**🌍 Solution : Disaster Prediction Methods**

**1. Using Weather Data (Simple ML Model)**

* **What it means:** We take rainfall, temperature, wind speed, humidity etc. from weather stations or APIs.
* **Why:** Heavy rainfall + steep slopes = landslide risk. Sudden rise in rainfall = flood risk.
* **How:**
  1. Collect weather data every hour/day.
  2. Store it in a table (like Excel sheet).
  3. Train a **machine learning model** (e.g., Random Forest or LSTM) to say: “If rain > X and slope is Y → landslide likely.”
* **Tools:** Python (pandas, scikit-learn, tensorflow).
* **Easy analogy:** Like teaching your brain “If clouds are black + strong wind → it may rain soon.”

**2. Using Satellite Images (Remote Sensing)**

* **What it means:** Satellites (like Sentinel or Landsat) take pictures of Earth daily.
* **Why:** They show rivers swelling (flood risk), soil getting weak (landslide), or cyclone clouds forming.
* **How:**
  1. Download satellite images (free via Google Earth Engine).
  2. Convert images into numbers (e.g., water area, vegetation cover).
  3. Train ML model with those numbers to predict floods/landslides.
* **Tools:** Google Earth Engine + Python.
* **Easy analogy:** Like checking CCTV footage of your house every day to see if water is rising.

**3. Crowdsourcing from Phones (Sensors)**

* **What it means:** Use tourists’ phones as sensors (accelerometer, GPS).
* **Why:** If many phones shake at the same time → earthquake.
* **How:**
  1. App collects small signals (phone shakes).
  2. Sends to server.
  3. Server checks if many users near one place had shaking → send alert.
* **Tools:** Your app + FastAPI backend.
* **Easy analogy:** Like many people shouting “I felt shaking!” at once → you confirm it’s real.

**5. Hybrid Approach (Best in real life)**

* Combine all:
  + Weather data + Satellite data + Mobile sensors.
* Why: If all signals say “danger” → send strong alert. If only one says → keep as “warning.”
* Example:
  + Rainfall rising fast (weather).
  + River looks wide in satellite photo.
  + Physics model says flood possible.
  + → All match = send alert immediately.

**🛠 Technologies for a Beginner**

* **Programming:** Python (simple and powerful for ML).
* **Data:** Weather APIs (like OpenWeather), free satellite data (Google Earth Engine).
* **ML Libraries:**
  + pandas (handle data tables),
  + scikit-learn (basic ML),
  + tensorflow or pytorch (deep learning).
* **Backend:** FastAPI (to connect model with your app).
* **Database:** PostgreSQL with PostGIS (to store maps + alerts).

**📱 How it connects to your App**

1. Model runs on server (Python + FastAPI).
2. It checks weather/satellite data every few minutes.
3. If “danger” → server sends **alert to app** (via Firebase push notification or SMS).
4. App shows alert + safe routes on map.

# 🌦 Section 1: Weather-based Features

### ✅ Parameters (Inputs for Model)

1. **Rainfall (mm/hour or mm/day)**
   * Heavy rainfall is the most direct signal for **floods** and **landslides**.
   * Example: If >100 mm in 24 hours → flood risk.
2. **Temperature (°C)**
   * Sudden rise → heatwave, drought risk.
   * Drop + high humidity → heavy rainfall chance.
3. **Humidity (%)**
   * High humidity + high temperature = thunderstorms or cyclones.
4. **Wind Speed (km/h)**
   * Strong winds indicate storms or cyclones.
   * Example: >60 km/h = storm warning.
5. **Wind Direction (degrees)**
   * Shows where cyclone or storm is moving.
6. **Air Pressure (hPa)**
   * Falling air pressure = storm/cyclone approaching.
   * Example: Drop below 1000 hPa = cyclone warning.

