Disaster Preparedness Companion

BACHELOR OF

TECHNOLOGY IN

COMPUTER SCIENCE AND ENGINEERING

BY

V. Venkata Ganesh (23505A0516)

T. Hema Harshitha (22501A05I1)

T.Sai Ritish (23505A0514)

V. Nitish Kumar (23505A0518)

Under the Guidance of

Mr. Michael Sadgun Rao Kona,

Assistant Professor

PRASAD V POTLURI SIDDHARTHA INSTITUTE OF TECHNOLOGY

(Permanently affiliated to JNTU: Kakinada, Approved by AICTE)

(An NBA & NAAC A+ accredited and ISO 9001:2015 certified institution)

Kanuru, Vijayawada-520007

2024-25

PRASAD V POTLURI

SIDDHARTHA INSTITUTE OF TECHNOLOGY

(Permanently affiliated to JNTU :: Kakinada, Approved by AICTE)

(An NBA & NAAC A+ accredited and ISO 9001:2015 certified institution)

Kanuru, Vijayawada – 520007

CERTIFICATE

This is to certify that the project report title **“Disaster Preparedness Companion”** is the bonafide work of **V. Venkata Ganesh (23505A0516), T. Hema Harshitha (22501A05I1), T. Sai Ritish (23505A0514), V. Nitish Kumar (23505A0518)** in partial fulfilment of completing the Academic project in Mobile App Development (20SA8651) during the academic year 2024-25.

Signature of the Incharge **Signature of the HOD**

INDEX

1.ABSTRACT

The Disaster Preparedness Companion is a mobile application designed to provide real-time alerts, evacuation routes, and emergency contact information during natural disasters using reliable API sources. The app features a clean and interactive user interface built with Android Studio and Java, allowing users to access disaster-related information based on their current location. Key details such as disaster type, severity, risk areas, weather data, and emergency contacts are displayed in a structured format.

The app integrates Retrofit, a popular HTTP client for API requests, ensuring fast and reliable data retrieval. A navigation drawer enhances usability, enabling users to access different sections such as Emergency Contacts, Safety Tips, Current Location Data, and Settings.

To improve user experience, the app features real-time alerts, interactive emergency maps, and error handling for invalid location inputs. The design follows modern UI/UX principles, ensuring responsiveness across different screen sizes. Future improvements may include AI-driven risk assessments, offline emergency guides, multilingual support, and push notifications for disaster warnings.

This app serves as a reliable and user-friendly disaster preparedness tool, helping users stay informed and safe during emergencies.

2.SDG Justification

The **Sustainable Development Goals (SDGs)** offer a framework for addressing global challenges. Below is a justification of how a **Weather App** can align with several SDGs:

1.SDG 3: Good Health and Well-being
The app provides real-time weather data, helping users prepare for extreme conditions, reducing health risks like heat strokes and respiratory issues.

2.SDG 11: Sustainable Cities and Communities
By offering weather data, the app aids communities in responding to extreme events like floods and storms, supporting sustainable urban planning.

3.SDG 13: Climate Action
The app raises awareness of climate change by providing real-time climate updates, empowering users to take climate-resilient actions.

4.SDG 9: Industry, Innovation, and Infrastructure
Weather Apps foster innovation, providing accurate data to industries like agriculture, construction, and transportation for better planning.

5.SDG 2: Zero Hunger
Weather data, such as rainfall and temperature forecasts, supports farmers in crop planning and reducing crop failure risks.

6.SDG7:Affordable and Clean Energy
The app helps users optimize energy use by providing weather forecasts, promoting solar energy use and reducing reliance on non-renewable sources.

3. INTRODUCTION

Weather conditions significantly impact our daily routines, travel, agriculture, and overall lifestyle. Access to real-time weather updates is crucial for planning and decision-making. The Disaster Preparedness Companion is developed to provide users with instant and accurate weather information, along with disaster-related alerts, using reliable API sources. This Android-based mobile application fetches live weather data, including temperature, humidity, longitude, latitude, and weather conditions, enhancing user convenience during emergencies.

The app is built using Android Studio and Java, leveraging Retrofit for efficient API communication. When a user enters a location or enables GPS tracking, the app fetches real-time weather data and disaster-related updates, displaying them in an easy-to-read format. A navigation drawer enhances user experience, offering quick access to key sections such as Emergency Contacts, Safety Tips, Current Location Data, and Settings.

The application follows a modular and scalable design, allowing smooth performance even with continuous API requests. The UI is simple yet interactive, ensuring that users can effortlessly obtain weather updates and disaster alerts. Additionally, features such as emergency contact management enable users to quickly access essential resources during crises.

Future improvements for the Disaster Preparedness Companion include extended weather forecasts, AI-driven risk assessments, and GPS-based location tracking for automatic disaster updates. Furthermore, features like push notifications for severe weather conditions and emergency alerts will be integrated to provide users with timely warnings.

