guard is technically feasible.

* + 1. **OPERATIONAL FEASIBILITY**

Operational feasibility evaluates whether the proposed system can be effectively implemented and operated within the organization. It considers factors such as user acceptance, ease of use, and the system’s ability to meet user requirements.

For Crisis Guard, the operational feasibility is assessed as follows:

* The system is designed with a user-friendly interface, making it accessible to users with varying levels of technical expertise.
* It supports multiple user roles (e.g., Admin, Camp Coordinators, Volunteers, Public, and Emergency Response Teams), ensuring it meets the needs of all stakeholders.
* The system’s emphasis on community involvement ensures widespread adoption and participation.
* Training programs and documentation will be provided to help users adapt to the new system.

Since Crisis Guard is designed to be intuitive and aligns with the operational needs of disaster management, it is operationally feasible.

* 1. **ANALYSIS MODELLING**
     1. **DATA FLOW DIAGRAM**

A data flow diagram (DFD) is a visual tool that illustrates how information flows through a process or system. It uses symbols such as rectangles, circles, and arrows, along with text labels, to represent data inputs, outputs, storage points, and the routes between them. DFDs can be classified into two types - logical and physical. The logical data flow diagram explains how data flows through a system to achieve specific business objectives, while the physical data flow diagram explains how the logical data flow is implemented. DFDs graphically depict the functions or processes involved in capturing, manipulating, storing, and distributing data between a system and its environment, as well as between different system components. As a result, they are a valuable communication tool for both users and system designers.

Data Flow Diagram uses the following symbols:

A rectangle is used to represent an external entity. They are the sources and destinations of information entering or leaving the system.

An open rectangle is used to represent a data store. They are the files or repositories that hold information, such as a database table or a form.

Represent processes that sow transformation or manipulation of data within the system

An arrow is used for showing the route the data takes between the external entities.

* + 1. **ER DIAGRAM**

An ER diagram, or Entity-Relationship diagram, is a visual representation used in database design to illustrate the logical structure of a database system. It depicts entities as rectangles, representing real-world objects, with attributes shown as ovals connected to their respective entities. Relationships between entities are represented as lines connecting them, depicting how they are related in the database schema, whether as one-to-one, one-to-many, or many-to-many. ER diagrams are essential for understanding the relationships between different entities within a database, aiding in the effective planning, building, and maintenance of databases by providing a clear blueprint of the database structure.

Entity-Relationship Diagram uses the following symbols:

These shapes are independent from other entities and are also called parent entities. They may also have a primary key and some weak entities depend on them.

Weak entities depend on some other entity type. They do not have any primary keys, and cannot exist without their parent entity.

Associative entities relate the instances of several entity types. They also contain attributes specific to the relationship between those entity instances.

Weak relationships are connections between a weak entity and its owner

Relationships refer to the associations between or among entities.

Derived attributes contain values that are calculated from related attribute values

Multivalued attributes are the attributes that can take more than one value.

An attribute is used for showing the characteristics or properties of an entity

* 1. **SYSTEM SPECIFICATION**
     1. **HARDWARE REQUIREMENT**

The hardware is the fundamental component where all the information and data are stored, both temporarily and permanently. Therefore, the hardware must be both dependable and cost-effective, and capable of supporting all application development. It should have the necessary speed to complete all the required jobs and executions efficiently.

Minimum Configuration

Processor: Core i3 or above CPU

Clock Speed: 2 GHz or higher

RAM: 8 GB or more

Hard Disk: 256 GB or more

Keyboard: Standard QWERTY 106 key

Mouse: Optical

Monitor: RGB

* + 1. **SOFTWARE REQUIREMENT**

The software specification encompasses the operating system and all additional applications and tools utilized in the development of the proposed system. This includes the operating system as well as any software employed in the process.

Operating System: Windows 8 or higher

Front End: HTML, Tailwind CSS, JavaScript, AJAX, Flutter Dart

Back End: MySQL, Python, Django

Browser: Google Chrome

IDE: Android Studio, Python 3.10, Visual Studio Code, SQLyog

* + 1. **SOFTWARE DESCRIPTION**

**Frontend:**

* **HTML (Hypertext Markup Language):** The standard markup language used to create and design documents on the World Wide Web. It provides the structure and content of web pages.
* **Tailwind CSS:** A utility-first CSS framework that provides pre-designed, low-level utility classes to style web interfaces quickly and efficiently.
* **JavaScript**: A high-level, interpreted programming language used to create dynamic and interactive web content.
* **AJAX (Asynchronous JavaScript and XML):** A set of web development techniques used to create interactive and dynamic web applications by exchanging data with a web server asynchronously.
* **Flutter Dart:** A programming language used with the Flutter framework to build cross-platform mobile applications with a single codebase.

**Backend:**

* **MySQL:** An open-source relational database management system (RDBMS) used to store and manage data efficiently.
* **Python:** A high-level, interpreted programming language known for its simplicity, readability, and versatility.
* **Django:** A high-level Python web framework that encourages rapid development and clean, pragmatic design.

**Development Tools:**

* **Android Studio:** An integrated development environment (IDE) used for developing Android applications.
* **Python 3.10:** The version of Python used for backend development, offering improved performance and new features.
* **Visual Studio Code:** A lightweight yet powerful source code editor developed by Microsoft, supporting multiple programming languages and frameworks.
* **SQLyog:** A graphical user interface (GUI) tool for managing MySQL databases, providing an intuitive interface for database operations.
* **Google Chrome:** A widely used web browser known for its speed, security, and compatibility with modern web technologies.

SYSTEM DESIGN

* 1. **INTRODUCTION**

System design is the process of defining the architecture, components, modules, and interfaces of a system to satisfy specified requirements. It involves transforming requirements gathered during system analysis into a blueprint that outlines how the system will be structured and how its various components will interact with each other to achieve the desired functionality. System design encompasses both high-level and low-level design decisions, addressing aspects such as system architecture, data storage, processing logic, user interface design, and integration with external systems.

During system design, designers consider factors such as scalability, performance, reliability, security, and maintainability to ensure that the system meets the needs of its users and stakeholders. They select appropriate technologies, frameworks, and design patterns to implement the system efficiently and effectively. System design also involves creating detailed specifications, diagrams, and documentation to communicate the design decisions to developers, testers, and other stakeholders involved in the development process.

Overall, system design plays a crucial role in the software development lifecycle, laying the foundation for building robust, scalable, and maintainable systems that meet the requirements and expectations of users and stakeholders. It bridges the gap between system analysis and system implementation, providing a roadmap for developers to follow during the development phase.

* 1. **INPUT DESIGN**

Input design refers to the process of creating a user interface through which users can input data into a system effectively and efficiently. It involves determining the types of data to be collected, designing forms or screens to capture the data, and defining validation rules and error handling mechanisms to ensure data accuracy and integrity. Input design considers factors such as user experience, usability, accessibility, and error prevention to create interfaces that are intuitive and user-friendly. By carefully designing input mechanisms, organizations can streamline data entry processes, minimize errors, and enhance overall system usability, ultimately improving user satisfaction and productivity.

The objectives of input design in system development include:

* 1. **Accuracy**: Ensuring that the input data captured accurately reflects the information intended by the user, minimizing errors and inconsistencies.
  2. **Completeness:** Collecting all necessary data required for the system to perform its functions effectively and comprehensively.
  3. **Efficiency:** Designing input mechanisms that allow users to input data quickly and with minimal effort, optimizing user productivity and system performance.
  4. **Clarity:** Creating clear and intuitive user interfaces that guide users through the data entry process, reducing confusion and the likelihood of input errors.
  5. **Consistency:** Maintaining consistency in input formats, layouts, and terminology across different parts of the system to enhance usability and user experience.
  6. **Validation:** Implementing validation checks and error handling mechanisms to verify the integrity and validity of input data, preventing incorrect or incomplete information from being processed.
  7. **Flexibility:** Designing input interfaces that can accommodate changes and updates to the system requirements or data input formats without significant disruption or rework.
  8. **Security:** Incorporating security measures to protect sensitive data during input, transmission, and storage, safeguarding against unauthorized access or data breaches.
  9. **OUTPUT DESIGN**

Output design refers to the process of presenting processed data or information to users in a meaningful and effective format. It involves determining the content, layout, and presentation of output reports, screens, or displays to meet the needs and expectations of users and stakeholders. Output design aims to provide users with relevant and actionable information in a clear, concise, and visually appealing manner, facilitating decision-making and enhancing user satisfaction. This process includes considerations such as formatting, visualization techniques, language, and accessibility to ensure that the output is understandable, accessible, and useful to the intended audience. Additionally, output design may involve customization options, such as user preferences or personalization features, to accommodate individual user needs and preferences. Overall, effective output design plays a crucial role in maximizing the usability, usefulness, and impact of the information provided by a system.

* 1. **DATABASE DESIGN**

Database design is the process of structuring and organizing data in a database system to ensure efficient storage, retrieval, and management of information. It involves defining the structure of tables, relationships between tables, data types, constraints, and indexes to meet the requirements of the application and optimize performance. Database design aims to minimize redundancy, ensure data integrity, and facilitate data manipulation and analysis. By carefully planning and designing the database schema, developers can create a robust and scalable database system that meets the needs of the organization and supports its business processes effectively.

* + 1. **NORMALIZATION**

Normalization is a database design technique aimed at organizing data efficiently in relational databases. By reducing redundancy and dependency, normalization enhances data integrity and minimizes anomalies during data manipulation. It involves a series of steps, or normal forms, such as First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF), which break down large tables into smaller ones and establish relationships between them using keys. Normalization is crucial for optimizing database performance, ensuring data consistency, and facilitating efficient data management and querying.

**First Normal Form**

This form eliminates repeating groups within a table, ensuring that each column contains atomic values (indivisible units) and that there are no repeating groups or arrays. It ensures that each attribute in a table is single-valued and that each column contains only one data value per row.

**Second Normal Form**

This form eliminates repeating groups within a table, ensuring that each column contains atomic values (indivisible units) and that there are no repeating groups or arrays. It ensures that each attribute in a table is single-valued and that each column contains only one data value per row.

**Third Normal Form**

3NF further refines the normalization process by eliminating transitive dependencies. It ensures that all non-key attributes are dependent only on the primary key and not on other non-key attributes. In other words, it removes any indirect relationships between non-key attributes by breaking them into separate tables.

* + 1. **TABLE DESIGN**

**Login Table**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Bigint | Primary Key |
| username | Varchar | Unique |
| password | Varchar | Not Null |
| type | Varchar | Not Null |

**Camp**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Bigint | Primary Key |
| camp\_name | Varchar | Unique |
| capacity | Integer | Not Null |
| district | Varchar | Not Null |
| city | Varchar | Not Null |
| pin | Bigint | Not Null |
| email | Email | Unique |
| contact\_no | Bigint | Unique |
| created\_at | Datetime | Not Null |
| updated\_at | Datetime | Not Null |

**Camp Coordinator**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Bigint | Primary Key |
| LOGIN | Bigint | Foreign Key |
| CAMP | Bigint | Foreign Key |
| name | Varchar | Not Null |
| gender | Varchar | Not Null |
| dob | Date | Not Null |
| district | Varchar | Not Null |
| city | Varchar | Not Null |
| pin | Bigint | Not Null |
| email | Email | Unique |
| contact\_no | Bigint | Unique |
| aadhaar\_number | Varchar | Unique |
| posted\_at | Date | Not Null |
| photo | File | Not Null |

**Guideline**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Bigint | Primary Key |
| COORDINATOR | Bigint | Foreign Key |
| guideline | File | Not Null |
| posted\_date | Date | Not Null |
| posted\_time | Time | Not Null |

**Public**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Bigint | Primary Key |
| LOGIN | Bigint | Foreign Key |
| name | Varchar | Not Null |
| gender | Varchar | Not Null |
| dob | Date | Not Null |
| district | Varchar | Not Null |
| city | Varchar | Not Null |
| pin | Bigint | Not Null |
| email | Varchar | Unique |
| contact\_no | Bigint | Unique |
| aadhaar\_number | Varchar | Unique |
| photo | File | Not Null |
| joined\_date | Date | Not Null |

**Notification**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Bigint | Primary Key |
| title | Varchar | Not Null |
| notification | Longtext | Not Null |
| posted\_date | Date | Not Null |
| posted\_time | Time | Not Null |

**Complaint**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Bigint | Primary Key |
| PUBLIC | Bigint | Foreign Key |
| complaint | Varchar | Not Null |
| posted\_at | Datetime | Not Null |
| reply | Varchar | Not Null |
| status | Varchar | Not Null |
| updated\_at | Datetime | Not Null |

**Emergency Team**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Bigint | Primary Key |
| LOGIN | Bigint | Foreign Key |
| department | Varchar | Not Null |
| district | Varchar | Not Null |
| city | Varchar | Not Null |
| pin | Bigint | Not Null |
| email | Email | Unique |
| contact\_no | Bigint | Unique |
| joined\_date | Date | Not Null |

**Volunteer**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Bigint | Primary Key |
| CAMP | Bigint | Foreign Key |
| LOGIN | Bigint | Foreign Key |
| name | Varchar | Not Null |
| gender | Varchar | Not Null |
| dob | Date | Not Null |
| district | Varchar | Not Null |
| city | Varchar | Not Null |
| pin | Bigint | Not Null |
| email | Email | Unique |
| contact\_no | Bigint | Unique |
| aadhaar\_number | Varchar | Unique |
| photo | File | Not Null |
| posted\_date | Date | Not Null |

**Needs**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| id | Primary Key | Primary Key |
| CAMP | Foreign Key | Foreign Key |
| category | Varchar | Not Null |
| product | Varchar | Not Null |
| quantity | Integer | Not Null |
| name | Varchar | Not Null |
| unit | Varchar | Not Null |
| added\_on | Date | Not Null |

**Emergency Request**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| Id | Bigint | Primary Key |
| PUBLIC | Bigint | Foreign Key |
| EMERGENCY\_TEAM | Bigint | Foreign Key |
| request | Varchar | Not Null |
| status | Varchar | Not Null |
| posted\_date | Date | Not Null |
| posted\_time | Time | Not Null |
| updated\_date | Date | Not Null |
| updated\_time | Time | Not Null |
| latitude | Float | Not Null |
| longitude | Float | Not Null |

**Member**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| Id | Bigint | Primary Key |
| CAMP | Bigint | Foreign Key |
| name | Varchar | Not Null |
| gender | Varchar | Not Null |
| dob | Date | Not Null |
| district | Varchar | Not Null |
| city | Varchar | Not Null |
| pin | Bigint | Not Null |
| email | Email | Unique |
| contact\_no | Bigint | Unique |
| aadhaar\_number | Varchar | Unique |
| photo | File | Not Null |
| joined\_date | Date | Not Null |

**Goods**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| Id | Bigint | Primary Key |
| PUBLIC | Bigint | Foreign Key |
| VOLUNTEER | Bigint | Foreign Kry |
| category | Varchar | Not Null |
| product | Varchar | Not Null |
| name | Varchar | Not Null |
| quantity | Integer | Not Null |
| unit | Varchar | Not Null |
| donated\_on | Datetime | Not Null |
| status | Varchar | Not Null |

**Stock**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| Id | Bigint | Primary Key |
| CAMP | Bigint | Foreign Key |
| category | Varchar | Not Null |
| product | Varchar | Not Null |
| name | Varchar | Not Null |
| quantity | Integer | Not Null |
| unit | Varchar | Not Null |
| added\_on | Datetime | Not Null |
| donated | Varchar | Not Null |

**Asset**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| Id | Bigint | Primary Key |
| MEMBER | Bigint | Foreign Key |
| VOLUNTEER | Bigint | Foreign Key |
| category | Varchar | Not Null |
| asset | Varchar | Not Null |
| description | Varchar | Not Null |
| status | Varchar | Not Null |
| missing\_date | Date | Not Null |
| posted\_date | Date | Not Null |

**Task**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| Id | Bigint | Primary Key |
| VOLUNTEER | Bigint | Foreign Key |
| task | Varchar | Not Null |
| posted\_date | Date | Not Null |
| status | Boolean | Not Null |

**Payment**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| Id | Bigint | Primary Key |
| PUBLIC | Bigint | Foreign Key |
| amount | Float | Not Null |
| payment\_date | Date | Not Null |
| payment\_time | Time | Not Null |

**Group Chat**

|  |  |  |
| --- | --- | --- |
| **FIELD NAME** | **FIELD TYPE** | **CONSTRAINTS** |
| Id | Bigint | Primary Key |
| CAMP | Bigint | Foreign Key |
| LOGIN | Bigint | Foreign Key |
| message | Longtext | Not Null |
| date | Date | Not Null |
| time | Time | Not Null |

* 1. **MODULE DESCRIPTION**

The major modules included in the system are:

**Admin Module**

The admin module handles all the activities managed by the administrator. This includes:

* Adding and managing camps, including their details and resources.
* Adding and managing camp coordinators to oversee camp operations.
* Sending guidelines and instructions to coordinators.
* Verifying and managing emergency team registrations.
* Sending notifications to users for important updates.
* Viewing and tracking donations received from the public.
* Replying to and resolving complaints submitted by users.

**Camp Coordinator Module**

The Camp Coordinator module is responsible for managing camp-related activities. It includes:

* Adding and managing camp stock to ensure adequate supplies.
* Adding and managing camp members for smooth operations.
* Registering missing assets to track lost or misplaced items.
* Adding and managing camp needs to identify required resources.
* Adding, verifying, and managing volunteers for camp activities.
* Group chat functionality to communicate with volunteers.
* Assigning tasks to volunteers for efficient task distribution.
* Viewing notifications to stay updated on system alerts.
* Editing profile details to maintain accurate information.

**Emergency Team Module**

The Emergency Team module is designed for disaster response and management. It includes:

* Registration for emergency team members to join the system.
* Viewing emergency requests submitted by the public for quick response.
* Viewing camps to locate and monitor relief operations.
* Viewing notifications to stay informed about updates.
* Landslide prediction to assess and mitigate risks.

**Volunteer Module**

The Volunteer module enables volunteers to contribute effectively to camp operations. It includes:

* Viewing camp needs to understand required resources.
* Collecting donated goods to ensure supplies reach the camps.
* Viewing assigned tasks to stay on top of responsibilities.
* Managing missing assets to assist in tracking lost items.
* Group chat functionality to communicate with other volunteers and coordinators.
* Viewing notifications to stay updated on tasks and alerts.
* Editing profile details to maintain accurate personal information.

**Public Module**

The Public module allows general users to interact with the system. It includes:

* Registration for users to access system features.
* Viewing camp needs to understand what resources are required.
* Donating goods to contribute to relief efforts.
* Registering as a volunteer to assist in camp operations.
* Paying donations to provide monetary support.
* Sending emergency requests to report urgent situations.
* Sending complaints to provide feedback or report issues.
* Viewing notifications to stay informed about updates.
* Editing profile details to maintain accurate personal information.

SYSTEM TESTING AND IMPLEMENTATION

* 1. **SYSTEM TESTING**
     1. **TYPES OF TESTING**

Software testing is a process of evaluating a software application or system to ensure that it meets specified requirements and functions correctly. The primary goal of software testing is to identify defects, errors, or bugs in the software and to verify that it meets quality standards and performs as expected. Testing involves executing the software under controlled conditions and comparing actual results with expected results to determine whether the software behaves as intended. Software testing can be performed at various stages of the software development lifecycle, including unit testing, integration testing, system testing, acceptance testing, and regression testing, each focusing on different aspects of the software's functionality and performance. Testing can be conducted manually by human testers or automated using testing tools and frameworks, depending on the complexity of the software and the requirements of the project. Overall, software testing plays a critical role in ensuring the reliability, usability, and quality of software products before they are released to users.

Software testing can identify defects, validate functionality, and ensure that software meets user requirements and quality standards. By detecting and fixing defects early in the development process, testing helps improve the reliability and stability of software products. It also enhances user satisfaction by validating that the software aligns with user needs and expectations. Additionally, testing helps mitigate risks associated with software development and deployment, building trust and confidence among stakeholders. Overall, software testing plays a crucial role in delivering high-quality, reliable software that meets user needs and performs as intended.

**White box testing**

White box testing, also known as clear box testing, glass box testing, or structural testing, is a software testing technique that involves examining the internal structure of the software being tested. In white box testing, testers have access to the source code, architecture, and design of the software, allowing them to design test cases based on this knowledge. The primary goal of white box testing is to ensure that all components and pathways within the software are functioning correctly and that all code paths are tested thoroughly. This technique focuses on verifying the correctness of individual functions, statements, and decision paths within the codebase, often using techniques such as code coverage analysis, path testing, and branch testing. White box testing is typically performed by developers or testers who have a deep understanding of the software's architecture and implementation details. It complements black box testing, which focuses on testing the software's functionality without knowledge of its internal structure. Overall, white box testing is an essential technique for validating the internal logic and behaviour of software systems, helping to identify defects and ensure the reliability and quality of the software.

**Black Box testing**

Black box testing, also known as behavioural testing or functional testing, is a software testing technique that focuses on testing the functionality of a software application without knowledge of its internal code structure, implementation details, or logic. In black box testing, testers interact with the software interface or user interface to validate that it behaves as expected based on specified requirements. Testers design test cases based on inputs, outputs, and system behaviours, without considering how the software processes or handles these inputs internally. The primary goal of black box testing is to assess the software's external behaviour, functionality, and usability from an end-user perspective. This technique helps identify defects, errors, or inconsistencies in the software's functionality, ensuring that it meets user requirements and quality standards. Black box testing is typically performed by testers who do not have access to the software's internal codebase, making it an effective method for evaluating the software's overall functionality and behaviour independently of its internal implementation.

* + 1. **TESTING STRATEGY**

**Condition testing**

Condition testing, also known as branch testing or decision testing, is a white-box testing technique that focuses on evaluating the logical conditions and decision points within a software program. Test cases are designed to exercise different branches of conditional statements, such as if-else statements and switch-case statements, to ensure that all possible outcomes are tested. The goal of condition testing is to verify that the software behaves correctly under different conditions and decision paths, identifying any logical errors or inconsistencies in the program's behaviour.

**Loop testing**

Loop testing is a white-box testing technique that focuses on testing the functionality of loops within a software program. Test cases are designed to exercise different aspects of loop execution, including loop initialization, loop body execution, and loop termination conditions. The goal of loop testing is to ensure that loops behave correctly under various scenarios, including valid and invalid input data, loop boundaries, and loop iterations. This technique helps identify defects such as off-by-one errors, infinite loops, and incorrect loop termination conditions.

