A Project Report on
“Disaster Management”

Submitted to

Mr. Gopal Krishna

Assistant Professor

Submitted by

Prerna Patwal
2006215
Mukund Kumar
2006200
Prince Raj
2006185
Rajeev Ranjan
2006201
Shivam Kumar
2006204
Alok Mishra
1706114

Subject Name: Software Engineering

Subject Code: CS6402

Session: CSE-2 B2 Sem-6

Department of Computer Science & Engineering National Institute of Technology Patna

April 2023

TABLE OF CONTENTS

SYNOPSIS…. 3

CHAPTER 1 SOFTWARE REQUIREMENTS SPECIFICATION

1.1 INTRODUCTION….5
1.2 ABBREVIATIONS AND DOCUMENT CONVENTIONS....6
1.3 OVERALL DESCRIPTION…. 6
1.4 SPECIFIC REQUIREMENTS.... 8

CHAPTER 2 DESIGN PHASE - UML DIAGRAMS

2.1 USE CASE DIAGRAM(UCD).... 10
2.2 CLASS DIAGRAM…. 13
2.3 DATA FLOW DIAGRAMS.... 15
2.4 ENTITY RELATIONSHIP DIAGRAM….19
2.5 SEQUENCE DIAGRAM….21
2.6 DEPLOYMENT DIAGRAM….23
2.7 COMPONENT DIAGRAM…. 25

CHAPTER 3 CODING PHASE

3.1 INTRODUCTION....27
3.2 CODE SNIPPETS....27
3.3 OUTPUT SCREENSHOTS.... 30

CHAPTER 4 TESTING PHASE

4.1 INTRODUCTION....34
4.2 SELENIUM….37

CHAPTER 5 MAINTENANCE PHASE

5.1 INTRODUCTION....38

CONCLUSION AND FUTURE SCOPE….40

Synopsis

1. Aim

The aim is to help emergency response teams, government agencies, and other organizations better prepare for, respond to, and recover from disasters. Real-time monitoring and alerts, Resource management and Communication and collaboration. improve the efficiency and effectiveness of disaster response efforts, ultimately saving lives and reducing the impact of disasters on affected communities.

2. Objective

The software can provide real-time information on disaster events, facilitate communication between stakeholders, help coordinate resources and personnel, track progress of response and recovery efforts, and enable data analysis for future planning and improvement. The software can also automate certain tasks, allowing emergency responders to focus on critical tasks, and increase overall efficiency and effectiveness in responding to disasters. The ultimate goal of a disaster management software is to reduce the impact of disasters on people, property, and the environment. To provide a personnel and resources allocation management system for the NDRF and be able to take in any extra available resources.

3. Motivation

A disaster management software system can help address these needs by providing a centralized platform for communication, data management, and resource coordination. It can also help automate certain tasks, such as alerting stakeholders, tracking the status of recovery efforts, and analyzing data for future planning.

Overall, the motivation behind creating a disaster management software system is to improve disaster preparedness, response, and recovery efforts, ultimately helping to save lives and minimize the impact of disasters on communities and the environment.

4. Introduction

During the time of disaster, it becomes very difficult for the NDRF to connect with trained NCC/NSS volunteers because of various reasons that they are not involved with community or change of contact numbers etc. There is a requirement to create a system/software where all

these volunteers can be enrolled NDRF is carrying out various awareness programs since its inception which involves the NCC/NSS volunteers. After a certain period, the contact with these volunteers is lost thus making the efforts futile. a) On a click of a Button, all connects can be activated. b) They can also be informed about various programs and periodical material can also be shared with all of them. c) These activities will keep them connected, interested, and involved. d) These resources can be used in any eventuality. These will be registered District wise, thus making them available in every district.

5. Problem statement

During a disaster, it is difficult for the NDRF to connect with NCC/NSS volunteers because of poor situational conditions and lack of organized information.

Generally these volunteers are seen to be less valuable than the trained NDRF response teams, hence using the time to organize them seeming to not be helpful as much as the first.

All the while, each passing second could mean the loss of another life and the risk of many others losing theirs.

Advising people about an upcoming disaster(using EWS) is not conducted fast enough, giving then local population little to no time to avoid or prepare for it accordingly.

6. Conclusion

The disaster management system software is designed to aid in the correlation of weather data, issue warnings and supply volunteers adequately to the disaster affected area. This system will allow the NDRF to form a network of individuals over an encompassed area. It will also calculate the time for publishing warnings and the allocation of resources and rescue personnel. The software can also help in avoiding the overall damage caused by man-made disasters by calculating and suggesting .

Chapter 1

Software Requirement Specification (SRS)

(Disaster Management System)

1. Introduction:

During the time of disaster, it becomes very difficult for the NDRF to connect with trained NCC/NSS volunteers because of various reasons that they are not involved with community or change of contact numbers etc. There is a requirement to create a system/software where all these volunteers can be enrolled NDRF is carrying out various awareness programs since its inception which involves the NCC/NSS volunteers. After a certain period, the contact with these volunteers is lost thus making the efforts futile. a) On a click of a Button, all connects can be activated. b) They can also be informed about various programs and periodical material can also be shared with all of them. c) These activities will keep them connected, interested, and involved. d) These resources can be used in any eventuality. These will be registered District wise, thus making them available in every district.

1.1 Purpose:

The purpose of this project is to examine and assess existing links, state of different initiatives, gaps and identify the needs and recommendations for strengthening for developing an efficient disaster management information network and early warning information systems in the country. With this scope the major aims of the present report are to identify the existing links and gaps in the disaster management information network through a secondary review of the "DMIC Need Assessment Report" (developed earlier in the project).To identify additional existing links and gaps in the disaster management information network at various levels through an analytical presentation of the current status of links from source to destination. To critically analyse the suggested improvements for strengthening the information network at various levels in the future.

1.2 Product Scope:

The system comprises of these major modules with their sub-modules as follows:

- The login page is where the system administrator enters their system credentials to gain access to the administrative side of the system.

- The page where an administrator can add a new earthquake is called the New Earthquake Page.
- **Earthquake List –** A page contains a list of earthquakes that may be manipulated to add, remove, or edit earthquakes.
- **New Tsunami –** This page allows administrators to add new tsunamis.
- **Tsunami List –** This is the page where you can see, edit, or delete the tsunami list.
- **New Volcanic Eruptions –** This is the page where a user can add new volcanic eruptions. The list of volcanic eruptions can be viewed, edited, or removed on the Volcanic Eruptions List website.
- **New User Page –** This is the page where an admin creates a new set of admin credentials.
- **Users list –** This is the page where you can see and manage all your newly added users.

1.3 Intended Audience:

- **Emergency Responders:** These are the first responders on the ground during a disaster, such as police, firefighters, and paramedics. They require real-time information on the disaster, including its location, severity, and the status of any victims. They also need to be able to communicate with other responders and coordinate resources to respond to the disaster.
- **Government Officials:** These are the officials responsible for managing the disaster response and recovery efforts, such as mayors, governors, and other government agencies. They require access to high-level information on the disaster, including the overall response strategy, allocation of resources, and communication with the public.
- **Non-Governmental Organizations (NGOs):** These are organizations that work in disaster response and recovery efforts, such as the Red Cross or other relief agencies. They require information on the disaster and its impact on the community, as well as the ability to coordinate resources and volunteers to assist with the response and recovery efforts.
- **Public:** These are the individuals affected by the disaster, such as residents and business owners. They require information on the disaster, including the severity, location, and any evacuation or shelter-in-place orders. They may also need assistance with finding resources and accessing services during the recovery period.

2. Abbreviations

DMS	Disaster Management System
SRS	Software Requirement Specification
SQL	Structured Programming Language
TU	Tools Used
API	Application Programming Interface
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheet
TOC	Table of content

2.1 Document Conventions

	Font style	Font	Font size
Heading	Bold	Times New Roman	16
Subheading	Bold	Times New Roman	14
Sub-sub-headings	Bold	Times New Roman	13
Content	Normal	Times New Roman	14

3.Overall Description

This allows the government to deal with the disaster situation. During the disaster it will become important for the government to deal with this condition in a good manner and in an effective manner.

3.1.1 System Interface

- Linux
- Windows

3.1.2 Software Interface

Front end: HTML, CSS, Javascript, Flask

- HTML: HTML is used to create the structure of a web page.
- CSS: (Cascading Style Sheets) Create attractive Layout
- Bootstrap: responsive design
- Flask: Backend Purpose

REST-API: MySQL

MySQL: MySQL is a database, widely used for accessing, querying, updating, and managing data in databases.

3.1.3 Hardware Interface:

- No hardware interface is required for the system.

3.2 User Classes and Characteristics.

The major User classes in the System would be:

1. Client

- New client needs to sign up or register giving complete details
- They can update information about the particular disaster .

2. Admin

- The admin has the supreme power of the application
- Admin provides approval to the client registration
- Admin is responsible for maintaining and updating the whole system.
- Admin has the responsibility to notify the client about the disaster condition.

3. Database

- It needs to be updated regarding any new additions.
- It has to be updated regarding any information from the client Trainer.
- Once the registration procedure is complete, the redundant data of the client is to be updated.

3.3 Operating Environment

This web application can be deployed on Linux or a Windows machine with MySQL server.

Minimum RAM 512MB

20GB Storage Space. Intel Dual Core Processor

Design Constraints and implementation constraints

Design constraints are the requirements that need to be met for the successful completion of the project i.e., without any glitches in the software and database.

1. Security:

The files in which the information regarding securities should be secured and should be protected against malicious deformations.

2. Fault Tolerance

Data should not become corrupted in case of system crash or power failure.

Assumptions and dependencies

There is no connection between user database and admin database.

Assuming that all user providing the good information.

Assuming that every company has its credentials and recognition before.

Dependencies:

HTML5, JS ,MYSQL, FLASK ENVIRONMENT

4. Specific Requirements

There are mainly five types of specific requirements included in this SRS, which are as follows:

4.1 Non-Functional Requirements

Aims to provide security, scalability, and make available all users i.e., availability, and make sure that performance of the software is maintained at all times. And also provides usability to the users i.e., the software is made in a simple and user-friendly interface.

4.2 Functional Requirements

1. **User Registration:**

A system/software should allow the NCC/NSS volunteers to register themselves by providing their contact details, location, and other relevant information.

2. District-wise Enrolment:

The system should have the provision to enrol the volunteers district-wise to make them available in every district.

3. Mobile Dissemination:

The system should disseminate information about the prediction of release of water based on rainfall in catchment areas to the affected public through mobile and other mediums.

4. Simulation Models:

The system should have simulation models that can accurately predict the data and provide a more comprehensive understanding of the situation.

4.2 Performance Requirements

The users must get the response within seconds. i.e., the response time of a particular function should be minimum.

Completely separate admin login at server side from the student interface ensures good performance.

The system would exhibit high performance because it would be well optimised. The admin logic would be clearly separate from the User Interface.

4.3 Safety Requirements

- Risk Reduction
- Integrity
- Clear and concise manner

4.4 Security Requirements

- Strong authentication
- Code integrity
- Avoid excess privilege
- Data security
- Code and data validation

- Avoid redundancy
- Provides user requirements

4.5 Software Quality Attributes

- Quality assurance
- Reusability
- Scalability
- Maintainability
- Efficiency

Chapter 2

Design Phase

Unified Modelling Language (UML) is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.

UML diagrams are designed to understand the system in a better and simple way. A single diagram is not enough to cover all the aspects of the system. Hence, UML defines various kinds of diagrams to cover most of the aspects of the system.

UML is not a programming language; it is rather a visual language. We use UML diagrams to portray the behavior and structure of a system. UML helps software engineers, businessmen and system architects with modeling, design and analysis.

UML Diagrams are of two types:

Structural Diagrams: These diagrams capture the static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.

Behavior Diagrams: These diagrams capture the dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

1.1 Use Case Diagram:

A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application. It depicts the high-level functionality of a system and tells how the user handles a system.

1.1.1 Components of use case diagram:

Components of use case diagrams include: Actor: The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data. Actors are stick figures that represent the people actually employing the use cases. System Boundary: A system boundary defines the scope of what a system will be. A system cannot have infinite functionality. So, it follows that use cases also need to have definitive limits defined. A system boundary of a use case

diagram defines the limits of the system. The system boundary is shown as a rectangle spanning all the use cases in the system. Use case: Horizontally shaped ovals that represent the different uses that a user might have. Associations: A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.

1.1.2 Relationships in Use Case

Use cases share different kinds of relationships. A relationship between two use cases is basically a dependency between the two use cases. Use case relationships can be one of the following:

Include: When a use case is depicted as using the functionality of another use case in a diagram, this relationship between the use cases is named as an include relationship.

Extend: In an extend relationship between two use cases, the child use case adds to the existing
functionality and characteristics of the parent use case. The stereotype "<>" identifies the relationship as an extend relationship.

Generalizations: A generalization relationship is also a parent-child relationship between use cases. The child use case in the generalization relationship has the underlying business process meaning but is an enhancement of the parent use case. In a use case diagram, generalization is shown as a directed arrow with a triangle arrowhead. The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.

