DISASTER MANAGEMENT

A MINI-PROJECT REPORT

Submitted by

SIVA BALAN K 230701316

SANJAI KUMARAN K 230701284

In partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER 2024

BONAFIDE CERTIFICATE

Certified that this project "DISASTER MANAGEMENT" is the bonafide work of "SIVABALAN,SANJAIKUMARAN" who carried out the project work under

my supervision.

SIGNATURE

MR G SARAVANA GOKUL

JANANEE ASSISTANT PROFESSOR

PROFESSOR

Dept. of Computer Science and Engg,

Rajalakshmi Engineering College

Chennai

SIGNATURE

MRS V

ASSISTANT

Dept. of Computer Science and Engg,

Rajalakshmi Engineering College

Chennai

This mini project report is submitted for the viva voce examination to be held on _____

TABLE OF CONTENTS

ABSTRACT:

The increasing frequency and intensity of natural and man-made disasters necessitate the development of efficient systems for disaster management. The **Disaster Management System (DMS)** is a software solution designed to support the planning, coordination, and response efforts during disasters, helping organizations and government agencies mitigate the effects of such events

This mini project aims to build a disaster management application using **Java** for the frontend and **MySQL** (or any relational DBMS) for the backend, offering a comprehensive system to handle disasterrelated data effectively.

The system facilitates real-time data entry, management, and analysis of disaster information, such as the type of disaster, affected areas, available resources, and ongoing response efforts. It includes features for resource management, disaster alerts, communication between stakeholders, and generating reports to assist in decision-making. The integration with a database ensures the seamless storage, retrieval, and management of data, ensuring that critical information is accessible and up-to-date.

By implementing the **Disaster Management System**, this project aims to demonstrate the potential of combining Java and a Database Management System (DBMS) to create a scalable, efficient, and userfriendly platform for disaster management. This application will not only enhance the coordination and communication during disaster events but also assist in making data-driven decisions to minimize the impact on affected populations. The project ultimately seeks to improve disaster response, save lives, and optimize resource allocation in critical situations.

INTRODUCTION:

In recent years, natural and man-made disasters have caused significant harm to communities around the world. Efficient disaster management is crucial in mitigating the impact of such events. A robust disaster management system helps in preparation, response, and recovery efforts, ensuring that the affected areas receive timely assistance and support.

The Disaster Management System (DMS) is a software application designed to aid in disaster management activities by leveraging technology to handle data related to disasters. The system will serve as a tool for government agencies, rescue teams, and NGOs involved in disaster management. By streamlining data storage, communication, and response coordination, the system aims to enhance the effectiveness and efficiency of disaster management operations.

This mini project is developed in Java as the primary programming language, utilizing object-oriented principles for creating a flexible and maintainable solution. The backend of the system is integrated with a Database Management System (DBMS), using technologies such as MySQL or Oracle, to store and manage disaster-related data, including disaster types, affected regions, a

SCOPE OF THE PROJECT

The Disaster Management System (DMS) developed in this project aims to address the challenges faced during disaster situations by providing an efficient, centralized platform for managing disaster-related data. The system is designed to assist various stakeholders, including government agencies, NGOs, rescue teams, and local authorities, by offering real-time information, resources, and communication tools. The scope of the project encompasses the following key areas:

1. Real-Time Disaster Data Management:
   - The system will enable the entry, retrieval, and update of disaster-related information such as the type of disaster (e.g., earthquake, flood, hurricane), affected regions, and severity.
   - Users will be able to log important data such as the number of casualties, damages, and the immediate needs of the affected regions.
2. Resource Management:
   - The system will provide a mechanism to manage resources such as rescue teams, medical supplies, relief materials, and equipment.
   - It will allow stakeholders to track the availability and allocation of resources, ensuring efficient deployment to the affected regions.
3. Communication and Alert System:
   - The system will include functionality for sending alerts and notifications to emergency contacts, rescue teams, and authorities via SMS or email in case of a disaster.
   - It will also allow communication between different disaster management teams and agencies to coordinate efforts effectively.
4. Reporting and Analytics:

- The system will offer analytical tools for generating real-time reports, which will aid decision-makers in assessing the current situation and planning the response.
- It will provide visualizations, charts, and summaries based on disaster data to make it easier for stakeholders to evaluate the severity and response needs.

5. Database Integration:
   - The project will utilize a Database Management System (DBMS) (e.g., MySQL, Oracle) for storing disaster data securely.
   - The database will ensure quick data retrieval and update, allowing multiple users to interact with the system without data loss or inconsistency.
6. User Interface and Role Management:
   - The system will provide an intuitive, user-friendly interface to ensure that even non-technical users (e.g., field personnel, local authorities) can access and manage data.
   - Role-based access control will be implemented to manage the permissions of different users (e.g., admin, rescue teams, local authorities), ensuring that sensitive information is only accessible to authorized personnel.
7. Scalability and Flexibility:
   - The system will be scalable to accommodate increasing amounts of data during large-scale disasters, ensuring high availability and reliability.
   - It will be designed in such a way that additional features (such as mobile integration or real-time monitoring) can be incorporated in the future.
8. Integration with External Systems (Optional):

- In the future, the system could be extended to integrate with other disaster management systems or geographic information systems (GIS) to enhance decision-making, such as tracking disaster spread and mapping relief distribution.

9. User Training and Documentation:

- Basic training will be provided for users to ensure they are familiar with the features and functionalities of the system.
- Comprehensive documentation will be available for both users and administrators, including how to manage data, generate reports, and use the alert system.

10. Limitations:

- The current version of the system will focus on smaller-scale disaster management and may not include advanced predictive capabilities or real-time tracking of disasters.
- The system will be a desktop/web-based solution, with mobile application development considered for future updates.

Future Enhancements

1. **Mobile Application Integration**
2. **Real-Time Disaster Monitoring and GeoLocation Tracking**
3. **Predictive Analytics and Machine Learning Integration**

4. Integration with Social Media and Crowdsourced Data

.

Key Features of the funcity management System in JAVA:

Introduction to Mini Project for Disaster Management System in Java with DBMS

In recent years, natural and man-made disasters have caused significant harm to communities around the world. Efficient disaster management is crucial in mitigating the impact of such events. A robust disaster management system helps in preparation, response, and recovery efforts, ensuring that the affected areas receive timely assistance and support.

The Disaster Management System (DMS) is a software application designed to aid in disaster management activities by leveraging technology to handle data related to disasters. The system will serve as a tool for government agencies, rescue teams, and NGOs involved in disaster management. By streamlining data storage, communication, and response coordination, the system aims to enhance the effectiveness and efficiency of disaster management operations.

This mini project is developed in Java as the primary programming language, utilizing object-oriented principles for creating a flexible and maintainable solution. The backend of the system is integrated with a Database Management System (DBMS), using technologies such as MySQL or Oracle, to store and manage disaster-related data, including disaster types, affected regions, and available resources.

Key Features of the System:

1. Disaster Data Management: The system will store critical information related to disasters, such as type, severity, location, and the response strategies.
2. Resource Management: It will track available resources such as medical supplies, rescue teams, and relief materials, ensuring they are efficiently allocated.
3. Communication & Alerts: The system can send alerts to emergency contacts, government officials, and rescue teams about ongoing disasters, using email or SMS services.
4. Reporting and Analytics: It will provide real-time data and analytical reports to decision-makers, enabling better disaster response planning.
5. Database Integration: A MySQL database will store all disaster-related information and ensure that the data is easily accessible, secure, and manageable.

Objectives:

- To build a system that supports real-time data entry and management during a disaster.
- To integrate disaster management modules with a database for efficient data retrieval and storage.
- To develop a user-friendly interface that can be used by both technical and non-technical personnel involved in disaster management.

In conclusion, this mini project aims to demonstrate how technology can enhance disaster management, providing timely and accurate data that will help in saving lives and reducing the impact of disasters. The combination of Java and DBMS creates a robust and scalable system, ensuring that the platform can handle large-scale disaster data efficiently.

The increasing frequency and intensity of natural and man-made disasters necessitate the development of efficient systems for disaster management. The Disaster Management System (DMS) is a software solution designed to support the planning, coordination, and response efforts during disasters, helping organizations and government agencies mitigate the effects of such events. This mini project aims to build a disaster management application using Java for the frontend and MySQL (or any relational DBMS) for the backend, offering a comprehensive system to handle disaster-related data effectively.

The system facilitates real-time data entry, management, and analysis of disaster information, such as the type of disaster, affected areas, available resources, and ongoing response efforts. It includes features for resource management, disaster alerts, communication between stakeholders, and generating reports to assist in decisionmaking. The integration with a database ensures the seamless storage, retrieval, and management of data, ensuring that critical information is accessible and up-to-date.

By implementing the Disaster Management System, this project aims to demonstrate the potential of combining Java and a Database Management System (DBMS) to create a scalable, efficient, and user-friendly platform for disaster management. This application will not only enhance the coordination and communication during disaster events but also assist in making data-driven decisions to minimize the impact on affected populations. The project ultimately seeks to improve disaster response, save lives, and optimize resource allocation in critical situations.