**Unit testing**

Unit testing is a software testing technique that focuses on testing individual units or components of a software application in isolation. Units may include functions, methods, classes, or modules, depending on the programming language and software architecture. Test cases are designed to verify the correctness of individual units, typically by providing inputs and comparing the actual outputs with expected outputs. Unit testing helps ensure that each unit of code performs its intended function correctly and independently of other units, facilitating early defect detection and debugging.

**Integration testing**

Integration testing is a software testing technique that focuses on testing the interactions and interfaces between different units or components of a software application. The goal of integration testing is to verify that individual units work together as expected when integrated into a larger system. Test cases are designed to test the interactions between units, including data flow, communication protocols, and interface compatibility. Integration testing helps identify defects such as interface mismatches, data corruption, and communication errors, ensuring that the software functions correctly as a whole.

* 1. **SYSTEM IMPLEMENTATION**

System implementation is the phase in the software development lifecycle where the designed system is put into practical use. It involves the actual construction of the system according to the specifications and requirements outlined during the design phase. System implementation typically includes activities such as coding, configuration, integration of components, database setup, and installation of software and hardware. This phase also involves testing the implemented system to ensure that it functions correctly and meets the specified requirements. Once the system has been successfully implemented and tested, it is deployed to users or customers for operational use. System implementation may also involve training users, documenting system processes and procedures, and providing ongoing support and maintenance. Overall, system implementation is a critical step in the software development process that transforms the design into a working system ready for use by stakeholders.

Some common steps typically involved in the system implementation process:

* **Planning:** Define objectives, scope, and resources.
* **Coding and Configuration**: Develop code and set up system components.
* **Integration:** Combine and ensure smooth operation of system parts.
* **Testing:** Verify system functionality and adherence to requirements.
* **Deployment:** Install and train users on the system.
* **Documentation:** Create user manuals and technical documents.
* **Monitoring and Support:** Provide ongoing maintenance and assistance.
* **Evaluation and Feedback:** Collect feedback for continuous improvement.

SYSTEM MAINTENANCE AND SECURITY

* 1. **SYSTEM MAINTENANCE**

System maintenance encompasses the ongoing activities aimed at ensuring the continued functionality, reliability, and performance of a software system after its deployment. These activities include identifying and resolving software defects, applying updates and upgrades to address security vulnerabilities and add new features, monitoring system performance to optimize resources, implementing backup and recovery procedures to safeguard data, managing security measures to protect against cyber threats, providing user support and assistance, updating system documentation to reflect changes, and managing changes to the system in a controlled manner. System maintenance is essential for maximizing the value of technology investments, ensuring business continuity, and delivering reliable services to users over the system's lifecycle.

* 1. **SYSTEM SECURITY**

System security refers to the measures and practices put in place to protect a computer system, network, or software application from unauthorized access, malicious attacks, data breaches, and other security threats. It encompasses various aspects, including:

* Access Control: Restricting access to authorized users and ensuring that only authenticated and authorized individuals or entities can access the system and its resources.
* Data Encryption: Encrypting sensitive data to prevent unauthorized access or interception, ensuring that data remains confidential and secure even if intercepted.
* Authentication and Authorization: Implementing mechanisms to verify the identity of users and determine their level of access rights to system resources based on their roles and permissions.
* Firewalls and Intrusion Detection Systems (IDS): Deploying firewalls and IDS to monitor network traffic, detect and prevent unauthorized access or malicious activities, and protect against external threats.
* Vulnerability Management: Regularly assessing and patching system vulnerabilities to address security weaknesses and minimize the risk of exploitation by attackers.
* Security Policies and Procedures: Establishing and enforcing security policies, guidelines, and procedures to govern the use, access, and management of system resources and data.
* Security Auditing and Monitoring: Conducting regular security audits and monitoring system activity to identify and respond to security incidents, anomalies, or breaches promptly.
* Disaster Recovery and Incident Response: Developing and implementing plans and procedures to recover from security incidents, data breaches, or system failures and mitigate their impact on the organization.

CONCLUSION

The Crisis Guard project represents a transformative approach to disaster management, leveraging cutting-edge technology to enhance preparedness, response, and recovery efforts. By integrating real-time data analysis, advanced weather prediction, and community-driven reporting, Crisis Guard has redefined how communities and authorities collaborate during emergencies. The system's emphasis on real-time monitoring, efficient resource allocation, and community involvement ensures a swift, coordinated, and effective response to disasters, ultimately saving lives and minimizing property damage.

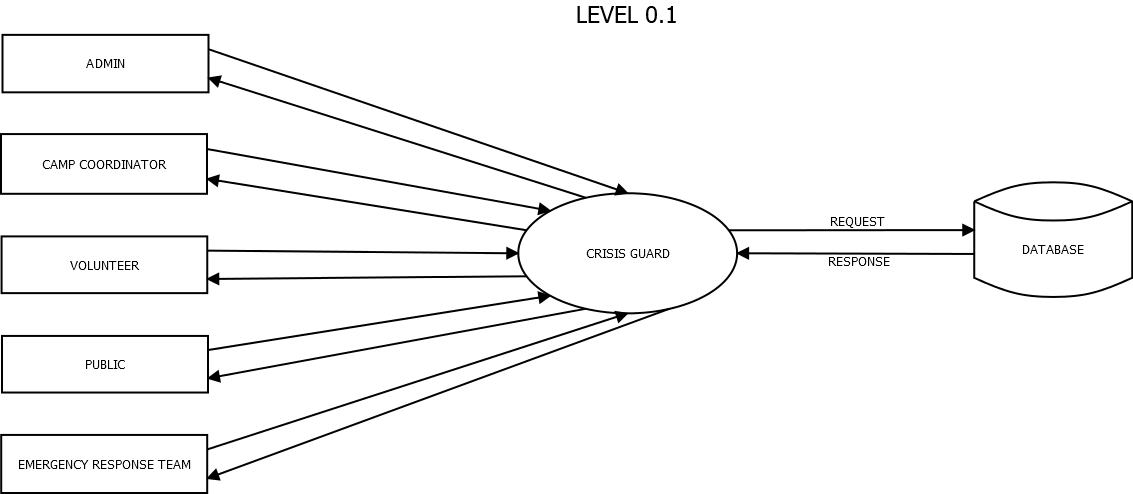
Through the development of Crisis Guard, we have demonstrated the potential of technology to address critical challenges in disaster management. The system's user-friendly interface, scalable architecture, and robust security measures make it a versatile tool adaptable to various disaster scenarios and geographic regions. The inclusion of advanced features such as landslide prediction and automated resource tracking further enhances its ability to mitigate risks and improve disaster recovery outcomes.

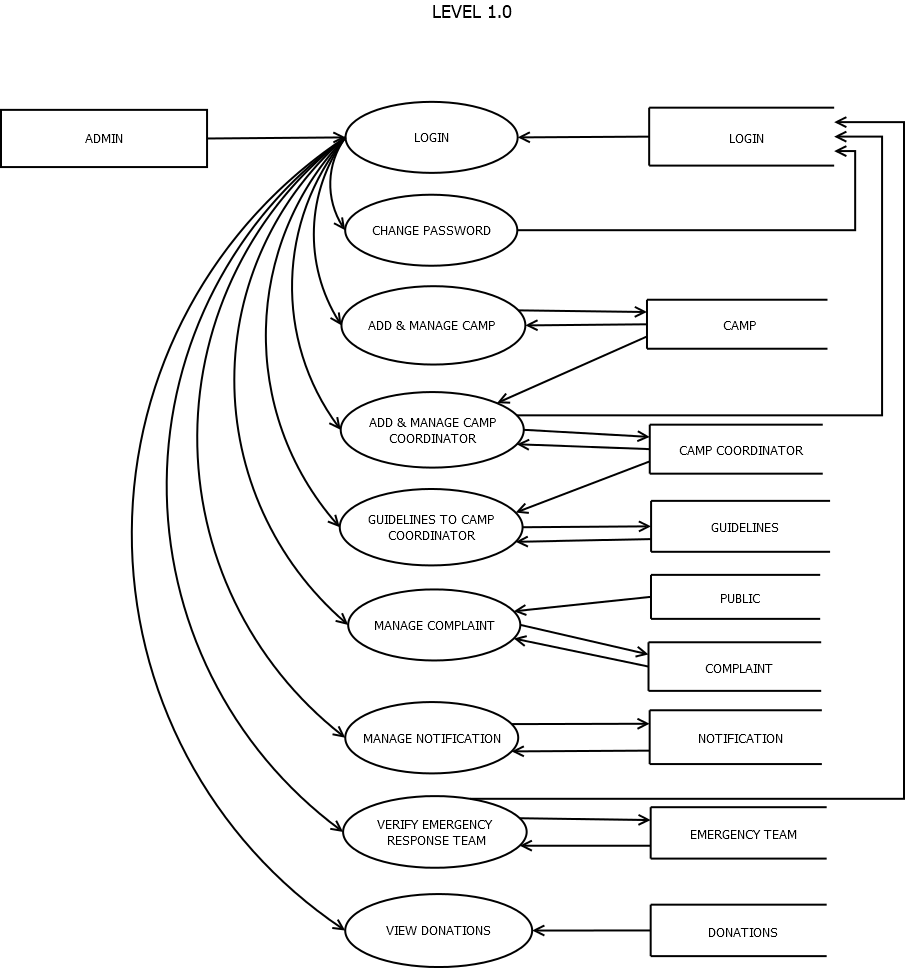
This project has not only honed our technical skills but also deepened our understanding of the importance of collaboration, innovation, and user-centric design in creating impactful solutions. As we look to the future, we envision further enhancements, such as integrating AI-driven predictive analytics, expanding community engagement features, and incorporating environmental impact assessments to make Crisis Guard even more comprehensive and effective.

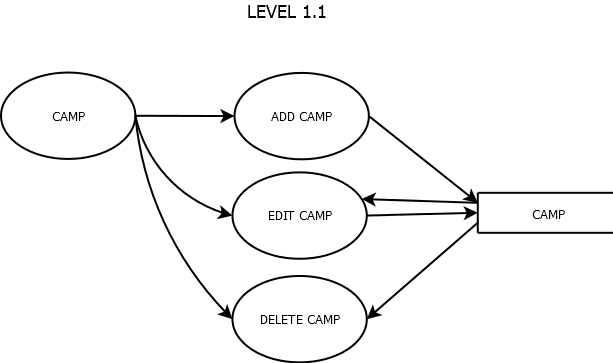
In conclusion, Crisis Guard is more than just a disaster management system—it is a platform that empowers communities, fosters resilience, and sets a new standard for disaster preparedness and response. Through continuous improvement and collaboration, we believe Crisis Guard has the potential to make a significant global impact, helping communities face disasters with confidence and resilience.

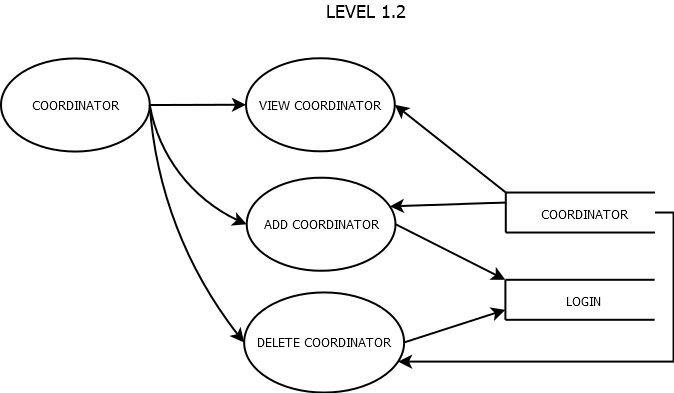
APPENDIX

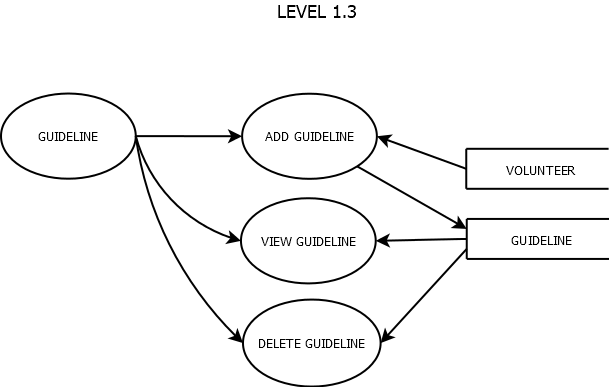
* 1. **DATA FLOW DIAGRAM**

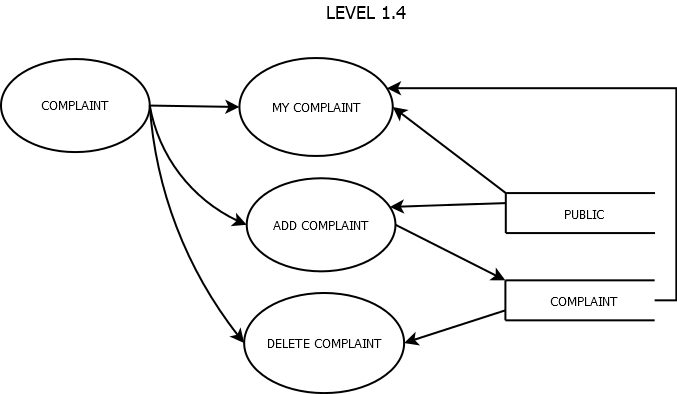
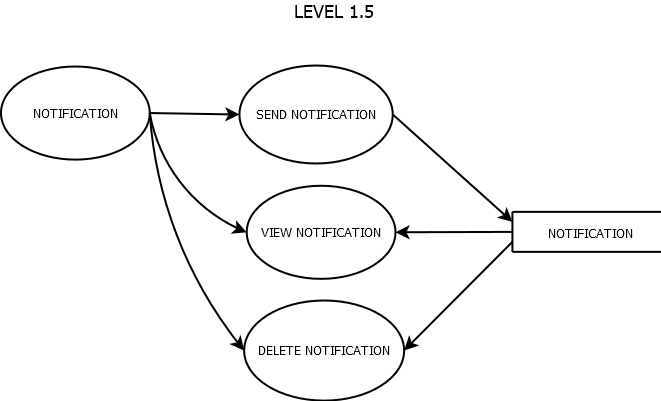
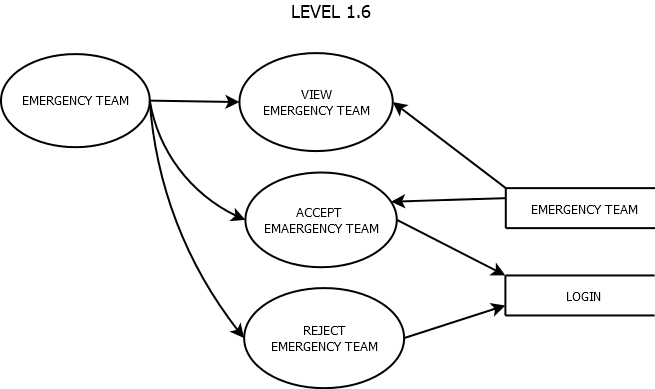


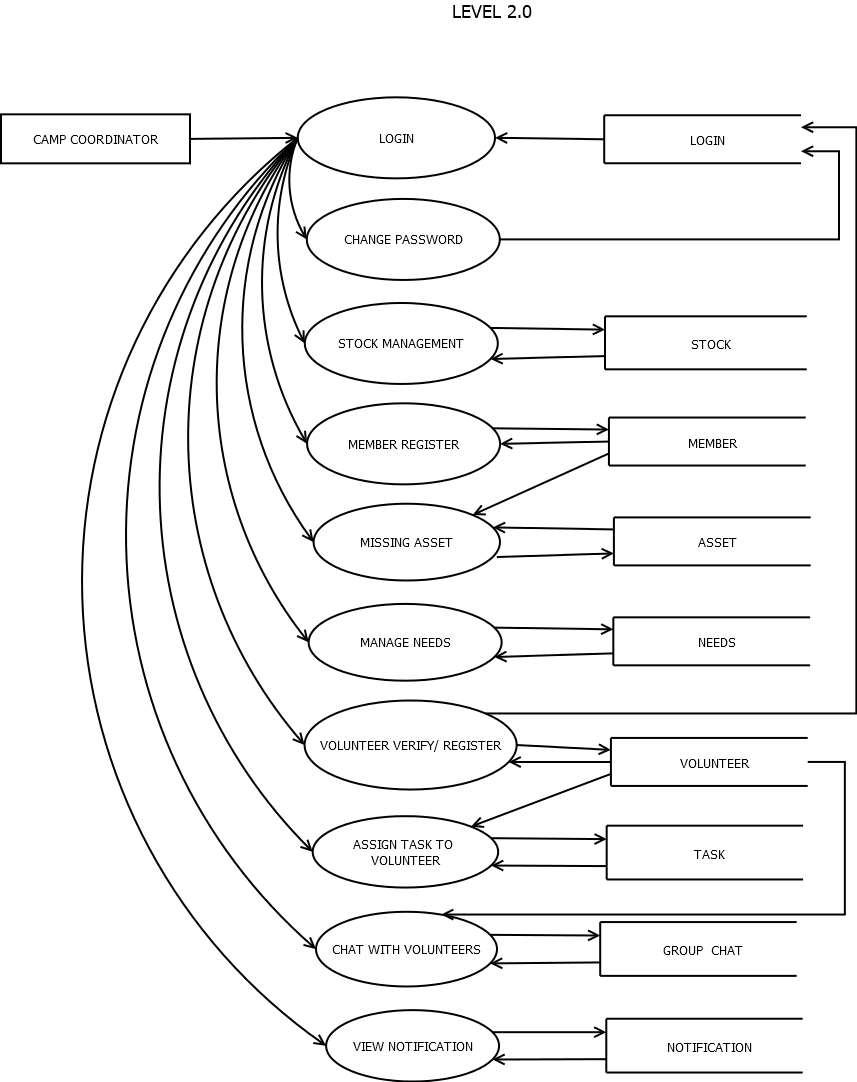


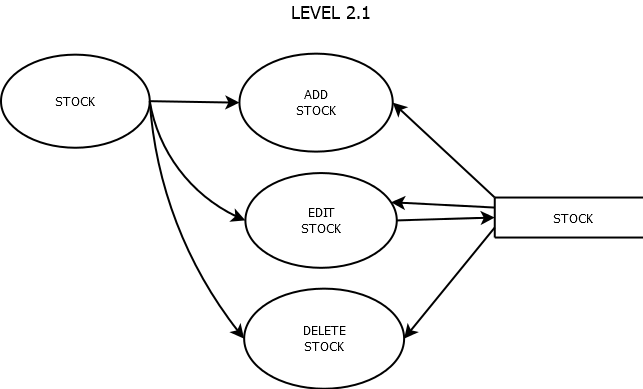


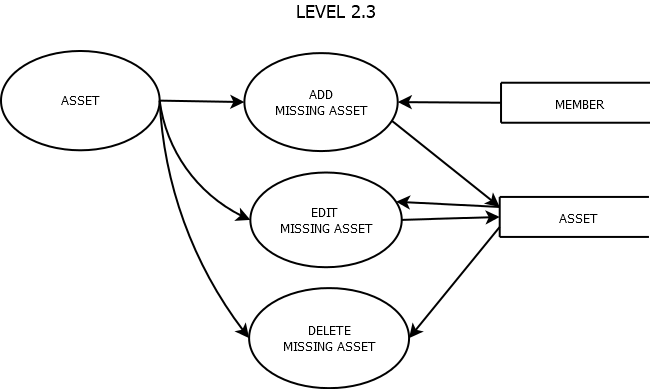
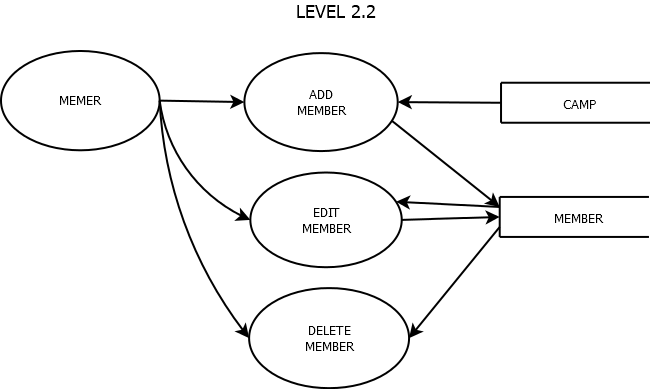


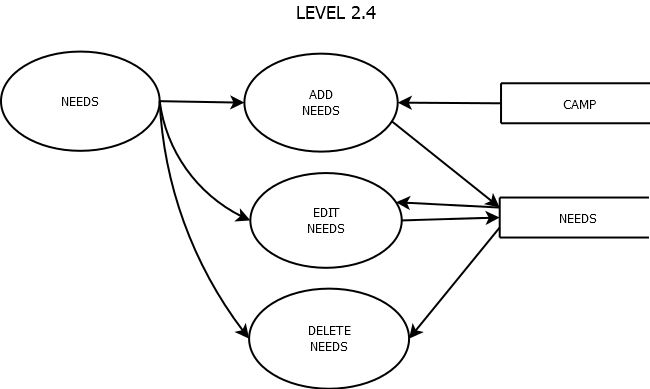


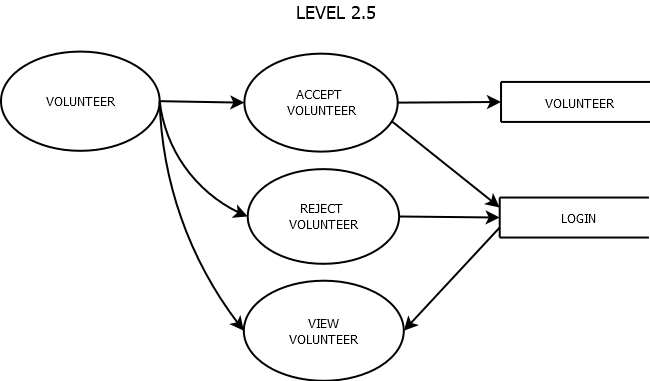
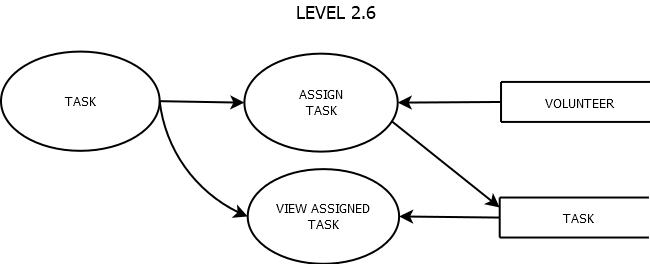