This Disaster Preparedness Companion is an efficient, lightweight, and user-friendly solution for obtaining instant weather updates and disaster alerts. It is designed to serve individuals who need quick access to emergency information while ensuring accuracy, reliability, and ease of use. As technology advances, further enhancements will make the app more intelligent and responsive, catering to the evolving needs of users worldwide.

4. OBJECTIVES AND SCOPE OF THE PROJECT

Objectives:

The primary goal of this project is to develop a mobile-based Disaster Preparedness Companion that provides users with real-time weather updates, disaster alerts, and emergency preparedness resources.

The key objectives of the project are:

1. **Real-time Weather & Disaster Alerts** – Implement reliable API sources to display temperature, humidity, weather conditions, and disaster-related warnings.
2. **Current Location-Based** Tracking – Enable GPS tracking to provide location-specific disaster and weather information.
3. **Emergency Contact Management** – Allow users to access and update emergency contact details quickly.
4. **Risk Area Identification** – Highlight high-risk areas prone to disasters based on real-time data.
5. **Seamless Navigation** – Integrate a navigation drawer to access Home, Emergency Contacts, Safety Tips, and Settings.
6. **Efficient API Communication** – Implement Retrofit library for fast and optimized API calls.
7. **User-friendly UI** – Ensure a responsive and visually appealing interface for easy usability.
8. **Performance Optimization** – Reduce loading times and optimize data usage for smooth operation.
9. **Future Enhancements** – Provide scope for additional features like AI-driven risk assessments, offline emergency guides, multilingual support, and push notifications for disaster warnings.

Scope of the Project:

This project is focused on building a functional and user-friendly Android-based Disaster Preparedness Companion that will benefit different users:

1. **General Users** – Individuals who need quick access to real-time weather and disaster alerts.
2. **Travelers** – People who want to check weather conditions and potential disaster risks before traveling.
3. **Emergency Responders** – First responders and authorities who require real-time disaster updates and emergency contacts.
4. **Students & Researchers** – Users who need climate and disaster data for studies and projects.
5. **Outdoor Enthusiasts** – Hikers, cyclists, and athletes who rely on weather forecasts and disaster alerts.

The app ensures real-time disaster tracking, emergency preparedness, and a user-friendly experience. With future scalability options, it is designed to be efficient, reliable, and easy to use, making it an essential tool for disaster awareness and preparedness

5. SOFTWARE USED

In the development of the WeatherApp, the following software technologies were utilized to ensure a responsive, efficient, and user-friendly experience:

Frontend Technologies • XML (Extensible Markup Language)

XML was used to design the user interface (UI) for the Android application. It defined the layout of various UI elements such as text fields, buttons, navigation menus, and weather icons, ensuring a structured and interactive design.

• Java

Java was used as the primary programming language to implement the app's logic and functionality.It ensured smooth communication between the UI components and backend services, handling user interactions and API calls efficiently.

• Android Studio

Android Studio served as the main IDE for developing and testing the WeatherApp. It provided various debugging tools, emulators, and performance optimization features, ensuring an efficient development process.

Backend Technologies:

- OpenWeatherMap API – Used to fetch real-time weather data such as temperature, humidity, wind speed, and weather conditions based on the entered city name. It provided accurate and up-to-date weather information, enhancing the app's reliability.
- Google API & Places API – Integrated for location-based services, allowing users to access disaster-prone areas and nearby emergency services.
- SQLite (Local Database) – Used for storing user data such as login credentials and profile details. It ensured that the app functions even without an internet connection, maintaining a seamless user experience.

API Integration:

- Retrofit (REST API Library) – Used to handle API requests and parse responses efficiently. It allowed seamless communication between the frontend and OpenWeatherMap API, ensuring smooth data retrieval.
- GSON (JSON Parser) – Used for converting JSON responses from the weather API into Java objects, making data handling easier.

Design and Documentation Tools:

- Canva – Used for designing UI elements, icons, and app prototypes. It ensured a visually appealing user interface that aligns with modern app design principles.
- Microsoft Word – Used for documenting the project, writing reports, and preparing structured documentation. It ensured clarity and proper record-keeping of the project's development process.

Version Control and Deployment:

- GitHub – Used for version control, collaboration, and code management. It ensured that all changes in the app were tracked, stored, and managed efficiently, allowing future updates and bug fixes.

Software Used:

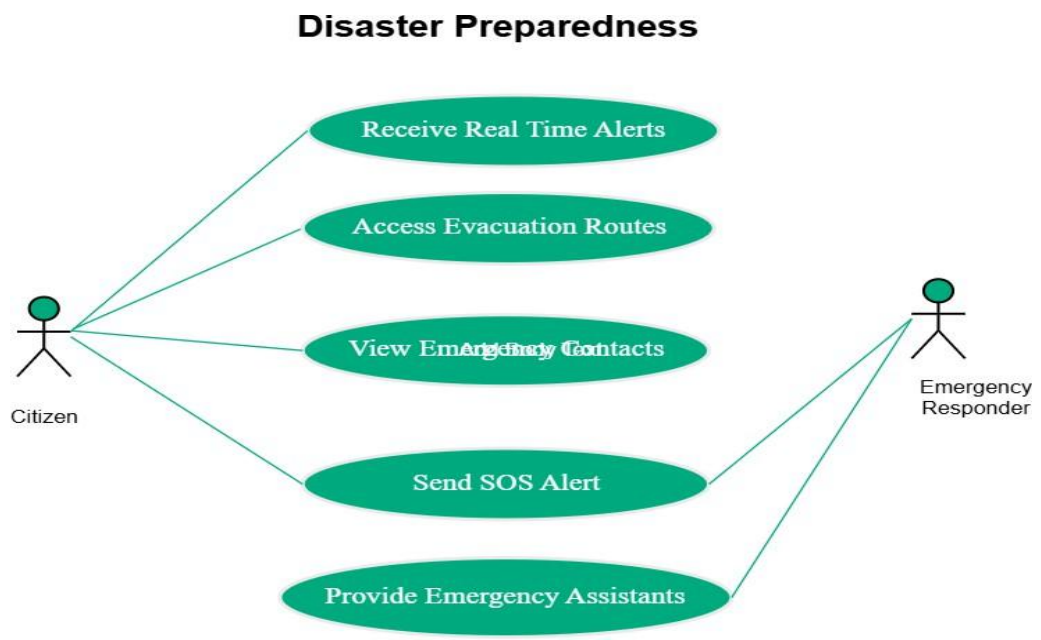
- Google API
- Weather API
- Places API
- Linear Layout

6. PROPOSED MODEL

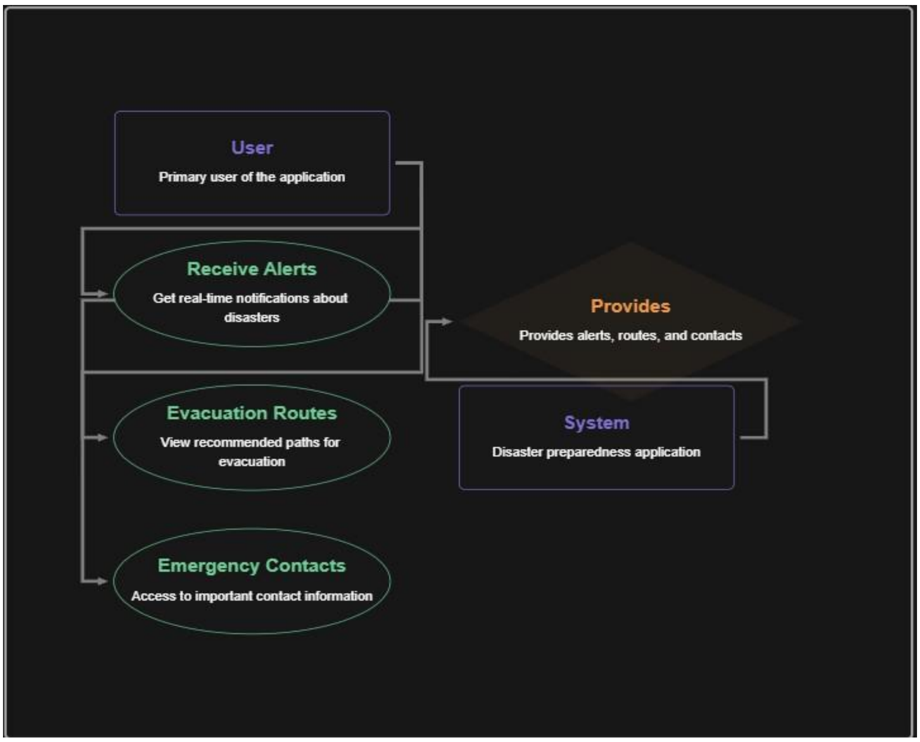
The Disaster Preparedness Companion follows a structured model designed to provide real-time disaster alerts, seamless navigation, and an enhanced user experience. This model ensures users receive accurate warnings, evacuation guidance, and emergency contacts while maintaining data security and offline accessibility.

1. **Emergency Alert System**
    - Provides real-time alerts for natural disasters such as earthquakes, floods, wildfires, and hurricanes.
    - Uses government and reliable API sources to ensure accuracy.
    - Sends push notifications for immediate awareness.
2. **Location-Based Risk Analysis**
    - Uses GPS tracking to determine disaster-prone areas.
    - Highlights evacuation routes and nearby shelters.
    - Displays localized weather forecasts and risk assessments.
3. **Emergency Contact & Resource Management**
    - Provides a list of essential emergency contacts (police, fire department, hospitals, relief centers).
    - Allows users to store and access their personal emergency contacts.
    - Offline accessibility ensures information is available without an internet connection.
4. **Navigation & Safety Resources**
    - Includes a navigation drawer for quick access to:
        - Home – Displays real-time alerts and weather updates.
        - Emergency Contacts – List of essential helplines.
        - Safety Tips – Guidelines for handling different disasters.
        - Settings – Customizable options such as notification preferences.
    - Provides step-by-step evacuation instructions during disasters.
5. **API Integration & Data Processing**
    - Uses OpenWeatherMap API for real-time weather updates.
    - Google Maps API for evacuation routes and safe zones.
    - Retrofit for efficient API communication.
    - JSON data parsing using GSON for smooth data handling.
6. **User Interface & Accessibility**
    - Designed with a user-friendly interface ensuring easy navigation.
    - Follows Material Design principles for visual appeal.
    - Supports multilingual options for accessibility.
    - Future enhancements include dark mode and voice alerts.
7. **Data Security & Offline Support**
    - Stores last fetched alerts and emergency contacts for offline access.
    - Implements encryption for secure data handling.
    - Ensures seamless performance even in low connectivity areas.
8. **Future Expansion & AI Integration**
    - AI-driven risk assessments to predict potential disasters based on historical data.
    - Smart assistant for voice-based emergency guidance.
    - Community-driven updates for real-time disaster information sharing.