Use Case diagram of Disaster management System:

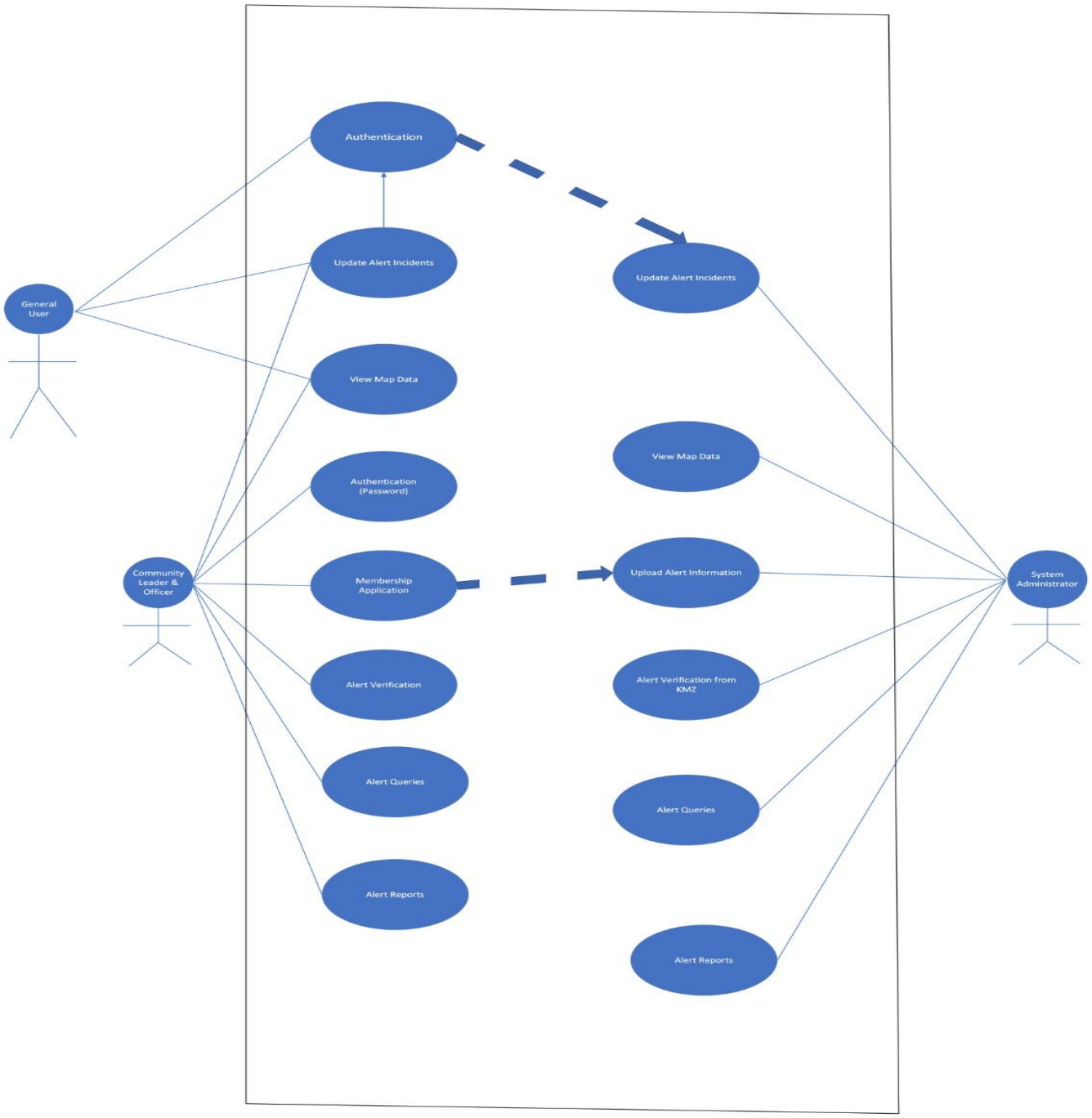


Fig. 01: Use case diagram

1.2 Class Diagram

A class diagram is a visual representation of class objects in a model system, categorized by class types. Each class type is represented as a rectangle with three compartments for the class name, attributes, and operations. Objects are represented as ovals that contain class names inside class name compartments.

The most widely used UML diagram is the class diagram. It is the building block of all object-oriented software systems. We use class diagrams to depict the static structure of a system by showing the system's classes, their methods and attributes. Class diagrams also help us identify relationships between different classes or objects.

Class diagrams are the main building blocks of every object-oriented method. The class diagram can be used to show the classes, relationships, interface, association, and collaboration. UML is standardized in class diagrams. Since classes are the building block of an application that is based on OOPs, the class diagram has an appropriate structure to represent the classes, inheritance, relationships, and everything that OOPs have in their context. It describes various kinds of objects and the static relationship between them. The main purpose to use class diagrams are:

- This is the only UML that can appropriately depict various aspects of the OOPs concept.
- Proper design and analysis of applications can be faster and efficient.
- It is the base for deployment and component diagrams.

Class diagrams are not only used to visualize the static view of the system but are also used to construct the executable code for forward and reverse engineering of any system.

The class diagram clearly shows the mapping with object-oriented languages such as Java, C++, etc. From practical experience, a class diagram is generally used for construction purposes.

UML class diagram of Disaster Management System:

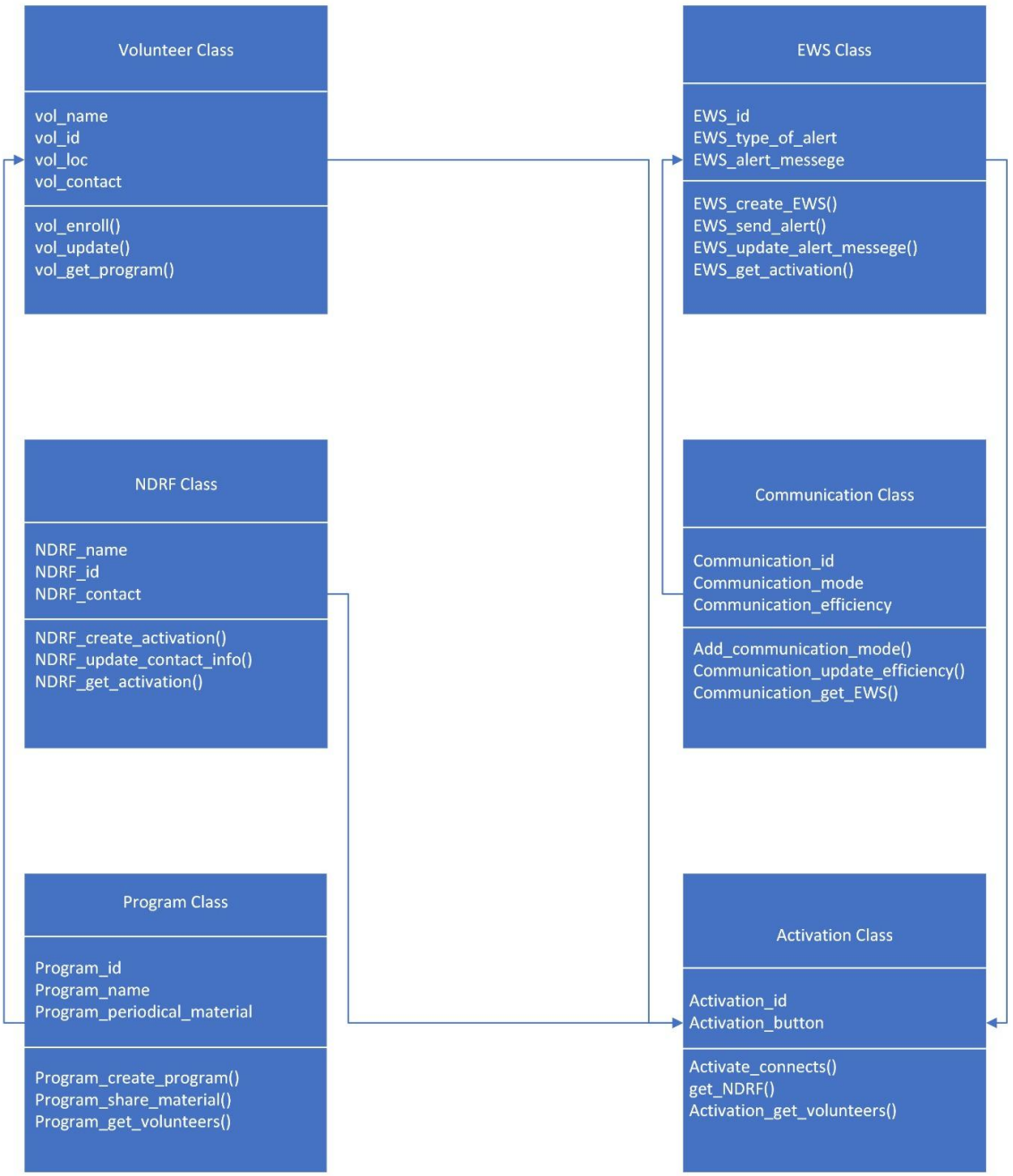


Fig. 02: class diagram

1.3 Data Flow Diagram (DFD)

The flow of data of a system or a process is represented by **data flow diagram (DFD)**. It also gives insight into the inputs and outputs of each entity and the process itself. DFD does not have control flow and no loops or decision rules are present. Specific operations depending on the type of data can be explained by a flowchart.

It is a graphical tool, useful for communicating with users, managers and other personnel. It is useful for analyzing existing as well as proposed systems.

Context level data flow diagram (DFD) is a basic overview of the whole system or process being analysed or modelled. It shows the interaction between a system and other external factors with which the system is designed to interface.

The data flow diagrams have 4 components:

- Process
- Data Flow
- Warehouse
- Terminator

DFD uses hierarchy to maintain transparency thus multi-level DFD's can be created. Levels of DFD are as follows:

- 0-level DFD: It represents the entire system as a single bubble and provides an overall picture of the system.
- 1-level DFD: It represents the main functions of the system and how they interact with each other.
- 2-level DFD: It represents the processes within each function of the system and how they interact with each other.
- 3-level DFD: It represents the data flow within each process and how the data is transformed and stored.

1.3.1 Zero level Data Flow Diagram (0 level DFD)

This is the Zero Level DFD of Disaster management, where we have elaborated the high-level process of Disaster information. It's a basic overview of the workings of the project regarding the disaster management and process being analyzed or modeled. It's designed to be an at-a-glance view of Skills, Registration and Login showing the system as a single high-level process. It should be easily understood by a wide audience, including the end users and the administration users.

High Level Entities and process flow of Disaster Management System:

- Managing all the users
- Managing all the Alerts
- Managing all the Risks
- Managing all the resource allocation
- Tracking management and login management

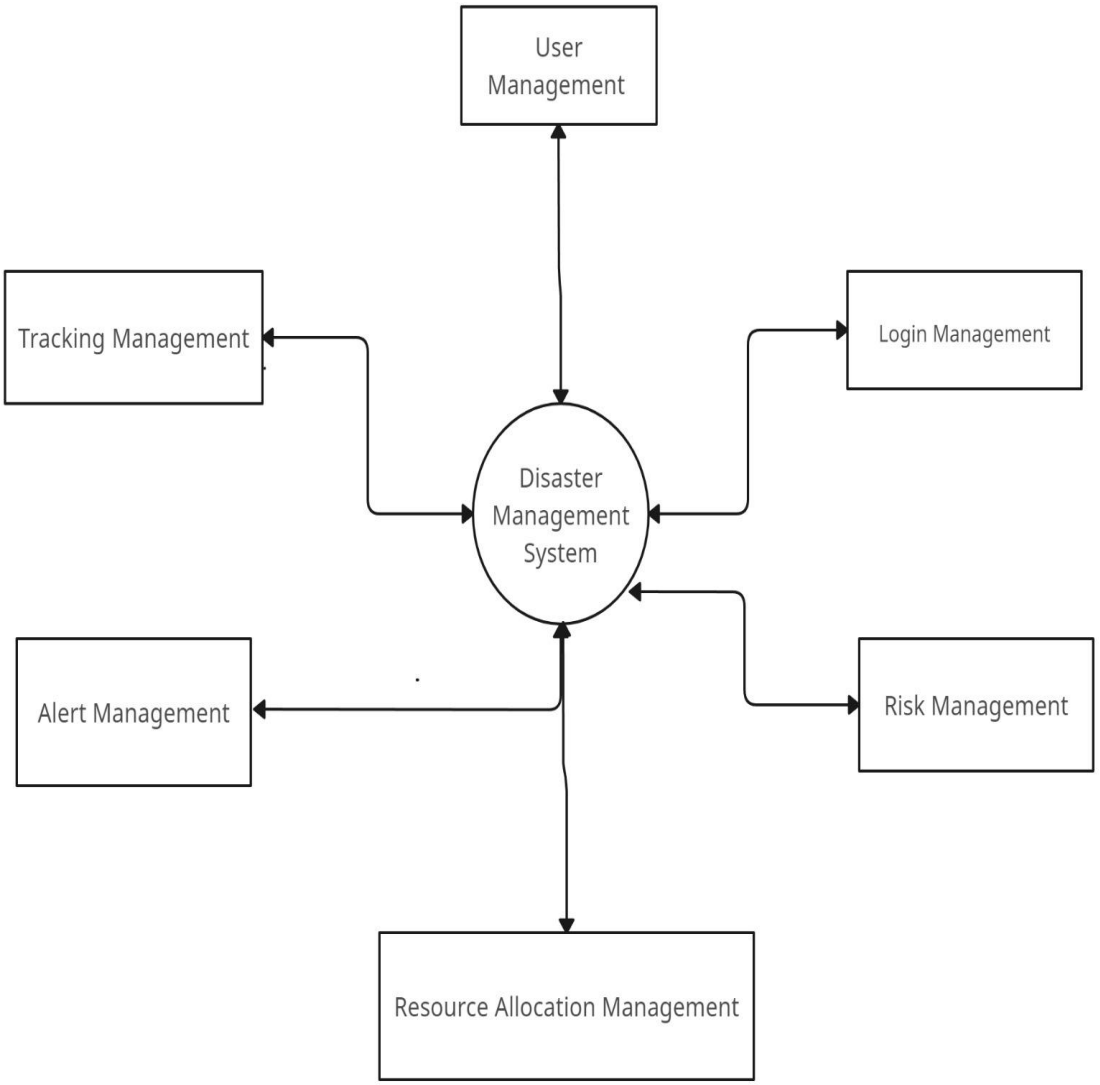


Fig. 03: 0 level DFD

1.3.2 First level Data Flow Diagram (1st level DFD)

First Level DFD (1st Level) of Disaster management shows how the system is divided into subsystems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the Disaster management system as a whole.

DFD Level 1 provides a more detailed breakout of pieces of the 0th level DFD.

Main entities of one level DFD are:

User management, Login management, Risk management, Tracking management, Alert management, Resource management allocation, Generate users report, Generate login report, Generate risk report, Generate tracking details report, Check alert, Check and generate, Resource allocation report.