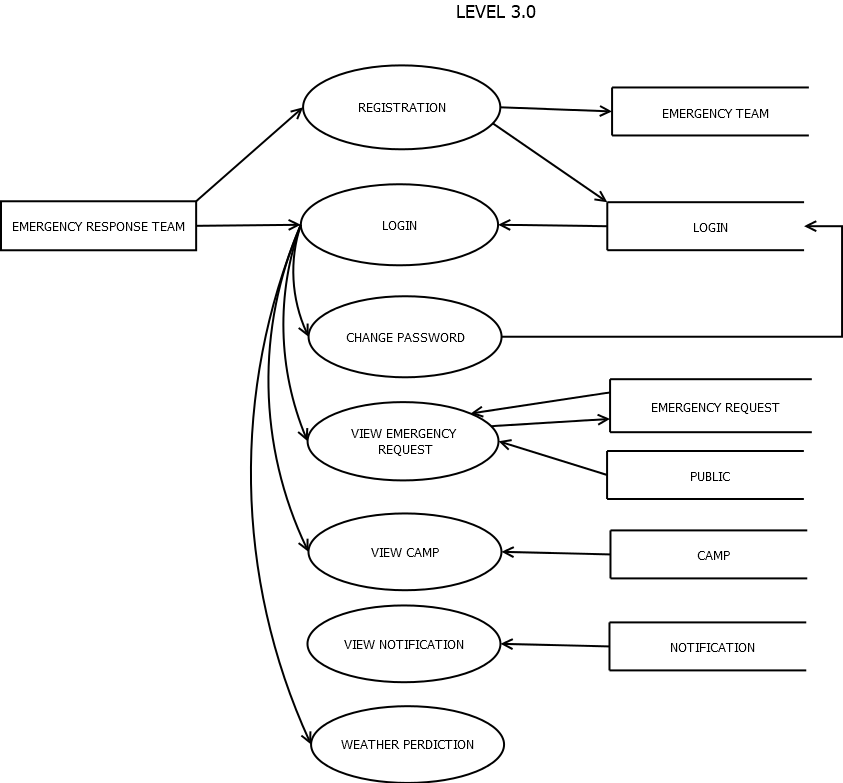


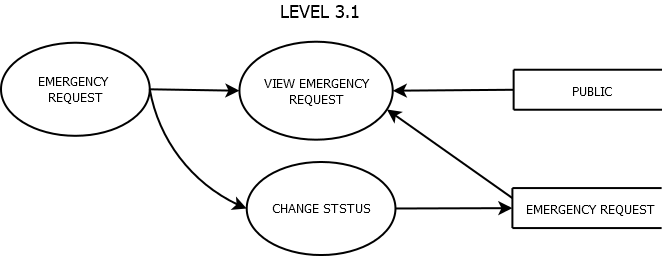


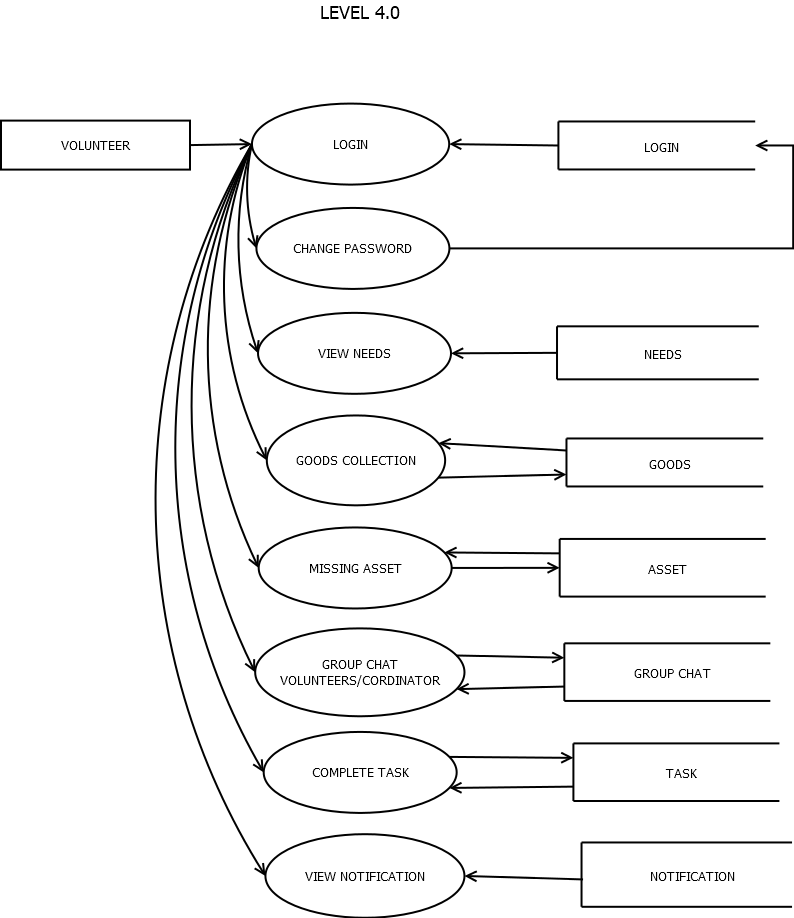


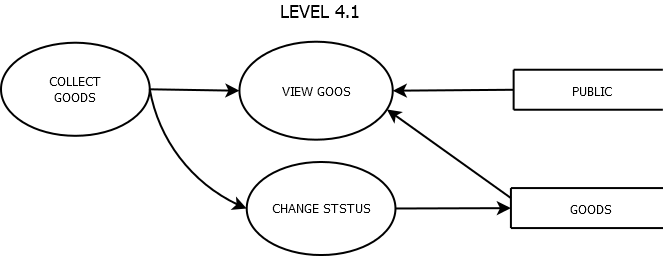


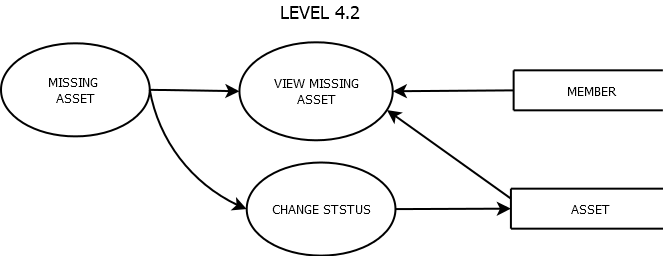
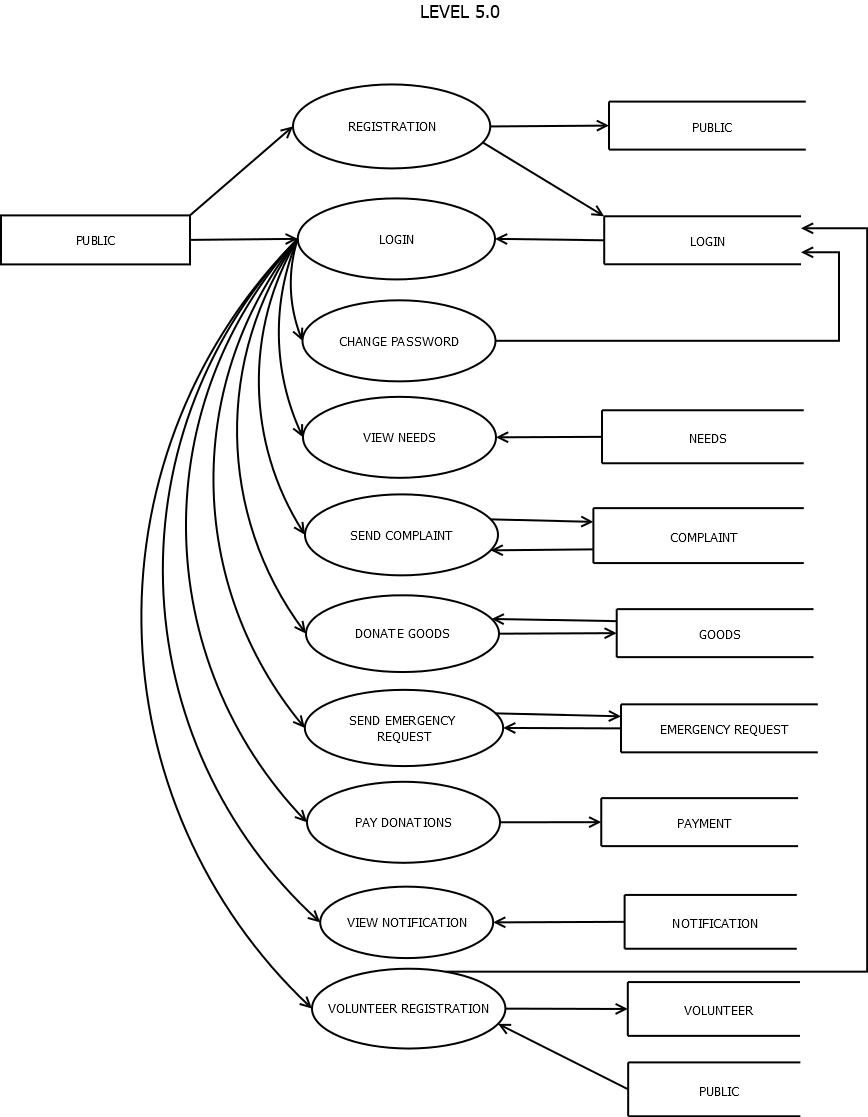


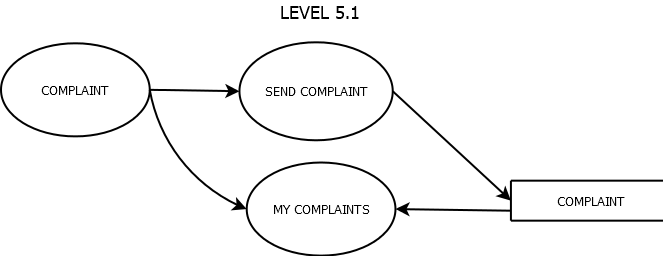
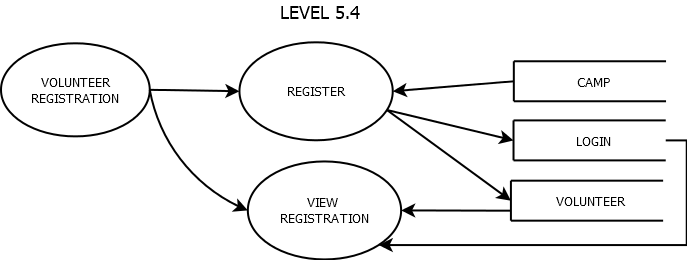
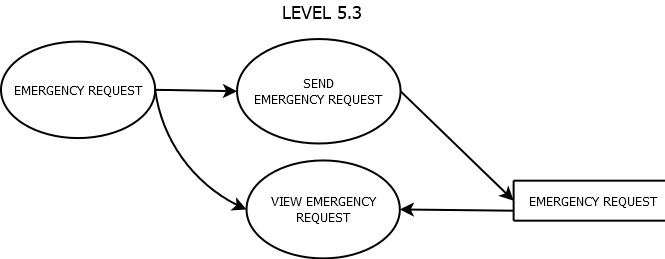
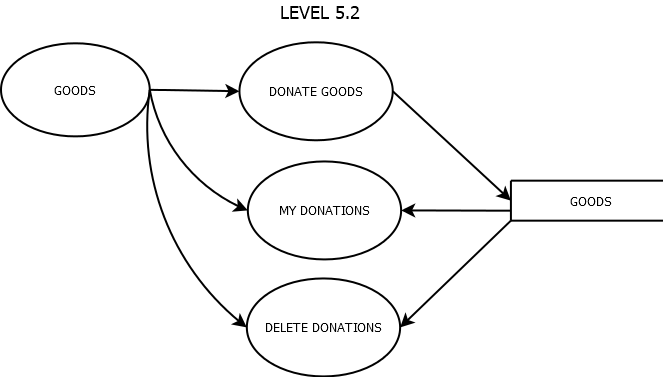




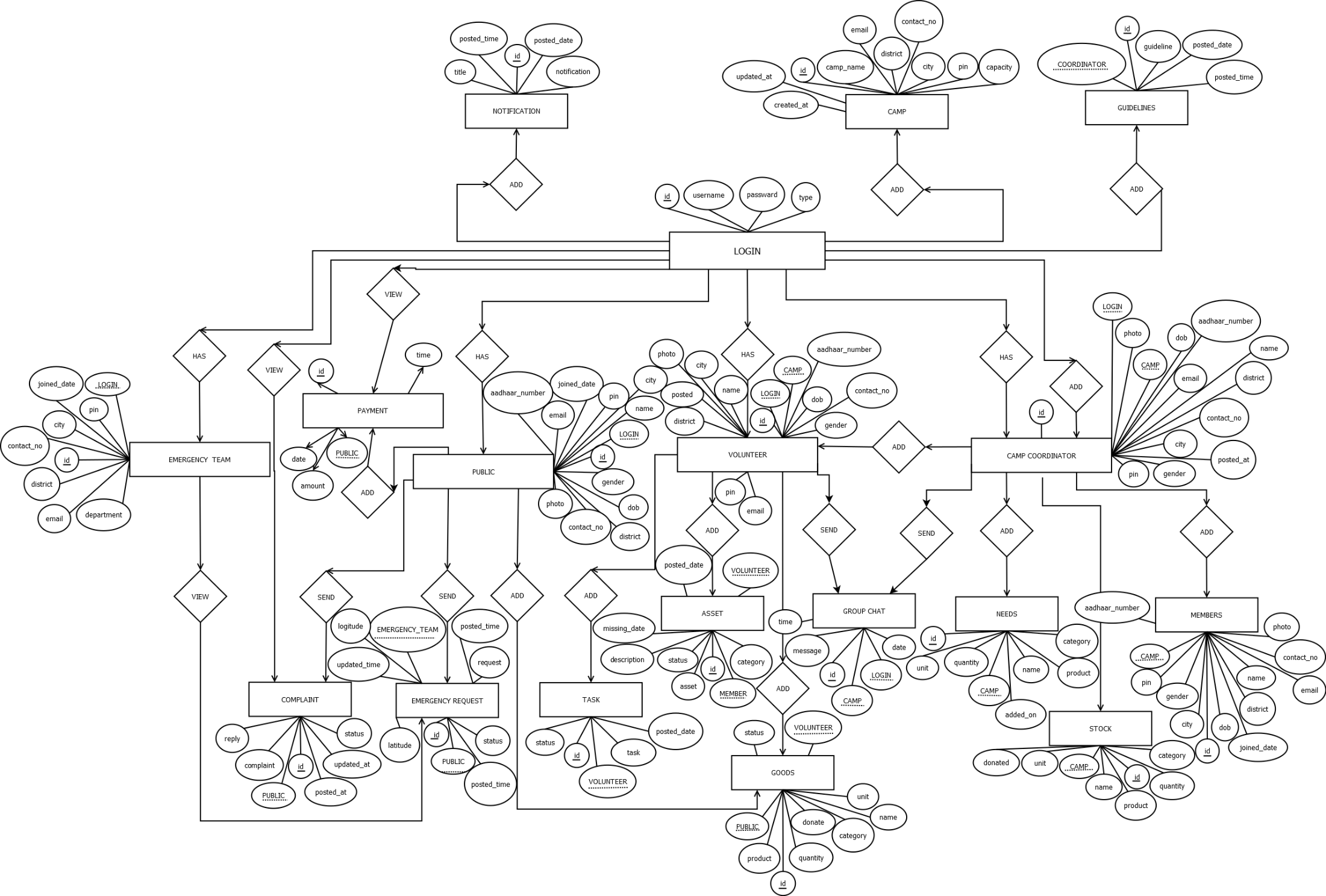








* 1. **ER DIAGRAM**

CODE SNIPPET

views.py

import datetime

import json

import os

from django.http import HttpResponse, JsonResponse

from django.shortcuts import redirect, render

from django.urls import reverse

import joblib

import requests

from CrisisGuard import settings

from app.models import \*

from app.river import get\_rivers\_near\_location

from app.sample import getmaindata

from django.contrib import messages, auth

from django.db.models import Count, F, FloatField, ExpressionWrapper,Sum

from django.core.files.storage import FileSystemStorage

from collections import defaultdict

from django.core.mail import send\_mail

from django.db.models import Q

from django.contrib.auth.decorators import login\_required

from django.utils import timezone

from datetime import timedelta

def login(request):

return render(request,'common/login.html')

def logout(request):

auth.logout(request)

return redirect('/')

def login\_post(request):

if request.method == 'POST':

name = request.POST.get('username', '').strip()

password = request.POST.get('password', '').strip()

if not name or not password:

messages.error(request, 'Please enter both username and password.')

return redirect('/')

try:

user = Login.objects.get(username\_\_iexact=name)

if user.password == password:

request.session['lid'] = user.id

ob1 = auth.authenticate(username="admin", password="admin")

if ob1 is not None:

auth.login(request, ob1)

dashboard\_mapping = {

'Admin': '/admin\_dashboard',

'Coordinator': '/coordinator\_dashboard',

'ERT': '/ert\_dashboard',

}

dashboard\_url = dashboard\_mapping.get(user.type)

if dashboard\_url:

messages.success(request, f'{user.type} Logged In')

return redirect(dashboard\_url)

else:

messages.error(request, 'Unknown user type.')

return redirect('/')

else:

messages.error(request, 'Invalid username or password.')

return redirect('/')

except Login.DoesNotExist:

messages.error(request, 'Invalid username or password.')

return redirect('/')

messages.error(request, 'Invalid request method.')

return redirect('/')

def forgotpass(request):

return render(request,'common/forgotpass.html')

@login\_required(login\_url='/')

def change\_password(request):

if request.method == "POST":

user\_id = request.session.get('lid')

if not user\_id:

messages.error(request, 'You are not logged in.')

return redirect('/logout')

if 'new\_password' in request.POST and 'confirm\_password' in request.POST:

new\_password = request.POST['new\_password']

confirm\_password = request.POST['confirm\_password']

user = Login.objects.get(id=user\_id)

if new\_password == confirm\_password:

user.password = new\_password

user.save()

messages.success(request, 'Password updated successfully!')

return redirect('/logout')

else:

messages.error(request, 'New passwords do not match.')

return render(request, 'common/change password.html')

elif 'current\_password' in request.POST:

current\_password = request.POST['current\_password']

try:

user = Login.objects.get(id=user\_id)

if user.password == current\_password:

return render(request, 'common/change password.html', {

'current\_password\_verified': True

})

else:

messages.error(request, 'Incorrect current password.')

return render(request, 'common/change password.html')

except Login.DoesNotExist:

messages.error(request, 'User does not exist.')

return redirect('/logout')

return render(request, 'common/change password.html')

def register\_ert(request):

return render(request,'common/register ert.html')

def register\_ert\_post(request):

department = request.POST["department"]

district = request.POST["district"]

city = request.POST["city"]

pin = request.POST["pin"]

contactno = request.POST["contactno"]

email = request.POST["email"]

username = request.POST["username"]

password = request.POST["password"]

if Login.objects.filter(username=username).exists():

messages.error(request, 'Username already exists!')

return redirect('/register\_ert')

if EmergencyTeam.objects.filter(email=email).exists():

messages.error(request, 'Email already exists!')

return redirect('/register\_ert')

if EmergencyTeam.objects.filter(contact\_no=contactno).exists():

messages.error(request, 'Contact number already exists!')

return redirect('/register\_ert')

ob = Login(username=username, password=password, type="Pending")

ob.save()

obj = EmergencyTeam(

LOGIN=ob,

department=department,

district=district,

city=city,

pin=pin,

contact\_no=contactno,

email=email

)

obj.save()

messages.success(request, 'Emergency Response Team registered!')

return redirect('/ert\_registration\_status')

def ert\_registration\_status(request):

ob=EmergencyTeam.objects.all()

return render(request,'common/status.html', {'val': ob})

@login\_required(login\_url='/')

def admin\_dashboard(request):

context = {

'total\_camp': Camp.objects.count(),

'total\_coordinator': CampCoordinator.objects.count(),

'total\_volunteer': Login.objects.filter(type='Volunteer').count(),

'total\_member': Member.objects.count(),

'total\_ert': Login.objects.filter(type='ERT').count(),

'total\_user': Public.objects.count(),

'total\_complaints\_pending': Complaint.objects.filter(status='Pending').count(),

'total\_complaints\_working\_on': Complaint.objects.filter(status='Working').count(),

'total\_complaints\_resolved': Complaint.objects.filter(status='Resolved').count(),

'total\_donations': '{:,}'.format(int(Payment.objects.aggregate(total=Sum('amount'))['total'] or 0)),

}

return render(request, 'admin/dashboard.html', context)

@login\_required(login\_url='/')

def admin\_add\_camp(request):

return render (request,'admin/add camp.html')

@login\_required(login\_url='/')

def admin\_add\_camp\_post(request):

camp = request.POST["camp"]

capacity = request.POST["capacity"]

district = request.POST["district"]

city = request.POST["city"]

pin = request.POST["pin"]

email = request.POST["email"]

contactno = request.POST["contactno"]

if Camp.objects.filter(camp\_name=camp).exists():

messages.error(request, 'Camp name already exists!')

return redirect('/admin\_add\_camp')

if Camp.objects.filter(email=email).exists():

messages.error(request, 'Email already exists!')

return redirect('/admin\_add\_camp')

if Camp.objects.filter(contact\_no=contactno).exists():

messages.error(request, 'Contact number already exists!')

return redirect('/admin\_add\_camp')

obj = Camp()

obj.camp\_name = camp

obj.capacity = capacity

obj.district = district

obj.city = city

obj.pin = pin

obj.email = email

obj.contact\_no = contactno

obj.save()

messages.success(request, 'Camp Added successfully!')

return redirect('/admin\_manage\_camp')