Use Case Diagram:



Entity-Relationship (ER) Diagram:

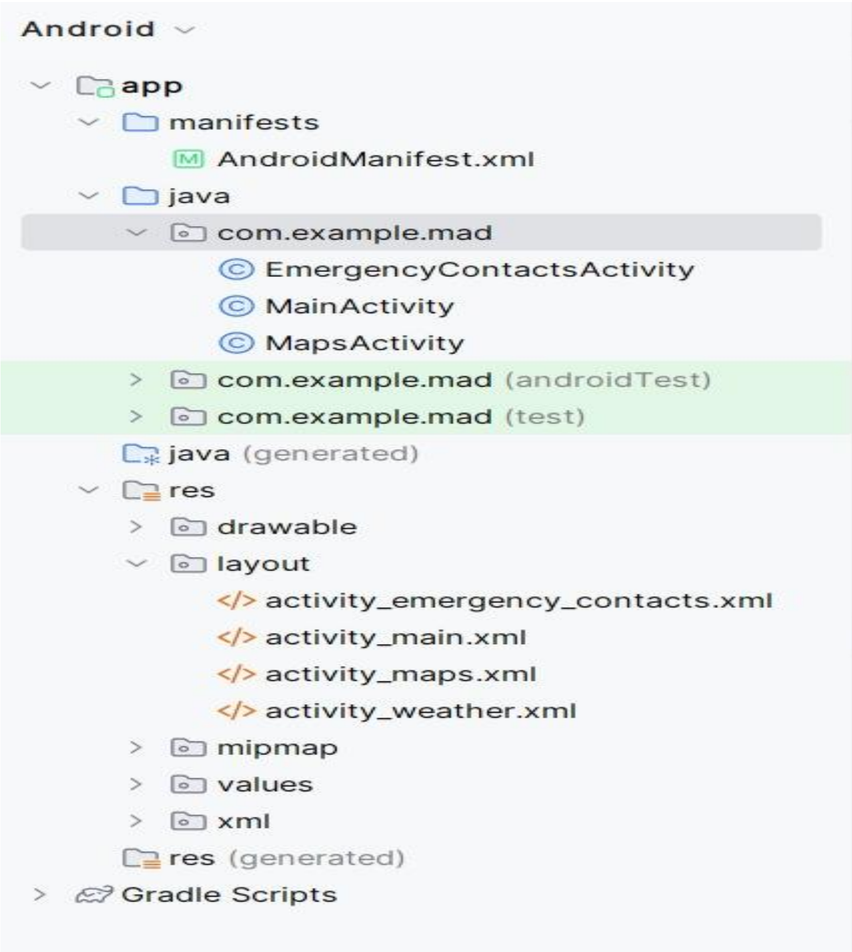


7. SAMPLE CODE

GitHub repository link:

https://github.com/SaiRitish514/Disaster-preparedness-Companion-

Folder Structure:



Program:

//MainActivity.java

```
package com.example.mad;
import android.annotation.SuppressLint;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    private Button ShowMap, btnEmergencyContacts;
```

```
    @SuppressLint("MissingInflatedId")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_main);

        // Initialize buttons
        ShowMap = findViewById(R.id.ShowMap);
        btnEmergencyContacts = findViewById(R.id.btnEmergencyContacts);

        // Show Map button click listener
        ShowMap.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, MapsActivity.class);
                startActivity(intent);
            }
        });
        // Emergency Contacts button click listener
        btnEmergencyContacts.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this, EmergencyContactsActivity.class);
                startActivity(intent);
            }
        });
    }
}
//MapsActivity
import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.os.Bundle;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import androidx.fragment.app.FragmentActivity;
import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.StringRequest;
```

```
import com.android.volley.toolbox.Volley;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
public class MapsActivity extends FragmentActivity implements OnMapReadyCallback {
   private GoogleMap mMap;
   private FusedLocationProviderClient fusedLocationClient;
   private static final String WEATHER_API_KEY = "f87f9331f1b2c100d015672ae1d4d5f8"; // Your
OpenWeatherMap API key
   private static final int LOCATION_PERMISSION_REQUEST_CODE = 1;
   private LinearLayout safePlacesLayout; // LinearLayout to display safe places addresses
   private TextView titleTextView; // TextView to display location status

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_maps);
      // Initialize fused location client
      fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);
      // Initialize the map
      SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
            .findFragmentById(R.id.map);
      if (mapFragment != null) {
         mapFragment.getMapAsync(this);
      }
      // Initialize the layout to show safe places addresses
      safePlacesLayout = findViewById(R.id.safePlacesLayout);
      // Initialize the title TextView
      titleTextView = findViewById(R.id.statusTitle);
   }
   @Override
   public void onMapReady(@NonNull GoogleMap googleMap) {
      mMap = googleMap;
      // Request location permission
      if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
            == PackageManager.PERMISSION_GRANTED) {
         getCurrentLocation();
      } else {
         ActivityCompat.requestPermissions(this,
               new String[]{Manifest.permission.ACCESS_FINE_LOCATION},
               LOCATION_PERMISSION_REQUEST_CODE);
      }
   }
   private void getCurrentLocation() {
      if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
```

```
            == PackageManager.PERMISSION_GRANTED) {

          // Enable the blue dot for current location
          mMap.setMyLocationEnabled(true);

          fusedLocationClient.getLastLocation()
                .addOnSuccessListener(this, location -> {
                  if (location != null) {
                     LatLng currentLocation = new LatLng(location.getLatitude(), location.getLongitude());
                     mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(currentLocation, 12f));

                     // Fetch weather for the current location
                     getWeather(location.getLatitude(), location.getLongitude());
                  } else {
                     Toast.makeText(MapsActivity.this, "Unable to fetch location",
Toast.LENGTH_SHORT).show();}});}}
   private void getWeather(double lat, double lon) {
      String url = "https://api.openweathermap.org/data/2.5/weather?lat=" + lat + "&lon=" + lon +
"&appid=" + WEATHER_API_KEY + "&units=metric";
      RequestQueue queue = Volley.newRequestQueue(this);
      StringRequest request = new StringRequest(Request.Method.GET, url,
            response -> {
               try {
                  JSONObject jsonResponse = new JSONObject(response);
                  String weather =
jsonResponse.getJSONArray("weather").getJSONObject(0).getString("description");
                  double temp = jsonResponse.getJSONObject("main").getDouble("temp");
                  double windSpeed = jsonResponse.getJSONObject("wind").getDouble("speed");
                  // Determine risk level
                  if (temp > 20 || temp < 0 || weather.contains("storm") || windSpeed > 20) {
                     titleTextView.setText("High-Risk Zone 🚨");
                     showHighRiskZone(lat, lon); // Show safe places if high-risk
                  } else if (weather.contains("rain") || windSpeed > 10) {
                     titleTextView.setText("Moderate Risk Zone ⚠");
                     showModerateRiskZone();
                  } else {
                     titleTextView.setText("Safe Zone 😊");
                     showSafeZone();
                  }
               } catch (JSONException e) {
                  e.printStackTrace();
                  Toast.makeText(this, "Error fetching weather data", Toast.LENGTH_SHORT).show();
               }
            },
```

```