4o mini

You said:

SCOPE FOR PROJECT ChatGPT

said:

ChatGPT

Scope of the Disaster Management System Project

The Disaster Management System (DMS) developed in this project aims to address the challenges faced during disaster situations by providing an efficient, centralized platform for managing disaster-related data. The system is designed to assist various stakeholders, including government agencies, NGOs, rescue teams, and local authorities, by offering real-time information, resources, and communication tools. The scope of the project encompasses the following key areas:

1. Real-Time Disaster Data Management:
   - The system will enable the entry, retrieval, and update of disaster-related information such as the type of disaster (e.g., earthquake, flood, hurricane), affected regions, and severity.
   - Users will be able to log important data such as the number of casualties, damages, and the immediate needs of the affected regions.
2. Resource Management:
   - The system will provide a mechanism to manage resources such as rescue teams, medical supplies, relief materials, and equipment.
   - It will allow stakeholders to track the availability and allocation of resources, ensuring efficient deployment to the affected regions.
3. Communication and Alert System:
   - The system will include functionality for sending alerts and notifications to emergency contacts, rescue teams, and authorities via SMS or email in case of a disaster.

- It will also allow communication between different disaster management teams and agencies to coordinate efforts effectively.

4. Reporting and Analytics:
   - The system will offer analytical tools for generating real-time reports, which will aid decision-makers in assessing the current situation and planning the response.
   - It will provide visualizations, charts, and summaries based on disaster data to make it easier for stakeholders to evaluate the severity and response needs.
5. Database Integration:
   - The project will utilize a Database Management System (DBMS) (e.g., MySQL, Oracle) for storing disaster data securely.
   - The database will ensure quick data retrieval and update, allowing multiple users to interact with the system without data loss or inconsistency.
6. User Interface and Role Management:
   - The system will provide an intuitive, user-friendly interface to ensure that even non-technical users (e.g., field personnel, local authorities) can access and manage data.
   - Role-based access control will be implemented to manage the permissions of different users (e.g., admin, rescue teams, local authorities), ensuring that sensitive information is only accessible to authorized personnel.
7. Scalability and Flexibility:
   - The system will be scalable to accommodate increasing amounts of data during large-scale disasters, ensuring high availability and reliability.

- It will be designed in such a way that additional features (such as mobile integration or real-time monitoring) can be incorporated in the future.

8. Integration with External Systems (Optional):

In the future, the system could be extended to integrate with other disaster management systems or geographic information systems (GIS) to enhance decision-making, such as tracking disaster spread and mapping relief distribution.

9. User Training and Documentation:

- Basic training will be provided for users to ensure they are familiar with the features and functionalities of the system.
- Comprehensive documentation will be available for both users and administrators, including how to manage data, generate reports, and use the alert system.

10. Limitations:

- The current version of the system will focus on smaller-scale disaster management and may not include advanced predictive capabilities or real-time tracking of disasters.
- The system will be a desktop/web-based solution, with mobile application development considered for future updates.

Conclusion:

The scope of this project covers the development of a disaster management system that streamlines the management of disaster data, communication, and resources through a user-friendly platform. By integrating Java for the frontend and a DBMS for the backend, this system aims to improve the efficiency and coordination of disaster management efforts, leading to better decision-making, faster response times, and ultimately, saving lives and resources.

Future Enhancements for the Disaster Management System

The Disaster Management System (DMS) developed in this project provides a foundational framework for disaster response and coordination. However, as technology and disaster management practices evolve, there are several potential future enhancements

that can be incorporated into the system to increase its functionality, scalability, and effectiveness. Below are some of the key future enhancements that could be implemented:

1. Mobile Application Integration
   - Current System: The system can be used via desktop/web interfaces.
   - Enhancement: Developing a mobile application (Android/iOS) will enable disaster management personnel, field workers, and citizens to access real-time disaster data and alerts from their smartphones. Mobile apps can also allow users to report disaster information, location-based issues, and send distress signals from remote areas.
   - Benefits: Increases accessibility, ensures quicker communication, and makes the system more versatile for use during on-the-ground disaster response.
2. Real-Time Disaster Monitoring and Geo-Location Tracking
   - Current System: The system provides data management, alerts, and reports but lacks real-time tracking of disaster events and affected regions.
   - Enhancement: Integrating Geographic Information Systems (GIS) or real-time disaster tracking tools (such as satellite data, weather updates, and GPS tracking) will enable real-time mapping of disaster impact zones. This can include tracking the spread of floods, wildfires, or hurricanes and visualizing it on a map.
   - Benefits: Improves situational awareness, helps coordinate response efforts based on real-time data, and supports resource allocation in real time.

3. Predictive Analytics and Machine Learning Integration Current System: The system relies on data input and manual reporting, with no advanced predictive capabilities.

   - Enhancement: Implementing predictive analytics and machine learning models could enable the system to forecast potential disaster events based on historical data, weather patterns, and environmental indicators. For instance, predicting flood risks or earthquake aftershocks using data trends.
   - Benefits: Allows proactive planning, early warnings, and the possibility to prevent or mitigate damage before disasters occur.

4. Integration with Social Media and Crowdsourced Data

   - Current System: The system is limited to data provided by authorized users and databases.
   - Enhancement: Incorporating social media feeds (e.g., Twitter, Facebook) and crowdsourced data from individuals in disaster-affected areas can provide real-time updates about the disaster situation from citizens. This could include reporting locations of damage, victims, and local conditions.
   - Benefits: Improves real-time situational awareness, broadens the information network, and enhances disaster response efforts by tapping into community-driven intelligence.

5. Integration with IoT (Internet of Things) Devices

   - Current System: The system does not leverage real-time environmental sensor data.
   - Enhancement: Integration with IoT sensors (e.g., air quality sensors, flood sensors, seismic sensors, weather stations) can provide real-time environmental data to further assess the disaster situation. For example, IoT-enabled sensors can

.

monitor rising water levels, track air quality during wildfires, or monitor seismic activity.

- Benefits: Provides up-to-date environmental data to improve situational analysis and response. It could also trigger automated alerts based on sensor thresholds.

6. Cloud-Based Infrastructure
   - Current System: The system uses a traditional DBMS (e.g., MySQL) which is typically hosted on local servers.
   - Enhancement: Migrating the backend to a cloud platform (e.g., AWS, Google Cloud, Microsoft Azure) would offer scalability, high availability, and faster access to the system, even in remote areas.
   - Benefits: Increases reliability, supports large-scale disaster events, and allows for better data storage and retrieval, even when local infrastructure is compromised.
7. Automated Resource Allocation System
   - Current System: Resource management is manually tracked and allocated.
   - Enhancement: A more advanced resource allocation module could be developed, using algorithms to automatically allocate available resources based on factors such as severity of the disaster, proximity to affected areas, and available supply.
   - Benefits: Optimizes resource deployment, reduces human error, and speeds up the response time.
8. Multi-Language and Multi-Region Support
   - Current System: The system may currently support only one language and region.
   - Enhancement: Adding multi-language support and multi-

- region capabilities will allow the system to be used in disaster situations across different countries, regions, and linguistic groups.

  Benefits: Makes the system globally applicable, ensuring that it can be used in various disaster-prone areas worldwide.

9. Offline Functionality
   - Current System: Requires internet access for data entry and retrieval.
   - Enhancement: Developing offline functionality would allow users, particularly field personnel, to enter data and perform essential functions in areas with poor or no internet connectivity. Once the connection is restored, the data would automatically sync with the central database.
   - Benefits: Ensures the system is functional in remote or disaster-affected areas where internet access is limited or unavailable.
10. Public Awareness and Communication Tools
    - Current System: The system is primarily designed for internal disaster management teams.
    - Enhancement: Add public-facing components such as a web portal or mobile app for citizens to receive disaster alerts, updates, and preparedness tips. Additionally, the system could include features for public participation, such as volunteering, donation management, and reporting issues.
    - Benefits: Engages the general public in disaster preparedness and response, facilitates easier communication with citizens, and promotes community involvement.

11. Post-Disaster Recovery and Impact Assessment

- Current System: The system handles disaster data and realtime response, but lacks functionality for long-term recovery.
- Enhancement: Add modules for post-disaster recovery, including damage assessment, infrastructure rebuilding, and tracking the recovery process. This could include financial and resource tracking, health assessments, and rebuilding efforts.
- Benefits: Provides a more comprehensive solution by assisting in both immediate response and long-term recovery efforts.

12. Integration with National and International Disaster Networks

- Current System: The system operates at a local or organizational level.
- Enhancement: Integrate with national and international disaster response networks, such as UN disaster relief agencies, FEMA (Federal Emergency Management Agency), or the Red Cross, to share critical data and coordinate largerscale disaster responses.
- Benefits: Ensures coordination between multiple agencies and countries, leading to more effective disaster management and resource allocation during large-scale disasters.