@login\_required(login\_url='/')

def admin\_manage\_camp(request):

camps = Camp.objects.annotate(

member\_count=Count('member'),

occupancy\_rate=ExpressionWrapper(

(F('member\_count') \* 100.0) / F('capacity'),

output\_field=FloatField()

)

)

camp\_data = []

for camp in camps:

try:

camp\_coordinator = camp.campcoordinator

except CampCoordinator.DoesNotExist:

camp\_coordinator = None

camp\_data.append({

'camp': camp,

'coordinator': camp\_coordinator

})

return render(request, 'admin/manage camp.html', {'camp\_data': camp\_data})

@login\_required(login\_url='/')

def admin\_search\_camp(request):

camp\_name = request.POST.get('campname', '')

camps = Camp.objects.filter(camp\_name\_\_icontains=camp\_name).annotate(

member\_count=Count('member'),

occupancy\_rate=ExpressionWrapper(

(F('member\_count') \* 100) / F('capacity'),

output\_field=FloatField()

)

)

camp\_data = []

for camp in camps:

try:

camp\_coordinator = camp.campcoordinator

except CampCoordinator.DoesNotExist:

camp\_coordinator = None

camp\_data.append({

'camp': camp,

'coordinator': camp\_coordinator

})

return render(request, 'admin/manage camp.html', {'camp\_data': camp\_data})

@login\_required(login\_url='/')

def admin\_edit\_camp(request,id):

request.session["campid"]=id

ob=Camp.objects.get(id=id)

return render(request, 'admin/edit camp.html',{"ob":ob})

@login\_required(login\_url='/')

def admin\_edit\_camp\_post(request):

camp = request.POST.get("camp")

capacity = request.POST.get("capacity")

district = request.POST.get("district")

city = request.POST.get("city")

pin = request.POST.get("pin")

contactno = request.POST.get("contactno")

email = request.POST.get("email")

camp\_id = request.session.get("campid")

obj = Camp.objects.get(id=camp\_id)

if Camp.objects.filter(camp\_name=camp).exclude(id=camp\_id).exists():

messages.error(request, 'Camp name already exists!')

return redirect(reverse('admin\_edit\_camp', args=[camp\_id]))

if Camp.objects.filter(email=email).exclude(id=camp\_id).exists():

messages.error(request, 'Email already exists!')

return redirect(reverse('admin\_edit\_camp', args=[camp\_id]))

if Camp.objects.filter(contact\_no=contactno).exclude(id=camp\_id).exists():

messages.error(request, 'Contact number already exists!')

return redirect(reverse('admin\_edit\_camp', args=[camp\_id]))

obj.camp\_name = camp

obj.capacity = capacity

obj.district = district

obj.city = city

obj.pin = pin

obj.contact\_no = contactno

obj.email = email

obj.save()

messages.success(request, 'Camp Edited successfully!')

return redirect('/admin\_manage\_camp')

@login\_required(login\_url='/')

def admin\_delete\_camp(request,id):

Camp.objects.get(id=id).delete()

messages.success(request, 'Camp Deleted')

return redirect('/admin\_manage\_camp')

@login\_required(login\_url='/')

def admin\_add\_coordinator(request):

camps = Camp.objects.filter(campcoordinator\_\_isnull=True)

return render(request, 'admin/add coordinator.html', {"camps": camps})

@login\_required(login\_url='/')

def admin\_add\_coordinator\_post(request):

camp = request.POST["camp"]

name = request.POST["name"]

gender = request.POST["gender"]

dob = request.POST["dob"]

contactno = request.POST["contactno"]

email = request.POST["email"]

district = request.POST["district"]

city = request.POST["city"]

pin = request.POST["pin"]

username = request.POST["username"]

password = request.POST["password"]

aadhaar\_number = request.POST["aadhaar\_number"]

photo = request.FILES["photo"]

if CampCoordinator.objects.filter(contact\_no=contactno).exists():

messages.error(request, 'Contact number already exists')

return redirect('/admin\_add\_coordinator')

if CampCoordinator.objects.filter(email=email).exists():

messages.error(request, 'Email already exists')

return redirect('/admin\_add\_coordinator')

if CampCoordinator.objects.filter(aadhaar\_number=aadhaar\_number).exists():

messages.error(request, 'Aadhaar number already exists')

return redirect('/admin\_add\_coordinator')

if Login.objects.filter(username=username).exists():

messages.error(request, 'Username already exists')

return redirect('/admin\_add\_coordinator')

fs = FileSystemStorage()

fsave = fs.save(photo.name, photo)

login\_obj = Login(username=username, password=password, type="Coordinator")

login\_obj.save()

coordinator\_obj = CampCoordinator(

LOGIN=login\_obj,

CAMP\_id=camp,

name=name,

gender=gender,

dob=dob,

contact\_no=contactno,

email=email,

district=district,

city=city,

pin=pin,

aadhaar\_number=aadhaar\_number,

photo=fsave )

coordinator\_obj.save()

messages.success(request, 'Camp Coordinator Added')

return redirect('/admin\_manage\_coordinator')

@login\_required(login\_url='/')

def admin\_manage\_coordinator(request):

ob=CampCoordinator.objects.all()

return render (request,'admin/manage coordinator.html',{'val':ob})

@login\_required(login\_url='/')

def admin\_search\_coordinator(request):

name = request.POST['coordinatorname']

ob = CampCoordinator.objects.filter(name\_\_icontains=name)

return render(request, 'admin/manage coordinator.html', {'val': ob})

@login\_required(login\_url='/')

def admin\_delete\_coordinator(request, id):

if CampCoordinator.objects.filter(id=id).exists():

coordinator = CampCoordinator.objects.get(id=id)

if coordinator.photo:

file\_path = os.path.join(settings.MEDIA\_ROOT, coordinator.photo.name)

if os.path.exists(file\_path):

os.remove(file\_path)

login = coordinator.LOGIN

login.delete()

coordinator.delete()

messages.success(request, 'Camp Coordinator Deleted')

else:

messages.error(request, 'Camp Coordinator not found')

return redirect('/admin\_manage\_coordinator')

@login\_required(login\_url='/')

def admin\_verify\_ert(request):

ob=EmergencyTeam.objects.filter(LOGIN\_\_type='Pending')

return render(request, 'admin/verify ert.html', {'val': ob})

@login\_required(login\_url='/')

def admin\_manage\_ert(request):

ob1=EmergencyTeam.objects.filter(LOGIN\_\_type='ERT')

ob2=EmergencyTeam.objects.filter(LOGIN\_\_type='Rejected')

return render(request, 'admin/manage ert.html', {'val1': ob1 ,'val2':ob2})

@login\_required(login\_url='/')

def admin\_accept\_ert(request, id):

try:

request.session["ERTid"] = id

login = Login.objects.get(id=id)

login.type = "ERT"

login.save()

emergency\_team = EmergencyTeam.objects.get(LOGIN=login)

subject = 'Emergency Team Application Accepted'

message = f'Dear {emergency\_team.city},{emergency\_team.department} Team,\n\nYour application to join the Emergency Team has been accepted. Welcome aboard!\n\nBest regards,\n Admin'

from\_email = settings.EMAIL\_HOST\_USER

recipient\_list = [emergency\_team.email]

send\_mail(subject, message, from\_email, recipient\_list, fail\_silently=False)

messages.success(request, 'Emergency Team Accepted and email sent!')

except Exception as e:

messages.error(request, f"Error: {str(e)}")

return redirect('/admin\_manage\_ert')

@login\_required(login\_url='/')

def admin\_reject\_ert(request, id):

try:

request.session["ERTid"] = id

login = Login.objects.get(id=id)

login.type = "Rejected"

login.save()

emergency\_team = EmergencyTeam.objects.get(LOGIN=login)

subject = 'Emergency Team Application Rejected'

message = f'Dear {emergency\_team.city},{emergency\_team.department} Team,\n\nWe regret to inform you that your application to join the Emergency Team has been rejected.\n\nBest regards,\Admin'

from\_email = settings.EMAIL\_HOST\_USER

recipient\_list = [emergency\_team.email]

send\_mail(subject, message, from\_email, recipient\_list, fail\_silently=False)

messages.error(request, 'Emergency Team Rejected and email sent!')

except Exception as e:

messages.error(request, f"Error: {str(e)}")

return redirect('/admin\_manage\_ert')

@login\_required(login\_url='/')

def admin\_search\_verify\_ert(request):

district = request.POST['district']

ob = EmergencyTeam.objects.filter(district\_\_icontains=district,LOGIN\_\_type='pending')

return render(request, 'admin/verify ert.html', {'val': ob})

@login\_required(login\_url='/')

def admin\_search\_accept\_ert(request):

district = request.POST['district']

ob = EmergencyTeam.objects.filter(district\_\_icontains=district,LOGIN\_\_type='ert')

ob2=EmergencyTeam.objects.filter(LOGIN\_\_type='Rejected')

return render(request, 'admin/manage ert.html', {'val1': ob,'val2':ob2})

@login\_required(login\_url='/')

def admin\_search\_reject\_ert(request):

district = request.POST['district']

ob = EmergencyTeam.objects.filter(district\_\_icontains=district,LOGIN\_\_type='rejected')

ob1=EmergencyTeam.objects.filter(LOGIN\_\_type='ERT')

return render(request, 'admin/manage ert.html', {'val1': ob1 ,'val2':ob})

@login\_required(login\_url='/')

def admin\_send\_notification(request):

return render (request,'admin/add notification.html')

@login\_required(login\_url='/')

def admin\_send\_notification\_post(request):

title=request.POST["title"]

notification=request.POST["notification"]

obj=Notification()

obj.title=title

obj.notification=notification

obj.save()

messages.success(request, 'Notification Sent')

return redirect('/admin\_manage\_notification')

@login\_required(login\_url='/')

def admin\_delete\_notification(request,id):

Notification.objects.get(id=id).delete()

messages.success(request, 'Notification Deleted')

return redirect('/admin\_manage\_notification')

@login\_required(login\_url='/')

def admin\_edit\_notification(request,id):

request.session["notificationid"]=id

ob=Notification.objects.get(id=id)

return render(request, 'admin/edit notification.html',{"ob":ob})

@login\_required(login\_url='/')

def admin\_reply\_complaint(request,id):

request.session["complaintid"]=id

ob=Complaint.objects.get(id=id)

return render(request, 'admin/reply complaint.html',{"ob":ob})

@login\_required(login\_url='/')

def admin\_reply\_complaint\_post(request):

status=request.POST["status"]

reply=request.POST["reply"]

obj = Complaint.objects.get(id=request.session["complaintid"])

obj.status=status

obj.reply=reply

obj.save()

messages.success(request, 'Complaint Replied')

return redirect('/admin\_manage\_complaint')

@login\_required(login\_url='/')

def coordinator\_dashboard(request):

user\_id = request.session.get('lid')

try:

coordinator = CampCoordinator.objects.get(LOGIN\_\_id=user\_id)

current\_camp = coordinator.CAMP

total\_members = Member.objects.filter(CAMP=current\_camp).count()

total\_volunteer = Volunteer.objects.filter(CAMP=current\_camp).count()

total\_missing\_asset = Asset.objects.filter(MEMBER\_\_CAMP=current\_camp, status='Pending').count()

total\_found\_asset = Asset.objects.filter(MEMBER\_\_CAMP=current\_camp, status='Found').count()

total\_needs = Needs.objects.filter(CAMP=current\_camp).count()

total\_ert = Login.objects.filter(type='ERT').count()

total\_products = Stock.objects.filter(CAMP=current\_camp).values('name').distinct().count()

except CampCoordinator.DoesNotExist:

total\_members = 0

total\_volunteer = 0

total\_missing\_asset = 0

total\_needs = 0

total\_products = 0

total\_found\_asset = 0

total\_ert = 0

context = {

'total\_members': total\_members,

'total\_volunteer': total\_volunteer,

'total\_missing\_asset': total\_missing\_asset,

'total\_needs': total\_needs,

'total\_products': total\_products,

'total\_found\_asset': total\_found\_asset,

'total\_ert': total\_ert,}

return render(request, 'coordinator/dashboard.html', context)

@login\_required(login\_url='/')

def coordinator\_profile(request):

user\_id = request.session.get('lid')

if not user\_id:

return redirect('/')

context = {}

try:

login\_user = Login.objects.get(id=user\_id)

if login\_user.type != 'Coordinator':

return redirect('/')

coordinator = CampCoordinator.objects.get(LOGIN=login\_user)

context['user\_type'] = 'Coordinator'

context['user\_details'] = {

'username': login\_user.username,

'type': login\_user.type,

'name': coordinator.name,

'gender': coordinator.gender,

'dob': coordinator.dob,

'district': coordinator.district,

'city': coordinator.city,

'pin': coordinator.pin,

'email': coordinator.email,

'contact\_no': coordinator.contact\_no,

'photo': coordinator.photo,

}

except Exception:

return redirect('/')

return render(request, 'coordinator/profile.html', context)

@login\_required(login\_url='/')

def update\_coordinator\_profile(request):

if request.method == 'POST':

user\_id = request.session.get('lid')

if not user\_id:

return redirect('/')

try:

login\_user = Login.objects.get(id=user\_id)

if login\_user.type != 'Coordinator':

return redirect('/')

new\_username = request.POST.get('username')

new\_email = request.POST.get('email')

new\_contact\_no = request.POST.get('contactno')

if Login.objects.filter(username=new\_username).exclude(id=user\_id).exists():

messages.error(request, 'Username already exists')

return redirect('coordinator\_profile')

if CampCoordinator.objects.filter(email=new\_email).exclude(LOGIN=login\_user).exists():

messages.error(request, 'Email already exists')

return redirect('coordinator\_profile')

if CampCoordinator.objects.filter(contact\_no=new\_contact\_no).exclude(LOGIN=login\_user).exists():

messages.error(request, 'Contact number already exists')

return redirect('coordinator\_profile')

coordinator = CampCoordinator.objects.get(LOGIN=login\_user)

coordinator.name = request.POST.get('name')

coordinator.gender = request.POST.get('gender')

coordinator.dob = request.POST.get('dob')

coordinator.district = request.POST.get('district')

coordinator.city = request.POST.get('city')

coordinator.pin = request.POST.get('pin')

coordinator.email = new\_email

coordinator.contact\_no = new\_contact\_no

login\_user.username = new\_username

if 'photoUpload' in request.FILES:

if coordinator.photo:

old\_photo\_path = os.path.join(settings.MEDIA\_ROOT, coordinator.photo.name)

if os.path.exists(old\_photo\_path):

os.remove(old\_photo\_path)

coordinator.photo = request.FILES['photoUpload']

coordinator.save()

login\_user.save()

messages.success(request, 'Profile updated successfully!')

return redirect('coordinator\_dashboard')

except Exception as e:

messages.error(request, f'Error updating profile: {e}')

return redirect('coordinator\_dashboard')

return redirect('/')

@login\_required(login\_url='/')

def coordinator\_add\_stock(request):

return render(request, 'coordinator/add stock.html')

@login\_required(login\_url='/')

def coordinator\_add\_stock\_post(request):

category = request.POST["category"]

product = request.POST["product"]

name = request.POST["name"]

quantity = request.POST["quantity"]

unit = request.POST["unit"]

obj = Stock()

obj.donated="Government"

obj.category= category

obj.product=product

obj.name=name

obj.quantity=quantity

obj.unit=unit

coordinator = CampCoordinator.objects.get(LOGIN\_\_id=request.session['lid'])

obj.CAMP = coordinator.CAMP

obj.save()

messages.success(request, 'Stock Added')

return redirect('/coordinator\_manage\_stock')

@login\_required(login\_url='/')

def coordinator\_manage\_stock(request):

login\_id = request.session.get('lid')

coordinator = CampCoordinator.objects.get(LOGIN\_id=login\_id)

ob = Stock.objects.filter(CAMP=coordinator.CAMP)

total\_quantities = ob.values('category', 'product', 'unit').annotate(total\_quantity=Sum('quantity'))

grouped\_quantities = defaultdict(list)

for item in total\_quantities:

grouped\_quantities[item['category']].append(item)

grouped\_quantities = dict(grouped\_quantities)

return render(request, 'coordinator/manage stock.html', {

"grouped\_quantities": grouped\_quantities

})

@login\_required(login\_url='/')

def coordinator\_detailed\_stock(request):

login\_id = request.session.get('lid')

coordinator = CampCoordinator.objects.get(LOGIN\_id=login\_id)

ob = Stock.objects.filter(CAMP=coordinator.CAMP)

return render(request, 'coordinator/detailed stock.html', {"val": ob})

@login\_required(login\_url='/')

def coordinator\_edit\_stock\_post(request):

quantity = request.POST["quantity"]

unit = request.POST["unit"]

obj = Stock.objects.get(id=request.session["stockid"])

obj.quantity=quantity

obj.unit= unit

obj.save()

messages.success(request, 'Stock Edited')

return redirect('/coordinator\_detailed\_stock')

@login\_required(login\_url='/')

def coordinator\_register\_member(request):

return render (request,'coordinator/register member.html')

@login\_required(login\_url='/')

def coordinator\_register\_member\_post(request):

name = request.POST["name"]

gender = request.POST["gender"]

dob = request.POST["dob"]

district = request.POST["district"]

city = request.POST["city"]

pin = request.POST["pin"]

aadhaar\_number = request.POST["aadhaar\_number"]

email = request.POST["email"]

contactno = request.POST["contactno"]

photo = request.FILES["photo"]

if Member.objects.filter(aadhaar\_number=aadhaar\_number).exists():

messages.error(request, 'Aadhaar number already exists!')

return redirect('/coordinator\_register\_member')

if Member.objects.filter(contact\_no=contactno).exists():

messages.error(request, 'Contact number already exists!')

return redirect('/coordinator\_register\_member')

if Member.objects.filter(email=email).exists():

messages.error(request, 'Email already exists!')

return redirect('/coordinator\_register\_member')

fs = FileSystemStorage()

fsave = fs.save(photo.name, photo)

obj = Member()

obj.name = name

obj.gender = gender

obj.dob = dob

obj.district = district

obj.city = city

obj.pin = pin

obj.aadhaar\_number = aadhaar\_number

obj.contact\_no = contactno

obj.email = email

obj.photo = fsave

coordinator = CampCoordinator.objects.get(LOGIN\_\_id=request.session['lid'])

obj.CAMP = coordinator.CAMP

obj.save()

messages.success(request, 'Member Registered')

return redirect('/coordinator\_manage\_members')

@login\_required(login\_url='/')

def coordinator\_manage\_members(request):

login\_id = request.session.get('lid')

coordinator = CampCoordinator.objects.get(LOGIN\_id=login\_id)

ob = Member.objects.filter(CAMP=coordinator.CAMP)

return render (request,'coordinator/manage member.html', {"val": ob})

@login\_required(login\_url='/')

def coordinator\_edit\_member\_post(request):

name = request.POST["name"]

gender = request.POST["gender"]

dob = request.POST["dob"]

district = request.POST["district"]

city = request.POST["city"]

pin = request.POST["pin"]

email = request.POST["email"]

contact\_no = request.POST["contact\_no"]

aadhaar\_number = request.POST["aadhaar\_number"]

obj = Member.objects.get(id=request.session["memberid"])

if Member.objects.filter(contact\_no=contact\_no).exclude(id=obj.id).exists():

messages.error(request, 'Contact number already exists')

return redirect('coordinator\_edit\_member', id=obj.id)

if Member.objects.filter(email=email).exclude(id=obj.id).exists():

messages.error(request, 'Email already exists')

return redirect('coordinator\_edit\_member', id=obj.id)

if Member.objects.filter(aadhaar\_number=aadhaar\_number).exclude(id=obj.id).exists():

messages.error(request, 'Aadhaar number already exists')

return redirect('coordinator\_edit\_member', id=obj.id)

obj.name = name

obj.gender = gender

obj.dob = dob

obj.district = district

obj.city = city

obj.pin = pin

obj.email = email

obj.contact\_no = contact\_no

obj.aadhaar\_number = aadhaar\_number

obj.save()

messages.success(request, 'Member Edited')

return redirect('/coordinator\_manage\_members')

@login\_required(login\_url='/')

def coordinator\_register\_missing\_asset(request):

login\_id = request.session.get('lid')

coordinator = CampCoordinator.objects.get(LOGIN\_id=login\_id)

names = Member.objects.filter(CAMP=coordinator.CAMP)

return render(request,'coordinator/register missing asset.html',{'names':names})

@login\_required(login\_url='/')

def coordinator\_register\_missing\_asset\_post(request):

member = request.POST["member"]

category = request.POST["category"]

assetname = request.POST["assetname"]

description = request.POST["description"]

date = request.POST["date"]

status = 'Pending'

obj = Asset()

obj.MEMBER\_id = member

obj.category = category

obj.asset = assetname

obj.description = description

obj.missing\_date =date

obj.status=status

obj.save()

messages.success(request, 'Missing Asset Registered')

return redirect('/coordinator\_manage\_missing\_asset')

@login\_required(login\_url='/')

def coordinator\_manage\_missing\_asset(request):

login\_id = request.session.get('lid')

coordinator = CampCoordinator.objects.get(LOGIN\_id=login\_id)

names = Member.objects.filter(CAMP=coordinator.CAMP)

ob = Asset.objects.filter(MEMBER\_\_CAMP=coordinator.CAMP)

return render(request, 'coordinator/manage asset registration.html', {'val': ob, 'names': names})

@login\_required(login\_url='/')

def coordinator\_edit\_asset\_registration\_post(request):

category = request.POST["category"]

asset = request.POST["asset"]

description = request.POST["description"]

date = request.POST["date"]

obj = Asset.objects.get(id=request.session["assetid"])

obj.category = category

obj.asset = asset

obj.description = description

obj.missing\_date = date

obj.save()

messages.success(request, 'Asset Registration Edited')

return redirect('/coordinator\_manage\_missing\_asset')

@login\_required(login\_url='/')

def coordinator\_search\_asset\_registration(request):

login\_id = request.session.get('lid')

coordinator = CampCoordinator.objects.get(LOGIN\_id=login\_id)

names = Member.objects.filter(CAMP=coordinator.CAMP)

if request.method == 'POST':

name = request.POST.get('Member', '').strip()

if name:

ob = Asset.objects.filter(MEMBER\_\_name\_\_icontains=name)

else:

ob = Asset.objects.none()

else:

ob = Asset.objects.none()

return render(request, 'coordinator/manage asset registration.html', {

'val': ob,

'names': names

})