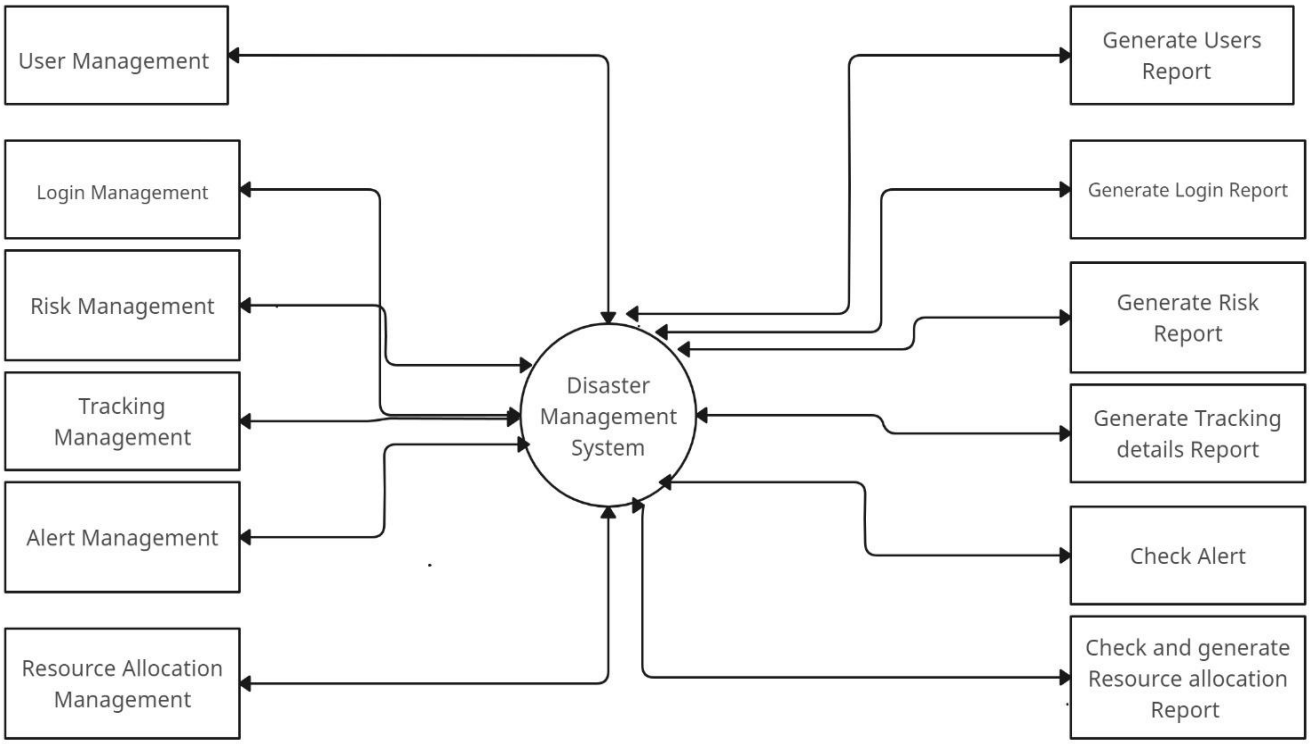


Fig. 04: 1st level DFD

1.3.3 Second level Data Flow Diagram (2nd level DFD)

The Second Level Data Flow Diagram (DFD) delves deeper into the various components of the Disaster Management system, building upon the foundation laid out in Level 1. Achieving a comprehensive understanding of the Disaster Management process may necessitate further refinement of its functionality. The initial level of DFD, or 1st Level, of the Disaster Management System outlines the various subsystems and processes involved. The 2nd Level DFD provides more granular details about admin,people, Govt.staff, Offices,benefits,target, and contact details.

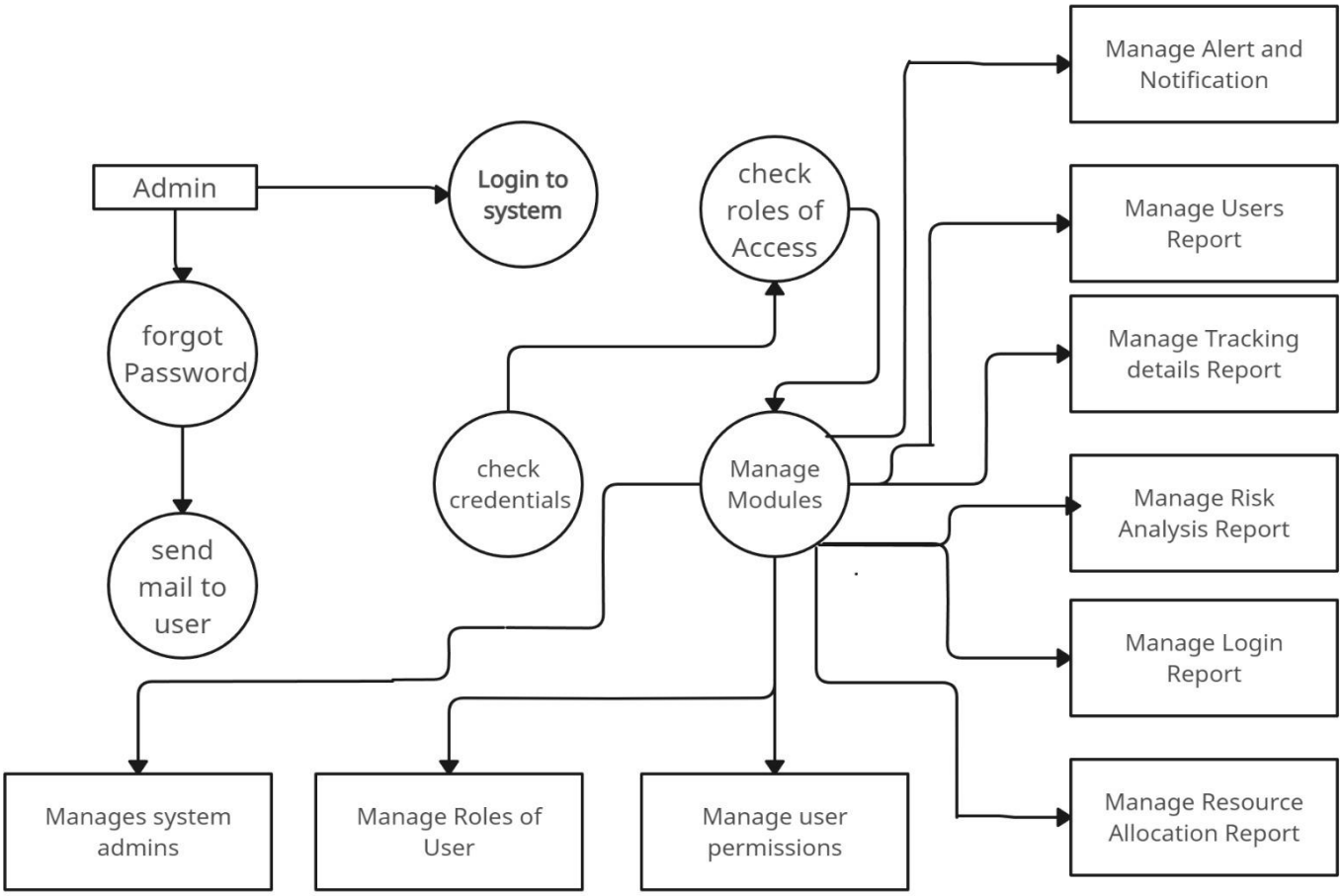


Fig. 05: 2nd level DFD

1.4 ER Diagram:

ER model stands for an Entity-Relationship model. It is a high-level data model. This model is used to define the data elements and relationship for a specified system. It develops a conceptual design for the database. It also develops a very simple and easy to design view of data.

1.4.1 Components of ER diagram:

Entity: An entity may be any object, class, person, or place. In the ER diagram, an entity can be represented as rectangles.

> **Weak Entity:** An entity that depends on another entity called a weak entity. The weak entity does not contain any key attribute of its own. The weak entity is represented by a double rectangle.

Attribute: The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute.

> **Key Attribute:** The key attribute is used to represent the main characteristics of an entity. It represents a primary key. The key attribute is represented by an ellipse with the text underlined.
>
> **Composite Attribute**: An attribute that composed of many other attributes is known as a composite attribute. The composite attribute is represented by an ellipse, and those ellipses are connected with an ellipse.
>
> **Multivalued Attribute:** An attribute can have more than one value. These attributes are known as a multivalued attribute. The double oval is used to represent multivalued attribute.
>
> **Derived Attribute:** An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed ellipse.

Relationship: A relationship is used to describe the relation between entities. Diamond or rhombus is used to represent the relationship.

> **One-to-One Relationship:** When only one instance of an entity is associated with the relationship, then it is known as one-to-one relationship.
>
> **One-to-many relationship:** When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.
>
> **Many-to-one relationship:** When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.
>
> **Many-to-many relationship:** When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to many relationships.

ER diagram of Disaster Management System:

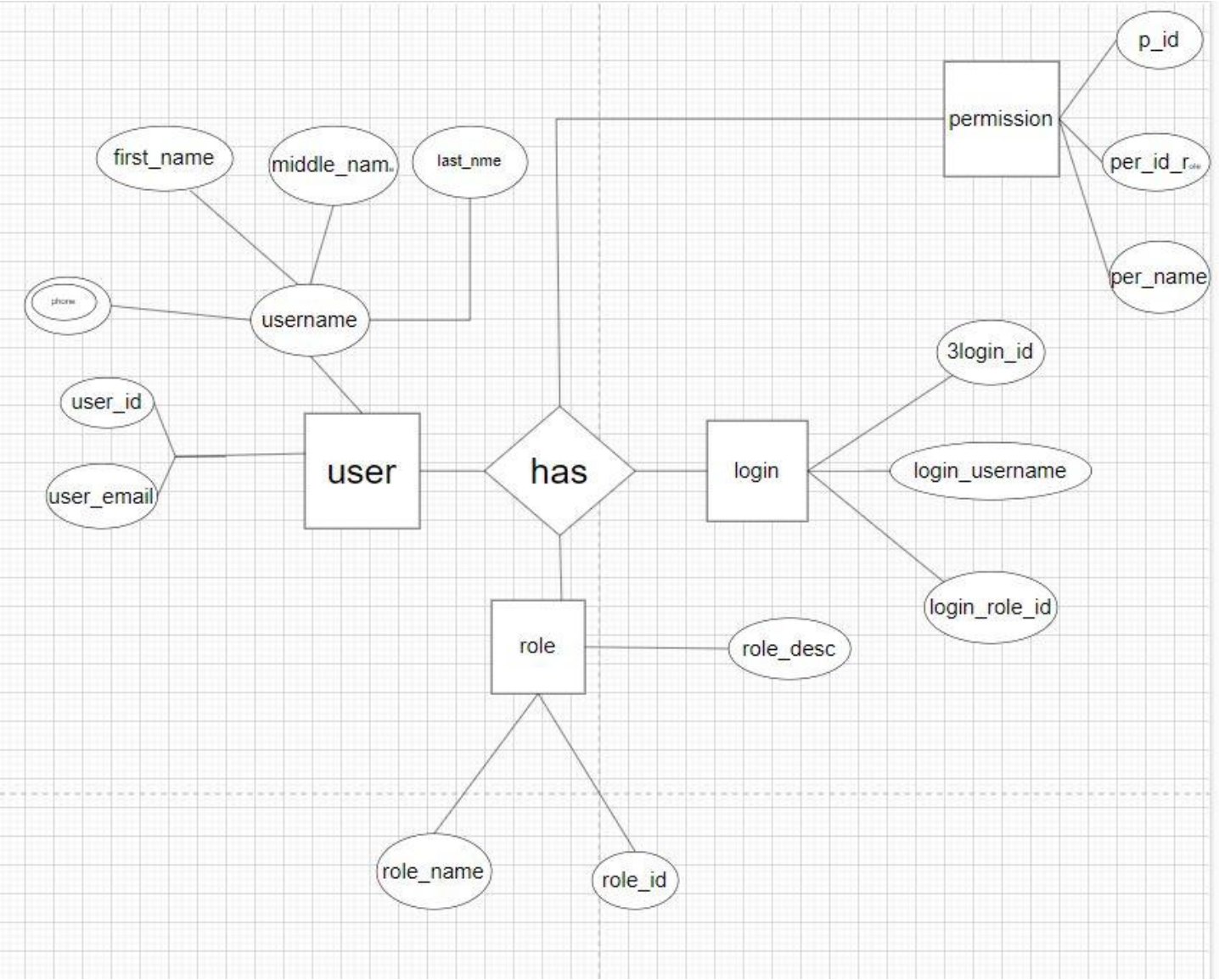


Fig. 06: ER Diagram

1.5 Sequence diagram

UML guides the creation of multiple types of diagrams such as interaction, structure and behaviour diagrams. A sequence diagram is the most used interaction diagram. Interaction diagrams An interaction diagram is used to show the interactive behaviour of a system.

Sequence Diagram Notations: A sequence diagram simply depicts interaction between objects in a sequential order i.e., the order in which these interactions take place.

Actors: An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.

Lifelines: A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically, each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram.

Messages: Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram. Messages can be broadly classified into the following categories:

Synchronous messages: A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed the processing of the message. The caller continues only when it knows that the receiver has processed the previous message i.e. it receives a reply message.

Asynchronous Messages: An asynchronous message does not wait for a reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not. We use a lined arrow head to represent an asynchronous message.

Create message: We use a Create message to instantiate a new object in the sequence diagram. There are situations when a particular message call requires the creation of an object. It is represented with a dotted arrow and create word labelled on it to specify that it is the create Message symbol.

Delete Message: We use a Delete Message to delete an object. When an object is deallocated memory or is destroyed within the system, we use the Delete Message symbol. It destroys the occurrence of the object in the system. It is represented by an arrow terminating with a x.

Self-Message: Certain scenarios might arise where the object needs to send a message to itself. Such messages are called Self Messages and are represented with a U-shaped arrow.

Reply Message: Reply messages are used to show the message being sent from the receiver to the sender. We represent a return/reply message using an open arrowhead

with a dotted line. The interaction moves forward only when a reply message is sent by the receiver.

Guards: To model conditions we use guards in UML. They are used when we need to restrict the flow of messages on the pretext of a condition being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process.