Conclusion:

The Disaster Management System offers a robust foundation for disaster response and management, and the future enhancements discussed above will significantly increase its capabilities and scope. By integrating new technologies like mobile apps, GIS, machine learning, IoT, and cloud infrastructure, the system will evolve into a more comprehensive and global tool that can improve the effectiveness of disaster response and recovery efforts. These future

.

enhancements will ensure that the system remains adaptable, scalable, and capable of addressing the increasingly complex challenges posed by natural and man-made disasters.

Key Features of Disaster Management System in Java with Predictive Analytics and Machine Learning Integration

Incorporating Predictive Analytics and Machine Learning (ML) into a Disaster Management System (DMS) enhances the system's ability to anticipate and manage disasters proactively. Below are key features of a disaster management system in Java with a focus on Predictive Analytics and Machine Learning Integration:

---

1. Real-Time Data Collection and Integration
   - Feature Description: The system gathers real-time data from various sources, such as weather APIs, sensors, satellite feeds, and social media to monitor evolving disaster conditions (e.g., hurricanes, earthquakes, floods).
   - Predictive Analytics Role: By using machine learning algorithms like regression models or time-series forecasting, the system can analyze historical data and predict future trends in disaster events, such as the likelihood of a natural disaster or the spread of a disaster in specific regions.
   - Benefit: Helps in monitoring live conditions, identifying potential threats, and providing early warnings.
2. Disaster Event Prediction (Risk Assessment)
   - Feature Description: The system can predict the occurrence of natural disasters, such as earthquakes, floods, or storms, based on historical data, patterns, and environmental factors.
   - Predictive Analytics Role: Machine learning models such as classification algorithms (e.g., decision trees, random forests, SVM) can be trained on datasets that include environmental conditions, seismic activity, and weather patterns to predict the likelihood of a disaster occurring in a given area.
   - Benefit: Allows governments and organizations to take preventive actions, such as evacuations, resource distribution,

and early warnings, thereby minimizing the loss of life and property damage.

3. Resource Allocation Optimization

- Feature Description: The system tracks available resources (e.g., medical teams, relief materials, rescue teams) and their current locations. It can recommend the best possible allocation of resources during a disaster.
- Machine Learning Role: ML optimization techniques, such as genetic algorithms or reinforcement learning, can be used to recommend optimal resource distribution based on realtime data (e.g., disaster intensity, affected population).
- Benefit: Ensures that resources are deployed efficiently and in the right areas, reducing the response time and improving disaster relief efforts.

4. Damage Assessment and Impact Prediction

- Feature Description: The system can predict the damage and impact of a disaster on infrastructure, human life, and the economy based on historical patterns.
- Machine Learning Role: Supervised learning models (e.g., regression models, neural networks) can be applied to assess the correlation between environmental conditions (like rainfall or seismic data) and damage reports to predict potential damage in future disaster scenarios.
- Benefit: Helps in quick recovery planning, financial planning, and identifying which areas require immediate intervention.

5. Automated Alert System with Predictive Warnings

- Feature Description: The system automatically sends out alerts (SMS, email, mobile app notifications) to the relevant authorities, organizations, and citizens based on predictions of imminent disasters.

- Machine Learning Role: ML models (e.g., decision trees, Naive Bayes) can be used to automatically trigger warnings when the system predicts a disaster event based on certain thresholds (e.g., if a flood level exceeds a predefined limit or if seismic activity surpasses a certain magnitude).
- Benefit: Provides early warning systems that allow people to prepare or evacuate in time, reducing casualties.

6. Post-Disaster Recovery Prediction

- Feature Description: After a disaster occurs, the system can forecast the time required for recovery in different regions based on the severity of the event and available resources.
- Machine Learning Role: Using predictive models (e.g., regression models, clustering), the system can estimate the recovery period by analyzing past recovery data and disaster impact. The system can also predict the need for further interventions in specific areas.
- Benefit: Assists disaster response teams in planning and allocating resources for long-term recovery and rebuilding efforts.

7. Crowdsourced Data Integration for Enhanced Predictive Models

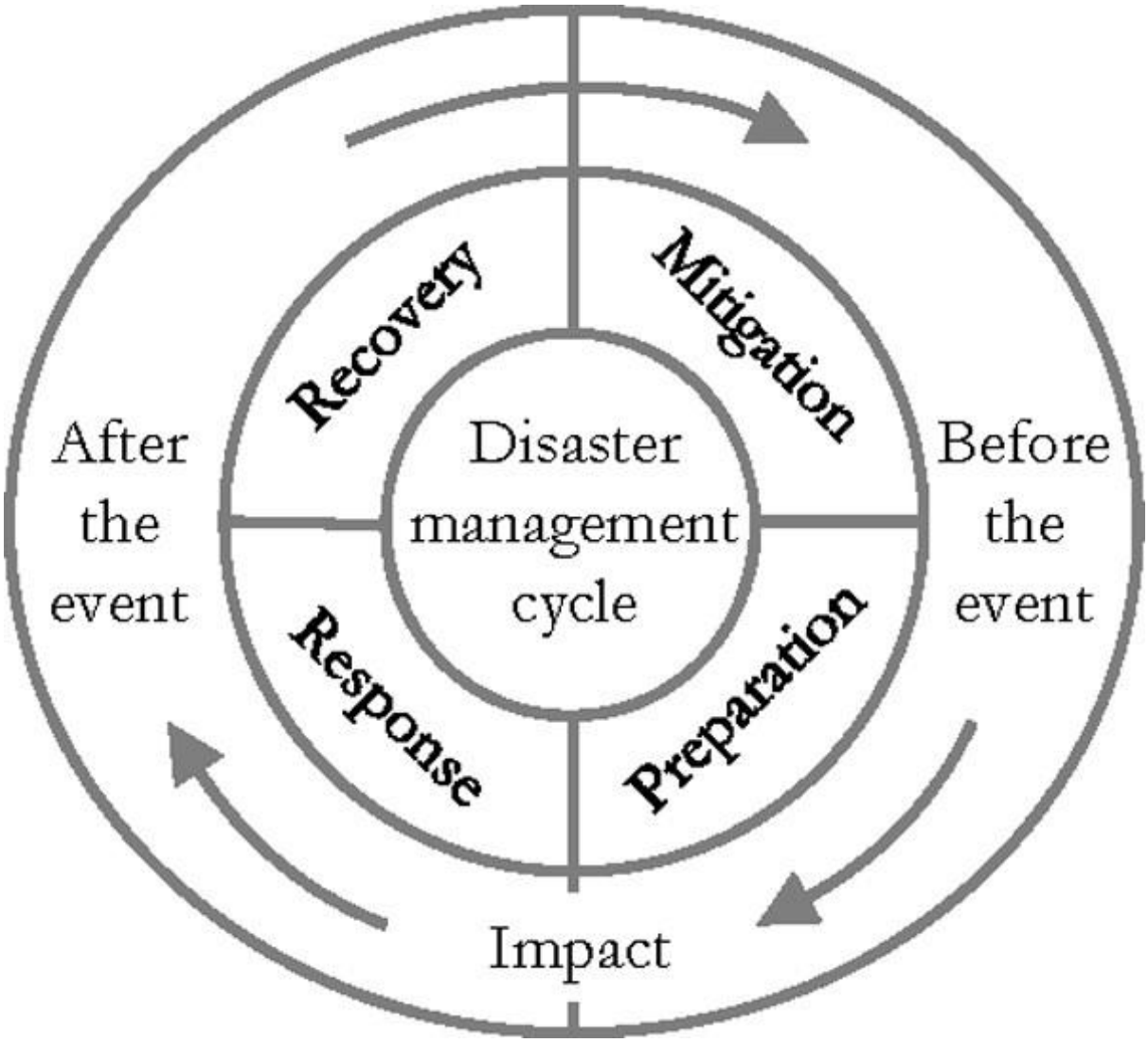
- Feature Description: The system allows citizens and field workers to report disaster-related information via mobile apps or web platforms (e.g., location of damage, blocked roads, casualties).
- Machine Learning Role: This crowdsourced data is integrated with predictive models, and unsupervised learning techniques (e.g., clustering) can be used to identify patterns and anomalies from the collected reports to improve disaster predictions and response strategies.

- Benefit: Improves the accuracy of predictive models and ensures that the system adapts to real-time changes in the environment.

8. Social Media Analysis for Disaster Prediction
   - Feature Description: The system integrates social media platforms (Twitter, Facebook, etc.) to gather information about real-time public sentiment, potential threats, and early warnings.
   - Machine Learning Role: Natural Language Processing (NLP) algorithms can analyze posts and tweets to detect patterns of disaster-related discussions (e.g., reports of unusual weather, sightings of wildfires, or storm warnings) to trigger early alerts and updates.
   - Benefit: Enhances predictive analytics by providing additional real-time data sources to anticipate and respond to disasters quicker.
9. Simulations for Disaster Scenarios
   - Feature Description: The system can simulate disaster scenarios (such as earthquakes, floods, and wildfires) to predict how they will unfold under various conditions.
   - Machine Learning Role: Simulation models can be integrated with machine learning techniques to run multiple disaster scenarios, using environmental data to predict the effects of various variables (e.g., weather conditions, population density).
   - Benefit: Helps authorities plan for different scenarios and develop strategies for resource allocation, evacuation plans, and other crisis response actions.
10. Real-Time Event Forecasting with Time-Series Analysis

- Feature Description: Time-series forecasting techniques can be employed to analyze continuous data streams, such as weather patterns, seismic activity, or river water levels, to predict the likelihood and intensity of future events.
- Machine Learning Role: ARIMA (AutoRegressive Integrated Moving Average) or more advanced deep learning models (like LSTM networks) can be applied to predict disaster events based on historical data.
- Benefit: Provides a real-time forecast of disaster events, enabling authorities to take timely preventive or mitigating acting.