@login\_required(login\_url='/')

def coordinator\_add\_needs(request):

ob=CampCoordinator.objects.get(LOGIN=request.session["lid"])

return render(request, 'coordinator/add needs.html',{"ob":ob})

@login\_required(login\_url='/')

def coordinator\_add\_needs\_post(request):

category = request.POST["category"]

product = request.POST["product"]

name = request.POST["name"]

quantity = request.POST["quantity"]

unit = request.POST["unit"]

coordinator = CampCoordinator.objects.get(LOGIN\_id=request.session["lid"])

obj = Needs()

obj.CAMP = coordinator.CAMP

obj.category = category

obj.product = product

obj.quantity = quantity

obj.name = name

obj.unit = unit

obj.save()

messages.success(request, 'Needs Added')

return redirect('/coordinator\_manage\_needs')

@login\_required(login\_url='/')

def coordinator\_edit\_needs\_post(request):

quantity = request.POST["quantity"]

unit = request.POST["unit"]

obj = Needs.objects.get(id=request.session["needsid"])

obj.quantity = quantity

obj.unit = unit

obj.save()

messages.success(request, 'Needs Edited')

return redirect('/coordinator\_manage\_needs')

@login\_required(login\_url='/')

def coordinator\_delete\_needs(request,id):

Needs.objects.get(id=id).delete()

messages.success(request, 'Needs Deleted')

return redirect('/coordinator\_manage\_needs')

@login\_required(login\_url='/')

def coordinator\_search\_needs(request):

login\_id = request.session.get('lid')

coordinator = CampCoordinator.objects.get(LOGIN\_id=login\_id)

product = request.POST['product']

ob = Needs.objects.filter(CAMP=coordinator.CAMP, product\_\_icontains=product)

return render(request, 'coordinator/manage needs.html', {"val": ob})

@login\_required(login\_url='/')

def coordinator\_volunteer\_registration\_post(request):

coid = CampCoordinator.objects.get(LOGIN=request.session['lid'])

name = request.POST["name"]

gender = request.POST["gender"]

dob = request.POST["dob"]

contactno = request.POST["contactno"]

email = request.POST["email"]

district = request.POST["district"]

city = request.POST["city"]

pin = request.POST["pin"]

aadhaar\_number = request.POST["aadhaar\_number"]

username = request.POST["username"]

password = request.POST["password"]

photo = request.FILES["photo"]

if Volunteer.objects.filter(contact\_no=contactno).exists():

messages.error(request, 'Contact number already exists!')

return redirect('/coordinator\_volunteer\_registration')

if Volunteer.objects.filter(email=email).exists():

messages.error(request, 'Email already exists!')

return redirect('/coordinator\_volunteer\_registration')

if Volunteer.objects.filter(aadhaar\_number=aadhaar\_number).exists():

messages.error(request, 'Aadhaar number already exists!')

return redirect('/coordinator\_volunteer\_registration')

if Login.objects.filter(username=username).exists():

messages.error(request, 'Username already exists!')

return redirect('/coordinator\_volunteer\_registration')

fs = FileSystemStorage()

fsave = fs.save(photo.name, photo)

ob = Login()

ob.username = username

ob.password = password

ob.type = "Volunteer"

ob.save()

obj = Volunteer()

obj.LOGIN = ob

obj.name = name

obj.gender = gender

obj.dob = dob

obj.contact\_no = contactno

obj.email = email

obj.district = district

obj.city = city

obj.pin = pin

obj.aadhaar\_number = aadhaar\_number

obj.photo = fsave

obj.CAMP = coid.CAMP

obj.save()

messages.success(request, 'Volunteer Registered')

return redirect('/coordinator\_manage\_volunteer')

@login\_required(login\_url='/')

def coordinator\_manage\_volunteer(request):

coid = CampCoordinator.objects.filter(LOGIN=request.session['lid']).first()

volunteers = Volunteer.objects.filter(LOGIN\_\_type='Volunteer',CAMP=coid.CAMP) if coid else []

return render(request, 'coordinator/manage volunteer.html', {"val": volunteers})

@login\_required(login\_url='/')

def coordinator\_search\_volunteer(request):

coid = CampCoordinator.objects.get(LOGIN=request.session['lid'])

name = request.POST.get('volunteername', '')

volunteers = Volunteer.objects.filter(CAMP=coid.CAMP, name\_\_icontains=name)

return render(request, 'coordinator/manage volunteer.html', {"val": volunteers})

@login\_required(login\_url='/')

def coordinator\_edit\_volunteer(request,id):

request.session["volunteerid"] = id

ob = Volunteer.objects.get(id=id)

return render(request, 'coordinator/edit volunteer.html', {"ob": ob})

@login\_required(login\_url='/')

def coordinator\_edit\_volunteer\_post(request):

name = request.POST["name"]

gender = request.POST["gender"]

dob = request.POST["dob"]

contactno = request.POST["contactno"]

email = request.POST["email"]

district = request.POST["district"]

city = request.POST["city"]

pin = request.POST["pin"]

aadhaar\_number = request.POST["aadhaar\_number"]

obj = Volunteer.objects.get(id=request.session["volunteerid"])

if Volunteer.objects.filter(contact\_no=contactno).exclude(id=obj.id).exists():

messages.error(request, 'Contact number already exists!')

return redirect(f'/coordinator\_edit\_volunteer/{obj.id}')

if Volunteer.objects.filter(email=email).exclude(id=obj.id).exists():

messages.error(request, 'Email already exists!')

return redirect(f'/coordinator\_edit\_volunteer/{obj.id}')

if Volunteer.objects.filter(aadhaar\_number=aadhaar\_number).exclude(id=obj.id).exists():

messages.error(request, 'Aadhaar number already exists!')

return redirect(f'/coordinator\_edit\_volunteer/{obj.id}')

obj.name = name

obj.gender = gender

obj.dob = dob

obj.contact\_no = contactno

obj.email = email

obj.district = district

obj.city = city

obj.pin = pin

obj.aadhaar\_number = aadhaar\_number

obj.save()

messages.success(request, 'Volunteer Edited')

return redirect('/coordinator\_manage\_volunteer')

@login\_required(login\_url='/')

def coordinator\_delete\_volunteer(request, id):

if Volunteer.objects.filter(id=id).exists():

volunteer = Volunteer.objects.get(id=id)

if volunteer.photo:

file\_path = os.path.join(settings.MEDIA\_ROOT, volunteer.photo.name)

if os.path.exists(file\_path):

os.remove(file\_path)

volunteer.delete()

messages.success(request, 'Volunteer Deleted')

else:

messages.error(request, 'Volunteer not found')

return redirect('/coordinator\_manage\_volunteer')

@login\_required(login\_url='/')

def coordinator\_verify\_volunteer(request):

coid = CampCoordinator.objects.filter(LOGIN=request.session['lid']).first()

obb = Volunteer.objects.filter(LOGIN\_\_type='VolPending', CAMP=coid.CAMP)

return render(request, 'coordinator/verify volunteer.html', {"val": obb})

@login\_required(login\_url='/')

def accept\_vol(request, id):

try:

login = Login.objects.get(id=id)

public\_user = Public.objects.get(LOGIN=login)

login.type = 'Volunteer'

login.save()

subject = 'Volunteer Application Accepted'

message = f'Dear {public\_user.name},\n\nYour volunteer application has been accepted. Welcome to the team!\n\nBest regards,\nThe Coordinator Team'

from\_email = settings.EMAIL\_HOST\_USER

recipient\_list = [public\_user.email]

send\_mail(subject, message, from\_email, recipient\_list, fail\_silently=False)

public\_user.delete()

messages.success(request, "Volunteer accepted successfully and email sent!")

except Exception as e:

messages.error(request, f"Error: {str(e)}")

return redirect('/coordinator\_verify\_volunteer')

@login\_required(login\_url='/')

def reject\_vol(request, id):

try:

login = Login.objects.get(id=id)

volunteer = Volunteer.objects.get(LOGIN=login)

public\_user = Public.objects.get(LOGIN=login)

volunteer.delete()

login.type = 'Public'

login.save()

subject = 'Volunteer Application Rejected'

message = f'Dear {public\_user.name},\n\nWe regret to inform you that your volunteer application has been rejected.\n\nBest regards,\nThe Coordinator Team'

from\_email = settings.EMAIL\_HOST\_USER

recipient\_list = [public\_user.email]

send\_mail(subject, message, from\_email, recipient\_list, fail\_silently=False)

messages.success(request, "Volunteer rejected successfully and email sent!")

except Exception as e:

messages.error(request, f"Error: {str(e)}")

return redirect('/coordinator\_verify\_volunteer')

@login\_required(login\_url='/')

def coordinator\_assign\_task(request):

coid = CampCoordinator.objects.filter(LOGIN=request.session['lid']).first()

volunteers = Volunteer.objects.filter(LOGIN\_\_type='Volunteer', CAMP=coid.CAMP) if coid else []

return render(request, 'coordinator/assign task.html', {"val": volunteers})

@login\_required(login\_url='/')

def coordinator\_task\_search\_volunteer(request):

coid = CampCoordinator.objects.get(LOGIN=request.session['lid'])

name = request.POST.get('volunteername', '')

volunteers = Volunteer.objects.filter(LOGIN\_\_type='Volunteer', CAMP=coid.CAMP, name\_\_icontains=name)

return render(request, 'coordinator/assign task.html', {"val": volunteers})

@login\_required(login\_url='/')

def coordinator\_task\_select\_volunteer(request, id):

request.session["volunteerid"] = id

volunteer = Volunteer.objects.get(id=id)

tasks = Task.objects.filter(VOLUNTEER=volunteer)

reversed\_tasks = tasks[::-1]

return render(request, 'coordinator/view task.html', {"ob": volunteer, "tasks": reversed\_tasks})

@login\_required(login\_url='/')

def coordinator\_delete\_task(request,id):

Task.objects.get(id=id).delete()

messages.success(request, 'Task Deleted')

return redirect('/coordinator\_task\_select\_volunteer/'+str(request.session["volunteerid"]))

@login\_required(login\_url='/')

def coordinator\_add\_task\_post(request):

task = request.POST["task"]

volunteer = Volunteer.objects.get(id=request.session["volunteerid"])

obj = Task()

obj.VOLUNTEER = volunteer

obj.task = task

obj.save()

messages.success(request, 'Task Added')

return redirect('/coordinator\_task\_select\_volunteer/'+str(request.session["volunteerid"]))

@login\_required(login\_url='/')

def ert\_dashboard(request):

user\_id = request.session.get('lid')

try:

emergency\_team = EmergencyTeam.objects.get(LOGIN\_\_id=user\_id)

total\_pending\_request = EmergencyRequest.objects.filter(EMERGENCY\_TEAM=emergency\_team, status='Pending').count()

total\_resolved\_request = EmergencyRequest.objects.filter(EMERGENCY\_TEAM=emergency\_team, status='Resolved').count()

total\_ontheway\_request = EmergencyRequest.objects.filter(EMERGENCY\_TEAM=emergency\_team, status='On The Way').count()

except EmergencyTeam.DoesNotExist:

total\_pending\_request = 0

total\_resolved\_request = 0

total\_ontheway\_request = 0

context = {

'total\_camp': Camp.objects.count(),

'total\_ert': Login.objects.filter(type='ERT').count(),

'total\_pending\_request': total\_pending\_request,

'total\_resolved\_request': total\_resolved\_request,

'total\_ontheway\_request': total\_ontheway\_request,

}

return render(request, 'emergency/dashboard.html', context)

@login\_required(login\_url='/')

def ert\_view\_camp(request):

camps = Camp.objects.annotate(

member\_count=Count('member'),

occupancy\_rate=ExpressionWrapper(

(F('member\_count') \* 100.0) / F('capacity'),

output\_field=FloatField()

)

)

camp\_data = []

for camp in camps:

try:

camp\_coordinator = camp.campcoordinator

except CampCoordinator.DoesNotExist:

camp\_coordinator = None

camp\_data.append({

'camp': camp,

'coordinator': camp\_coordinator

})

return render(request, 'emergency/view camp.html', {'camp\_data': camp\_data})

@login\_required(login\_url='/')

def ert\_search\_camp(request):

district = request.POST.get('district', '')

camps = Camp.objects.filter(district\_\_icontains=district).annotate(

member\_count=Count('member'),

occupancy\_rate=ExpressionWrapper(

(F('member\_count') \* 100) / F('capacity'),

output\_field=FloatField()

)

)

camp\_data = []

for camp in camps:

try:

camp\_coordinator = camp.campcoordinator

except CampCoordinator.DoesNotExist:

camp\_coordinator = None

camp\_data.append({

'camp': camp,

'coordinator': camp\_coordinator

})

return render(request, 'emergency/view camp.html', {'camp\_data': camp\_data})

@login\_required(login\_url='/')

def ert\_view\_emergency\_request(request):

ob = EmergencyRequest.objects.filter(EMERGENCY\_TEAM\_\_LOGIN\_id=request.session['lid']).order\_by('-posted\_date', '-posted\_time')

return render(request, 'emergency/emergency request.html', {"val": ob})

@login\_required(login\_url='/')

def ert\_popup\_emergency\_request(request):

if request.headers.get('x-requested-with') == 'XMLHttpRequest':

ob = EmergencyRequest.objects.filter(EMERGENCY\_TEAM\_\_LOGIN\_id=request.session['lid']).order\_by('-posted\_date', '-posted\_time')

data = [{

'id': req.id,

'name': req.PUBLIC.name,

'photo\_url': req.PUBLIC.photo.url,

'city': req.PUBLIC.city,

'district': req.PUBLIC.district,

'pin': req.PUBLIC.pin,

'email': req.PUBLIC.email,

'contact\_no': req.PUBLIC.contact\_no,

'request': req.request,

'posted\_date': req.posted\_date,

'posted\_time': req.posted\_time,

'status': req.status

} for req in ob]

return JsonResponse(data, safe=False)

else:

# Handle initial page load

return render(request, 'emergency/emergency request.html')

@login\_required(login\_url='/')

def ert\_search\_emergency\_request(request):

status = request.POST['status']

ob = EmergencyRequest.objects.filter(status\_\_icontains=status)

return render (request,'emergency/emergency request.html',{'val':ob})

@login\_required(login\_url='/')

def ert\_select\_emergency\_request(request,id):

request.session["emergencyid"]=id

ob=EmergencyRequest.objects.get(id=id)

return render(request, 'emergency/select emergency request.html',{"ob":ob})

@login\_required(login\_url='/')

def ert\_status\_emergency\_post(request):

status=request.POST["status"]

obj = EmergencyRequest.objects.get(id=request.session["emergencyid"])

obj.status=status

obj.save()

messages.success(request, 'Update Status')

return redirect('/ert\_view\_emergency\_request')

def logincode(request):

print(request.POST)

un = request.POST['username']

pwd = request.POST['password']

print(un, pwd)

try:

ob = Login.objects.get(username=un, password=pwd)

if ob is None:

data = {"task": "invalid"}

else:

print("in user function")

data = {"task": "valid", "lid": ob.id,"type":ob.type}

r = json.dumps(data)

print(r)

return HttpResponse(r)

except:

data = {"task": "invalid"}

r = json.dumps(data)

print(r)

return HttpResponse(r)

def public\_registration(request):

if request.method == 'POST':

name = request.POST["name"]

gender = request.POST["gender"]

dob = request.POST["dob"]

contactno = request.POST["contactno"]

email = request.POST["email"]

district = request.POST["district"]

city = request.POST["city"]

pin = request.POST["pin"]

username = request.POST["username"]

password = request.POST["password"]

aadhaar\_number = request.POST["aadhaar"]

photo = request.FILES["image"]

if Login.objects.filter(username=username).exists():

return JsonResponse({"status": "error", "message": "Username already exists"})

if Public.objects.filter(email=email).exists():

return JsonResponse({"status": "error", "message": "Email already exists"})

if Public.objects.filter(contact\_no=contactno).exists():

return JsonResponse({"status": "error", "message": "Contact number already exists"})

if Public.objects.filter(aadhaar\_number=aadhaar\_number).exists():

return JsonResponse({"status": "error", "message": "Aadhaar number already exists"})

fs = FileSystemStorage()

fsave = fs.save(photo.name, photo)

login\_obj = Login(username=username, password=password, type="Public")

login\_obj.save()

public\_obj = Public(

LOGIN=login\_obj,

name=name,

gender=gender,

dob=dob,

contact\_no=contactno,

email=email,

district=district,

city=city,

pin=pin,

aadhaar\_number=aadhaar\_number,

photo=fsave

)

public\_obj.save()

messages.success(request, 'Volunteer Registered')

return JsonResponse({"status": "ok"})

return JsonResponse({"status": "error", "message": "Invalid request method"})

def verify\_current\_password(request):

if request.method == 'POST':

id = request.POST['id']

current\_password = request.POST['current\_password']

if Login.objects.filter(id=id, password=current\_password).exists():

return JsonResponse({"status": "ok"})

else:

return JsonResponse({"status": "error", "message": "Current password is incorrect"})

return JsonResponse({"status": "error", "message": "Invalid request method"})

def changepassword(request):

if request.method == 'POST':

id = request.POST['id']

new\_password = request.POST['new\_password']

confirm\_password = request.POST['confirm\_password']

if new\_password == confirm\_password:

ob = Login.objects.get(id=id)

ob.password = new\_password

ob.save()

return JsonResponse({"status": "ok"})

else:

return JsonResponse({"status": "error", "message": "New password and confirm password do not match"})

return JsonResponse({"status": "error", "message": "Invalid request method"})

def view\_notification(request):

ob=Notification.objects.all()

print(ob,"HHHHHHHHHHHHHHH")

mdata=[]

for i in ob:

data={'title':i.title,'notification':i.notification,'posted\_date':i.posted\_date,'posted\_time':i.posted\_time,'id':i.id}

mdata.append(data)

print(mdata)

return JsonResponse({"status":"ok","data":mdata})

def public\_view\_needs(request):

ob=Needs.objects.all()

print(ob,"HHHHHHHHHHHHHHH")

mdata=[]

for i in ob:

data={'CAMP':i.CAMP.camp\_name,'category':i.category,'product':i.product,'quantity':str(i.quantity),'name':i.name,'unit':i.unit,'added\_on':i.added\_on,'id':i.id}

mdata.append(data)

print(mdata)

return JsonResponse({"status":"ok","data":mdata})

def public\_donate\_goods(request):

lid=request.POST['lid']

category=request.POST["category"]

product=request.POST["product"]

name=request.POST["name"]

quantity = request.POST["quantity"]

unit=request.POST["unit"]

obj=Goods()

obj.PUBLIC=Public.objects.get(LOGIN\_id=lid)

obj.category=category

obj.product=product

obj.name=name

obj.quantity=quantity

obj.unit=unit

obj.status="Pending"

obj.save()

return JsonResponse({"status":"ok"})

def public\_view\_donations(request):

lid = request.GET.get('lid')

if lid is None:

return JsonResponse({"status": "error", "message": "Missing lid parameter"}, status=400)

ob = Goods.objects.filter(PUBLIC\_\_LOGIN\_id=lid)

mdata = []

for i in ob:

data = {

'id': i.id,

'category': i.category,

'product': i.product,

'name': i.name,

'quantity': i.quantity,

'donated\_on': str(i.donated\_on),

'status': i.status,

'volunteer\_name': i.VOLENTEER.name if i.VOLENTEER else 'No Volunteer Assigned',

'volunteer\_phone': i.VOLENTEER.contact\_no if i.VOLENTEER else 'No Volunteer Assigned'

}

mdata.append(data)

return JsonResponse({"status": "ok", "data": mdata})

def delete\_donation(request):

id = request.GET.get('id')

if id is None:

return JsonResponse({"status": "error", "message": "Missing id parameter"}, status=400)

try:

donation = Goods.objects.get(id=id)

donation.delete()

return JsonResponse({"status": "ok", "message": "Donation deleted successfully"})

except Goods.DoesNotExist:

return JsonResponse({"status": "error", "message": "Donation not found"}, status=404)

except Exception as e:

return JsonResponse({"status": "error", "message": str(e)}, status=500)

def public\_vol\_registration(request):

try:

login\_id = request.POST['login\_id']

camp\_id = request.POST['camp\_id']

public = Public.objects.get(LOGIN\_id=login\_id)

camp = Camp.objects.get(id=camp\_id)

login = public.LOGIN

login.type = 'VolPending'

login.save()

Volunteer.objects.create(

CAMP=camp,

LOGIN=login,

name=public.name,

gender=public.gender,

dob=public.dob,

district=public.district,

city=public.city,

pin=public.pin,

email=public.email,

aadhaar\_number = public.aadhaar\_number,

contact\_no=public.contact\_no,

photo=public.photo

)

return JsonResponse({"status": "ok"})

except Exception as e:

return JsonResponse({"status": "error", "message": str(e)})

def get\_camps(request):

ob = Camp.objects.all()

mdata = []

for i in ob:

data = {'camp\_name': i.camp\_name,'id': i.id}

mdata.append(data)

print(mdata)

return JsonResponse({"status": "ok", "data": mdata})

def public\_pay\_donation(request):

lid = request.POST['lid']

amount = request.POST['amount']

obj = Payment()

obj.PUBLIC = Public.objects.get(LOGIN\_id=lid)

obj.amount = amount

obj.save()

return JsonResponse({"status": "ok"})

def public\_view\_payments(request):

lid = request.GET.get('lid')

if lid is None:

return JsonResponse({"status": "error", "message": "Missing lid parameter"}, status=400)

ob = Payment.objects.filter(PUBLIC\_\_LOGIN\_id=lid)

mdata = []

for i in ob:

data = {

'id': i.id,

'amount': i.amount,

'payment\_date': str(i.payment\_date),

'payment\_time': str(i.payment\_time)