### Dataset Example (Excel Table Format)

| **Date** | **Rainfall (mm)** | **Temp (°C)** | **Humidity (%)** | **Wind Speed (km/h)** | **Wind Dir (°)** | **Pressure (hPa)** | **Disaster (Yes/No)** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2025-09-01 | 120 | 28 | 85 | 40 | 180 | 998 | Yes (Flood) |
| 2025-09-02 | 10 | 32 | 60 | 12 | 90 | 1010 | No |
| 2025-09-03 | 85 | 26 | 90 | 25 | 200 | 995 | Yes (Landslide) |
| 2025-09-04 | 0 | 35 | 30 | 8 | 270 | 1015 | No |

### 🌍 Data Sources (Free & Beginner-friendly)

1. **OpenWeatherMap API**
   * Free tier available (needs signup).
   * Provides rainfall, temperature, humidity, wind, pressure.
   * Example API call:
   * api.openweathermap.org/data/2.5/weather?q=Indore&appid=YOUR\_API\_KEY
   * Response gives JSON with rainfall, temp, humidity etc.
2. **IMD (India Meteorological Department)**
   * Provides Indian rainfall & weather data.
   * Useful for historical + real-time.
3. **NOAA Climate Data (USA but global coverage)**
   * Historical weather datasets.

### 🛠 Tools to Collect + Train

* **Python Libraries**
  + requests → fetch API data.
  + pandas → store in table format.
  + scikit-learn → train ML models (Random Forest).
* **Flow**
  1. Collect data every hour/day → store in CSV/DB.
  2. Clean it (handle missing values).
  3. Add disaster label (Yes/No) from past records.
  4. Train model (e.g., predict flood based on rainfall + pressure + humidity).

### 🎯 Easy Analogy

Like a person looking outside:

* If **clouds are dark + strong wind + air pressure low** → you guess a storm is coming.  
  That’s exactly what the model learns!

# 🛰 Section 2: Satellite Image Features (Remote Sensing)

### ✅ Parameters (Inputs for Model from Satellite Images)

1. **NDWI (Normalized Difference Water Index)**
   * Detects **water bodies** (rivers, lakes, flooded land).
   * Formula: (Green – NIR) / (Green + NIR).
   * If NDWI suddenly increases in city area → flood risk.
2. **NDVI (Normalized Difference Vegetation Index)**
   * Detects **vegetation health**.
   * Healthy plants = slope stable.
   * Low NDVI = weak soil = landslide risk.
3. **Land Surface Temperature (LST)**
   * From infrared bands.
   * Sudden rise = drought, sudden drop = cyclone cloud cover.
4. **Soil Moisture (from microwave satellites)**
   * Wet soil on steep slope → landslide risk.
5. **Cloud Cover & Cyclone Patterns**
   * Satellites can track cloud spirals → cyclone prediction.

### 📊 Dataset Example (Excel Table Format)

| **Date** | **NDWI (Water Index)** | **NDVI (Vegetation Index)** | **Land Temp (°C)** | **Soil Moisture (%)** | **Cloud Cover (%)** | **Disaster (Yes/No)** |
| --- | --- | --- | --- | --- | --- | --- |
| 2025-09-01 | 0.45 | 0.60 | 27 | 40 | 75 | No |
| 2025-09-02 | 0.80 | 0.20 | 24 | 85 | 90 | Yes (Flood) |
| 2025-09-03 | 0.30 | 0.15 | 26 | 80 | 50 | Yes (Landslide) |
| 2025-09-04 | 0.20 | 0.70 | 35 | 20 | 10 | No |

### 🌍 Data Sources (Free & Beginner-friendly)

1. **Google Earth Engine (GEE)**
   * Free, browser-based, beginner-friendly.
   * Datasets: Sentinel-1, Sentinel-2, MODIS, Landsat.
   * Example: Sentinel-2 gives NDVI, NDWI directly.
2. **Copernicus (Sentinel Satellites)**
   * Sentinel-1 (radar, works even in clouds) → floods & soil moisture.
   * Sentinel-2 (optical) → NDVI, NDWI, vegetation, land changes.
3. **NASA Earthdata**
   * MODIS & GPM (rainfall, land surface temp).