            error -> Toast.makeText(this, "Error fetching weather data", Toast.LENGTH_SHORT).show());

        queue.add(request);
    }
    private void showHighRiskZone(double lat, double lon) {
        // Fetch and display nearby safe places dynamically based on the current location
        fetchSafePlaces(lat, lon);
    }
    private void fetchSafePlaces(double lat, double lon) {
        // 1. Fetch weather data using OpenWeatherMap API or any other weather API
        String weatherApiUrl = "https://api.openweathermap.org/data/2.5/weather?lat=" + lat + "&lon=" +
lon + "&appid=YOUR_API_KEY";

        // Make an HTTP request to fetch weather data (use a library like Retrofit or OkHttp for network
calls)
        // Here we just simulate a good weather condition (sunny) for the sake of demonstration
        String weatherCondition = "sunny"; // This should be dynamically fetched from the weather API

        // 2. Use Google Places API to fetch nearby safe places based on the weather condition (e.g., parks,
hospitals)
        String placesApiUrl = "https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=" +
lat + "," + lon + "&radius=5000&type=park&key=YOUR_API_KEY";

        // Simulate fetching nearby safe places based on the weather condition and proximity
        List<LatLng> safePlaces = new ArrayList<>();
        // Example: Fetching 3 safe places based on nearby parks
        safePlaces.add(new LatLng(lat + 0.01, lon + 0.01)); // Nearby park
        safePlaces.add(new LatLng(lat - 0.02, lon - 0.02)); // Nearby park
        safePlaces.add(new LatLng(lat + 0.03, lon - 0.03)); // Nearby park

        // 3. Display safe places on the map
        for (LatLng place : safePlaces) {
            mMap.addMarker(new MarkerOptions().position(place).title("Safe Place"));
        }

        // Optionally, zoom to the first safe place
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(safePlaces.get(0), 12f));

        // 4. Display safe place addresses or weather conditions (this can also be dynamic)
        displaySafePlacesAddresses(safePlaces, lat, lon, weatherCondition);
    }

    private void showModerateRiskZone() {
        // Display moderate-risk zone message on screen or map
        titleTextView.setText("Moderate Risk Zone ⚠");
```

```
    }

    private void showSafeZone() {
        // Display safe zone message on screen or map
        titleTextView.setText("Safe Zone 😊");
    }
    private void displaySafePlacesAddresses(List<LatLng> safePlaces, double lat, double lon, String
weatherCondition) {
        Geocoder geocoder = new Geocoder(this);
        // Clear previous safe places addresses
        safePlacesLayout.removeAllViews();
        for (LatLng place : safePlaces) {
            try {
                List<Address> addresses = geocoder.getFromLocation(place.latitude, place.longitude, 1);
                if (addresses != null && !addresses.isEmpty()) {
                    Address address = addresses.get(0);
                    StringBuilder fullAddress = new StringBuilder();
                    // Get full address (street, city, country)
                    for (int i = 0; i <= address.getMaxAddressLineIndex(); i++) {
                        fullAddress.append(address.getAddressLine(i)).append("\n");
                    }
                    // Calculate distance from current location to this safe place
                    float[] results = new float[1];
                    Location.distanceBetween(lat, lon, place.latitude, place.longitude, results);
                    float distance = results[0]; // Distance in meters
                    // Create a TextView for each safe place
                    TextView addressTextView = new TextView(this);
                    addressTextView.setText("📍 Safe Place:\n" + fullAddress.toString() +
                            "\n📏 Distance: " + distance + " meters" +
                            "\n☀ Weather: " + weatherCondition);
                    addressTextView.setPadding(20, 15, 20, 15);
                    addressTextView.setTextSize(16);
                    addressTextView.setTextColor(getResources().getColor(android.R.color.white));
  addressTextView.setBackground(getResources().getDrawable(R.drawable.ic_launcher_background));
                    // Set margin between safe place entries
                    LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(
                            LinearLayout.LayoutParams.MATCH_PARENT,
                            LinearLayout.LayoutParams.WRAP_CONTENT
                    );
                    params.setMargins(0, 10, 0, 10);  // 10dp space between items
                    addressTextView.setLayoutParams(params);

                    // Add the TextView to the LinearLayout
                    safePlacesLayout.addView(addressTextView);}
```

```
            } catch (IOException e) {
                e.printStackTrace();
                Toast.makeText(this, "Error getting address", Toast.LENGTH_SHORT).show();}}
    }
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull
int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (requestCode == LOCATION_PERMISSION_REQUEST_CODE) {
            if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED)
{
                getCurrentLocation();
            } else {
                Toast.makeText(this, "Permission denied", Toast.LENGTH_SHORT).show();
            }
        }
    }
}
```