}

mdata.append(data)

return JsonResponse({"status": "ok", "data": mdata})

def public\_view\_profile(request):

lid = request.POST.get('lid', None)

if not lid:

return JsonResponse({"status": "error", "message": "Missing login ID"})

try:

ob = Public.objects.filter(LOGIN\_id=lid)

if not ob.exists():

return JsonResponse({"status": "error", "message": "No user found"})

mdata = []

for i in ob:

login\_instance = i.LOGIN

data = {

'name': i.name,

'gender': i.gender,

'dob': str(i.dob),

'district': i.district,

'city': i.city,

'pin': str(i.pin),

'email': i.email,

'contact\_no': str(i.contact\_no),

'photo': i.photo.url if i.photo else "",

'id': str(i.id),

'aadhaar\_number': str(i.aadhaar\_number),

'joined\_date': str(i.joined\_date),

'login\_type': login\_instance.type

}

mdata.append(data)

return JsonResponse({"status": "ok", "data": mdata})

except Exception as e:

return JsonResponse({"status": "error", "message": str(e)})

def volunteer\_view\_needs(request):

lid = request.GET.get('lid')

volunteer = Volunteer.objects.get(LOGIN\_id=lid)

ob = Needs.objects.filter(CAMP=volunteer.CAMP)

mdata = []

for i in ob:

data = {

'name': i.name,

'unit': i.unit,

'category': i.category,

'product': i.product,

'quantity': i.quantity,

'added\_on': i.added\_on.strftime('%Y-%m-%d %H:%M:%S'),

'id': i.id,

'CAMP': i.CAMP.camp\_name if i.CAMP.camp\_name else "Unknown Camp"

}

mdata.append(data)

return JsonResponse({"status": "ok", "data": mdata})

def vol\_view\_goods(request):

goods = Goods.objects.filter(status='Pending')

mdata = []

for item in goods:

data = {

'id': item.id,

'category': item.category,

'product': item.product,

'name': item.name,

'quantity': item.quantity,

'donated\_on': item.donated\_on.strftime('%Y-%m-%d %H:%M:%S'),

'status': item.status,

'volunteer\_name': item.VOLENTEER.name if item.VOLENTEER else 'No Volunteer Assigned',

'volunteer\_phone': item.VOLENTEER.contact\_no if item.VOLENTEER else 'No Volunteer Assigned'

}

mdata.append(data)

return JsonResponse({"status": "ok", "data": mdata})

def update\_pickup(request):

print(request.POST,'++++++++++++==================')

gid = request.POST['gid']

lid = request.POST['lid']

goods = Goods.objects.get(id=gid)

goods.VOLENTEER = Volunteer.objects.get(LOGIN\_id=lid)

goods.status = 'Ready for pickup'

goods.save()

return JsonResponse({"status": "ok"})

def vol\_cancel\_pickup(request):

gid = request.POST.get('gid')

try:

goods = Goods.objects.get(id=gid)

goods.VOLENTEER = None

goods.status = 'Pending'

goods.save()

return JsonResponse({"status": "ok"})

except Goods.DoesNotExist:

return JsonResponse({"status": "error", "message": "Goods not found"})

def vol\_collected\_goods(request):

gid = request.POST['gid']

goods = Goods.objects.get(id=gid)

goods.status = 'Donated'

goods.save()

ob=Stock()

ob.CAMP\_id=goods.VOLENTEER.CAMP.id

ob.category=goods.category

ob.product=goods.product

ob.name=goods.name

ob.quantity=goods.quantity

ob.unit=goods.unit

ob.donated=goods.PUBLIC.name

ob.save()

return JsonResponse({"status": "ok"})

def vol\_show\_completed\_pickup(request):

lid = request.POST.get('lid')

ob = Goods.objects.filter(VOLENTEER\_\_LOGIN\_id=lid, status='Donated')

mdata = []

for i in ob:

data = {

'id': i.id,

'category': i.category,

'product': i.product,

'name': i.name,

'quantity': i.quantity,

'donated\_on': i.donated\_on.strftime('%Y-%m-%d %H:%M:%S'),

'public\_name': i.PUBLIC.name,

'public\_phone': i.PUBLIC.contact\_no

}

mdata.append(data)

return JsonResponse({"status": "ok", "data": mdata})

def volunteer\_view\_task(request):

lid = request.GET.get('lid')

if lid is None:

return JsonResponse({"status": "error", "message": "Missing lid parameter"}, status=400)

ob = Task.objects.filter(VOLUNTEER\_\_LOGIN\_id=lid)

mdata = []

for i in ob:

data = {

'id': i.id,

'task': i.task,

'assigned\_on': str(i.posted\_date),

'status': i.status

}

mdata.append(data)

return JsonResponse({"status": "ok", "data": mdata})

def toggle\_status\_old(request):

tid = request.POST.get('tid')

if tid is None:

return JsonResponse({"status": "error", "message": "Missing tid parameter"}, status=400)

try:

ob = Task.objects.get(id=tid)

ob.status = '0' if ob.status == '1' else '1'

ob.save()

return JsonResponse({"status": "ok", "new\_status": ob.status})

except Task.DoesNotExist:

return JsonResponse({"status": "error", "message": "Task not found"}, status=404)

def toggle\_status(request):

tid = request.POST.get('tid')

if tid is None:

return JsonResponse({"status": "error", "message": "Missing tid parameter"}, status=400)

try:

ob = Task.objects.get(id=tid)

ob.status = '1' if ob.status == '0' else '0'

ob.save()

return JsonResponse({"status": "ok", "new\_status": ob.status})

except Task.DoesNotExist:

return JsonResponse({"status": "error", "message": "Task not found"}, status=404)

def volunteer\_view\_profile(request):

lid = request.POST.get('lid')

print(lid,"+++++++++++++")

i = Volunteer.objects.get(LOGIN=lid)

data = [{

'name': i.name,

'gender': i.gender,

'dob': str(i.dob),

'district': i.district,

'city': i.city,

'pin': str(i.pin),

'email': i.email,

'contact\_no': str(i.contact\_no),

'photo': i.photo.url if i.photo else "",

'aadhaar\_number': str(i.aadhaar\_number),

'joined\_date': str(i.posted\_date),

'id': str(i.id)

}]

print(data)

return JsonResponse({"status": "ok", "data": data})

def volunteer\_edit\_profile(request):

lid = request.POST.get('lid', None)

name = request.POST.get('name', None)

gender = request.POST.get('gender', None)

dob = request.POST.get('dob', None)

district = request.POST.get('district', None)

city = request.POST.get('city', None)

pin = request.POST.get('pin', None)

email = request.POST.get('email', None)

contact\_no = request.POST.get('contact\_no', None)

aadhaar\_number = request.POST.get('aadhaar\_number', None)

if Volunteer.objects.filter(email=email).exclude(LOGIN\_id=lid).exists():

return JsonResponse({'status': 'error', 'message': 'Email already exists'})

if Volunteer.objects.filter(contact\_no=contact\_no).exclude(LOGIN\_id=lid).exists():

return JsonResponse({'status': 'error', 'message': 'Contact number already exists'})

if Volunteer.objects.filter(aadhaar\_number=aadhaar\_number).exclude(LOGIN\_id=lid).exists():

return JsonResponse({'status': 'error', 'message': 'Aadhaar number already exists'})

p = Volunteer.objects.get(LOGIN\_id=lid)

if 'photo' in request.FILES:

photo = request.FILES['photo']

fs = FileSystemStorage()

fp = fs.save(photo.name, photo)

p.photo = fp

p.save()

p.name = name

p.gender = gender

p.dob = dob

p.district = district

p.city = city

p.pin = pin

p.email = email

p.contact\_no = contact\_no

p.aadhaar\_number = aadhaar\_number

p.save()

return JsonResponse({'status': 'ok'})

def public\_send\_complaint(request):

lid = request.POST['lid']

complaint = request.POST['complaint']

obj=Complaint()

obj.PUBLIC=Public.objects.get(LOGIN\_id=lid)

obj.complaint=complaint

obj.status="Pending"

obj.reply="Not Replied"

obj.save()

return JsonResponse({"status":"ok"})

def updatelocation(request):

lid = request.POST['lid']

lati = request.POST['lat']

loni = request.POST['lon']

print(lati,loni,"kkkkkkkkkkkkkkk")

return JsonResponse({"status":"ok"})

def send\_emergency\_request(request):

tid=request.POST['tid']

lid=request.POST['lid']

latitude=request.POST['latitude']

longitude=request.POST['longitude']

request=request.POST['request']

obj=EmergencyRequest()

obj.PUBLIC=Public.objects.get(LOGIN\_id=lid)

obj.EMERGENCY\_TEAM=EmergencyTeam.objects.get(id=tid)

obj.latitude=latitude

obj.longitude=longitude

obj.request=request

obj.status="Pending"

obj.save()

return JsonResponse({"status":"ok"})

def public\_view\_emergency\_team(request):

ob = EmergencyTeam.objects.filter(LOGIN\_\_type="ERT")

mdata = []

for i in ob:

data = {'department': i.department,

'district' : i.district,

'city' : i.city,

'pin' : i.pin,

'email' : i.email,

'contact\_no' : i.contact\_no,

'joined\_date' : i.joined\_date,

'id': i.id

}

mdata.append(data)

print(mdata)

return JsonResponse({"status": "ok", "data": mdata})

def vol\_view\_missing\_asset(request):

lid = request.POST['lid']

print(lid)

obx=Volunteer.objects.get(LOGIN\_id=lid)

ob = Asset.objects.filter(MEMBER\_\_CAMP\_id=obx.CAMP.id)

mdata = []

for i in ob:

data = {

'id': i.id,

'category': i.category,

'asset': i.asset,

'description': i.description,

'status': i.status,

'missing\_date': i.missing\_date.strftime('%Y-%m-%d %H:%M:%S'),

'posted\_date': i.posted\_date,

}

mdata.append(data)

print(mdata)

return JsonResponse({"status": "ok", "data": mdata})

def user\_viewchat(request):

cid =request.POST["camp\_id"]

obv=Volunteer.objects.get(LOGIN\_\_id=cid)

res = GroupChat.objects.filter(CAMP\_\_id=obv.CAMP.id).order\_by("id")

l = []

for i in res:

if str(i.LOGIN\_id) == cid:

l.append({"id": i.id, "msg": i.message, "from": i.LOGIN\_id, "date": i.date, })

else:

l.append({"id": i.id, "msg": i.LOGIN.username+"("+i.LOGIN.type+"): "+i.message, "from": i.LOGIN\_id, "date": i.date, })

return JsonResponse({"status":"ok",'data':l})

def user\_sendchat(request):

LOGIN\_id=request.POST['lid']

print(LOGIN\_id)

msg=request.POST['message']

obv=Volunteer.objects.get(LOGIN\_\_id=LOGIN\_id)

from datetime import datetime

c=GroupChat()

c.LOGIN\_id=LOGIN\_id

c.CAMP=obv.CAMP

c.message=msg

c.date=datetime.now()

c.save()

return JsonResponse({'status':"ok"})

def cod\_view\_chat(request):

cid = request.POST["camp\_id"]

obv=Volunteer.objects.get(LOGIN\_\_id=cid)

res = GroupChat.objects.filter(CAMP\_\_id=obv.CAMP.id).order\_by("id")

l = []

for i in res:

if str(i.LOGIN\_id) == cid:

l.append({"id": i.id, "msg": i.message, "from": i.LOGIN\_id, "date": i.date, })

else:

l.append({"id": i.id, "msg": i.LOGIN.username+"("+i.LOGIN.type+"): "+i.message, "from": i.LOGIN\_id, "date": i.date, })

return JsonResponse({"status":"ok",'data':l})

@login\_required(login\_url='/')

def cod\_groupchat(request):

ob=CampCoordinator.objects.get(LOGIN\_id=request.session['lid'])

return render(request,"coordinator/group chat.html",{"cname":ob.CAMP.camp\_name})

def coun\_insert\_chat(request,msg):

print("===",msg)

# id== camp select

obc=CampCoordinator.objects.get(LOGIN\_id=request.session['lid'])

ob=GroupChat()

ob.LOGIN=Login.objects.get(id=request.session['lid'])

ob.CAMP=obc.CAMP

ob.message=msg

ob.date=datetime.datetime.now().strftime("%Y-%m-%d")

ob.save()

return JsonResponse({"task":"ok"})

def coun\_msg(request):

obc = CampCoordinator.objects.get(LOGIN\_\_id=request.session['lid'])

ob1 = GroupChat.objects.filter(CAMP\_\_id=obc.CAMP.id)

combined\_chat = ob1

combined\_chat = combined\_chat.order\_by('id')

res = []

today = timezone.now().date()

yesterday = today - timedelta(days=1)

for i in combined\_chat:

message\_date = i.date

if message\_date == today:

date\_label = "Today"

elif message\_date == yesterday:

date\_label = "Yesterday"

else:

date\_label = message\_date.strftime("%Y-%m-%d")

sender = None

photo\_url = None

sender\_name = "Unknown"

sender = Volunteer.objects.filter(LOGIN=i.LOGIN).first()

if sender:

photo\_url = sender.photo.url if sender.photo else None

sender\_name = sender.name

else:

sender = CampCoordinator.objects.filter(LOGIN=i.LOGIN).first()

if sender:

photo\_url = sender.photo.url if sender.photo else None

sender\_name = sender.name

res.append({

"from\_id": i.LOGIN.id,

"msg": i.message,

"date": date\_label,

"time": i.time.strftime("%I:%M %p"),

"chat\_id": i.id,

"photo\_url": photo\_url,

"sender\_name": sender\_name,

})

obu = CampCoordinator.objects.get(LOGIN\_\_id=request.session['lid'])

return JsonResponse({"data": res, "name": obu.name, "user\_lid": obu.LOGIN.id})

API\_KEY = '59066fda027f4b7ea3d64384a7b32ede'

def get\_weather(request):

if request.method == 'POST':

try:

data = json.loads(request.body)

lat = data.get('latitude')

lon = data.get('longitude')

distance\_str = "1000"

# Get river data

river\_data = get\_rivers\_near\_location(lat, lon)

if "error" in river\_data:

print(river\_data["error"])

else:

print(f"Rivers near location ({lat}, {lon}):")

for river in river\_data:

if isinstance(river["distance\_km"], (float, int)):

distance\_str = float(river['distance\_km'])

break

else:

distance\_str = "Distance not available"

break

elevation, soil\_type, slope\_angle = getmaindata(float(lat), float(lon))

print(elevation, soil\_type, slope\_angle, distance\_str)

if lat is None or lon is None:

return JsonResponse({'error': 'Invalid coordinates'}, status=400)

api\_url = f'https://api.weatherbit.io/v2.0/current?lat={lat}&lon={lon}&key={API\_KEY}'

response = requests.get(api\_url)

weather\_data = response.json()

print(weather\_data)

except json.JSONDecodeError:

return JsonResponse({'error': 'Invalid JSON'}, status=400)

knn = joblib.load(r"C:\Users\shaha\Desktop\Project\Web\app\knn-model.joblib")

try:

st = [

'Fluvisols',

'Andosols',

'Arenosols',

'Chernozem',

'Gleysols',

'Histosols',

'Kastanozems',

'Luvisols',

'Nitisols',

'Regosols',

'Vertisols',

'Solonchaks',

'Podzols',

'Alisols',

'Cambisols',

'Calcisols',

'Phaeozems',

'Acrisols',

'Plinthosols'

]

row = [float(lat), float(lon), float(elevation), float(distance\_str), float(slope\_angle),weather\_data['data'][0]['precip'],st.index(soil\_type),weather\_data['data'][0]['rh'],weather\_data['data'][0]['rh']] # Adjust as necessary

if len(row) != knn.n\_features\_in\_:

raise ValueError(f"Expected {knn.n\_features\_in\_} features, but got {len(row)}")

res = knn.predict([row])

print(res, "++++++++++++++++++")

if res[0] == 0:

return JsonResponse({'status': 'ok', 'val': 'Non-landslide','weather\_data':weather\_data,'st':soil\_type,'river':distance\_str,'altitude':elevation,'rainfall':weather\_data['data'][0]['precip']})

else:

return JsonResponse({'status': 'not ok', 'val': 'Landslide','weather\_data':weather\_data,'st':soil\_type,'river':distance\_str,'altitude':elevation,'rainfall':weather\_data['data'][0]['precip']})

except ValueError as e:

print("Error in prediction:", str(e))

return JsonResponse({'error': 'Prediction failed', 'details': str(e)}, status=500)

return JsonResponse({'status': 'not ok', 'val': 'Landslide','weather\_data':weather\_data})

def map\_start(request):

return render(request,'emergency/map\_start.html')

def get\_predicttt(request):

if request.method == 'POST':

try:

lat = float(request.POST['lat'])

lon = float(request.POST['lon'])

distance\_str = "1000"

river\_data = get\_rivers\_near\_location(lat, lon)

if "error" in river\_data:

print(river\_data["error"])

else:

print(f"Rivers near location ({lat}, {lon}):")

for river in river\_data:

if isinstance(river["distance\_km"], (float, int)):

distance\_str = float(river['distance\_km'])

break

else:

distance\_str = "Distance not available"

break

elevation, soil\_type, slope\_angle = getmaindata(float(lat), float(lon))

print(elevation, soil\_type, slope\_angle, distance\_str)

if lat is None or lon is None:

return render(request,"emergency/result.html",{"val":"invalid"})

api\_url = f'https://api.weatherbit.io/v2.0/current?lat={lat}&lon={lon}&key={API\_KEY}'

response = requests.get(api\_url)

weather\_data = response.json()

print(weather\_data)

except json.JSONDecodeError as e:

print(e)

return render(request, "emergency/result.html", {"val": "invalid"})

knn = joblib.load(r"C:\Users\shaha\Desktop\Project\Web\app\knn-model1.joblib")

try:

st = [

'Fluvisols',

'Andosols',

'Arenosols',

'Chernozem',

'Gleysols',

'Histosols',

'Kastanozems',

'Luvisols',

'Nitisols',

'Regosols',

'Vertisols',

'Solonchaks',

'Podzols',

'Alisols',

'Cambisols',

'Calcisols',

'Phaeozems',

'Acrisols',

'Plinthosols'

]

try:

row = [float(lat), float(lon), float(elevation), float(distance\_str), float(slope\_angle),weather\_data['data'][0]['precip'],st.index(soil\_type),weather\_data['data'][0]['rh'],weather\_data['data'][0]['rh']] # Adjust as necessary

except:

row = [float(lat), float(lon), float(elevation), float(distance\_str), float(slope\_angle),1,st.index(soil\_type),1,1] # Adjust as necessary

if len(row) != knn.n\_features\_in\_:

raise ValueError(f"Expected {knn.n\_features\_in\_} features, but got {len(row)}")

res = knn.predict([row])

print(res, "++++++++++++++++++")

if res[0] == 0:

try:

return render(request,"emergency/result.html",{'status': 'ok', 'val': 'Non-landslide','weather\_data':weather\_data,'st':soil\_type,'river':distance\_str,'altitude':elevation,'rainfall':weather\_data['data'][0]['precip'],'city\_name':weather\_data['data'][0]['city\_name']})

except:

return render(request,"emergency/result.html",{'status': 'ok', 'val': 'Non-landslide','weather\_data':weather\_data,'st':soil\_type,'river':distance\_str,'altitude':elevation,'rainfall':5,'city\_name':""})

else:

try:

return HttpResponse({'status': 'not ok', 'val': 'Landslide','weather\_data':weather\_data,'st':soil\_type,'river':distance\_str,'altitude':elevation,'rainfall':weather\_data['data'][0]['precip'],'city\_name':weather\_data['data'][0]['city\_name']})

except:

return HttpResponse({'status': 'not ok', 'val': 'Landslide','weather\_data':weather\_data,'st':soil\_type,'river':distance\_str,'altitude':elevation,'rainfall':5,'city\_name':""})

except ValueError as e:

print("Error in prediction:", str(e))

return JsonResponse({'error': 'Prediction failed', 'details': str(e)}, status=500)

return JsonResponse({'status': 'not ok', 'val': 'Landslide','weather\_data':weather\_data})