Sequence diagram of Disaster Management System:

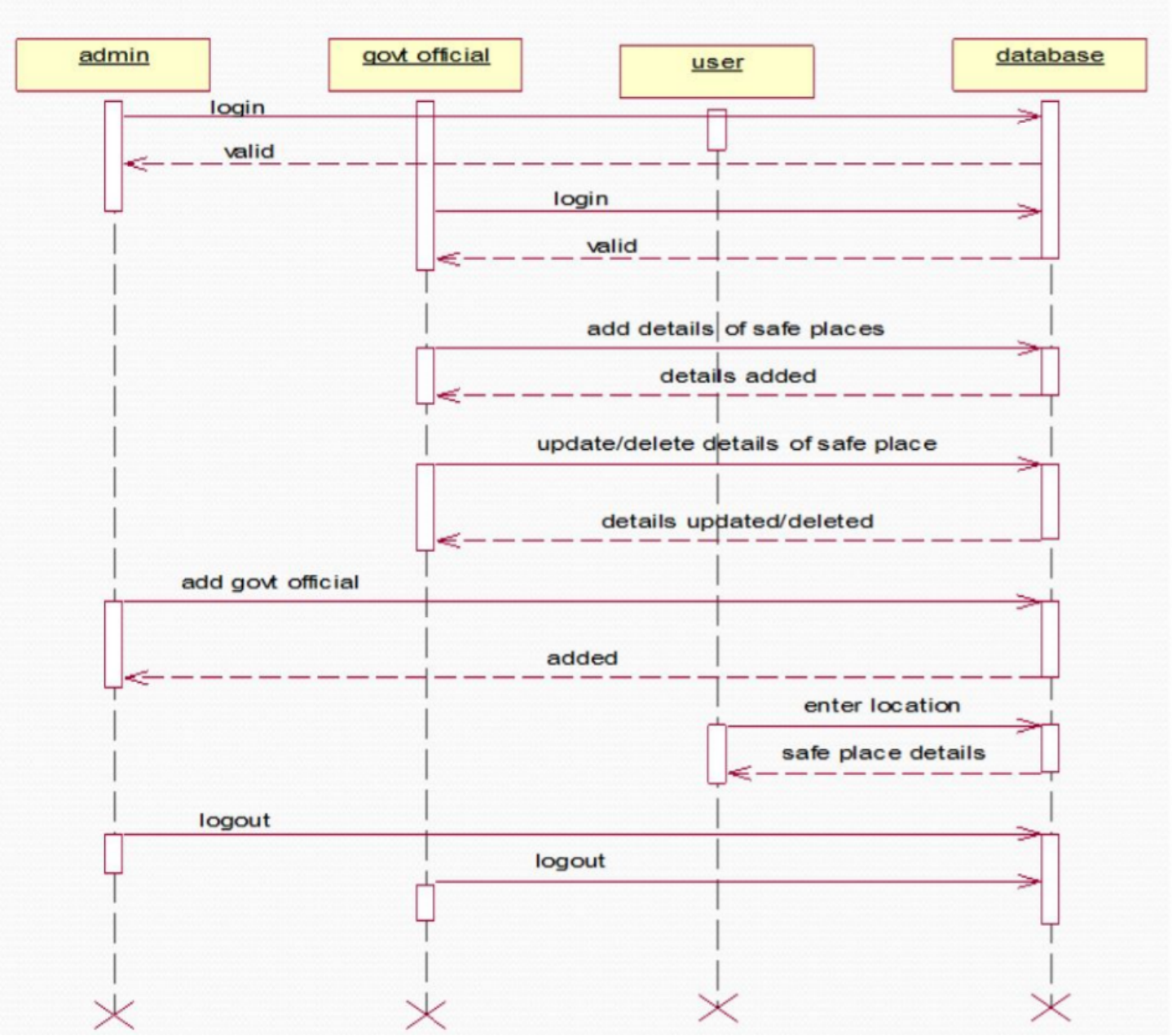


Fig. 06: Sequence Diagram

1.6 Deployment Diagram:

A deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments, and the middleware connecting them. Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware. Deployment diagrams help model the hardware topology of a system compared to other UML diagram types which mostly outline the logical components of a system.

Deployment Diagram Notations

In order to draw a deployment diagram, you need to first become familiar with the following deployment diagram notations and deployment diagram elements.

Node: A node, represented as a cube, is a physical entity that executes one or more components, subsystems or executables. A node could be a hardware or software element.

Artifacts: Artifacts are concrete elements that are caused by a development process. Examples of artifacts are libraries, archives, configuration files, executable files etc.

Communication Association: This is represented by a solid line between two nodes. It shows the path of communication between nodes.

Devices: A device is a node that is used to represent a physical computational resource in a system. An example of a device is an application server.

Deployment specifications: Deployment specifications is a configuration file, such as a text file or an XML document. It describes how an artifact is deployed on a node.

Deployment diagram of Disaster Management System:

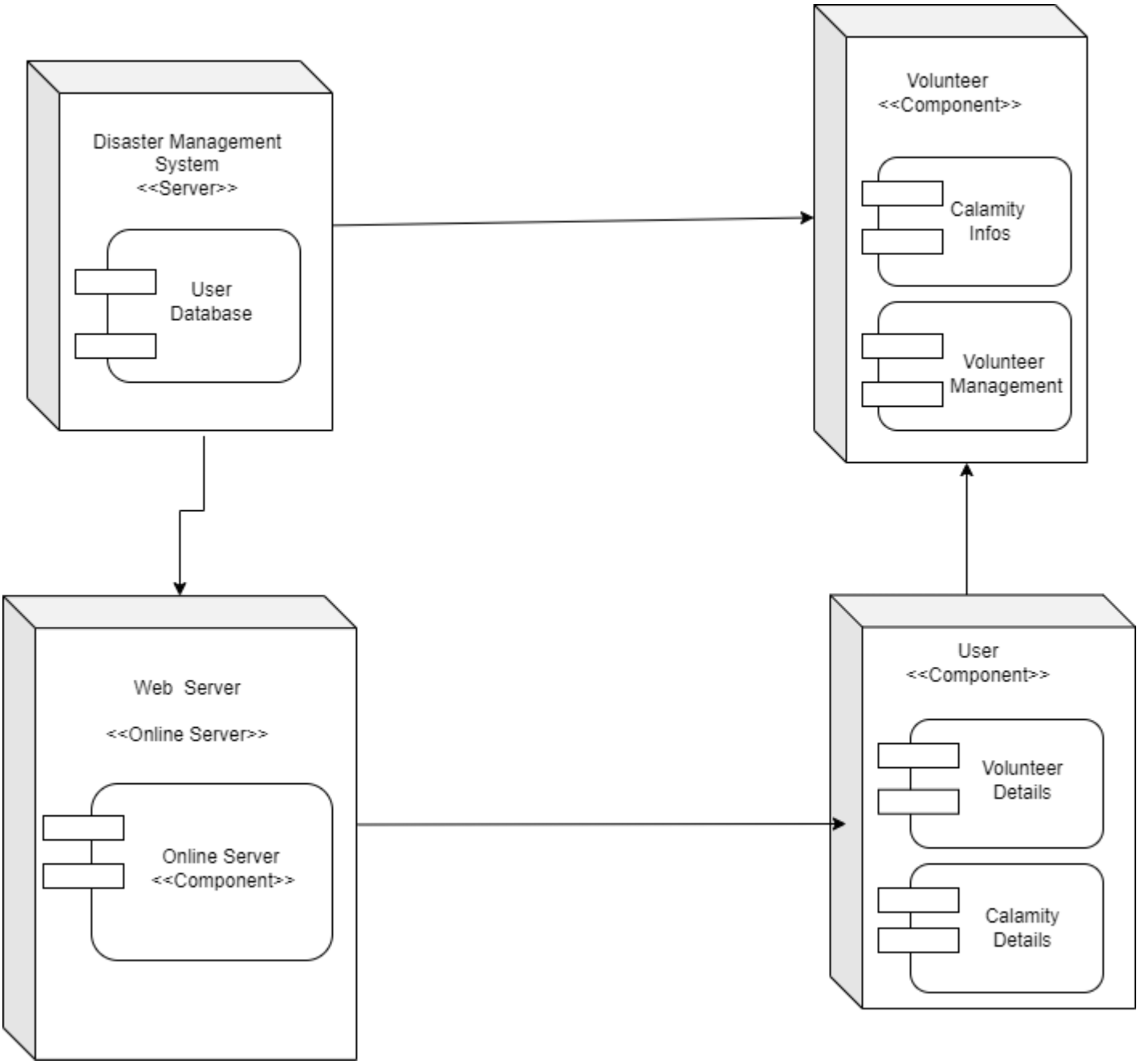


Fig. 07: Deployment Diagram

1.7 component diagram:

Component diagrams are used to visualize the organization of system components and the dependency relationships between them.

Component Diagram Symbols: We have explained below the common component diagram notations that are used to draw a component diagram.

Component

There are three ways the component symbol can be used.

i. Rectangle with the component stereotype (the text <>). The component stereotype is usually used above the component name to avoid confusing the shape with a class icon.
ii. Rectangle with the component icon in the top right corner and the name of the component.
iii. Rectangle with the component icon and the component stereotype.

Required Interface: Interfaces in component diagrams show how components are wired together and interact with each other.

Port: Port (represented by the small square at the end of a required interface or provided interface) is used when the component delegates the interfaces to an internal class.

Dependencies: Although you can show more detail about the relationship between two components using the ball-and-socket notation (provided interface and required interface), you can just as well use a dependency arrow to show the relationship between two components

Component diagram of Disaster Management System:

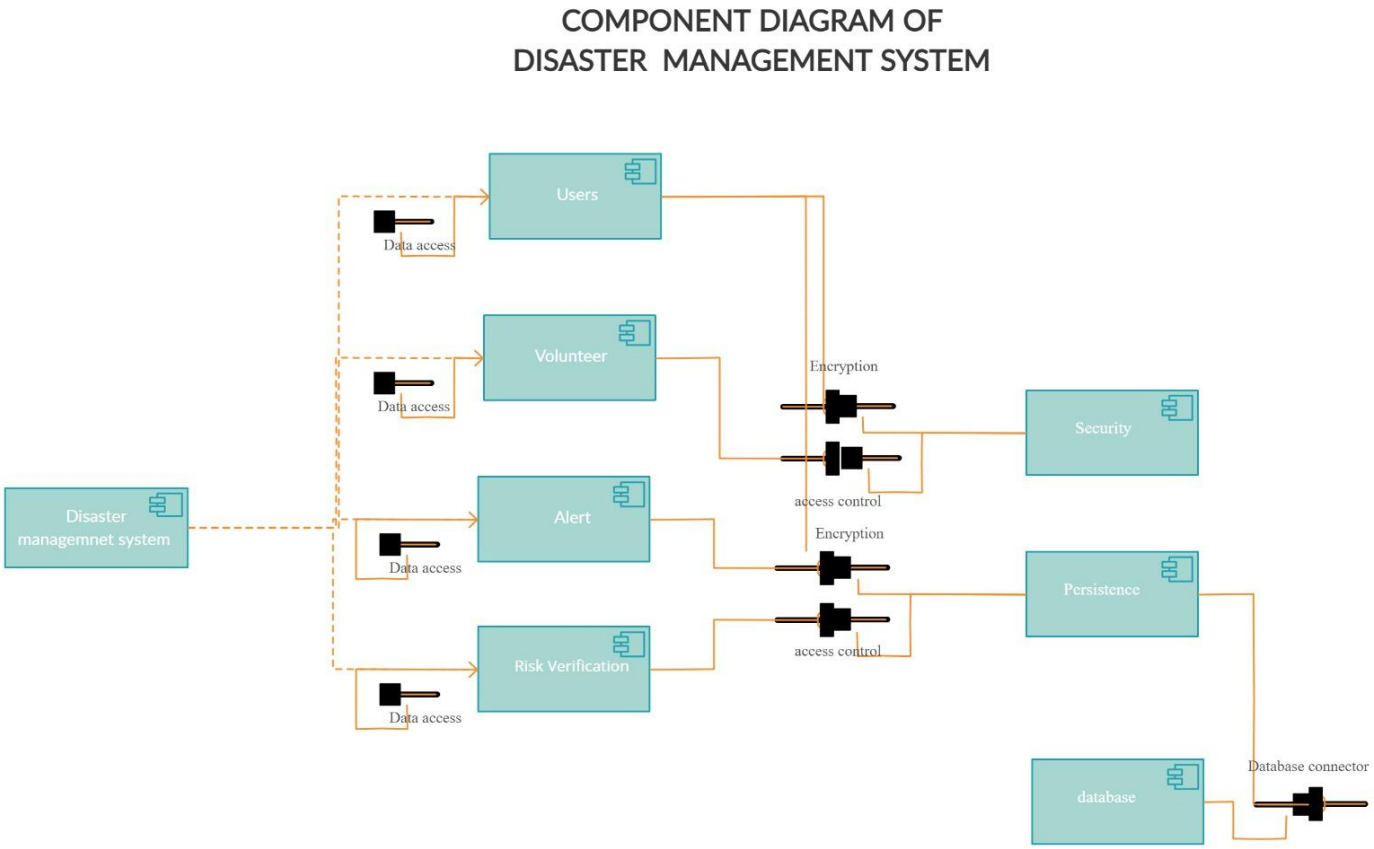


Fig. 08: Component Diagram

Chapter 3

Coding Phase

3.1 Introduction

During the Coding phase the design of the software system is translated into a programming language format which is used by a computer. This involves the implementation of system components
identified in the analysis and design phases. The aim of the coding is to produce a high-quality system which can be performed in any situation.

At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

This is also known as coding or building or developing phase. It is known as the implementation phase at most of Software engineering blogs, while it may be correct from developers' perspective not from the overall process perspective as we will see that the implementation phase has different activities.

During the coding phase, developers analyse the feasibility of each coding language and begin programming according to coding specifications. Without proper coding, the product won't function according to the customer's specifications, and new codes may need to be implemented. Coding phase also includes static code analysis to avoid any type of errors.