//EmergencyContactActivity

```
  package com.example.mad;
import android.os.Bundle;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
public class EmergencyContactsActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_emergency_contacts);
        // Example Emergency Contacts
        String contacts = " 🚨 Emergency Contacts 🚨 \n\n" +
                "📞 Police: 100\n" +
                "🚑 Ambulance: 108\n" +
                "🔥 Fire Brigade: 101\n" +
                " 📡 Disaster Helpline: 112\n" +
                " 🏥 Local Hospital: +91 98765 43210";
        // Set the contacts to TextView
        TextView contactTextView = findViewById(R.id.txtEmergencyContacts);
        contactTextView.setText(contacts);
    }
}
```

8.RESULT/OUTPUT SCREENS

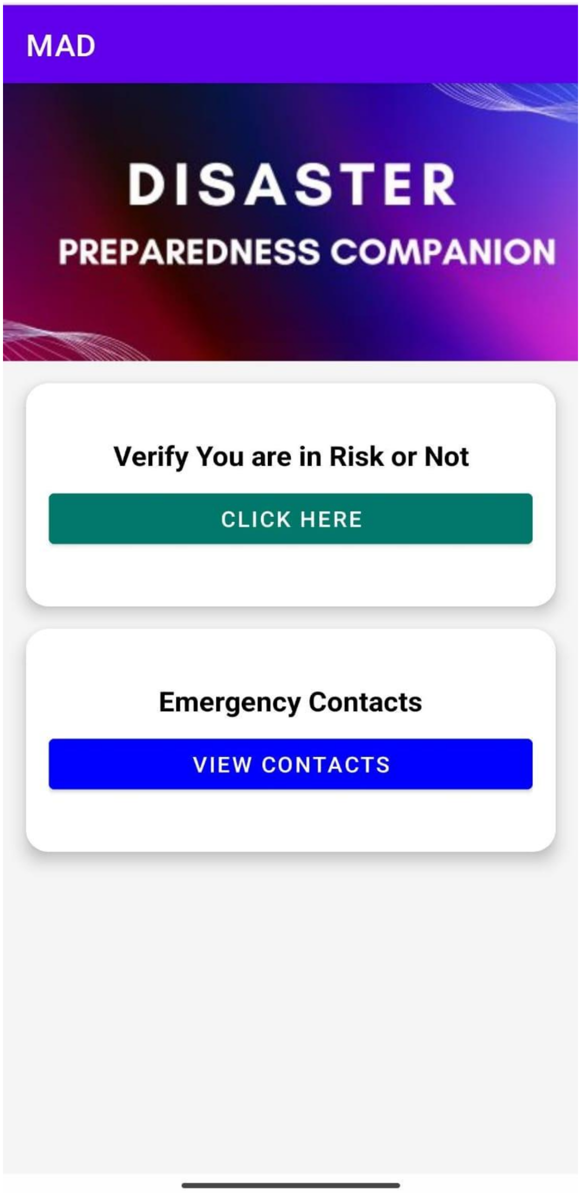


Fig 1: Home Page

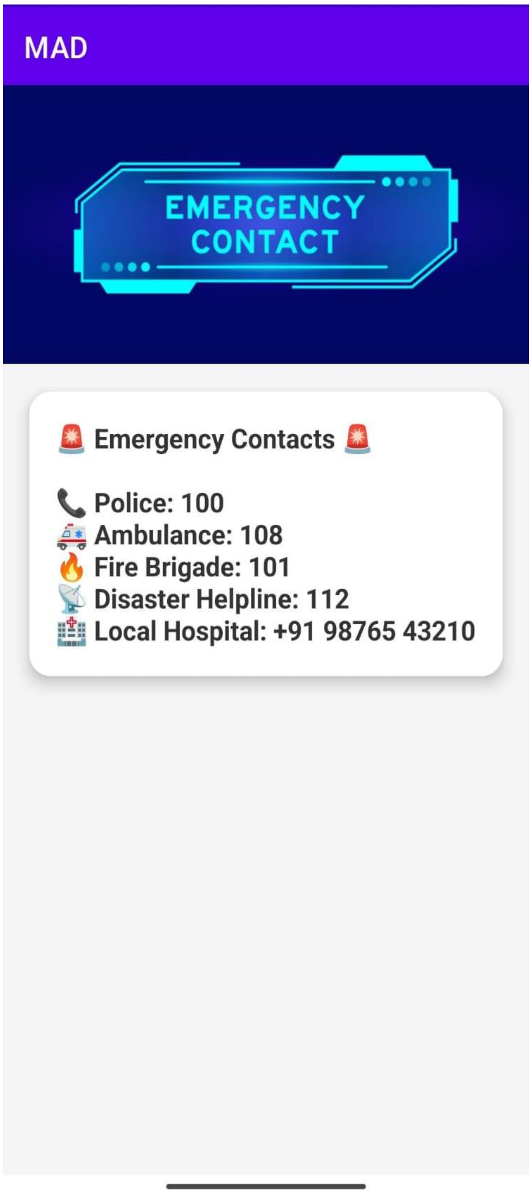
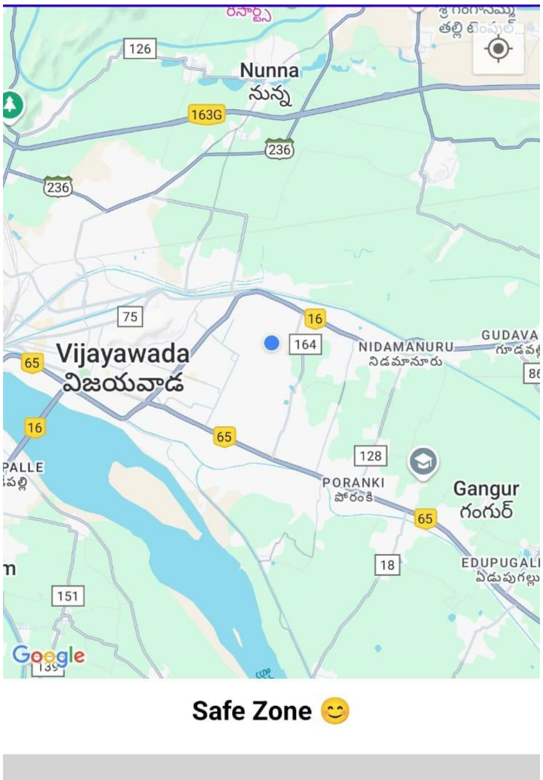


Fig 2 : Emergency Contacts
After Click on View Contacts in Fig(1).



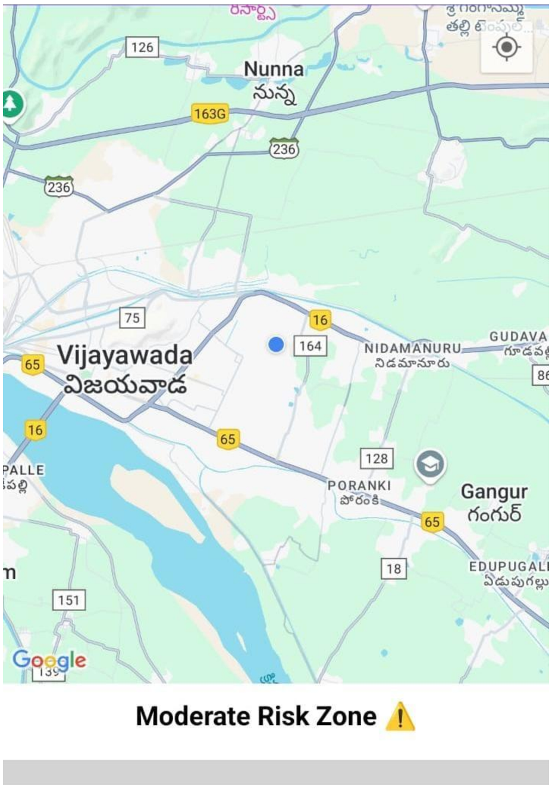


Fig 3
This is the safe zone for
Your Current Location.