DART CODE

LOGIN

import 'dart:convert';  
import 'package:flutter/material.dart';  
import 'package:shared\_preferences/shared\_preferences.dart';  
import 'package:http/http.dart' as http;  
import 'package:crisisguard/Registration.dart';  
import 'package:crisisguard/Public/Home.dart';   
import 'package:crisisguard/Volunteer/Home.dart';  
import 'Public/location.dart';  
class Login extends StatefulWidget {  
 const Login({super.key});  
 @override  
 State<Login> createState() => \_LoginState();  
}  
class \_LoginState extends State<Login> {  
 final TextEditingController usernameController = TextEditingController();  
 final TextEditingController passwordController = TextEditingController();  
 final \_formKey = GlobalKey<FormState>();  
 bool isLoading = false;   
 @override  
 Widget build(BuildContext context) {  
 return Scaffold(  
 appBar: AppBar(  
 title: Row(  
 children: [  
 Image.asset(  
 'assets/logo/logo.png',   
 width: 30,  
 height: 30,  
 ),  
 const SizedBox(width: 10),  
 const Text(  
 "Crisis Guard",  
 style: TextStyle(  
 fontSize: 24,  
 fontWeight: FontWeight.*bold*,  
 color: Colors.*white*,  
 ),  
 ),  
 ],  
 ),  
 backgroundColor: Colors.*blue*.shade800,  
 elevation: 10,  
 ),  
 body: Container(  
 decoration: BoxDecoration(  
 gradient: LinearGradient(  
 begin: Alignment.*topCenter*,  
 end: Alignment.*bottomCenter*,  
 colors: [  
 Colors.*blue*.shade200,  
 Colors.*blue*.shade50,  
 ],  
 ),  
 ),  
 child: SafeArea(  
 child: Center(  
 child: SingleChildScrollView(  
 child: Padding(  
 padding: const EdgeInsets.all(20.0),  
 child: Form(  
 key: \_formKey,  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 crossAxisAlignment: CrossAxisAlignment.center,  
 children: [  
 const Text(  
 "Login",  
 style: TextStyle(  
 fontSize: 35,  
 fontWeight: FontWeight.*bold*,  
 color: Color(0xFF0D47A1),  
 ),  
 ),  
 const SizedBox(height: 40),  
 Padding(  
 padding: const EdgeInsets.symmetric(horizontal: 20.0),  
 child: ClipRRect(  
 borderRadius: BorderRadius.circular(15),  
 child: TextFormField(  
 controller: usernameController,  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 labelText: "Username",  
 hintText: "Enter your username",  
 filled: true,  
 fillColor: Colors.*white*,  
 contentPadding: const EdgeInsets.symmetric(  
 vertical: 15,  
 horizontal: 20,  
 ),  
 labelStyle: TextStyle(  
 color: Colors.*blue*.shade800,  
 fontSize: 16,  
 ),  
 hintStyle: TextStyle(  
 color: Colors.*grey*.shade500,  
 fontSize: 16,  
 ),  
 ),  
 style: TextStyle(  
 color: Colors.*blue*.shade800,  
 fontSize: 16,  
 ),  
 validator: (value) {  
 if (value == null || value.isEmpty) {  
 return 'Please enter your username';  
 }  
 return null;  
 },  
 ),  
 ),  
 ),  
 const SizedBox(height: 20),  
 Padding(  
 padding: const EdgeInsets.symmetric(horizontal: 20.0),  
 child: ClipRRect(  
 borderRadius: BorderRadius.circular(15),  
 child: TextFormField(  
 controller: passwordController,  
 obscureText: true,  
 decoration: InputDecoration(  
 border: InputBorder.*none*,  
 labelText: "Password",  
 hintText: "Enter your password",  
 filled: true,  
 fillColor: Colors.*white*,  
 contentPadding: const EdgeInsets.symmetric(  
 vertical: 15,  
 horizontal: 20,  
 ),  
 labelStyle: TextStyle(  
 color: Colors.*blue*.shade800,  
 fontSize: 16,  
 ),  
 hintStyle: TextStyle(  
 color: Colors.*grey*.shade500,  
 fontSize: 16,  
 ),  
 ),  
 style: TextStyle(  
 color: Colors.*blue*.shade800,  
 fontSize: 16,  
 ),  
 validator: (value) {  
 if (value == null || value.isEmpty) {  
 return 'Please enter your password';  
 }  
 return null;  
 },  
 ),  
 ),  
 ),  
 const SizedBox(height: 30),   
 Padding(  
 padding: const EdgeInsets.all(8.0),  
 child: ElevatedButton(  
 onPressed: isLoading  
 ? null   
 : () async {  
 if (\_formKey.currentState!.validate()) {  
 setState(() {  
 isLoading = true;   
 });  
 final sh = await SharedPreferences.*getInstance*();  
 String Uname = usernameController.text.toString();  
 String Passwd = passwordController.text.toString();  
 String url = sh.getString("url").toString();  
  
 try {  
 var data = await http.post(  
 Uri.*parse*(url + "logincode"),  
 body: {  
 'username': Uname,  
 "password": Passwd,  
 },  
 );  
 var jasondata = json.decode(data.body);  
 String status = jasondata['task'].toString();  
 String type = jasondata['type'].toString();  
  
 if (status == "valid") {  
 if (type == 'Public' || type == 'VolPending') {  
 String lid = jasondata['lid'].toString();  
 sh.setString("lid", lid);  
 Navigator.*push*(  
 context,  
 MaterialPageRoute(builder: (context) => home()),  
 );  
 } else if (type == 'Volunteer' ) {  
 String lid = jasondata['lid'].toString();  
 sh.setString("lid", lid);  
 Navigator.*push*(  
 context,  
 MaterialPageRoute(builder: (context) => VolunteerHome()),  
 );  
 } else {  
 ScaffoldMessenger.*of*(context).showSnackBar(  
 SnackBar(  
 content: Text("Error: Invalid user type"),  
 backgroundColor: Colors.*red*,  
 duration: Duration(seconds: 3),  
 ),  
 );  
 }  
 } else {  
 ScaffoldMessenger.*of*(context).showSnackBar(  
 SnackBar(  
 content: Text("Invalid username or password"),  
 backgroundColor: Colors.*red*,  
 duration: Duration(seconds: 3),  
 ),  
 );  
 }  
 } catch (e) {  
 ScaffoldMessenger.*of*(context).showSnackBar(  
 SnackBar(  
 content: Text("An error occurred. Please try again."),  
 backgroundColor: Colors.*red*,  
 duration: Duration(seconds: 3),  
 ),  
 );  
 } finally {  
 setState(() {  
 isLoading = false;   
 });  
 }  
 }  
 },  
 child: isLoading  
 ? CircularProgressIndicator(  
 color: Colors.*white*,  
 )  
 : const Text(  
 "Login",  
 style: TextStyle(  
 fontSize: 18,  
 fontWeight: FontWeight.*bold*,  
 color: Colors.*white*,  
 ),  
 ),  
 style: ElevatedButton.*styleFrom*(  
 backgroundColor: Colors.*blue*.shade800,  
 padding: const EdgeInsets.symmetric(  
 vertical: 15,  
 horizontal: 20,  
 ),  
 shape: RoundedRectangleBorder(  
 borderRadius: BorderRadius.circular(15),  
 ),  
 ),  
 ),  
 ),  
 const SizedBox(height: 20),  
 GestureDetector(  
 onTap: () {  
 Navigator.*push*(  
 context,  
 MaterialPageRoute(builder: (context) => SignUpForm()),  
 );  
 },  
 child: RichText(  
 text: const TextSpan(  
 text: "Don't have an account? ",  
 style: TextStyle(  
 fontSize: 16,  
 color: Colors.*black*,  
 ),  
 children: <TextSpan>[  
 TextSpan(  
 text: 'Register here',  
 style: TextStyle(  
 fontSize: 16,  
 fontWeight: FontWeight.*bold*,  
 color: Colors.*blue*,  
 ),  
 ),  
 ],  
 ),  
 ),  
 ),  
 ],  
 ),  
 ),  
 ),  
 ),  
 ),  
 ),  
 ),  
 );  
 }  
}

PUBLIC DRAWER

import 'package:crisisguard/Public/ViewDonation.dart';

import 'package:crisisguard/Public/home.dart';

import 'package:crisisguard/Public/viewneeds.dart';

import 'package:crisisguard/Public/DonateGoods.dart';

import 'package:crisisguard/ViewNotification.dart';

import 'package:crisisguard/Public/VolunteerRegistration.dart';

import 'package:flutter/material.dart';

import '../Login.dart';

import 'ViewEmergencyTeam.dart';

import 'Payment.dart';

import 'Profile.dart';

import 'SendComplaint.dart';

import 'ViewComplaint.dart';

import 'ViewPayDonation.dart';

class Drawerclass extends StatelessWidget {

const Drawerclass({Key? key}) : super(key: key);

@override

Widget build(BuildContext context) {

return Drawer(

width: 280,

elevation: 0,

child: Container(

color: Colors.white,

child: ListView(

padding: EdgeInsets.zero,

children: <Widget>[

DrawerHeader(

decoration: BoxDecoration(

gradient: LinearGradient(

colors: [Colors.blue[700]!, Colors.blue[500]!],

begin: Alignment.topLeft,

end: Alignment.bottomRight,

),

),

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Image.asset(

'assets/logo/logo.png'

width: 60,

height: 60,

),

SizedBox(height: 10),

Text(

"Crisis Guard",

style: TextStyle(

color: Colors.white,

fontSize: 24,

fontWeight: FontWeight.bold,

),

),

SizedBox(height: 5),

Text(

"Your Safety, Our Priority",

style: TextStyle(

color: Colors.white.withOpacity(0.8),

fontSize: 14,

fontWeight: FontWeight.w400,

),

),

],

),

),

\_buildListTile(

icon: Icons.home\_outlined,

title: "Home",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => const home()));

},

),

\_buildListTile(

icon: Icons.assignment\_outlined,

title: "Camp Needs",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => ViewNeedsPage()));

},

),

\_buildExpansionTile(

icon: Icons.card\_giftcard\_outlined,

title: "Donate Goods",

children: [

Padding(

padding: const EdgeInsets.only(left: 24.0),

child: \_buildListTile(

icon: Icons.add\_circle\_outline,

title: "Donate",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => DonateGoodsForm()));

},

),

),

Padding(

padding: const EdgeInsets.only(left: 24.0),

child: \_buildListTile(

icon: Icons.list\_alt\_outlined,

title: "My Donations",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => PublicViewDonations()));

},

),

),

],

),

\_buildListTile(

icon: Icons.people\_outlined,

title: "Volunteer Registration",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => PublicVolunteerRegistration()));

},

),

\_buildExpansionTile(

icon: Icons.currency\_rupee\_rounded,

title: "Pay Donation",

children: [

Padding(

padding: const EdgeInsets.only(left: 24.0),

child: \_buildListTile(

icon: Icons.add\_card\_sharp,

title: "Donate",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => PaymentPage()));

},

),

),

Padding(

padding: const EdgeInsets.only(left: 24.0),

child: \_buildListTile(

icon: Icons.payments\_outlined,

title: "Past Donation",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => PublicViewPayments()));

},

),

),

],

),

\_buildListTile(

icon: Icons.emergency\_outlined,

title: "Emergency Request",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => PublicViewEmergencyTeam()));

},

),

\_buildListTile(

icon: Icons.notifications\_outlined,

title: "Notification",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => ViewNotificationPage()));

},

),

\_buildExpansionTile(

icon: Icons.error\_outline,

title: "Complaint",

children: [

Padding(

padding: const EdgeInsets.only(left: 24.0),

child: \_buildListTile(

icon: Icons.send\_rounded,

title: "Send Complaint",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => PublicSendComplaint()));

},

),

),

Padding(

padding: const EdgeInsets.only(left: 24.0),

child: \_buildListTile(

icon: Icons.reviews\_outlined,

title: "My Complaint",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => PublicViewComplaints()));

},

),

),

],

),

\_buildListTile(

icon: Icons.person\_outlined,

title: "Profile",

onTap: () {

Navigator.push(context, MaterialPageRoute(builder: (context) => PublicViewProfile()));

},

),

Divider(

color: Colors.grey[300],

thickness: 1,

indent: 20,

endIndent: 20,

),

LogoutFeature.buildLogoutListTile(context),

],

),

),

);

}

Widget \_buildListTile({required IconData icon, required String title, required VoidCallback onTap}) {

return ListTile(

leading: Container(

padding: EdgeInsets.all(8),

decoration: BoxDecoration(

color: Colors.blue[800]!.withOpacity(0.1),

borderRadius: BorderRadius.circular(8),

),

child: Icon(

icon,

color: Colors.blue[800],

size: 24,

),

),

title: Text(

title,

style: TextStyle(

color: Colors.grey[800],

fontSize: 16,

fontWeight: FontWeight.w500,

),

),

onTap: onTap,

hoverColor: Colors.blue[50],

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(10),

),

);

}

Widget \_buildExpansionTile({required IconData icon, required String title, required List<Widget> children}) {

return ExpansionTile(

leading: Container(

padding: EdgeInsets.all(8),

decoration: BoxDecoration(

color: Colors.blue[800]!.withOpacity(0.1),

borderRadius: BorderRadius.circular(8),

),

child: Icon(

icon,

color: Colors.blue[800],

size: 24,

),

),

title: Text(

title,

style: TextStyle(

color: Colors.grey[800],

fontSize: 16,

fontWeight: FontWeight.w500,

),

),

children: children,

);

}

}

class LogoutFeature {

static void showLogoutConfirmationDialog(BuildContext context) {

showDialog(

context: context,

builder: (BuildContext context) {

return AlertDialog(

title: Text("Logout Confirmation"),

content: Text("Are you sure you want to logout?"),

actions: <Widget>[

TextButton(

child: Text("Cancel"),

onPressed: () {

Navigator.of(context).pop();

},

),

TextButton(

child: Text("Logout"),

onPressed: () {

Navigator.of(context).pop();

performLogout(context);

},

),

],

);

},

);

}

static void performLogout(BuildContext context) {

Navigator.pushAndRemoveUntil(

context,

MaterialPageRoute(builder: (context) => const Login()),

(Route<dynamic> route) => false,

);

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(

content: Text("You have been logged out."),

backgroundColor: Colors.blue[700],

),

);

}

static Widget buildLogoutListTile(BuildContext context) {

return ListTile(

leading: Container(

padding: EdgeInsets.all(8),

decoration: BoxDecoration(

color: Colors.blue[800]!.withOpacity(0.1),

borderRadius: BorderRadius.circular(8),

),

child: Icon(

Icons.logout\_outlined,

color: Colors.blue[800],

size: 24,

),

),

title: Text(

"Logout",

style: TextStyle(

color: Colors.grey[800],

fontSize: 16,

fontWeight: FontWeight.w500,

),

),

onTap: () {

showLogoutConfirmationDialog(context);

},

hoverColor: Colors.blue[50],

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(10),

),

);

}

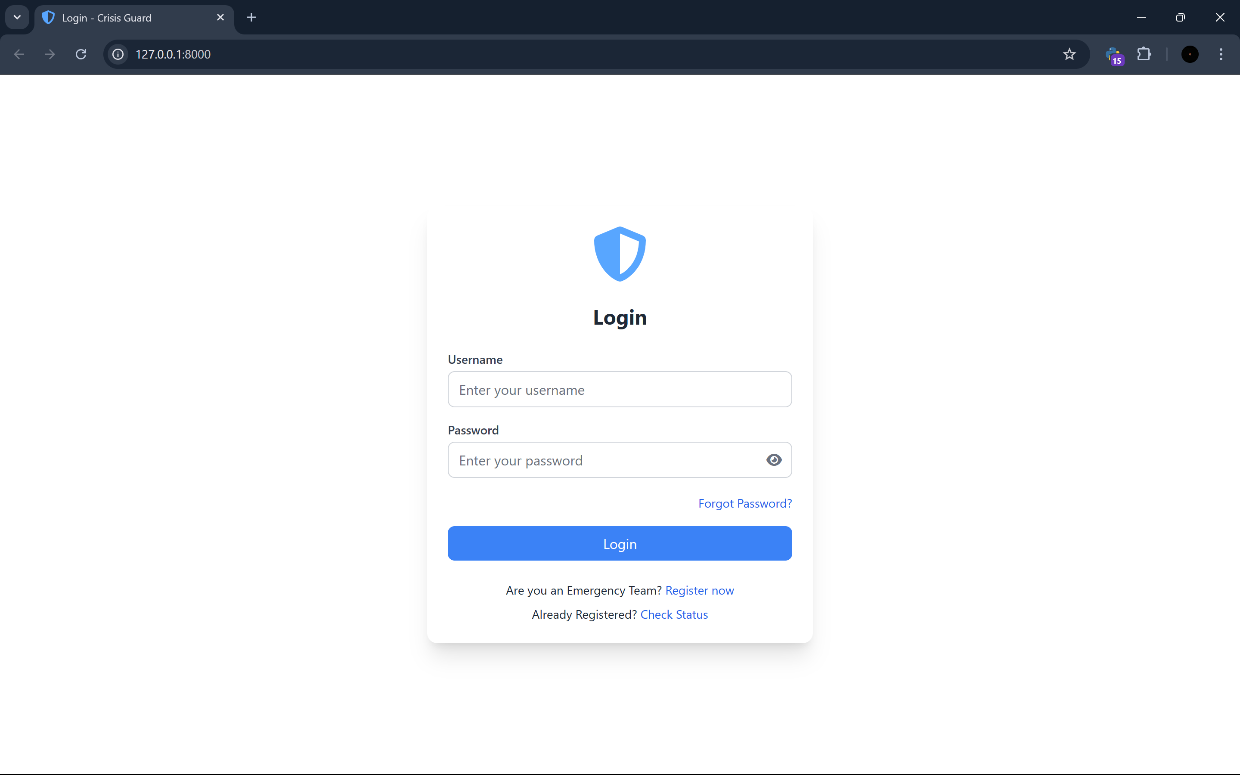
}

VOLUNTEER DRAWER

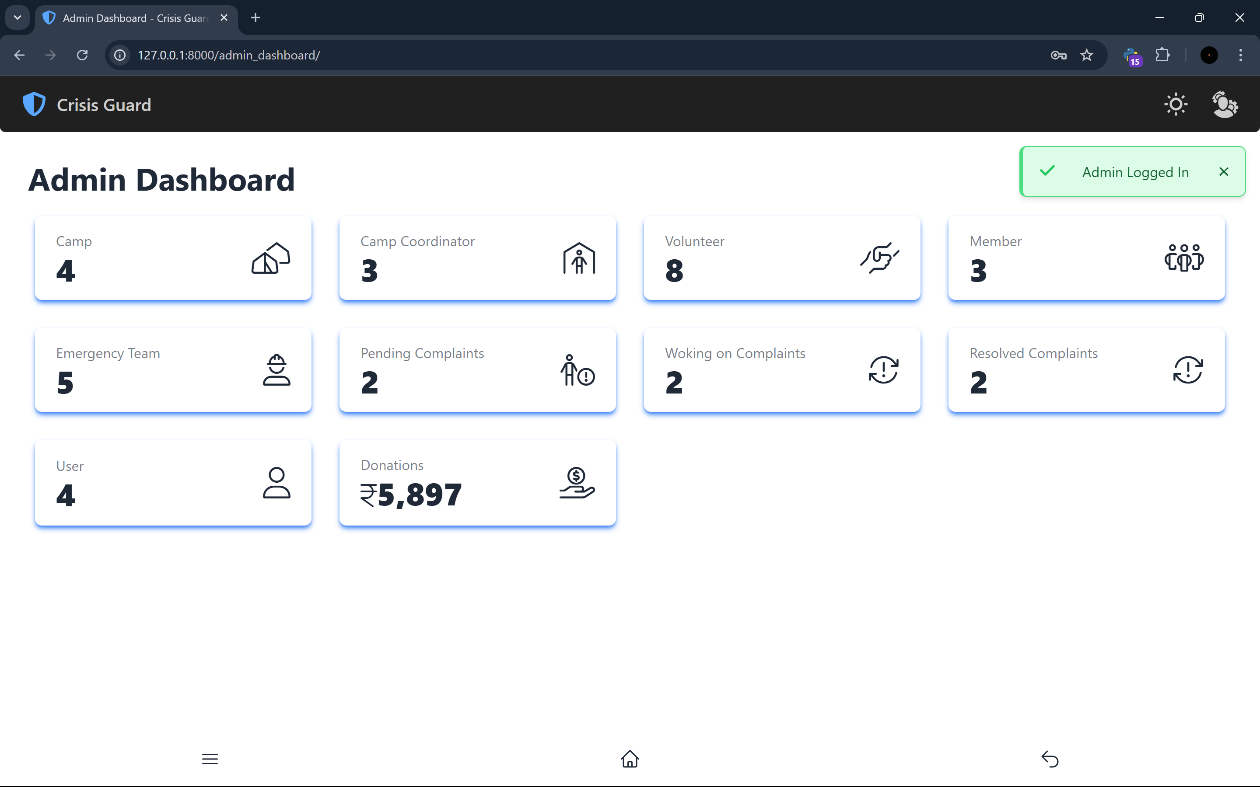
import 'package:crisisguard/Volunteer/ViewNeeds.dart';  
import 'package:crisisguard/Volunteer/Home.dart';  
import 'package:flutter/material.dart';  
import '../Login.dart';  
import '../ViewNotification.dart';  
import 'CompletedPickUp.dart';  
import 'GroupChat.dart';  
import 'ViewGoods.dart';  
import 'Profile.dart';  
import 'ViewMissingAsset.dart';  
import 'ViewPickUp.dart';  
import 'ViewTask.dart';   
class VolunteerDrawerClass extends StatelessWidget {  
 const VolunteerDrawerClass({Key? key}) : super(key: key);  
 @override  
 Widget build(BuildContext context) {  
 return Drawer(  
 width: 280,   
 elevation: 0,   
 child: Container(  
 color: Colors.*white*,   
 child: ListView(  
 padding: EdgeInsets.*zero*,  
 children: <Widget>[  
 DrawerHeader(  
 decoration: BoxDecoration(  
 gradient: LinearGradient(  
 colors: [Colors.*blue*[700]!, Colors.*blue*[500]!],   
 begin: Alignment.*topLeft*,  
 end: Alignment.*bottomRight*,  
 ),  
 ),  
 child: Column(  
 mainAxisAlignment: MainAxisAlignment.center,  
 crossAxisAlignment: CrossAxisAlignment.start,  
 children: [  
 Image.asset(  
 'assets/logo/logo.png',   
 width: 60,   
 height: 60,   
 ),  
 SizedBox(height: 10),  
 Text(  
 "Crisis Guard",  
 style: TextStyle(  
 color: Colors.*white*,  
 fontSize: 24,  
 fontWeight: FontWeight.*bold*,  
 ),  
 ),  
 SizedBox(height: 5),  
 Text(  
 "Your Safety, Our Priority",  
 style: TextStyle(  
 color: Colors.*white*.withOpacity(0.8),  
 fontSize: 14,  
 fontWeight: FontWeight.*w400*,  
 ),  
 ),  
 ],  
 ),  
 ),  
 \_buildListTile(  
 icon: Icons.*home\_outlined*,  
 title: "Home",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => const VolunteerHome()));  
 },  
 ),  
 \_buildListTile(  
 icon: Icons.*assignment\_outlined*,  
 title: "View Needs",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => VolViewNeedsPage()));  
 },  
 ),  
 \_buildExpansionTile(  
 icon: Icons.*volunteer\_activism\_outlined*,   
 title: "Goods",  
 children: [  
 Padding(  
 padding: const EdgeInsets.only(left: 24.0),   
 child: \_buildListTile(  
 icon: Icons.*add\_circle\_outline*,  
 title: "View Goods",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => VolViewGoods()));  
 },  
 ),  
 ),  
 Padding(  
 padding: const EdgeInsets.only(left: 24.0),   
 child: \_buildListTile(  
 icon: Icons.*list\_alt\_outlined*,  
 title: "My Pick Up",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => VolViewPickupPage()));  
 },  
 ),  
 ),  
 Padding(  
 padding: const EdgeInsets.only(left: 24.0),   
 child: \_buildListTile(  
 icon: Icons.*list\_alt\_outlined*,  
 title: "Completed Pick Up",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => VolCompletedPickUpPage()));  
 },  
 ),  
 ),  
 ],  
 ),  
 \_buildListTile(  
 icon: Icons.*task\_alt*,  
 title: "Task",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => VolViewTasksPage()));  
 },  
 ),  
 \_buildListTile(  
 icon: Icons.*web\_asset*,  
 title: "Missing Asset",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => VolViewMissingAsset()));  
 },  
 ),  
 \_buildListTile(  
 icon: Icons.*notifications\_outlined*,  
 title: "Notification",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => ViewNotificationPage()));  
 },  
 ),  
 \_buildListTile(  
 icon: Icons.*chat*,  
 title: "Chat",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => GroupChat()));  
 },  
 ),  
 \_buildListTile(  
 icon: Icons.*person\_outlined*,  
 title: "Profile",  
 onTap: () {  
 Navigator.*push*(context, MaterialPageRoute(builder: (context) => VolViewProfile()));  
 },  
 ),  
 Divider(  
 color: Colors.*grey*[300],   
 thickness: 1,  
 indent: 20,  
 endIndent: 20,  
 ),  
 \_buildListTile(  
 icon: Icons.*logout\_outlined*,  
 title: "Logout",  
 onTap: () {  
 \_showLogoutConfirmationDialog(context);  
 },  
 ),  
 ],  
 ),  
 ),  
 );  
 }  
 Widget \_buildListTile({required IconData icon, required String title, required VoidCallback onTap}) {  
 return ListTile(  
 leading: Container(  
 padding: EdgeInsets.all(8),  
 decoration: BoxDecoration(  
 color: Colors.*blue*[800]!.withOpacity(0.1),   
 borderRadius: BorderRadius.circular(8),  
 ),  
 child: Icon(  
 icon,  
 color: Colors.*blue*[800],   
 size: 24,  
 ),  
 ),  
 title: Text(  
 title,  
 style: TextStyle(  
 color: Colors.*grey*[800],   
 fontSize: 16,  
 fontWeight: FontWeight.*w500*,  
 ),  
 ),  
 onTap: onTap,  
 hoverColor: Colors.*blue*[50],   
 shape: RoundedRectangleBorder(  
 borderRadius: BorderRadius.circular(10),  
 ),  
 );  
 }  
 Widget \_buildExpansionTile({required IconData icon, required String title, required List<Widget> children}) {  
 return ExpansionTile(  
 leading: Container(  
 padding: EdgeInsets.all(8),  
 decoration: BoxDecoration(  
 color: Colors.*blue*[800]!.withOpacity(0.1),   
 borderRadius: BorderRadius.circular(8),  
 ),  
 child: Icon(  
 icon,  
 color: Colors.*blue*[800],   
 size: 24,  
 ),  
 ),  
 title: Text(  
 title,  
 style: TextStyle(  
 color: Colors.*grey*[800],   
 fontSize: 16,  
 fontWeight: FontWeight.*w500*,  
 ),  
 ),  
 children: children,  
 );  
 }  
 void \_showLogoutConfirmationDialog(BuildContext context) {  
 showDialog(  
 context: context,  
 builder: (BuildContext context) {  
 return AlertDialog(  
 title: Text("Logout Confirmation"),  
 content: Text("Are you sure you want to logout?"),  
 actions: <Widget>[  
 TextButton(  
 child: Text("Cancel"),  
 onPressed: () {  
 Navigator.*of*(context).pop();  
 },  
 ),  
 TextButton(  
 child: Text("Logout"),  
 onPressed: () {  
 Navigator.*of*(context).pop();   
 \_performLogout(context);  
 },  
 ),  
 ],  
 );  
 },  
 );  
 }  
 void \_performLogout(BuildContext context) {  
 Navigator.*pushAndRemoveUntil*(  
 context,  
 MaterialPageRoute(builder: (context) => const Login()),  
 (Route<dynamic> route) => false,  
 );  
 ScaffoldMessenger.*of*(context).showSnackBar(  
 SnackBar(  
 content: Text("You have been logged out."),  
 backgroundColor: Colors.*blue*[700],   
 ),  
 );  
 }  
}