3.2 Code Snippets

app.py > ...

```
26
27  @app.route('/login', methods = ['POST', 'GET'])
28  def login():
29      if request.method == 'POST' :
30          email = request.form.get('email')
31          fname = request.form.get('fname')
32          lname = request.form.get('lname')
33          phone = request.form.get('phone')
34          password1 = request.form.get('password1')
35          aadhar = request.form.get('aadhar')
36          pan = request.form.get('pan')
37          address = request.form.get('address')
38
39          cur = db.connection.cursor()
40          cur.execute("insert into adminn values(%s,%s,%s, %s, %s, %s, %s, %s )", (fname, lname, phone, email, aadhar, pan, password1, address))
41          # cur.execute("INSERT INTO USER values (%s, %s, %s, %s, %s)", (uname, fname, lname, email, password1))
42          db.connection.commit()
43
44          return redirect(url_for('home'))
45      return render_template('login.html')
46
47
48  @app.route('/delete/<int:id>')
49  def delete(id):
50      sql_query = "delete from calamity where contact_no = %s"
51      print(id)
```

contact.html M | index.html M | live_maps.html M | showcalamity.html M | services.html M | updatedb.html M | login.html M | app.py M

templates > showcalamity.html > html > body > nav.navbar.navbar-expand-lg.navbar-light.fixed-top.h-color > div.container > div#navbarResponsive.collapse.navbar-collapse > ul.navbar-nav.m

```
    </nav>


<div class="container1" style="position: relative; top:100px">
    <h1>Registered Candidates</h1>

    <table>
        <tr>
            <th>Name</th>
            <th>Address</th>
            <th>Contact Number</th>
            <th>Weather Calamity</th>
            <th>Action</th>
        </tr>


        {% for x in person %}
            <tr >
            <th style="background-color: aliceblue; color:black">{{x[0]}}</th>
            <th style="background-color: aliceblue; color:black">{{x[1]}}</th>
            <th style="background-color: aliceblue; color:black">{{x[2]}}</th>
            <th style="background-color: aliceblue; color:black">{{x[3]}}</th>

            <th style="background-color: aliceblue; color:black"><button type="button"><a href="delete/{{x[2]}}">Solved</a></button>
            </th>

            </tr>
        {% endfor %}
    </table>

</div>
</body>
</html>
```

contact.html M | index.html M | live_maps.html M | showcalamity.html M | services.html M | updatedb.html M | login.html M | app.py M

templates > updatedb.html > html > head > title

```
                <a href="/login" style="position:relative; top:15px"><i class="fas fa-user-plus fa-3x " style="color:rgb(255, 255, 255);"></i><
            </li>

        </ul>
    </div>
</div>
</nav>


<form method="POST" action="{{ url_for('showcalamity') }}" style="position: relative; top:100px">
    <h1 >Calamity Form</h1>
    <label for="name">Name</label>
    <input type="text" id="name" name="name" required>

    <label for="address">Address</label>
    <input type="text" id="address" name="address" required>

    <label for="contact">Contact Number</label>
    <input type="tel" id="contact" name="contact" pattern="[0-9]{10}" required>

    <label for="calamity">Weather Calamity</label>
    <select id="calamity" name="calamity" required>
        <option value="">-- Select a Calamity --</option>
        <option value="Flood">Flood</option>
        <option value="Cyclone">Cyclone</option>
        <option value="Drought">Drought</option>
        <option value="Heat Wave">Heat Wave</option>
    </select>

    <input type="submit" value="Submit">
</form>
</body>
</html>
```

contact.html M | index.html M | live_maps.html M | showcalamity.html M | services.html M | updatedb.html M | login.html M | app.py M

templates > index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="{{ url_for('static', filename='/stylesheet/style.css') }}">
    <link rel="stylesheet" hhref="{{ url_for('static', filename='/stylesheet/style.css') }}">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/css/bootstrap.min.css">
    <link rel = "script" href="{{ url_for('static', filename='static/script/services.html') }}">
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.1/css/all.min.css"rel="stylesheet"/>
    <link href="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/3.6.0/mdb.min.css" rel="stylesheet"/>
    <link href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap"rel="stylesheet"/>
    <script type="text/javascript"src="https://cdnjs.cloudflare.com/ajax/libs/mdb-ui-kit/3.6.0/mdb.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.0/dist/js/bootstrap.bundle.min.js"></script>
    <link rel="stylesheet" href="https://pro.fontawesome.com/releases/v5.10.0/css/all.css" integrity="sha384-AYmEC3Yw5cVb3ZcuHtOA93w35dYTsvhLPVnY
    <title>Emergency-Connect</title>
</head>
<body>

    <nav class="navbar navbar-expand-lg navbar-light fixed-top h-color" style="height:80px;">
        <div class="container">
          <a class="navbar-brand" href="#" style="color:white; font-size:30px;font-weight:bold;font-family:Roboto;position:absolute;left:20px;">E
          <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarResponsive" aria-controls="navbarResponsi
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="collapse navbar-collapse" id="navbarResponsive">
            <ul class="navbar-nav ms-auto">
              <li class="video__icon nav-item">
                  <a href="/live_maps"><div class="circle--outer"></div></a>
                  <a href="/live_maps"><div class="circle--inner"></div></a>
                  <a href="/live_maps"> <p style="font-size: 15px;font-weight:bolder;">LIVE</p></a>
              </li>

              <li class="nav-item active">
                <a class="nav-link" href="/" style="color:white;font-size:20px; padding:20px">Home</a>
              </li>
```

contact.html M | index.html M | live_maps.html M | showcalamity.html M | services.html M | updatedb.html M | login.html M | app.py M

templates > index.html

```
        <div><i class="fas fa-quote-left fa-2x q-color" style="position: relative; left: 215px;top:80px;" ><i class="fas fa-quote-right"style="pos
      <center><div id="example2">Your Little effort can give others a second chance to live life !</div></center>
      </div>
      <br><br><br>
      <div class="sidehoverbar">
        <button type="button" class="article  mx-auto d-block mt-5">Donate Now  <img src="/static/img/coins (1).png" height="40px" data
      </div>
    <div style="font-size: 36px;font-family:Roboto Slab;font-weight:700;color:#BE0A0A;text-shadow: 0px 4px 4px rgba(0, 0, 0, 0.25);position:absol

    <div class="h-div">
      <button type="button" class="btn btn-success btn-rounded" style="font-size: 30px;width:250px;height:70px;background-color:#1F672B;border-ra
    </div>
    <br><br><br><br><br><br><br><br><br>

   <!-- Footer -->
 <footer class=" text-center text-white f-color ">
   <!-- Grid container -->
   <div class="container p-4">
     <!-- Section: Social media -->
     <section class="mb-4" id="pad" >
       <!-- Facebook -->
       <button type="button" class="btn btn-primary btn-lg btn-floating" ><i class="fab fa-facebook-f" ></i> </button>
       <!-- Twitter -->
       <button type="button" class="btn btn-primary btn-lg btn-floating" style="background-color: #55acee;" ><i class="fab fa-twitter"></i></butto
       <!-- Google -->
       <button type="button" class="btn btn-primary btn-lg btn-floating"  style="background-color: #dd4b39;"><i class="fab fa-google"></i></button
       <!-- Instagram -->
       <button type="button" class="btn btn-primary btn-lg btn-floating"  style="background-color: #ac2bac;"><i class="fab fa-instagram"></i></but
       <!--WhatsApp-->
       <button type="button" class="btn btn-primary btn-lg btn-floating" style="background-color: #25d366;" ><i class="fab fa-whatsapp"></i></butt
     </section>
     <!-- Section: Social media -->

     <!-- Section: Form -->
     <section class="">
       <form action="">
         <!--Grid row-->
```

contact.html M | index.html M | live_maps.html M | showcalamity.html M | services.html M | updatedb.html M | login.html M | app.py M

templates > index.html

```
<header>

<div id="carouselExampleCaptions" class="carousel slide" data-bs-ride="carousel">
  <div class="carousel-indicators">
    <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="0" class="active" aria-current="true" aria-label="S
    <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="1" aria-label="Slide 2"></button>
    <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="2" aria-label="Slide 3"></button>
  </div>
  <div class="carousel-inner">
    <div class="carousel-item active" style="background-image: url('https://media12.s-nbcnews.com/i/MSNBC/Components/Video/202103/Australi
      <div class="carousel-caption">
        <h5>Amphan Cyclone</h5>
        <p><b>Super Cyclone Amphan Takes 72 lives, Damaged Around 5500 House in West Bengal</p></b>
      </div>
    </div>
    <div class="carousel-item" style="background-image: url('https://media12.s-nbcnews.com/i/MSNBC/Components/Video/202103/Australia-floo
      <div class="carousel-caption">
        <h5>Second slide label</h5>
        <p>Some representative placeholder content for the second slide.</p>
      </div>
    </div>
    <div class="carousel-item" style="background-image: url('https://media12.s-nbcnews.com/i/MSNBC/Components/Video/202103/Australia-floo
      <div class="carousel-caption">
        <h5>Third slide label</h5>
        <p>Some representative placeholder content for the third slide.</p>
      </div>
    </div>
  </div>
  <button class="carousel-control-prev" type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide="prev">
    <span class="carousel-control-prev-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Previous</span>
  </button>
  <button class="carousel-control-next" type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide="next">
    <span class="carousel-control-next-icon" aria-hidden="true"></span>
    <span class="visually-hidden">Next</span>
  </button>
</div>
```

app.py > ...

```
@app.route('/updatedb', methods = ['POST', 'GET'])
def updatedb():
    if request.method =='POST':
        name = request.form['name']
        address = request.form['address']
        contact = request.form['contact']
        calamity = request.form['calamity']
        sql_query = "INSERT INTO calamity (name, address, contact_no, calamity) VALUES (%s, %s, %s, %s)"
        values = (name, address, contact, calamity)
        cur = db.connection.cursor()
        cur.execute(sql_query, values)
        db.connection.commit()
        # sendmail()
        return render_template('updatedb.html')
    return render_template('updatedb.html')