### 🛠 Tools to Collect + Train

* **Google Earth Engine (GEE)**
  + JavaScript/Python API to extract NDVI, NDWI, soil moisture.
  + Example GEE Python code:
  + import ee
  + ee.Initialize()
  + # Load Sentinel-2 image
  + img = ee.ImageCollection("COPERNICUS/S2") \
  + .filterDate('2025-09-01', '2025-09-02') \
  + .mean()
  + # Calculate NDVI
  + ndvi = img.normalizedDifference(['B8', 'B4'])
* **Python Libraries**
  + geemap → for GEE maps.
  + rasterio → read satellite rasters.
  + pandas → store extracted values.

### 🎯 Easy Analogy

Imagine satellite images as **CCTV cameras for Earth**:

* If water spreads outside the river boundary → CCTV shows flood.
* If hill looks brown (no green plants) → slope weak, landslide risk.
* If white spiral clouds appear → cyclone alert.

# 📱 Section 3: Crowdsourcing from Phones (Sensors)

### ✅ Parameters (Inputs from Mobile Devices)

1. **Accelerometer Data**
   * Detects shaking/vibration.
   * If many phones in one area shake → possible **earthquake**.
2. **GPS Location (Lat, Long)**
   * Needed to map where shaking or flood is happening.
   * Helps identify the disaster zone.
3. **Gyroscope Data**
   * Detects tilt/movement of phone.
   * Can confirm unusual ground vibration.
4. **Barometer / Pressure Sensor**
   * Some phones have it.
   * Drop in pressure → storm/cyclone detection.
5. **User Reports (Crowdsourced Evidence)**
   * Tourists can upload **photos, short videos, voice messages** from disaster site.
   * Helps validate sensor signals.

### Dataset Example (Collected via App)

| **User ID** | **Latitude** | **Longitude** | **Accelerometer (m/s²)** | **Gyroscope (°/s)** | **Pressure (hPa)** | **Report (Yes/No)** | **Disaster Confirmed** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| U101 | 22.72 | 75.87 | 15.2 | 3.5 | 1008 | No | No |
| U102 | 22.73 | 75.89 | 25.6 | 6.2 | 1005 | Yes (Flood pic) | Yes (Flood) |
| U103 | 22.75 | 75.90 | 28.1 | 7.8 | 1002 | No | Yes (Earthquake) |
| U104 | 22.76 | 75.92 | 9.1 | 2.0 | 1010 | No | No |

### Data Sources

* **Directly from User Phones** (via your app).
* Sensors used:
  + Accelerometer, GPS, Gyroscope, Barometer.
* **User-Generated Data**:
  + Photos, videos, or “I felt shaking” button inside app.

### Tools to Collect + Train

1. **Frontend (Mobile App)**
   * Android/iOS: Access accelerometer, gyroscope, GPS via built-in APIs.
   * Example (Android): SensorManager class.
2. **Backend (Server)**
   * FastAPI / Django → to receive and store sensor data.
   * Database: PostgreSQL with PostGIS → store coordinates.
3. **Analysis**
   * Aggregate signals:
     + If >50 users in 2 km radius report shaking → confirm earthquake.
     + If rainfall data + user flood photos match → confirm flood.

### Easy Analogy

Socho ek jagah 50 log khade hain:

* Agar sab ek saath chillayein **“Mujhe zameen hilli mehsoos hui”** → pakka earthquake.
* Agar 20 log ek hi area se **flood ki photo bhejein** → pakka flood.

Phones yahi kaam karenge — **automatic sensors + human confirmation**.