SCREENSHORT

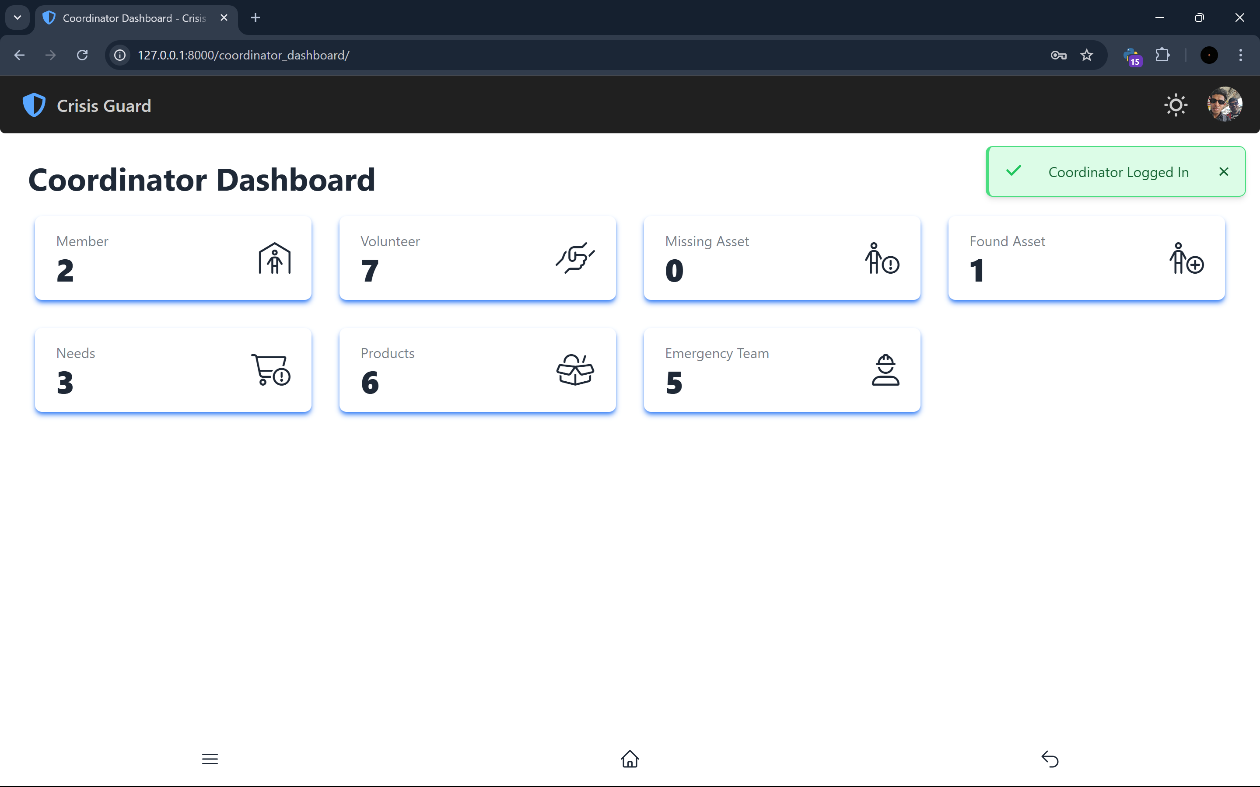
Login Page (Web)

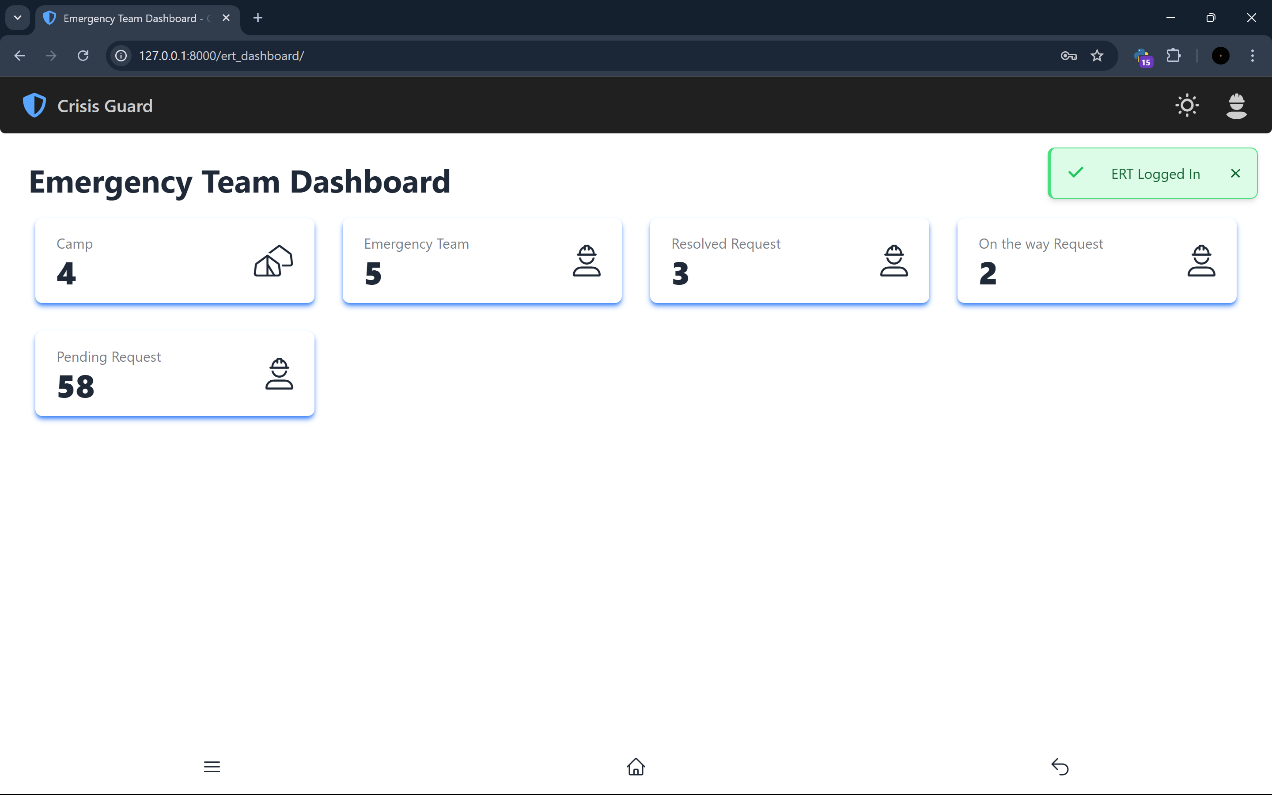


Admin Dashboard(Web)

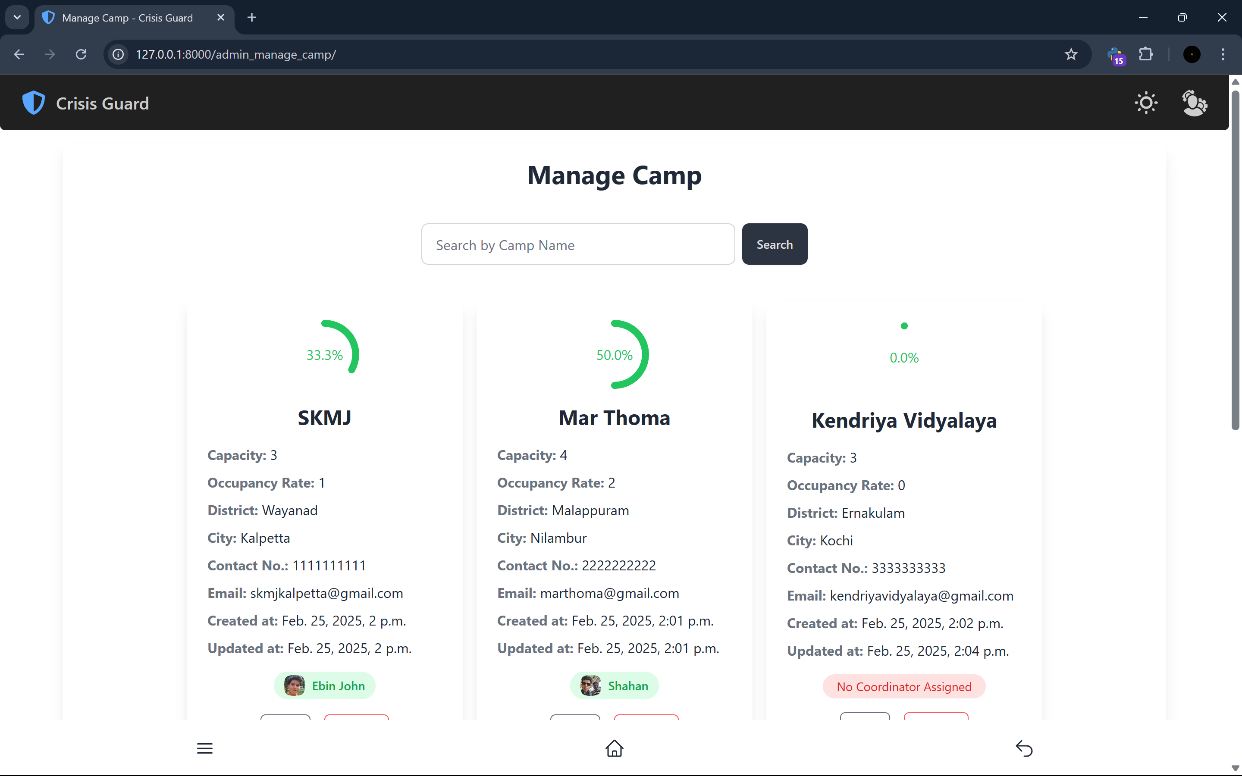


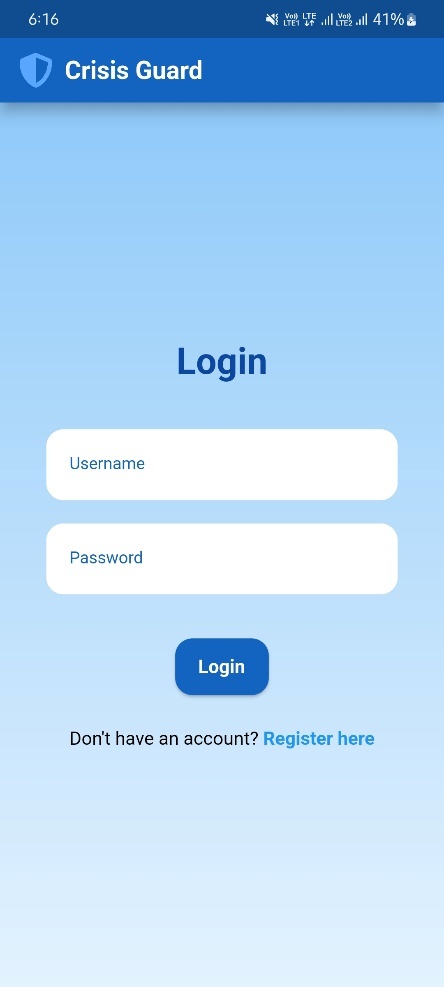
Coordinator dashboard(Web)

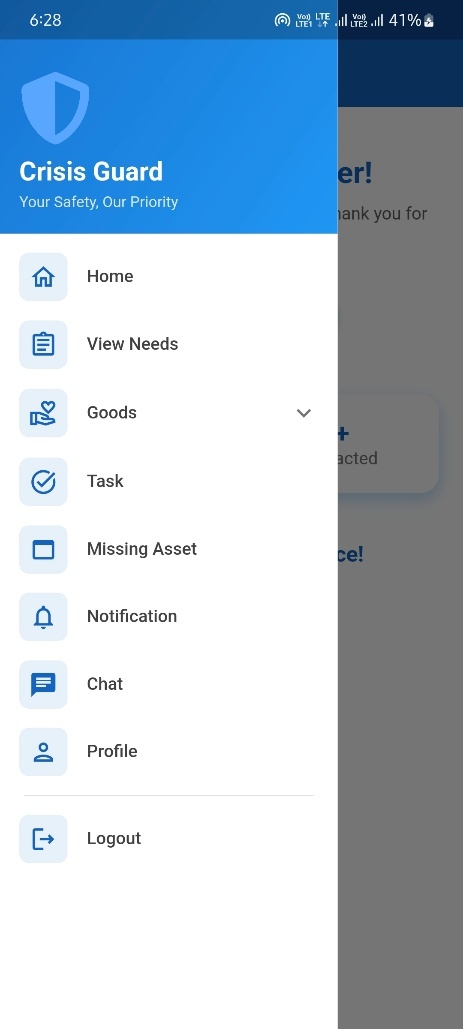


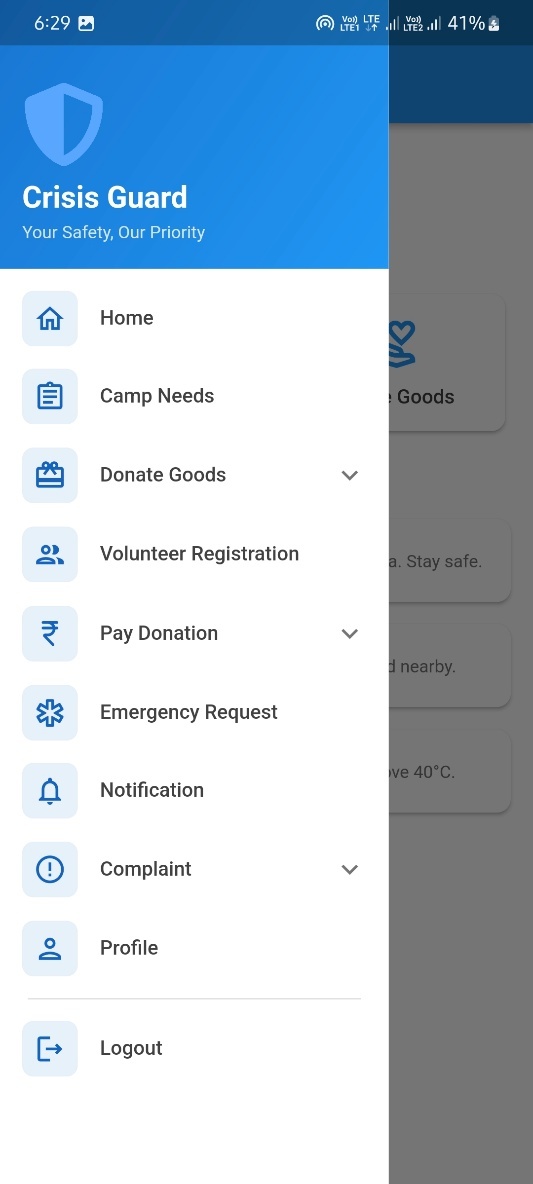
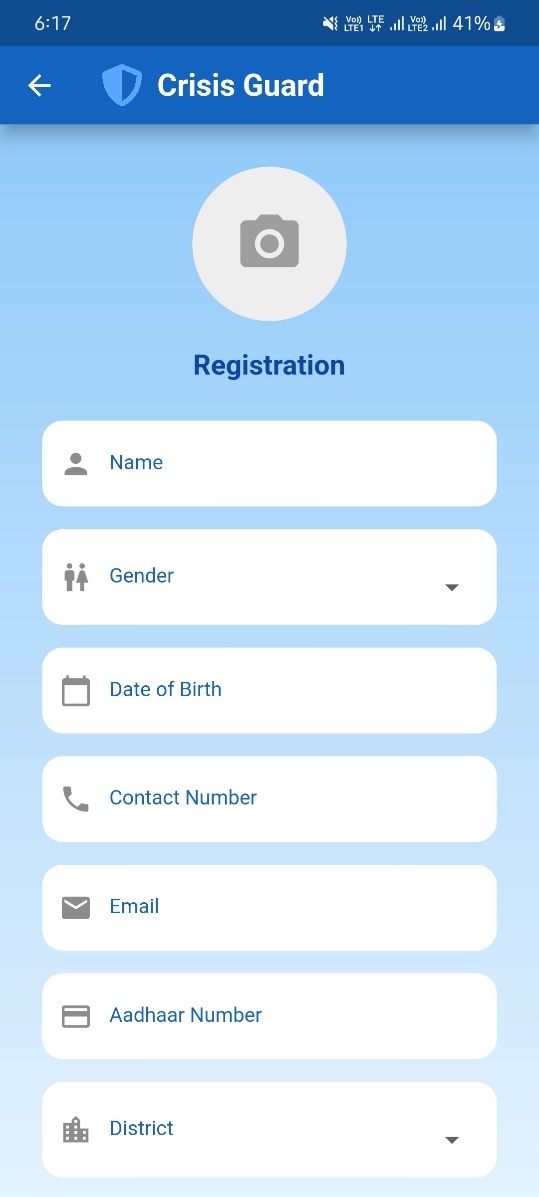
Emergency Team Dashboard(Web)

Manage Camp



Login (Android) Volunteer Drawer (Android)



Public Drawer (Android) Public Registration (Android)

FUTURE SCOPE

The Crisis Guard system is designed to continuously evolve, offering more advanced features and capabilities to enhance disaster management and response. In the future, the system can integrate AI-driven predictive analytics to provide even more accurate early warnings for disasters such as floods, landslides, and extreme weather events. This will enable authorities to take proactive measures, reducing the impact on communities.

Additionally, Crisis Guard can expand its functionality to include real-time audio and video analysis for disaster monitoring. By leveraging steganography techniques, the system can securely transmit critical information, such as emergency alerts or resource requests, through multimedia formats, ensuring data integrity and confidentiality. This added layer of security will be crucial in high-stakes disaster scenarios.

The system can also incorporate blockchain technology to enhance transparency and accountability in resource allocation and donation tracking. This will build trust among users and ensure that resources are distributed efficiently and fairly during emergencies.

Furthermore, Crisis Guard can explore integration with smart city infrastructure, enabling seamless coordination between disaster management systems and other urban services, such as traffic control, emergency services, and public communication networks. This will ensure a more unified and efficient response during disasters.

To improve user adoption, Crisis Guard can focus on user-friendly design and multilingual support, making the system accessible to a wider audience. Gamification strategies, such as rewarding users for reporting hazards or volunteering, can also encourage greater community participation.

Finally, the system can incorporate environmental impact assessment tools to help authorities understand the long-term effects of disasters on ecosystems and communities. This will aid in planning sustainable recovery and rebuilding efforts.

By addressing challenges such as user adoption, regulatory compliance, and balancing security with usability, Crisis Guard can continue to innovate and provide even more secure, efficient, and impactful disaster management solutions. Continuous advancements in this field will ensure that communities are better prepared, more resilient, and capable of facing disasters with confidence.

BIBLIOGRAPHY

[www.djangoproject.com](http://www.djangoproject.com/)

[www.tailwindcss.com](http://www.tailwindcss.com)

[www.stackoverflow.com](http://www.stackoverflow.com)

[www.geeksforgeeks.org](http://www.geeksforgeeks.org)

[www.w3school.in](http://www.w3school.in/)

www.daisyui.com