def fetchdetails():
    cur = db.connection.cursor()
    abc = "select * from calamity"
    rr = cur.execute(abc)
    detail = cur.fetchall()
    cur.close()
    # print(detail)
    return detail
```

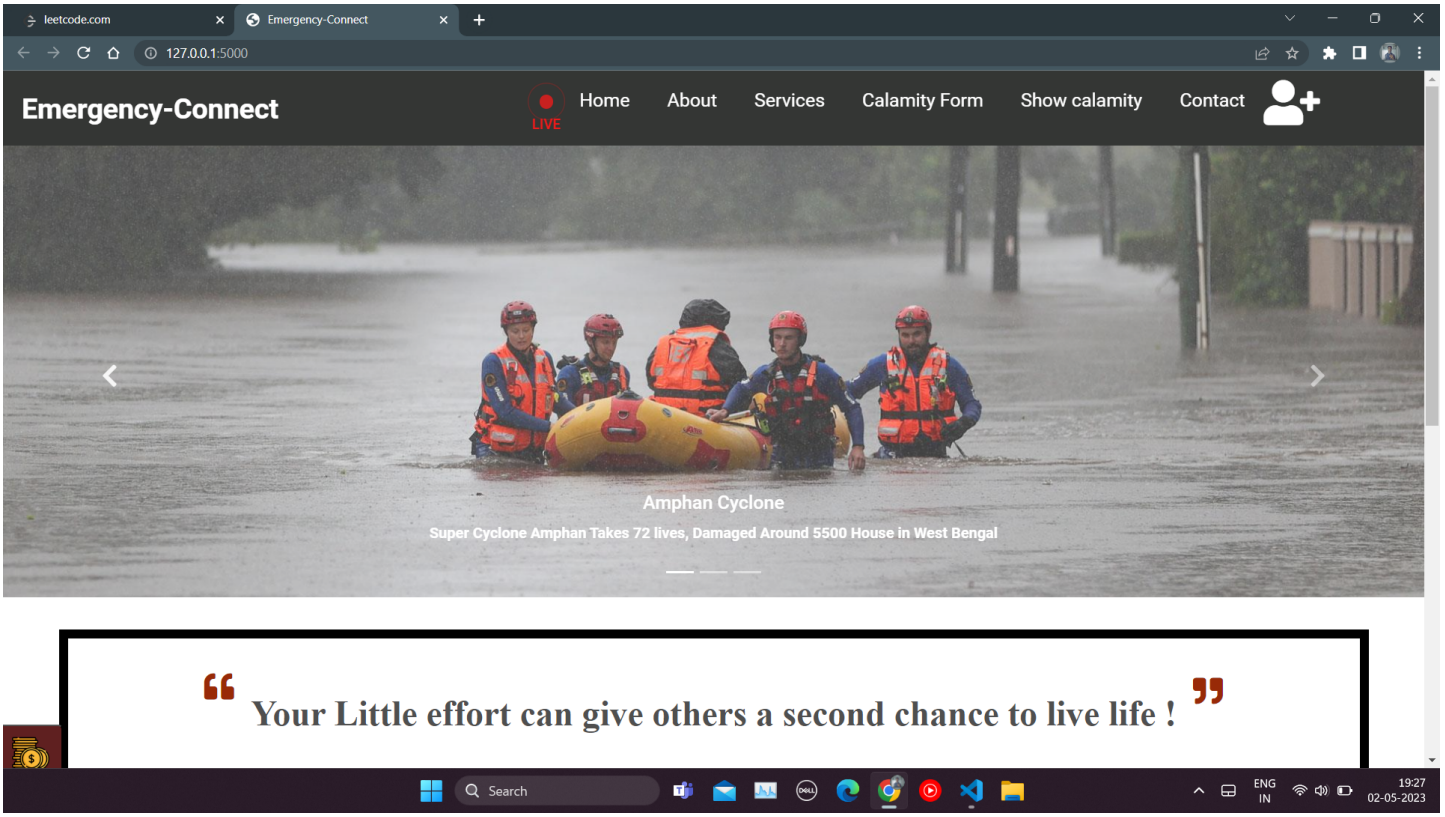
PROBLEMS OUTPUT DEBUG CONSOLE COMMENTS OUTLINE OPEN EDITORS TERMINAL

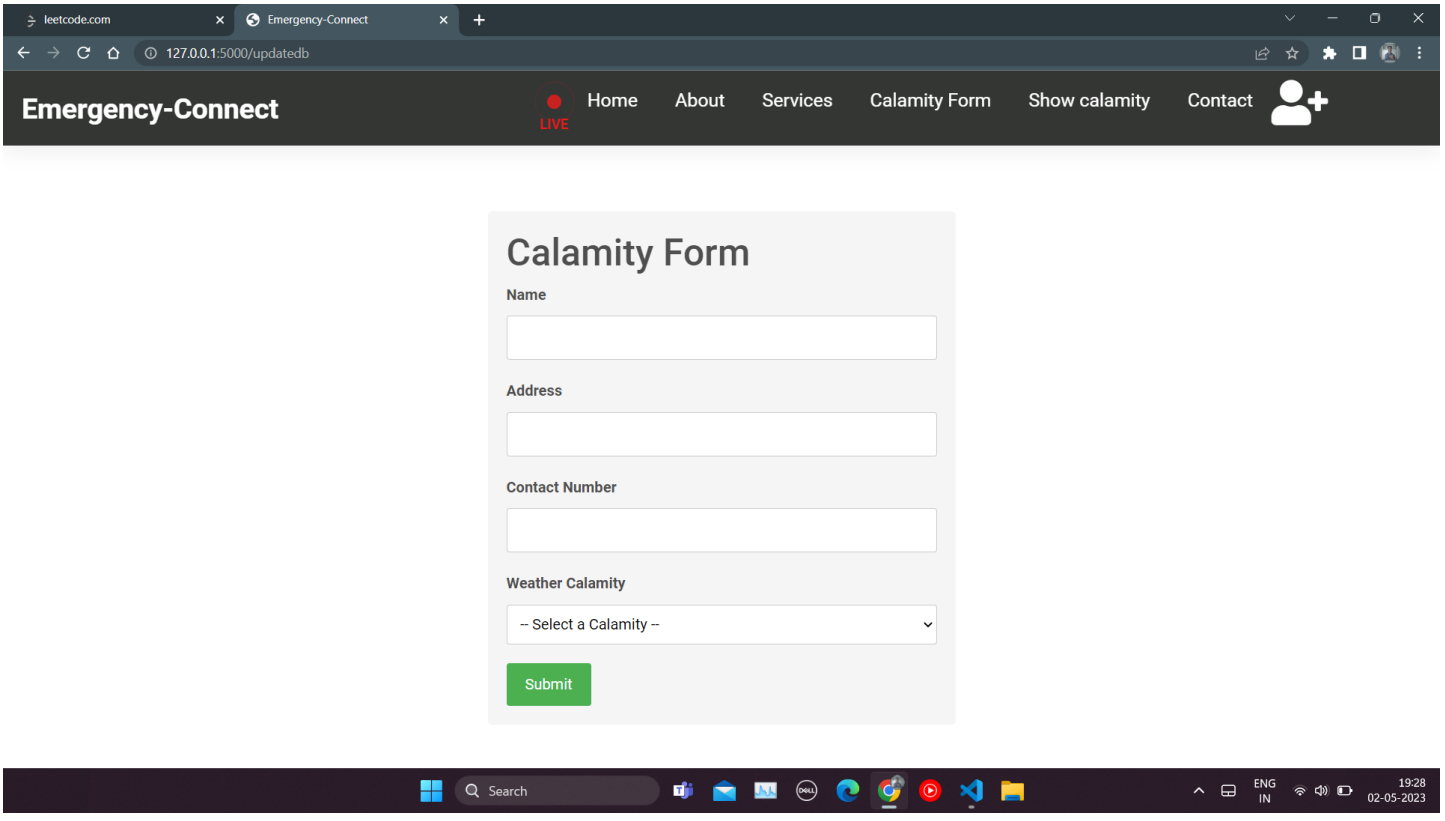
Code

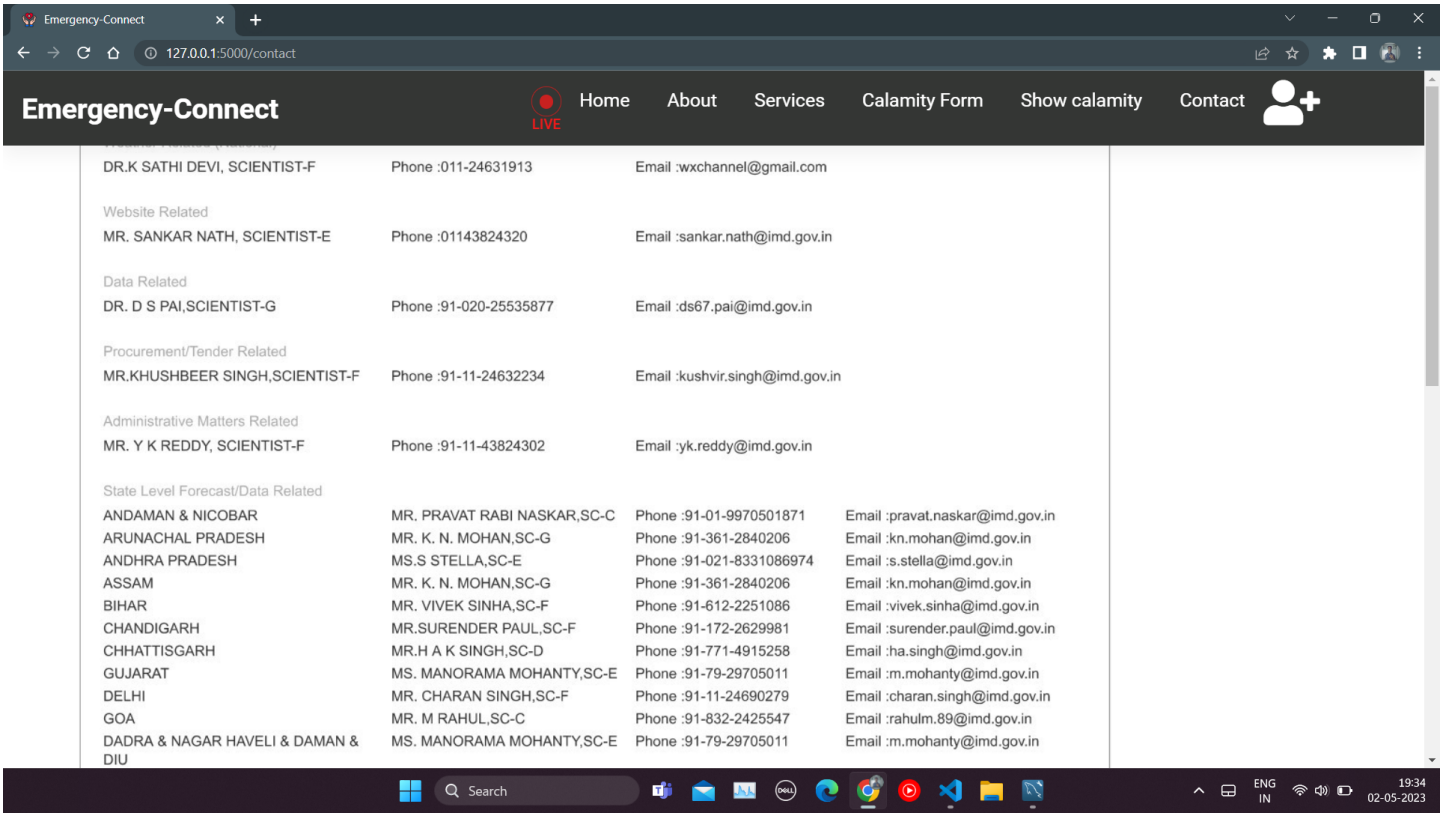
app.py > ...

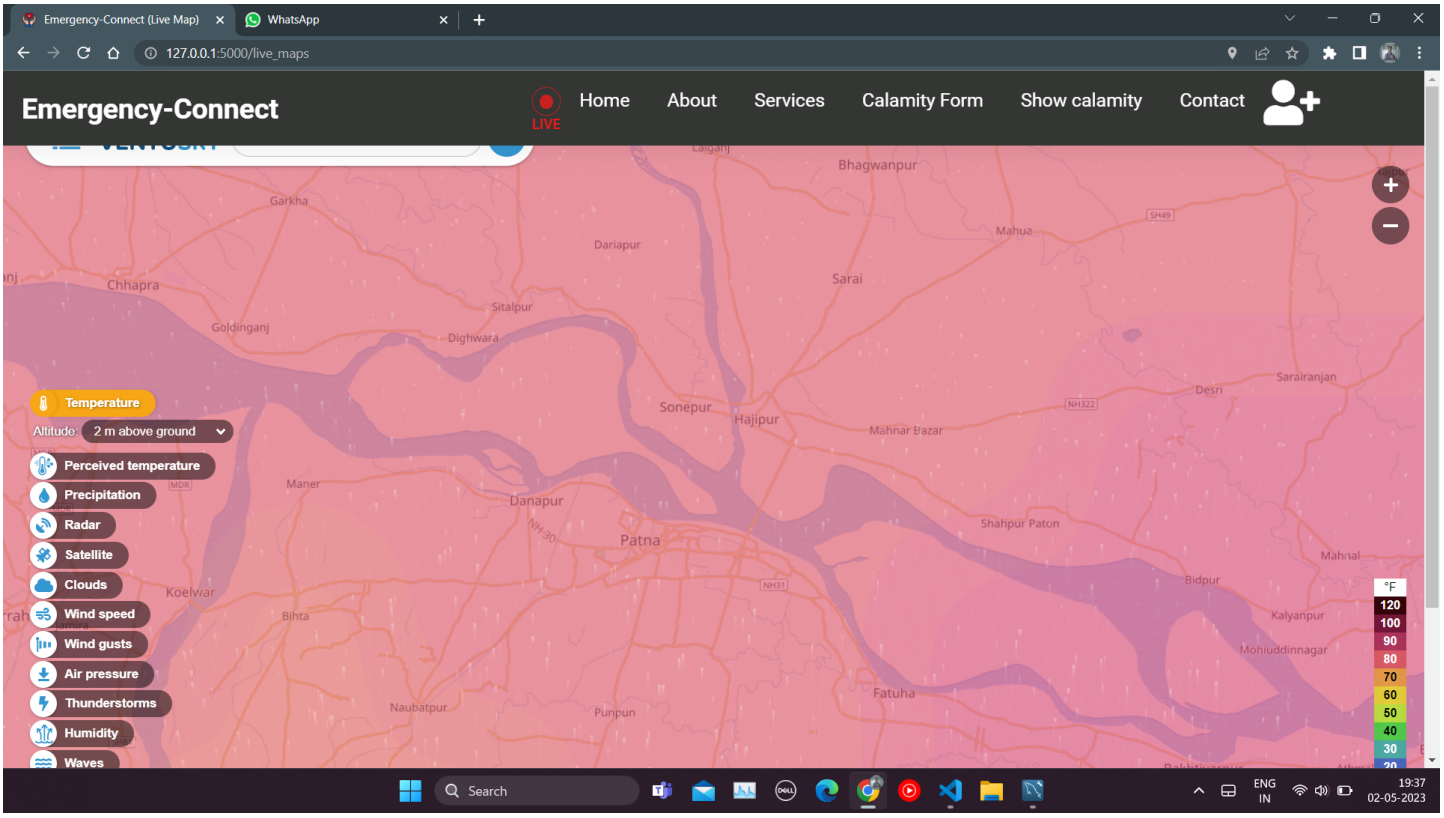
```
99
100
101 def fetchdetails():
102     cur = db.connection.cursor()
103     abc = "select * from calamity"
104     rr = cur.execute(abc)
105     detail = cur.fetchall()
106     cur.close()
107     # print(detail)
108     return detail
109
110 def sendmail(name, contactno, address, calamity):
111     import smtplib
112     server = smtplib.SMTP('smtp.googlemail.com', 587)
113     server.starttls()
114     server.login('rajeevranjanjop@gmail.com', 'FirstJobapply@17')
115     body = name+" "+contactno+" " +address+" "+calamity+"."
116     server.sendmail('rajeevranjanjop@gmail.com', 'mukundwh8@gmail.com', body)
117     print('sent')
118
119
120 if __name__ == '__main__':
121     app.run(debug=True)
122
```

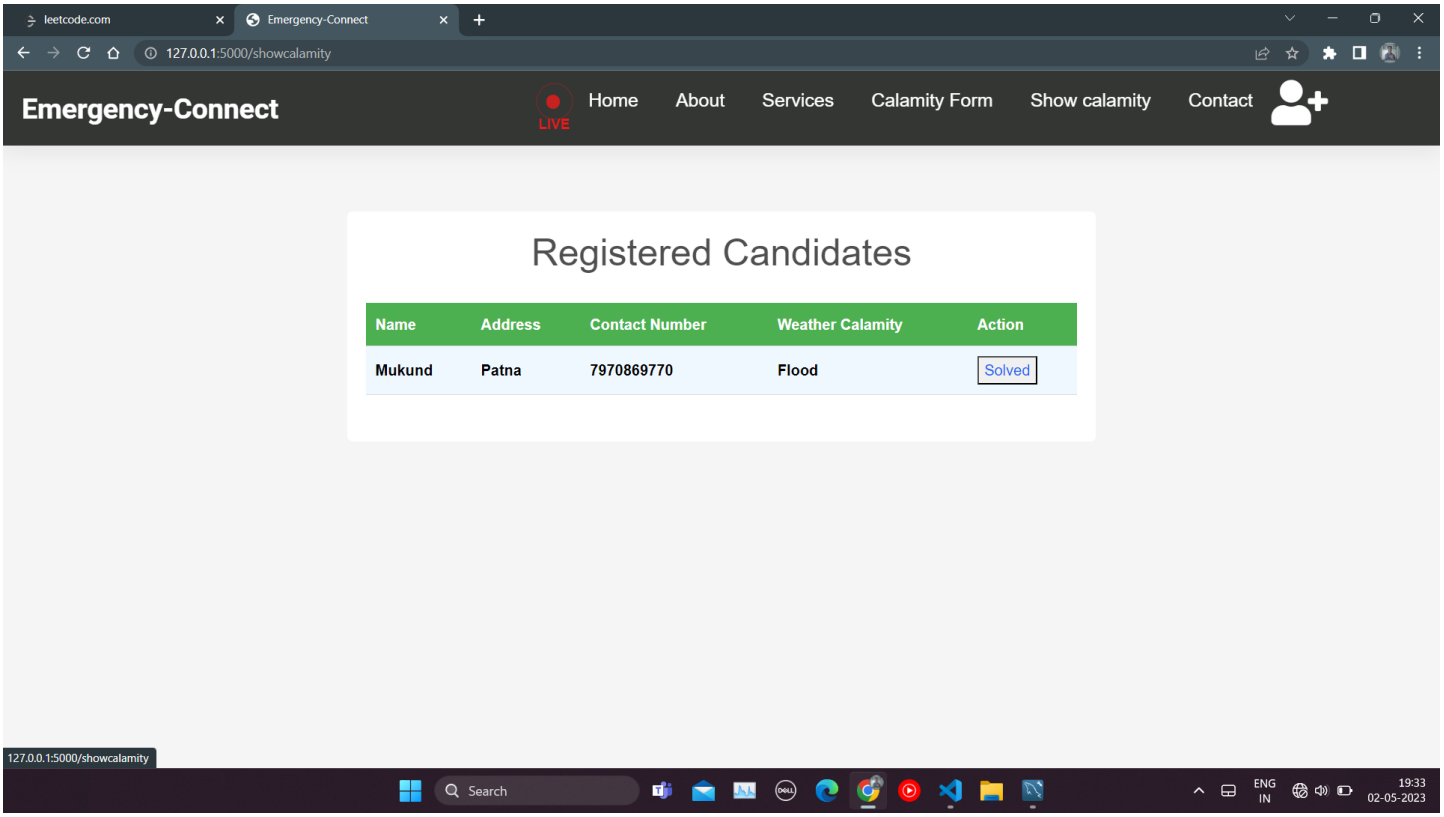
3.3 Output Screenshots

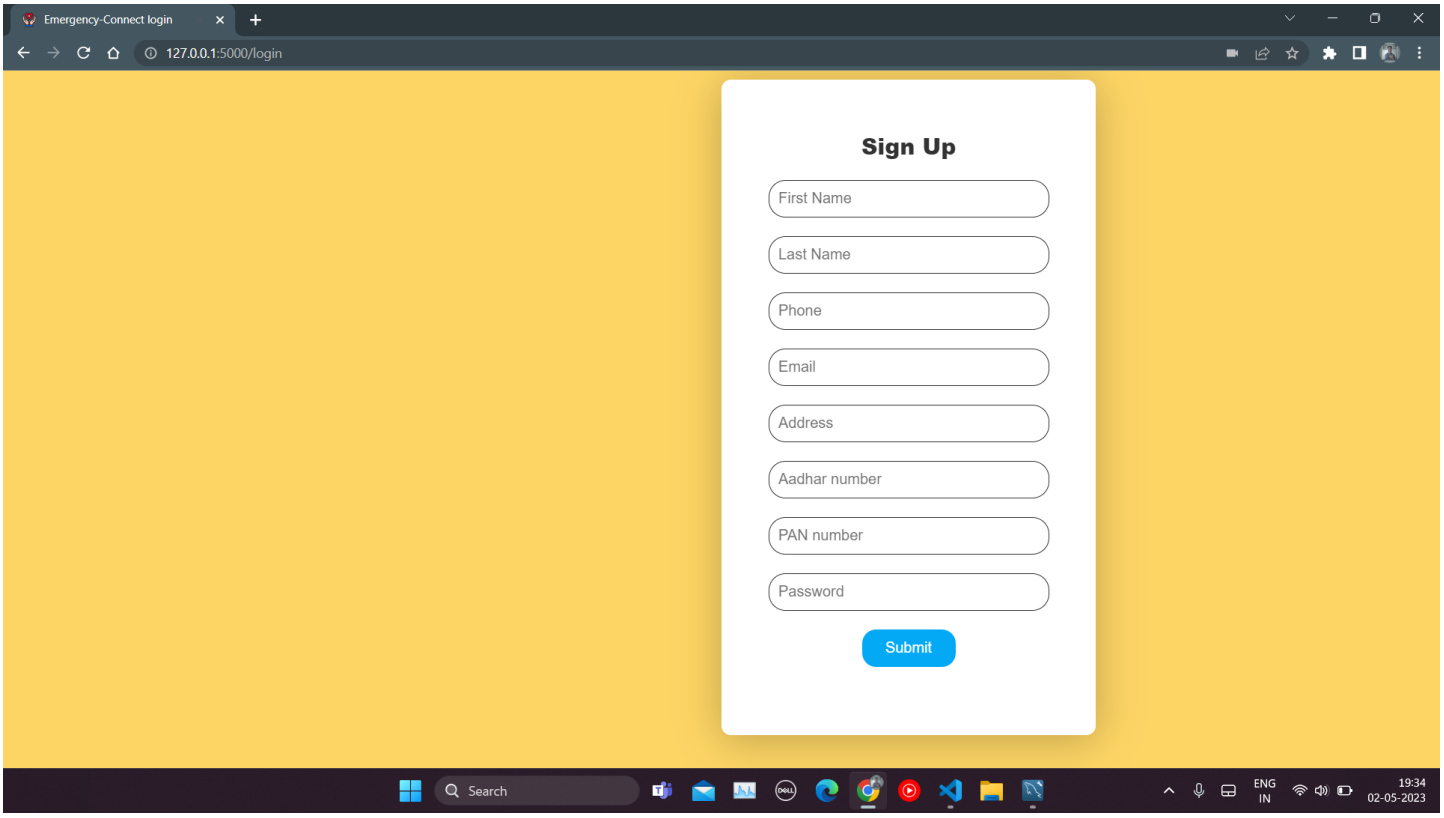


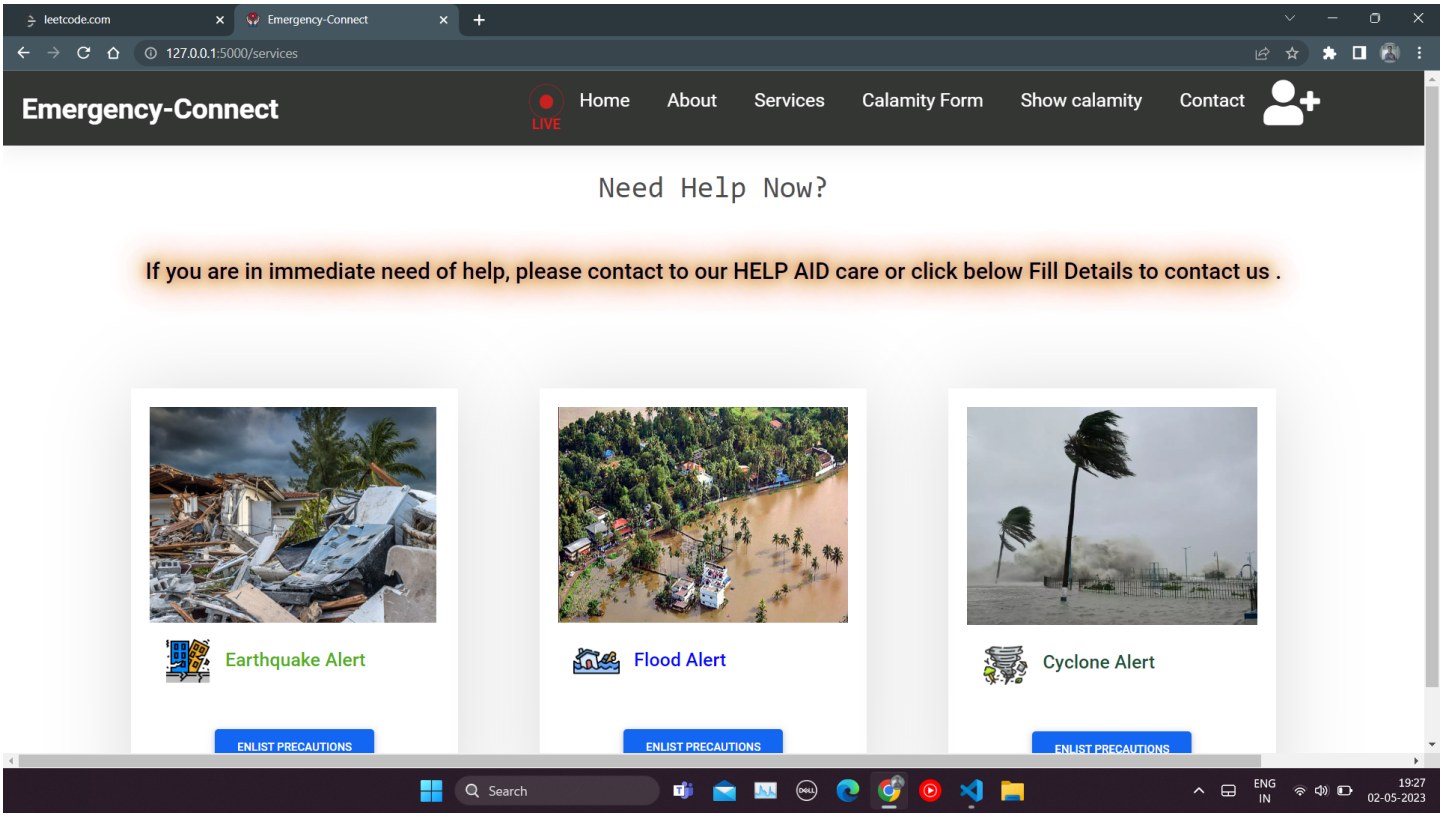


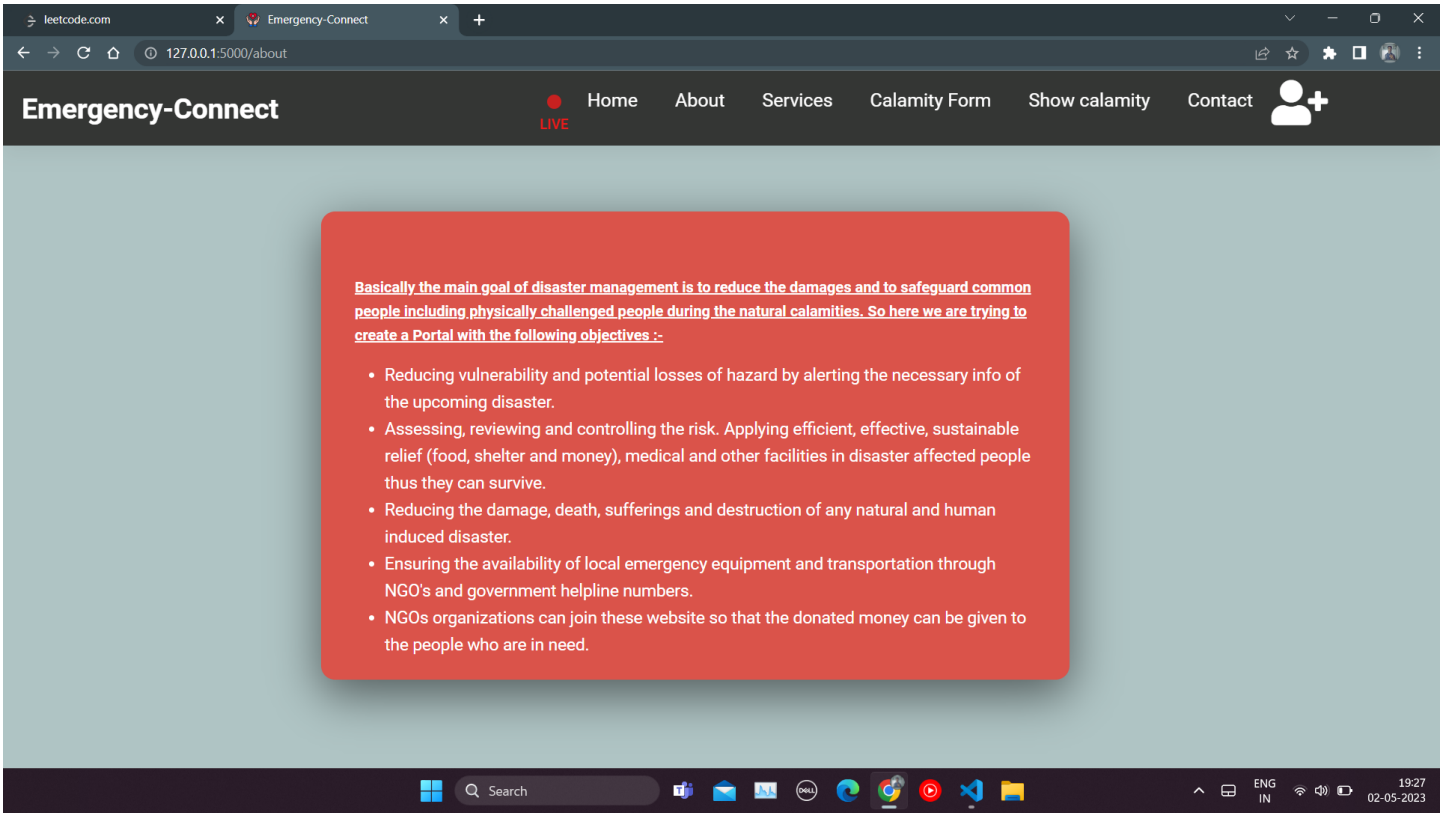

Emergency-Connect
127.0.0.1:5000/contact
Emergency-Connect
LIVE
Home
About
Services
Calamity Form
Show calamity
Contact
DR.K SATHI DEVI, SCIENTIST-F Phone :011-24631913 Email :wxchannel@gmail.com
Website Related
MR. SANKAR NATH, SCIENTIST-E Phone :01143824320 Email :sankar.nath@imd.gov.in
Data Related
DR. D S PAI,SCIENTIST-G Phone :91-020-25535877 Email :ds67.pai@imd.gov.in
Procurement/Tender Related
MR.KHUSHBEER SINGH,SCIENTIST-F Phone :91-11-24632234 Email :kushvir.singh@imd.gov.in
Administrative Matters Related
MR. Y K REDDY, SCIENTIST-F Phone :91-11-43824302 Email :yk.reddy@imd.gov.in
State Level Forecast/Data Related
ANDAMAN & NICOBAR MR. PRAVAT RABI NASKAR,SC-C Phone :91-01-9970501871 Email :pravat.naskar@imd.gov.in
ARUNACHAL PRADESH MR. K. N. MOHAN,SC-G Phone :91-361-2840206 Email :kn.mohan@imd.gov.in
ANDHRA PRADESH MS.S STELLA,SC-E Phone :91-021-8331086974 Email :s.stella@imd.gov.in
ASSAM MR. K. N. MOHAN,SC-G Phone :91-361-2840206 Email :kn.mohan@imd.gov.in
BIHAR MR. VIVEK SINHA,SC-F Phone :91-612-2251086 Email :vivek.sinha@imd.gov.in
CHANDIGARH MR.SURENDER PAUL,SC-F Phone :91-172-2629981 Email :surender.paul@imd.gov.in
CHHATTISGARH MR.H A K SINGH,SC-D Phone :91-771-4915258 Email :ha.singh@imd.gov.in
GUJARAT MS. MANORAMA MOHANTY,SC-E Phone :91-79-29705011 Email :m.mohanty@imd.gov.in
DELHI MR. CHARAN SINGH,SC-F Phone :91-11-24690279 Email :charan.singh@imd.gov.in
GOA MR. M RAHUL,SC-C Phone :91-832-2425547 Email :rahulm.89@imd.gov.in
DADRA & NAGAR HAVELI & DAMAN & DIU MS. MANORAMA MOHANTY,SC-E Phone :91-79-29705011 Email :m.mohanty@imd.gov.in
Search
ENG IN
19:34
02-05-2023


Emergency-Connect (Live Map)
WhatsApp
127.0.0.1:5000/live_maps
Emergency-Connect
LIVE
Home
About
Services
Calamity Form
Show calamity
Contact
Bhagwanpur
Garkha
Dariapur
Mahua
Chhapra
Sarai
Sitalpur
Goldinganj
Dighwara
Sonepur
Hajipur
Mahnar Bazar
Desri
Sarairanjan
Maner
Danapur
Patna
Shahpur Paton
Mahnal
Bidpur
Koelwar
Bihta
Kalyanpur
Mohiuddinnagar
Fatuha
Naubatpur
Punpun
Temperature
Altitude
2 m above ground
Perceived temperature
Precipitation
Radar
Satellite
Clouds
Wind speed
Wind gusts
Air pressure
Thunderstorms
Humidity
Waves
°F
120
100
90
80
70
60
50
40
30
Search
ENG IN
19:37
02-05-2023


leetcode.com
Emergency-Connect
127.0.0.1:5000/showcalamity
Emergency-Connect
LIVE
Home
About
Services
Calamity Form
Show calamity
Contact
Registered Candidates
Name
Address
Contact Number
Weather Calamity
Action
Mukund
Patna
7970869770
Flood
Solved
127.0.0.1:5000/showcalamity
Search
ENG
IN
19:33
02-05-2023


Emergency-Connect login
127.0.0.1:5000/login
Sign Up
First Name
Last Name
Phone
Email
Address
Aadhar number
PAN number
Password
Submit
Search
ENG
IN
19:34
02-05-2023


leetcode.com
Emergency-Connect
127.0.0.1:5000/services
Emergency-Connect
LIVE
Home
About
Services
Calamity Form
Show calamity
Contact
Need Help Now?
If you are in immediate need of help, please contact to our HELP AID care or click below Fill Details to contact us .
Earthquake Alert
ENLIST PRECAUTIONS
Flood Alert
ENLIST PRECAUTIONS
Cyclone Alert
ENLIST PRECAUTIONS
Search
ENG
IN
19:27
02-05-2023


leetcode.com
Emergency-Connect
127.0.0.1:5000/about
Emergency-Connect
LIVE
Home
About
Services
Calamity Form
Show calamity
Contact
Basically the main goal of disaster management is to reduce the damages and to safeguard common people including physically challenged people during the natural calamities. So here we are trying to create a Portal with the following objectives :-
Reducing vulnerability and potential losses of hazard by alerting the necessary info of the upcoming disaster.
Assessing, reviewing and controlling the risk. Applying efficient, effective, sustainable relief (food, shelter and money), medical and other facilities in disaster affected people thus they can survive.
Reducing the damage, death, sufferings and destruction of any natural and human induced disaster.
Ensuring the availability of local emergency equipment and transportation through NGO's and government helpline numbers.
NGOs organizations can join these website so that the donated money can be given to the people who are in need.
Search
ENG
IN
19:27