ERDIAGRAM


Recovery
Mitigation
After the event
Disaster management cycle
Before the event
Response
Preparation
Impact

Code implementation:

Main: login page :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;
```

```
public class Loginpage extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton loginButton;
    private JButton registerButton;
    private JLabel messageLabel;

    public Loginpage() {
        // Frame settings
        setTitle("Login Page");
        setExtendedState(JFrame.MAXIMIZED_BOTH); // Set
window to full screen
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create main panel with white background
        JPanel mainPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                // Load background image
                ImageIcon icon = new ImageIcon("flood_background.jpg");
// Update with your image path
                Image image = icon.getImage();

                // Scale the image to fit the entire window
                g.drawImage(image, 0, 0, getWidth(), getHeight(), this);
```

```
        }
    };
    mainPanel.setLayout(new BorderLayout(10, 10));
    mainPanel.setBorder(BorderFactory.createEmptyBorder(50, 50, 50, 50)); // Increased padding

    // Create form panel
    JPanel formPanel = new JPanel(new GridBagLayout());
    formPanel.setBackground(new Color(255, 255, 255, 150)); // Semi-transparent background
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.insets = new Insets(10, 10, 10, 10); // Increased spacing

    // UI Components with styled fonts and colors - increased sizes for better visibility
    Font labelFont = new Font("Arial", Font.BOLD, 18); // Increased font size
    Font fieldFont = new Font("Arial", Font.PLAIN, 18); // Increased font size

    JLabel titleLabel = new JLabel("User Login", SwingConstants.CENTER);
    titleLabel.setFont(new Font("Arial", Font.BOLD, 36)); // Increased title size
    titleLabel.setForeground(new Color(51, 51, 51));

    JLabel usernameLabel = new JLabel("Username:");
```

```
        usernameLabel.setFont(labelFont);
        usernameField = new JTextField(20);
        usernameField.setFont(fieldFont);
        usernameField.setPreferredSize(new Dimension(300, 40)); // Increased field size

        JLabel passwordLabel = new JLabel("Password:");
        passwordLabel.setFont(labelFont);
        passwordField = new JPasswordField(20);
        passwordField.setFont(fieldFont);
        passwordField.setPreferredSize(new Dimension(300, 40)); // Increased field size

        loginButton = new JButton("Login");
        registerButton = new JButton("Register");
        messageLabel = new JLabel("", SwingConstants.CENTER);
        messageLabel.setForeground(Color.RED);
        messageLabel.setFont(new Font("Arial", Font.BOLD, 16));

        // Style buttons - both in blue with increased size
        Color buttonBlue = new Color(0, 120, 215);
        Dimension buttonSize = new Dimension(150, 50); // Increased button size

        loginButton.setBackground(buttonBlue);
        loginButton.setForeground(Color.BLACK);
        loginButton.setFocusPainted(false);
```

```
loginButton.setFont(new Font("Arial", Font.BOLD, 18));
loginButton.setPreferredSize(buttonSize);

registerButton.setBackground(buttonBlue);
registerButton.setForeground(Color.BLACK);
registerButton.setFocusPainted(false);
registerButton.setFont(new Font("Arial", Font.BOLD, 18));
registerButton.setPreferredSize(buttonSize);

// Add components to form panel with increased spacing
gbc.gridx = 0;
gbc.gridy = 0;
gbc.gridwidth = 2;
gbc.insets = new Insets(0, 0, 50, 0); // Extra space below title
formPanel.add(titleLabel, gbc);

gbc.insets = new Insets(10, 10, 10, 10);
gbc.gridy = 1;
gbc.gridwidth = 1;
formPanel.add(usernameLabel, gbc);

gbc.gridx = 1;
formPanel.add(usernameField, gbc);

gbc.gridx = 0;
gbc.gridy = 2;
```

```
        formPanel.add(passwordLabel, gbc);

        gbc.gridx = 1;
        formPanel.add(passwordField, gbc);

        // Button panel with increased spacing
        JPanel buttonPanel = new JPanel(new
FlowLayout(FlowLayout.CENTER, 20, 3));
        buttonPanel.setBackground(Color.WHITE);
        buttonPanel.add(loginButton);
        buttonPanel.add(registerButton);

        gbc.gridx = 0;
        gbc.gridy = 3;
        gbc.gridwidth = 2;
        gbc.insets = new Insets(30, 0, 20, 0); // Extra space above
buttons
        formPanel.add(buttonPanel, gbc);

        gbc.gridy = 4;
        gbc.insets = new Insets(20, 0, 0, 0);
        formPanel.add(messageLabel, gbc);

        // Center the form in the main panel
        JPanel centeringPanel = new JPanel(new GridBagLayout());
        centeringPanel.setBackground(Color.WHITE);
```

```
        centeringPanel.add(formPanel);

        // Add centering panel to main panel
        mainPanel.add(centeringPanel, BorderLayout.CENTER);

        // Add main panel to frame
        add(mainPanel);

        // Action listener for the login button
        loginButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String username = usernameField.getText();
                String password = new
String(passwordField.getPassword());
                if (validateLogin(username, password)) {
                    messageLabel.setForeground(new Color(60, 179, 113));
                    messageLabel.setText("Login successful!");
                    dispose();  // Close the login page window
                    new Dashboard().setVisible(true);  // Open the
Dashboard page
                } else {
                    messageLabel.setForeground(Color.RED);
                    messageLabel.setText("Invalid credentials.");
                }
            }
```

```
        });

        // Action listener for the register button
        registerButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                new Register();  // Open Register page
            }
        });
    }

    // Method to validate login credentials against the database
    private boolean validateLogin(String username, String password) {
        String url =
"jdbc:mysql://localhost:3306/unisoft?useSSL=false&allowPublicKey
Retrieval=true&serverTimezone=UTC";
        String user = "root";
        String pass = "siva2005";

        try (Connection con = DriverManager.getConnection(url, user,
pass)) {
            String query = "SELECT * FROM users WHERE username
= ? AND password = ?";
            PreparedStatement pst = con.prepareStatement(query);
            pst.setString(1, username);
            pst.setString(2, password);
```

```
            ResultSet rs = pst.executeQuery();
            return rs.next();  // Returns true if the username and password
are found in the database
        } catch (SQLException e) {
            e.printStackTrace();
            messageLabel.setText("Database error.");
            return false;
        }
    }

    // Main method to launch the login page
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                try {
                    // Set system look and feel

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelCl
assName());
                } catch (Exception e) {
                    e.printStackTrace();
                }
                Loginpage loginPage = new Loginpage();
                loginPage.setVisible(true);  // Show login page
            }
        });
```

```
    }
}
```

Register :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class Register extends JFrame {
    private JTextField usernameField;
    private JPasswordField passwordField;
    private JButton registerButton;
    private JLabel messageLabel;

    public Register() {
        // Setting up the frame
        setTitle("Register Page");
        setSize(300, 200);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new GridLayout(4, 2, 5, 5));
```

```
        // Creating components
        JLabel usernameLabel = new JLabel("Username:");
        usernameField = new JTextField();

        JLabel passwordLabel = new JLabel("Password:");
        passwordField = new JPasswordField();

        registerButton = new JButton("Register");
        messageLabel = new JLabel("", SwingConstants.CENTER);

        // Adding components to frame
        add(usernameLabel);
        add(usernameField);
        add(passwordLabel);
        add(passwordField);
        add(new JLabel());  // Empty cell
        add(registerButton);
        add(messageLabel);

        // Action listener for register button
        registerButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String username = usernameField.getText();
                String password = new
String(passwordField.getPassword());
```

```
            if (registerUser(username, password)) {
                messageLabel.setText("Registration successful!");
            } else {
                messageLabel.setText("Registration failed.");
            }
        }
    });
}

private boolean registerUser(String username, String password) {
    String url =
"jdbc:mysql://localhost:3306/unisoft?useSSL=false&allowPublicKey
Retrieval=true&serverTimezone=UTC";
    String user = "root";
    String pass = "siva2005";

    try (Connection con = DriverManager.getConnection(url, user,
pass)) {
        String query = "INSERT INTO users (username, password)
VALUES (?, ?)";
        PreparedStatement pst = con.prepareStatement(query);
        pst.setString(1, username);
        pst.setString(2, password);
        pst.executeUpdate();
        return true;
    } catch (SQLException e) {
        e.printStackTrace();
```

```
            messageLabel.setText("Database error.");
            return false;
        }
    }
}
```

Dashboard :