# Section 5: Hybrid Approach (Best in Real Life)

### ✅ Parameters (Combination of All Sources)

1. **Weather Features (from Section 1)**
   * Rainfall, temperature, humidity, wind, air pressure.
2. **Satellite Features (from Section 2)**
   * NDWI (water spread), NDVI (vegetation health), soil moisture, land surface temperature, cyclone clouds.
3. **Crowdsourced Features (from Section 4)**
   * Accelerometer shakes, GPS location clusters, barometer drops, user photo/video reports.

### Dataset Example (All-in-One Table)

| **Date** | **Rainfall (mm)** | **Humidity (%)** | **NDWI** | **NDVI** | **Soil Moisture (%)** | **Slope (°)** | **DEM Elevation (m)** | **Accel. Avg (m/s²)** | **User Reports** | **Disaster (Yes/No)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2025-09-01 | 120 | 85 | 0.80 | 0.20 | 85 | 35 | 250 | 25.6 | 10 (Flood pics) | Yes (Flood) |
| 2025-09-02 | 10 | 60 | 0.20 | 0.70 | 20 | 10 | 150 | 9.1 | 0 | No |
| 2025-09-03 | 85 | 90 | 0.30 | 0.15 | 80 | 40 | 500 | 28.1 | 7 (Landslide) | Yes (Landslide) |
| 2025-09-04 | 0 | 30 | 0.10 | 0.80 | 15 | 5 | 100 | 8.0 | 0 | No |

### Data Sources (Mix of All)

* **Weather APIs** → OpenWeather, IMD, NOAA.
* **Satellite Images** → Google Earth Engine, Sentinel, MODIS.
* **DEM & Soil** → USGS, FAO, ISRO Bhuvan.
* **Crowdsourced Sensors** → From your app (accelerometer, GPS, barometer, reports).

### Tools & Workflow

1. **Data Collection Layer**
   * Weather API → hourly/daily data.
   * GEE → satellite indices (NDVI, NDWI).
   * DEM/soil maps → static input.
   * App sensors → real-time input.
2. **Data Fusion Layer**
   * Combine all inputs into one database (PostgreSQL + PostGIS).
   * Use ETL pipeline (Extract-Transform-Load) with Python.
3. **Model Layer**
   * **Machine Learning**: Random Forest, XGBoost, or LSTM (time series).
   * **Rule-based check**:
     + If rainfall > threshold + NDWI high + slope steep = landslide risk.
     + If sensors confirm shaking + USGS quake data = earthquake confirm.
4. **Alert Layer**
   * FastAPI backend checks predictions.
   * If multiple signals = "Strong Alert".
   * If only one = "Warning".
   * Push notifications via Firebase / SMS.

### Easy Analogy

Socho ek doctor ke paas **4 reports** aayi:

* Thermometer: Bukhar.
* X-ray: Chest infection.
* Blood test: WBC high.
* Patient khud bol raha: "Bohot thakan hai."

👉 Agar sab confirm karte hain → doctor sure hai ki infection hai.  
Waise hi hybrid model me agar **Weather + Satellite + Phones** sab bolte hain "Flood aa raha hai" → system pakka alert bhejta hai.