02-05-2023

Chapter 4
Testing Phase

4.1 Introduction

The testing phase includes quality assurance testing (QA), system integration testing (SIT), and user acceptance testing (UAT). Quality assurance testing (QA) tests against the customer's procedures and policies. System integration testing (SIT) tests how the software interacts with other software applications.

The primary purpose of the Test Phase is to determine whether the automated system/application software or other IT solution developed or acquired and preliminarily tested during the Development Phase is ready for implementation.

The testing is done to ensure that the entire application works according to the customer requirements. After testing, the QA and testing team might find some bugs or defects and communicate the same with the developers.

In order to uncover the error present in different phases we have the concept of level of testing.

The basic levels of testing are:

- **Unit Testing:**

  It focuses verification effort on the smallest unit of software i.e. the module. Using the detailed design & the process. Specification testing is done to uncover the error within the boundary of the module. All modules must be successful in the unit test before the start of integration testing begins.

- **Integration Testing:**

  After the unit testing, we have to perform the integration testing. The goal here is to see if the module can be integrated properly, the emphasis begins on the testing between modules. This testing activity can be considered as testing the design & hence the emphasis on testing module interaction.

- **System Testing:**

  Here the entire software system is tested. The reference document for this process is the requirement document, & the goal of the operating system to see if software meets its requirements.

- **Acceptance Testing:**

  It is performed with realistic data of the client to demonstrate that the software is working satisfactorily. Testing here is focused on external behaviour of the system: the internal logic of the program is not emphasized.

➢ **White Box Testing:**

  This is the unit testing method where a unit will be taken at a time & tested thoroughly at a statement level to find the maximum possible error.

➢ **Black Box Testing:**

This testing method considers a module as a single unit and checks the unit as interface and communication with other modules rather than getting into details at statement level. Here the module will be treated as black box that will take some input and generate output. Outputs for a given set of input combination are forwarded to the other module.

Testing Plan:

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. Testing presents an interesting anomaly for the software engineer.

- **Testing Objective includes**

  Testing is a process of executing a program with the intent of finding an error. A good test case is one that has a probability of finding an as yet undiscovered error. A successful test is one that uncovers an undiscovered error.

- **Testing Principles**
  - ➔ All tests should be traceable to end user requirements.
  - ➔ Tests should be planned long before testing begins.
  - ➔ Testing should begin on a small scale and progress towards testing in large.
  - ➔ Exhaustive testing is not possible.
  - ➔ To be most effective, testing should be conducted by an independent third party.

Test cases:

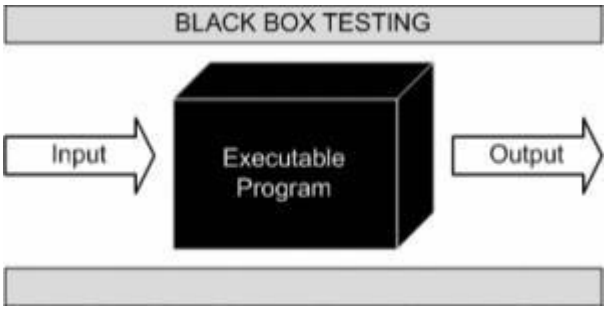
TEST CASE	EXPECTED RESULT	ACTUAL RESULT	RESULT STATUS
1	Login	Admin Login , Member Login , And Trainer Login	PASS
2	Registration	Registration Of member And Trainer.	PASS
3	Check Details	Admin can check the details of every member and every trainer which is registered. Trainer can check the details of Students.	PASS
4	Schedule the dates	Trainers will schedule the date and time for the member workout.	PASS

5	Notification To Students	Trainers sends the email or the message to the member whenever the they	PASS
		miss the workout.	

Test Results:

WHITE BOX TESTING (also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing) is a software testing method in which the internal structure/design/implementation of the item being tested is known to the tester. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. White box testing is testing beyond the user interface and into the nitty gritty of a system. This method is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one clearly sees.

BLACK BOX TESTING, also known as Behavioural Testing is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see.

Black Box testing method is applicable to the following levels of software testing:

- Integration Testing
- System Testing
- Acceptance Testing

Unit Testing:

Unit testing concentrates verification on the smallest element of the program, the module. Using the detailed design description important control paths are tested to establish errors within the bounds of the module. In this system each sub module is tested individually as per the unit testing such as campaign, lead, contact etc are tested individually. Their input field validations are tested.

Integration Testing:

Once all the individual units have been tested there is a need to test how they were put together to ensure no data is lost across the interface, one module does not have an adverse impact on another and a function is not performed correctly. After unit testing each and every submodule is tested with integrating each other.

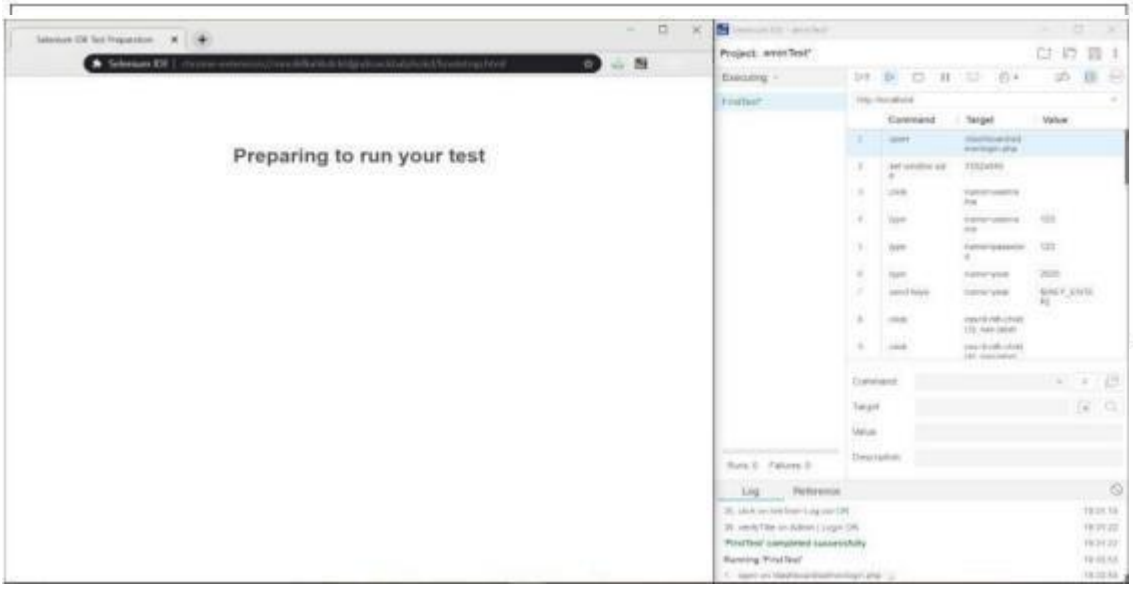