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class Dashboard extends JFrame {
    private JButton addReportButton;
    private JButton viewReportsButton;
    private JButton removeReportsButton;
    private JButton logoutButton;
    private ImageIcon backgroundImage; // Store the image as
ImageIcon

    public Dashboard() {
        // Load the flood background image
        backgroundImage = new ImageIcon("flood_background.jpg");
// Replace with your image path

        // Setting up the frame
        setTitle("Disaster Management Dashboard");
```

```
        setExtendedState(JFrame.MAXIMIZED_BOTH); // Set
window to full screen
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create main panel with image background
        JPanel mainPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                // Draw the background image scaled to the panel's size
                if (backgroundImage != null) {
                    Image img = backgroundImage.getImage(); // Convert
to Image
                    g.drawImage(img, 0, 0, getWidth(), getHeight(), this); //
Scale the image to fit the screen
                }
            }
        };
        mainPanel.setLayout(new BorderLayout());

        // Header Panel
        JPanel headerPanel = new JPanel(new BorderLayout());
        headerPanel.setOpaque(false);
        headerPanel.setBorder(BorderFactory.createEmptyBorder(30,
50, 30, 50));
```

```
        JLabel titleLabel = new JLabel("Disaster Management System", SwingConstants.CENTER);
        titleLabel.setFont(new Font("Arial", Font.BOLD, 40));
        titleLabel.setForeground(Color.black); // Adjust to ensure visibility on the background image
        headerPanel.add(titleLabel, BorderLayout.CENTER);

        // Buttons Panel
        JPanel buttonsPanel = new JPanel(new GridBagLayout());
        buttonsPanel.setOpaque(false);
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(20, 20, 20, 20);

        // Create and style buttons
        Dimension buttonSize = new Dimension(300, 80);
        Font buttonFont = new Font("Arial", Font.BOLD, 20);
        Color buttonBlue = new Color(0, 120, 215);

        addReportButton = createStyledButton("Add Disaster Report", buttonSize, buttonFont, buttonBlue);
        viewReportsButton = createStyledButton("View Disaster Reports", buttonSize, buttonFont, buttonBlue);
        removeReportsButton = createStyledButton("Remove Disaster Reports", buttonSize, buttonFont, buttonBlue);
        logoutButton = createStyledButton("Logout", new Dimension(200, 60), buttonFont, new Color(220, 53, 69));
```

```
        // Add buttons to panel
        gbc.gridx = 0;
        gbc.gridy = 0;
        buttonsPanel.add(addReportButton, gbc);

        gbc.gridy = 1;
        buttonsPanel.add(viewReportsButton, gbc);

        gbc.gridy = 2;
        buttonsPanel.add(removeReportsButton, gbc);

        gbc.gridy = 3;
        gbc.insets = new Insets(50, 20, 20, 20); // Extra space above
logout
        buttonsPanel.add(logoutButton, gbc);

        // Add panels to main panel
        mainPanel.add(headerPanel, BorderLayout.NORTH);
        mainPanel.add(buttonsPanel, BorderLayout.CENTER);

        // Add main panel to frame
        add(mainPanel);

        // Action listeners
        addReportButton.addActionListener(new ActionListener() {
            @Override
```

```
        public void actionPerformed(ActionEvent e) {
            new AddReportPage().setVisible(true); // Placeholder class for adding reports
        }
    });

    viewReportsButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            new ViewDisasterPage().setVisible(true); // Placeholder class for viewing disaster reports
        }
    });

    removeReportsButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            new RemoveDisasterPage().setVisible(true); // Placeholder class for removing disaster reports
        }
    });

    logoutButton.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            dispose();
```

```
            new Loginpage().setVisible(true); // Placeholder class for
the login page
        }
    });

    // Add hover effects to buttons
    addHoverEffect(addReportButton);
    addHoverEffect(viewReportsButton);
    addHoverEffect(removeReportsButton);
    addHoverEffect(logoutButton);
}

private JButton createStyledButton(String text, Dimension size,
Font font, Color backgroundColor) {
    JButton button = new JButton(text);
    button.setPreferredSize(size);
    button.setFont(font);
    button.setBackground(backgroundColor);
    button.setForeground(Color.WHITE);
    button.setFocusPainted(false);
    button.setBorderPainted(false);
    button.setOpaque(true);
    return button;
}

private void addHoverEffect(JButton button) {
```

```
        Color originalColor = button.getBackground();
        Color darkerColor = darkenColor(originalColor);

        button.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                button.setBackground(darkerColor);
            }

            public void mouseExited(java.awt.event.MouseEvent evt) {
                button.setBackground(originalColor);
            }
        });
    }

    private Color darkenColor(Color color) {
        float[] hsb = Color.RGBtoHSB(color.getRed(), color.getGreen(),
color.getBlue(), null);
        return Color.getHSBColor(hsb[0], hsb[1], Math.max(0, hsb[2] -
0.1f));
    }

    public static void main(String[] args) {
        try {
            // Set system look and feel

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelCl
assName());
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }

        SwingUtilities.invokeLater(() -> {
            Dashboard dashboard = new Dashboard();
            dashboard.setVisible(true);
        });
    }
}
```

Addreportpage :

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;

public class AddReportPage extends JFrame {
    private JTextField disasterNameField, disasterTypeField,
locationField, severityField;
    private JButton submitButton;
    private JTable disasterTable;
    private DefaultTableModel tableModel;

    public AddReportPage() {
```

```
        setTitle("Add Disaster Report");
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setExtendedState(JFrame.MAXIMIZED_BOTH); // Full-screen mode

        // Left Panel: Form for adding disasters
        JPanel formPanel = new JPanel(new GridBagLayout());
        formPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.insets = new Insets(10, 10, 10, 10);
        gbc.anchor = GridBagConstraints.WEST;

        // Adding components to form panel
        JLabel disasterNameLabel = new JLabel("Disaster Name:");
        disasterNameLabel.setFont(new Font("Arial", Font.BOLD, 18));
        disasterNameField = new JTextField(15);

        JLabel disasterTypeLabel = new JLabel("Disaster Type:");
        disasterTypeLabel.setFont(new Font("Arial", Font.BOLD, 18));
        disasterTypeField = new JTextField(15);

        JLabel locationLabel = new JLabel("Location:");
        locationLabel.setFont(new Font("Arial", Font.BOLD, 18));
        locationField = new JTextField(15);
```

```
JLabel severityLabel = new JLabel("Severity:");
severityLabel.setFont(new Font("Arial", Font.BOLD, 18));
severityField = new JTextField(15);

submitButton = new JButton("Submit Report");
submitButton.setFont(new Font("Arial", Font.BOLD, 16));

// Positioning components
gbc.gridx = 0;
gbc.gridy = 0;
formPanel.add(disasterNameLabel, gbc);

gbc.gridx = 1;
formPanel.add(disasterNameField, gbc);

gbc.gridx = 0;
gbc.gridy = 1;
formPanel.add(disasterTypeLabel, gbc);

gbc.gridx = 1;
formPanel.add(disasterTypeField, gbc);

gbc.gridx = 0;
gbc.gridy = 2;
formPanel.add(locationLabel, gbc);
```

```
        gbc.gridx = 1;
        formPanel.add(locationField, gbc);

        gbc.gridx = 0;
        gbc.gridy = 3;
        formPanel.add(severityLabel, gbc);

        gbc.gridx = 1;
        formPanel.add(severityField, gbc);

        gbc.gridx = 1;
        gbc.gridy = 4;
        gbc.anchor = GridBagConstraints.CENTER;
        formPanel.add(submitButton, gbc);

        // Right Panel: Table for displaying disasters
        JPanel tablePanel = new JPanel(new BorderLayout());
        JLabel tableLabel = new JLabel("Disaster Reports",
JLabel.CENTER);
        tableLabel.setFont(new Font("Arial", Font.BOLD, 20));
        tablePanel.add(tableLabel, BorderLayout.NORTH);

        tableModel = new DefaultTableModel(new String[]{"Disaster
Name", "Disaster Type", "Location", "Severity", "Report Date"}, 0);
        disasterTable = new JTable(tableModel);
        JScrollPane scrollPane = new JScrollPane(disasterTable);
```

```
        tablePanel.add(scrollPane, BorderLayout.CENTER);

        // Load disaster list
        loadDisasterList();

        // JSplitPane for dividing the window
        JSplitPane splitPane = new
JSplitPane(JSplitPane.HORIZONTAL_SPLIT, formPanel,
tablePanel);
        splitPane.setDividerLocation(500); // Initial position of the
divider
        splitPane.setResizeWeight(0.5); // Equal distribution of space
initially

        // Add split pane to frame
        add(splitPane);

        // Action listener for submit button
        submitButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                addDisasterReport();
            }
        });
    }

    private void addDisasterReport() {
```

```
        String disasterName = disasterNameField.getText();
        String disasterType = disasterTypeField.getText();
        String location = locationField.getText();
        String severity = severityField.getText();


        // Database connection details
        String url =
"jdbc:mysql://localhost:3306/unisoft?useSSL=false&allowPublicKey
Retrieval=true&serverTimezone=UTC";
        String user = "root";  // Change to your MySQL username
        String pass = "siva2005";  // Change to your MySQL password


        try (Connection con = DriverManager.getConnection(url, user,
pass)) {
            // Insert SQL query for disaster reports
            String query = "INSERT INTO disaster_reports
(disaster_name, disaster_type, location, severity, report_date)
VALUES (?, ?, ?, ?, ?)";
            PreparedStatement pst = con.prepareStatement(query);
            pst.setString(1, disasterName);
            pst.setString(2, disasterType);
            pst.setString(3, location);
            pst.setString(4, severity);
            pst.setDate(5, new Date(System.currentTimeMillis())); //
Current date for report_date


            // Execute the query
```

```
            pst.executeUpdate();

            // Show success message
            JOptionPane.showMessageDialog(this, "Disaster report added
successfully!");

            // Reload disaster list
            loadDisasterList();

            // Clear input fields
            disasterNameField.setText("");
            disasterTypeField.setText("");
            locationField.setText("");
            severityField.setText("");

        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error adding report: "
+ e.getMessage());
        }
    }

    private void loadDisasterList() {
        // Database connection details
        String url =
"jdbc:mysql://localhost:3306/unisoft?useSSL=false&allowPublicKey
Retrieval=true&serverTimezone=UTC";
```

```
        String user = "root";  // Change to your MySQL username
        String pass = "siva2005";  // Change to your MySQL password

        try (Connection con = DriverManager.getConnection(url, user,
pass)) {
            // Query to fetch disaster reports
            String query = "SELECT disaster_name, disaster_type,
location, severity, report_date FROM disaster_reports";
            PreparedStatement pst = con.prepareStatement(query);
            ResultSet rs = pst.executeQuery();

            // Clear existing rows in the table
            tableModel.setRowCount(0);

            // Populate table with data from the database
            while (rs.next()) {
                tableModel.addRow(new Object[]{
                        rs.getString("disaster_name"),
                        rs.getString("disaster_type"),
                        rs.getString("location"),
                        rs.getString("severity"),
                        rs.getDate("report_date")
                });
            }
        } catch (SQLException e) {
            e.printStackTrace();
```

```
            JOptionPane.showMessageDialog(this, "Error loading disaster
list: " + e.getMessage());
        }
    }

    // Main method for testing the AddReportPage form
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new AddReportPage().setVisible(true);  // Show
AddReportPage
            }
        });
    }
}
```