Fig 4
This is Moderate Zone for
Your Current Location.

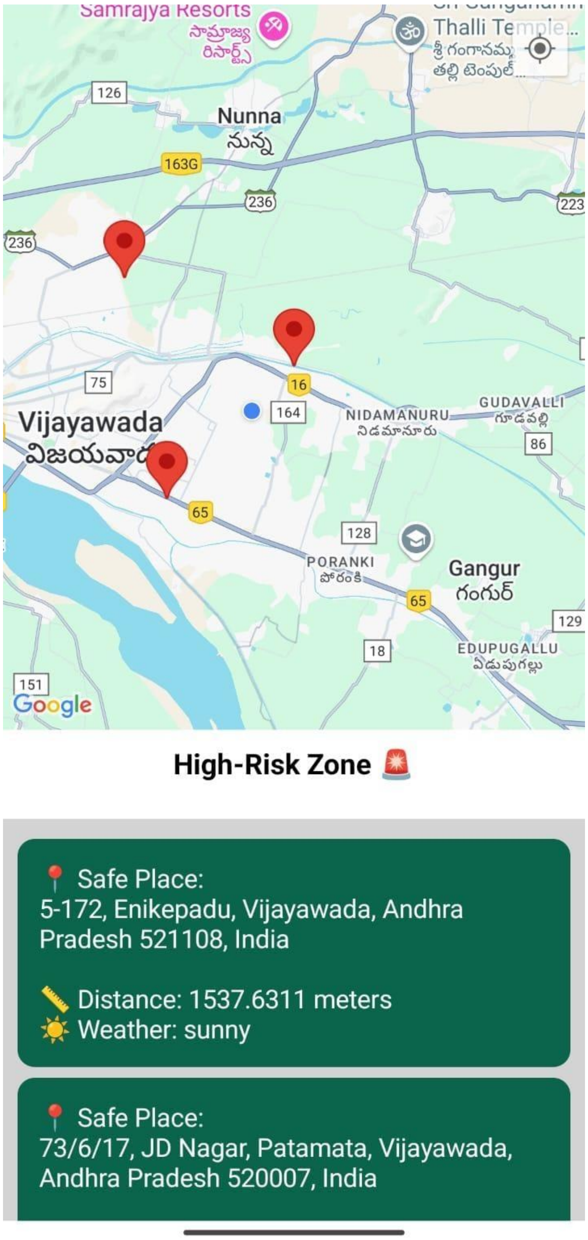


Fig 5 : This is For Risk Conditions in Your Current Location and Shows Some Safe Places along with Distances and also Show Weather Conditions of current and Safe Places.

9.CONCLUSION & FUTURE ENHANCEMENTS

This project successfully implements real-time disaster alerts, evacuation routes, and emergency contact management in an Android application. The Disaster Preparedness Companion provides users with crucial disaster-related information, including real-time weather updates, risk area identification, and emergency preparedness resources.

By integrating reliable APIs, the app ensures efficient and seamless data retrieval, fetching real-time disaster warnings in a structured manner. The use of GPS tracking enhances user safety by providing location-specific alerts and evacuation routes. Additionally, robust error handling ensures that users receive meaningful feedback, such as alerts for high-risk areas and network connectivity issues.

The interface is designed to be intuitive, allowing users to access emergency contacts, view evacuation routes, and receive real-time alerts with ease. The implementation follows best coding practices, ensuring modularity, readability, and maintainability. Key enhancements include improved UI feedback, precise disaster alert notifications, and enhanced user interaction.

This project can be further expanded by adding AI-driven risk assessment, multilingual support, offline emergency guides, and push notifications for critical warnings. Overall, this application provides a solid foundation for disaster preparedness and showcases the effective use of modern Android development techniques to enhance user safety and awareness.

10. REFERENCES

GitHub repository links:

https://github.com/SaiRitish514/Disaster-preparedness-Companion-

1. **OpenWeatherMap API** - https://openweathermap.org/api
2. **Google Places API** - https://developers.google.com/places/web-service/intro
3. **SQLite for Android -** https://developer.android.com/training/data-storage/sqlite
4. **Android Development Documentation -** https://developer.android.com/docs
5. **Google API Services** - https://developers.google.com/
6. **Android App Development Guide** - https://developer.android.com/guide
7. **Emergency Disaster Call Details** - https://www.fema.gov/emergency-managers
8. **Google Maps API** - https://developers.google.com/maps/documentation