System Testing for Current System:

Modules of the project. We are testing whether the system is giving correct output or not. All the modules were integrated and the flow of information among different modules was checked. It was also checked whether the flow of data is as per the requirements or not. It was also checked that whether any particular module is non-functioning or not i.e. once the integration is over each and every module is functioning in its entirety or not. In this level of testing we tested the following: - Whether all the forms are properly working or not. Whether all the forms are properly linked or not. Whether all the images are properly displayed or not. Whether data travel is proper or not.

4.2 Selenium

Selenium is free and open source and it is an automation tool. It provides a playback tool for functional testing for learning to test scripting language. Selenium is an API that works with Java, C#, perl, ruby, python to automate browser activities. It provides a user-friendly interface that helps to create and execute scripts easily. It provides multi-browser support, meaning one selenium script for all browsers.

Selenium Architecture:



Selenium Output:

Chapter 5
Maintenance Phase

5.1 Introduction

Maintenance is crucial for the Disaster management system as it ensures the system's smooth functioning and helps to identify and fix any issues that arise. The system must be continuously maintained to ensure it remains reliable, efficient, and effective.

Types of Maintenance for Disaster Management System:

There are several types of maintenance that can be performed for a Disaster Management System:

- **Corrective Maintenance**: This type of maintenance involves fixing issues or problems with the system after they have occurred. This can include bug fixes, software patches, and hardware replacements.
- **Preventive Maintenance**: This type of maintenance involves regular system checks, updates, and maintenance to prevent issues from occurring in the first place. This can include routine system backups, security updates, and hardware maintenance.
- **Predictive Maintenance:** This type of maintenance involves using data and analytics to predict potential issues or failures in the system before they occur. This can include monitoring system performance and setting up alerts for potential issues.
- **Adaptive Maintenance:** This type of maintenance involves making changes to the system to adapt to changing business requirements or user needs. This can include adding new features, upgrading hardware, or changing software configurations.
- **Perfective Maintenance:** This type of maintenance involves making improvements to the system to enhance its functionality or performance. This can include optimizing system speed, improving user experience, or adding new integrations.

The maintenance process for the Disaster Management System includes the following steps:

- Problem identification
- Analysis of the problem
- Design of the solution
- Implementation of the solution
- Testing of the solution
- Deployment of the solution

Tools and Techniques for Maintaining Disaster Management System:

There are several tools and techniques that can be used to maintain the Disaster Management System, such as:

- Version control systems
- Debugging tools
- Testing tools
- Code analysis tools
- Refactoring tools

Challenges in Maintaining s Disaster Management System:

The Disaster Management System may face several challenges during maintenance, such as:

- Lack of documentation
- Legacy code
- Limited resources
- Changing requirements
- Time constraints

Best Practices for Maintaining Disaster Management System:

To ensure successful maintenance of the Disaster Management System, it is important to follow best practices such as:

- Establishing a maintenance team
- Creating a maintenance plan
- Keeping the documentation up to date
- Using version control systems
- Performing regular code reviews
- Implementing automated testing
- Refactoring code as needed

Conclusion and Future Scope

In conclusion, a disaster management system plays a crucial role in mitigating the impact of disasters by providing timely and effective response to the affected population. Such a system involves multiple stakeholders, including government agencies, non-governmental organizations, and local communities, working in close collaboration to ensure a coordinated and efficient response.

The future scope of disaster management systems lies in leveraging emerging technologies such as artificial intelligence, machine learning, and data analytics to enhance the efficiency and effectiveness of response efforts. For instance, predictive analytics can be used to forecast the occurrence and severity of disasters, enabling authorities to take proactive measures to prevent or mitigate their impact.

In addition, the use of drones and other unmanned aerial vehicles can facilitate rapid assessment of the disaster-affected areas and aid in search and rescue operations. Furthermore, the integration of social media and other digital platforms can help to disseminate critical information to the affected population in real-time, enabling them to take necessary precautions and make informed decisions.

Overall, the future of disaster management systems lies in the convergence of technology and human expertise, with the ultimate goal of ensuring the safety and well-being of communities affected by disasters.