Removedisaterpage :

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableColumn;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;

public class RemoveDisasterPage extends JFrame {
```

```
    private JTable disasterTable;
    private DefaultTableModel tableModel;
    private JButton removeButton;

    public RemoveDisasterPage() {
        setTitle("Remove Disaster Reports");
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setExtendedState(JFrame.MAXIMIZED_BOTH); // Full-
screen mode

        JPanel mainPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                Graphics2D g2d = (Graphics2D) g;
                Color color1 = new Color(240, 248, 255); // AliceBlue
                Color color2 = new Color(230, 230, 250); // Lavender
                GradientPaint gp = new GradientPaint(0, 0, color1, 0,
getHeight(), color2);
                g2d.setPaint(gp);
                g2d.fillRect(0, 0, getWidth(), getHeight());
            }
        };
        mainPanel.setLayout(new BorderLayout());

        // Header label
```

```
        JLabel headerLabel = new JLabel("Disaster Reports",
SwingConstants.CENTER);
        headerLabel.setFont(new Font("Arial", Font.BOLD, 36)); //
Make the header large
        headerLabel.setForeground(new Color(0, 0, 128)); // Dark blue
color for the header
        headerLabel.setBorder(BorderFactory.createEmptyBorder(30,
10, 30, 10)); // Add padding around the header
        mainPanel.add(headerLabel, BorderLayout.NORTH);


        // Set up table with larger fonts and row height
        tableModel = new DefaultTableModel(new Object[]{"ID",
"Disaster Name", "Type", "Location", "Severity", "Date"}, 10);
        disasterTable = new JTable(tableModel);
        disasterTable.setFont(new Font("Arial", Font.PLAIN, 18)); //
Larger font for the table content
        disasterTable.setRowHeight(40); // Increase row height


        // Set custom column width
        TableColumn column =
disasterTable.getColumnModel().getColumn(0); // ID column
        column.setPreferredWidth(100); // Increase width for ID
column
        column = disasterTable.getColumnModel().getColumn(1); //
Disaster Name column
        column.setPreferredWidth(200); // Increase width for Disaster
Name column
        column = disasterTable.getColumnModel().getColumn(2); //
Disaster Type column
```

```
        column.setPreferredWidth(150); // Adjust Disaster Type column width
        column = disasterTable.getColumnModel().getColumn(3); // Location column
        column.setPreferredWidth(150); // Adjust Location column width
        column = disasterTable.getColumnModel().getColumn(4); // Severity column
        column.setPreferredWidth(100); // Adjust Severity column width
        column = disasterTable.getColumnModel().getColumn(5); // Date column
        column.setPreferredWidth(120); // Adjust Date column width

        JScrollPane scrollPane = new JScrollPane(disasterTable);

        // Set up Remove button with larger size and increased font
        removeButton = new JButton("Remove Selected Disaster");
        removeButton.setFont(new Font("Arial", Font.BOLD, 24)); // Larger font for the button
        removeButton.setPreferredSize(new Dimension(400, 60)); // Larger button
        removeButton.setEnabled(false); // Initially disabled until a row is selected

        // Enable button when a row is selected
        disasterTable.getSelectionModel().addListSelectionListener(e -> {
```

```
            removeButton.setEnabled(disasterTable.getSelectedRow() != -
1);
        });

        // Button panel
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(removeButton);

        // Add components to frame
        setLayout(new BorderLayout());
        add(mainPanel, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.CENTER);
        add(buttonPanel, BorderLayout.SOUTH);

        // Load data from the database
        loadReports();

        // Action listener for the remove button
        removeButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                removeSelectedDisaster();
            }
        });
    }
```

```
    private void loadReports() {
        String url =
"jdbc:mysql://localhost:3306/unisoft?useSSL=false&allowPublicKey
Retrieval=true&serverTimezone=UTC";
        String user = "root"; // MySQL username
        String pass = "siva2005"; // MySQL password


        try (Connection con = DriverManager.getConnection(url, user,
pass)) {
            String query = "SELECT * FROM disaster_reports"; //
Query to fetch all reports
            PreparedStatement pst = con.prepareStatement(query);
            ResultSet rs = pst.executeQuery();

            // Clear previous data in the table
            tableModel.setRowCount(0);

            // Iterate through the result set and add each row to the table
            while (rs.next()) {
                int id = rs.getInt("id");
                String disasterName = rs.getString("disaster_name");
                String disasterType = rs.getString("disaster_type");
                String location = rs.getString("location");
                String severity = rs.getString("severity");
                Date reportDate = rs.getDate("report_date");

                // Add row to table model
```

```
                tableModel.addRow(new Object[]{id, disasterName,
disasterType, location, severity, reportDate});
            }


        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error loading reports:
" + e.getMessage());
        }
    }


    private void removeSelectedDisaster() {
        int selectedRow = disasterTable.getSelectedRow();
        if (selectedRow == -1) {
            JOptionPane.showMessageDialog(this, "No disaster
selected!");
            return;
        }


        int id = (int) tableModel.getValueAt(selectedRow, 0); // Get ID
from the selected row


        String url =
"jdbc:mysql://localhost:3306/unisoft?useSSL=false&allowPublicKey
Retrieval=true&serverTimezone=UTC";
        String user = "root"; // MySQL username
        String pass = "siva2005"; // MySQL password
```

```
        try (Connection con = DriverManager.getConnection(url, user,
pass)) {
            String query = "DELETE FROM disaster_reports WHERE id
= ?"; // Query to delete the selected report
            PreparedStatement pst = con.prepareStatement(query);
            pst.setInt(1, id);


            // Execute deletion
            int rowsAffected = pst.executeUpdate();
            if (rowsAffected > 0) {
                // Remove the row from the table model
                tableModel.removeRow(selectedRow);
                JOptionPane.showMessageDialog(this, "Disaster report
removed successfully!");
            } else {
                JOptionPane.showMessageDialog(this, "Failed to remove
the disaster report!");
            }


        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error removing
report: " + e.getMessage());
        }
    }
```

```
    // Main method for testing the RemoveDisasterPage form
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new RemoveDisasterPage().setVisible(true); // Show
RemoveDisasterPage
            }
        });
    }
}
```

View disater page :

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.sql.*;

public class ViewDisasterPage extends JFrame {
    private JTable disasterTable;
    private DefaultTableModel tableModel;

    public ViewDisasterPage() {
        // Set up the frame
        setTitle("View Disaster Reports");
        setExtendedState(JFrame.MAXIMIZED_BOTH); // Full-
screen mode
```

```
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        // Main panel with gradient background
        JPanel mainPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                Graphics2D g2d = (Graphics2D) g;
                Color color1 = new Color(240, 248, 255); // AliceBlue
                Color color2 = new Color(230, 230, 250); // Lavender
                GradientPaint gp = new GradientPaint(0, 0, color1, 0,
getHeight(), color2);
                g2d.setPaint(gp);
                g2d.fillRect(0, 0, getWidth(), getHeight());
            }
        };
        mainPanel.setLayout(new BorderLayout());

        // Header label
        JLabel headerLabel = new JLabel("Disaster Reports",
SwingConstants.CENTER);
        headerLabel.setFont(new Font("Arial", Font.BOLD, 36));
        headerLabel.setBorder(BorderFactory.createEmptyBorder(20,
10, 20, 10));
        mainPanel.add(headerLabel, BorderLayout.NORTH);

        // Table setup
```

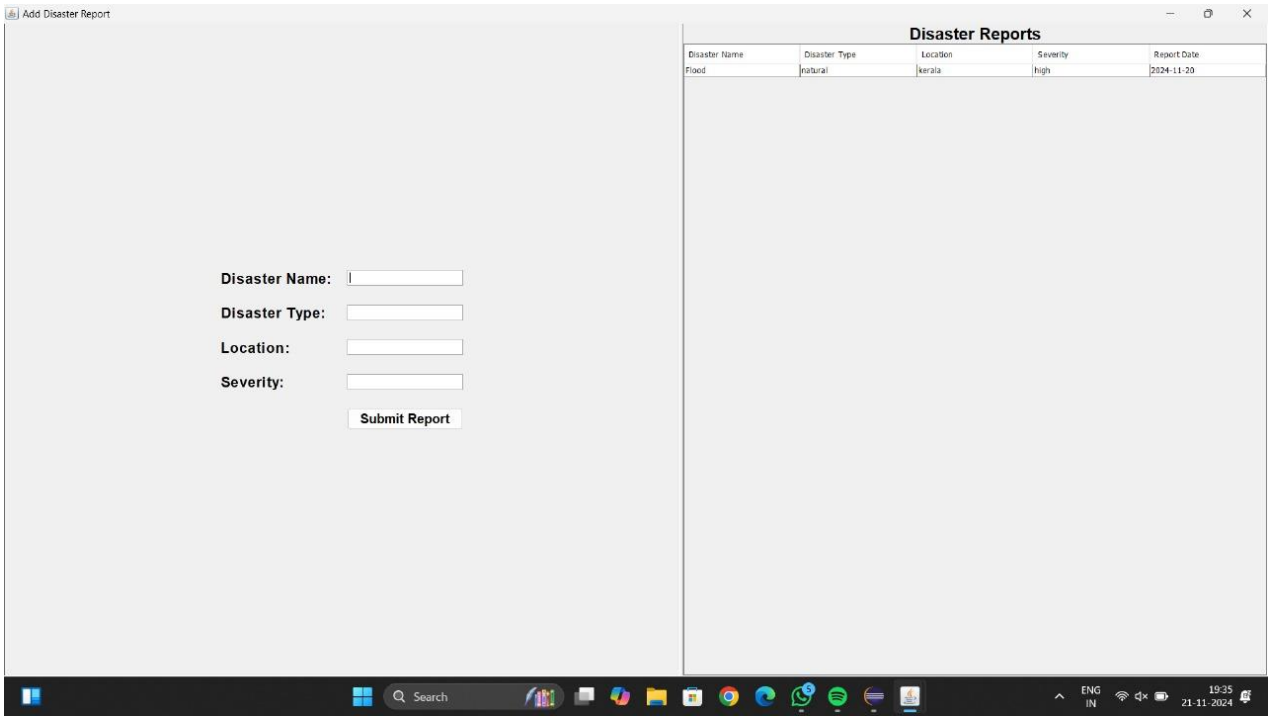
```
        tableModel = new DefaultTableModel(new Object[]{"ID",
"Disaster Name", "Type", "Location", "Severity", "Date"}, 0);
        disasterTable = new JTable(tableModel);
        disasterTable.setFont(new Font("Arial", Font.PLAIN, 16));
        disasterTable.setRowHeight(25);
        JScrollPane scrollPane = new JScrollPane(disasterTable);

        // Add table to center
        mainPanel.add(scrollPane, BorderLayout.CENTER);

        // Load disaster data
        loadDisasters();

        // Add main panel to frame
        add(mainPanel);
    }

    private void loadDisasters() {
        String url =
"jdbc:mysql://localhost:3306/unisoft?useSSL=false&allowPublicKey
Retrieval=true&serverTimezone=UTC";
        String user = "root"; // MySQL username
        String pass = "siva2005"; // MySQL password

        try (Connection con = DriverManager.getConnection(url, user,
pass)) {
            String query = "SELECT * FROM disaster_reports";
```

```
            PreparedStatement pst = con.prepareStatement(query);
            ResultSet rs = pst.executeQuery();

            // Clear any existing data
            tableModel.setRowCount(0);

            // Populate the table with data
            while (rs.next()) {
                int id = rs.getInt("id");
                String disasterName = rs.getString("disaster_name");
                String disasterType = rs.getString("disaster_type");
                String location = rs.getString("location");
                String severity = rs.getString("severity");
                Date date = rs.getDate("report_date");

                tableModel.addRow(new Object[]{id, disasterName,
disasterType, location, severity, date});
            }

        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error loading disaster
reports: " + e.getMessage());
        }
    }
```

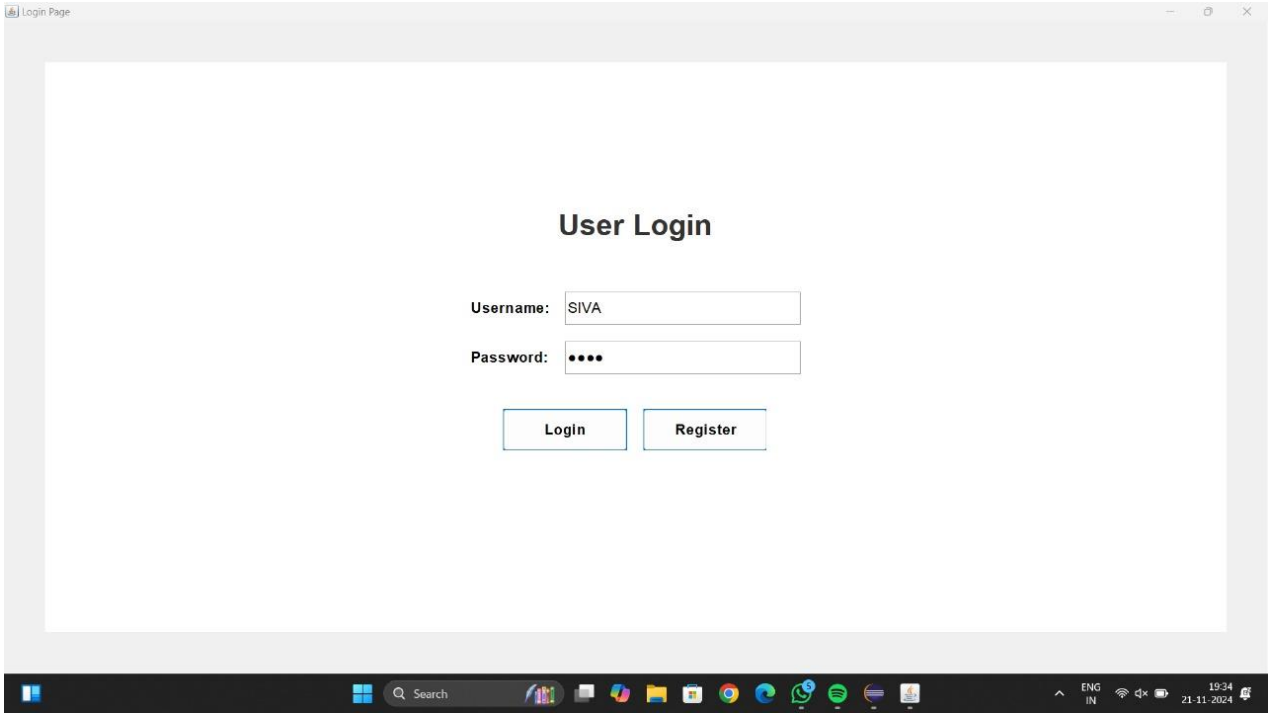
```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            ViewDisasterPage viewDisasterPage = new
ViewDisasterPage();
            viewDisasterPage.setVisible(true);
        });
    }
}
```

OUTPUT:

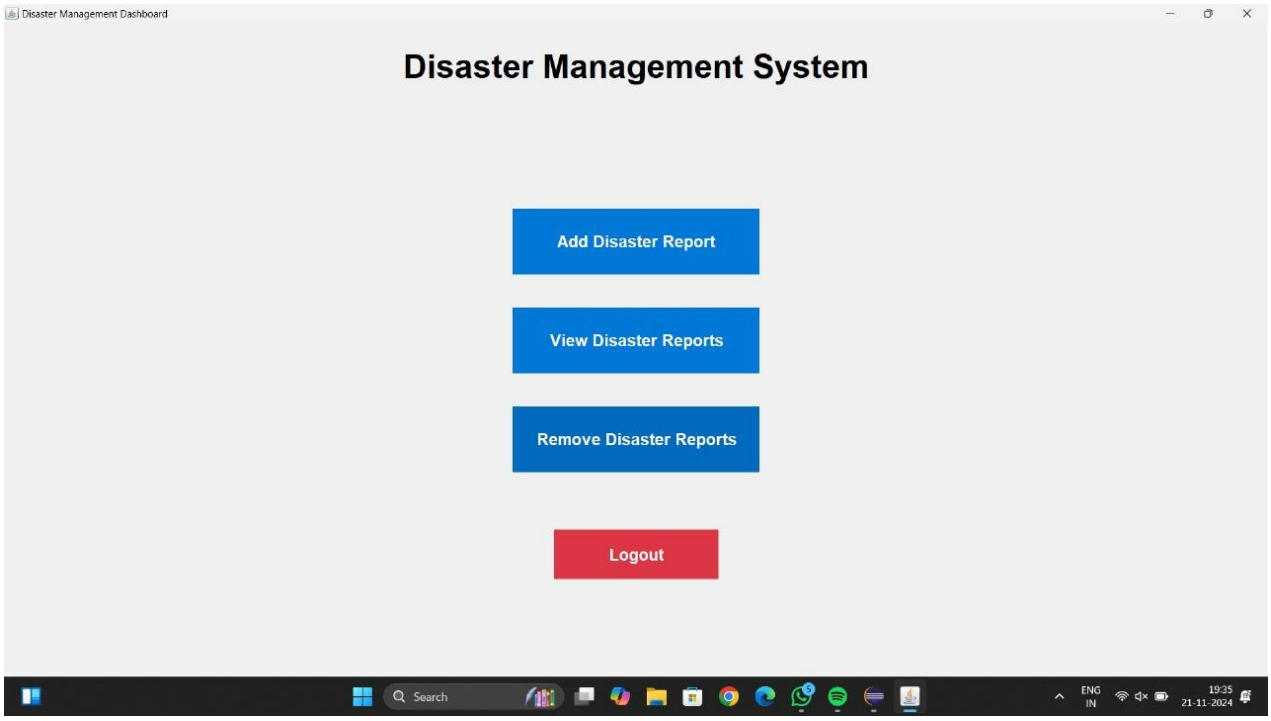
1)Login Page



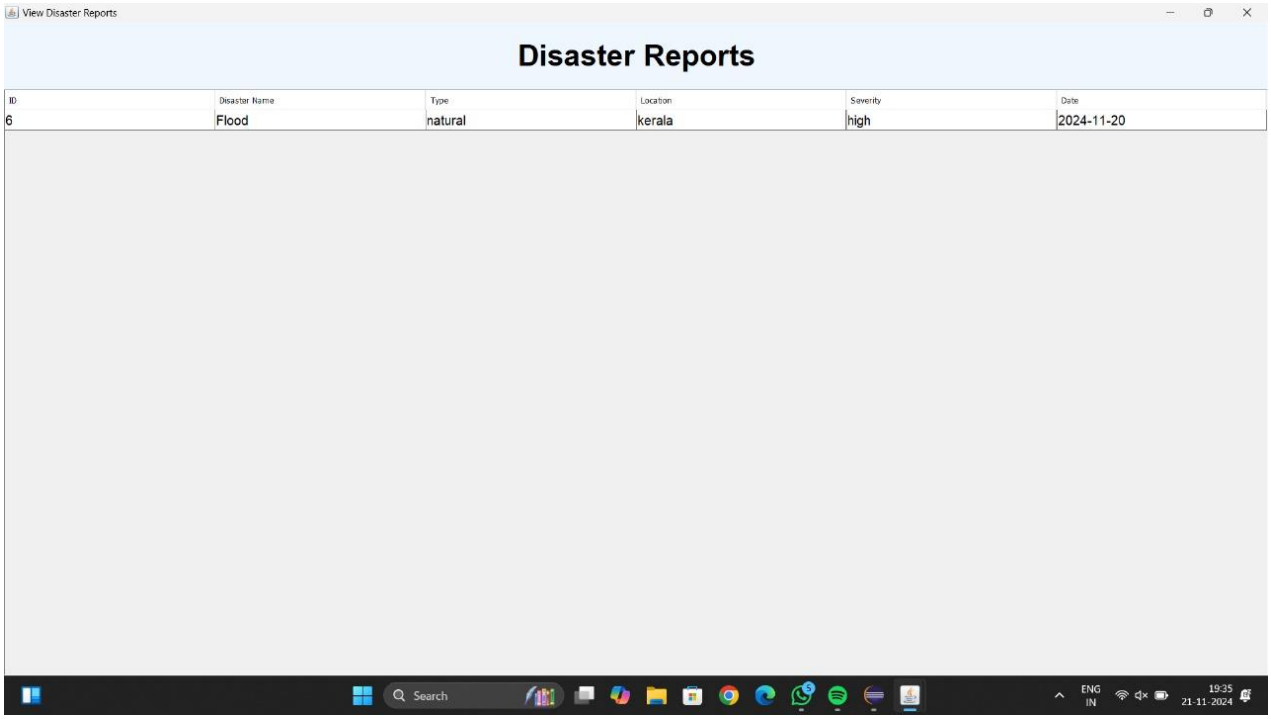
2)VIEW View



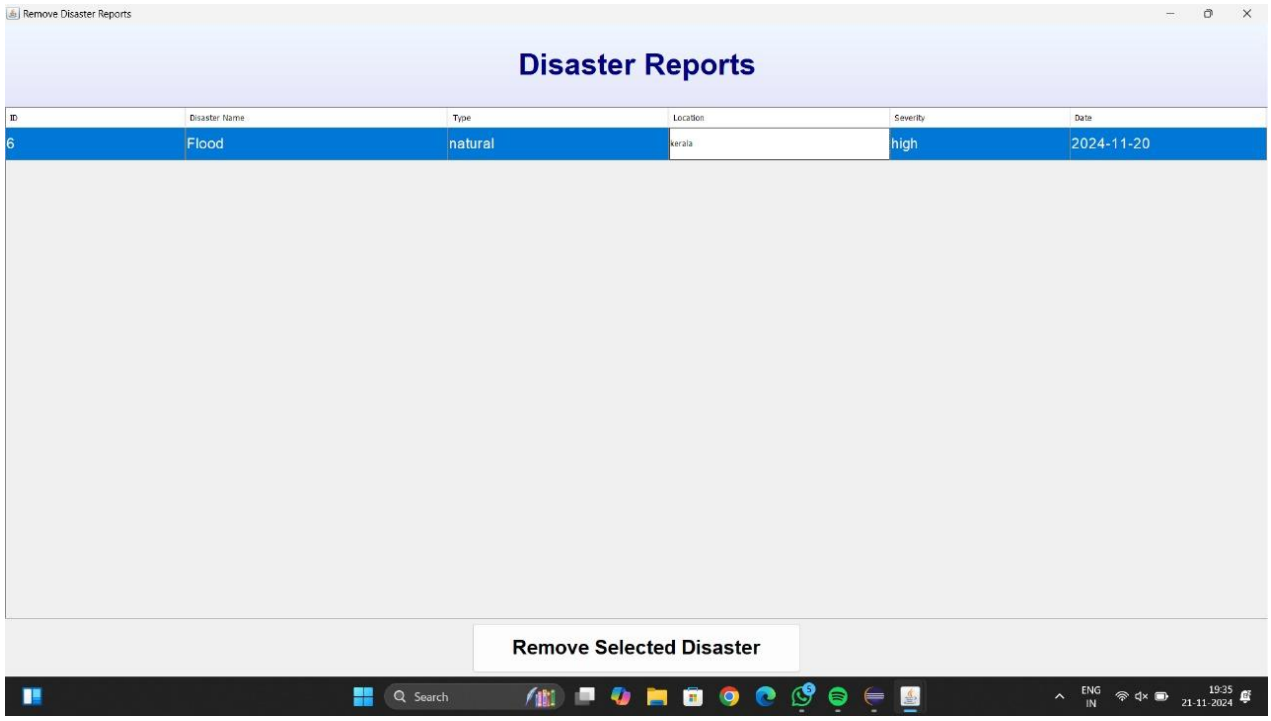
3)Adding Record



4)Viewing the record being added



5.FINAL



Conclusion:

In conclusion, the Disaster Management System developed as part of this Java mini-project demonstrates the potential of technology in mitigating the effects of natural and man-made disasters. By integrating features such as real-time alerts, resource tracking, victim assistance, and efficient coordination, the system empowers users to respond effectively to emergencies.

Through this project, key programming concepts like objectoriented design, database management, and user interface development were applied, showcasing how Java can be leveraged to build scalable and practical solutions. Furthermore, the project highlights the importance of proactive disaster management and emphasizes the role of technology in saving lives and minimizing damage.

Future improvements could include incorporating machine learning for predictive analytics, cloud integration for scalability, and mobile application support for wider accessibility. Overall, this mini-project not only serves as a proof-of-concept but also as a stepping stone for developing robust disaster management tools for real-world applications.