| **Method** | **What it Uses** | **Key Parameters** | **Data Sources** | **Tools/Tech** | **Best For** |
| --- | --- | --- | --- | --- | --- |
| **1. Weather Data (Simple ML)** | Hourly/Daily weather records | Rainfall, Temperature, Humidity, Wind Speed/Direction, Air Pressure | OpenWeather API, IMD, NOAA | Python (pandas, scikit-learn, tensorflow) | Floods, Landslides, Heatwaves, Cyclones |
| **2. Satellite Images (Remote Sensing)** | Satellite pictures of Earth | NDWI (water spread), NDVI (vegetation), Soil Moisture, Land Temp, Cloud Patterns | Google Earth Engine, Sentinel, Landsat, NASA Earthdata | GEE, geemap, rasterio, Python ML | Flood detection, Landslide risk, Cyclone monitoring |
| **3. Physics-based Models (Simulation)** | Science formulas + maps | Rainfall Intensity, Soil Type, Soil Moisture, Slope Angle, DEM Elevation, Drainage Density, Land Use | USGS DEM, FAO Soil Maps, ISRO Bhuvan, HydroSHEDS | HEC-HMS (Floods), TRIGRS (Landslides), QGIS | Accurate flood & landslide simulation |
| **4. Crowdsourcing from Phones (Sensors)** | Tourists’ mobile sensors | Accelerometer (shaking), GPS, Gyroscope, Barometer, User Reports (photo/video) | Directly from app users | Android/iOS Sensors, FastAPI backend, PostgreSQL + PostGIS | Earthquakes, Real-time flood/landslide reports |
| **5. Hybrid Approach (Combination)** | All the above combined | Weather + Satellite + Physics + Sensors | Mix of all APIs + satellite + app | Python ML (Random Forest, XGBoost, LSTM), FastAPI, Firebase for alerts | Most reliable real-time prediction & alerts |

# 🔹 Solutions for "No Network" Areas to give ALERT

### ✅ 1. **Cell Broadcast (like Government Emergency Alerts)**

* This is the same system that shows “Earthquake Alert” or “Flood Warning” pop-ups on your phone **without internet**.
* The alerts are sent **directly from telecom towers** → no mobile data required.
* Industry-level apps cannot do this on their own (requires **government + telecom permissions**).
* If you partner with local government/telecom, your disaster alerts can also be broadcast through this system.

### ✅ 2. **Satellite-based Alerts (for Tourists)**

* If the tourist’s mobile is connected with a **satellite SMS receiver app** or **wearable device** (e.g., Garmin inReach, SPOT device), they can receive alerts **without mobile towers**.
* Already used in trekking/adventure apps like **AllTrails, Garmin Explore**.
* Your app can integrate with **satellite alert APIs** (Garmin, Iridium).

#### Step 1: What are Garmin / Iridium Devices?

* Small, pocket-sized **satellite communicators**.
* Example: **Garmin inReach Mini 2**.
* Features:
  + 2-way messaging via satellite.
  + **SOS button** → immediately alerts the nearest rescue center with GPS location.
  + Works **without mobile network**.

### ✅ 3. **Offline Mesh Networking (App-to-App Communication)**

* If some phones have network and others don’t → messages can be spread **phone-to-phone** using **Bluetooth / WiFi Direct / LoRa Mesh**.
* Example: **Bridgefy App** (used in Hong Kong protests) → worked without internet by spreading alerts device-to-device.
* In React Native, you can use:
  + react-native-wifi-p2p
  + react-native-ble-plx (Bluetooth Low Energy)
* Use case: One tourist gets network → alert received → shared locally with nearby tourists via Bluetooth/WiFi.

### ✅ 4. **Offline Pre-downloaded Safety Data**

* When the user installs the app (or whenever they get internet), the app downloads **hazard maps + safe shelters** for offline use.
* If real-time alerts can’t be received due to no network, the app can still:
  + Show: “⚠️ No Network – Stay on marked safe trails only.”
  + Use phone’s **sensors (accelerometer + GPS + barometer)** to detect sudden earthquakes or landslides.
* Example: **Google’s Android Earthquake Alerts** works even without internet → it uses phone sensors + crowd-sourced detection.

### ✅ 5. **Community/Rescue Volunteers with Radio**

* Disaster-prone hill stations already have **wireless radio networks (HAM, police, forest departments)**.
* Your backend can notify these volunteers → they spread alerts via loudspeakers or radios.

### ✅ 6. **Satellite SOS in Modern Smartphones**

* New smartphones (iPhone 14+, Samsung Galaxy S24 Ultra) have **built-in Satellite SOS features**.
* Even without mobile signal → tourists can send emergency SMS via satellite.
* Example: iPhone 14 “Emergency SOS via Satellite” → user points phone towards the sky → message goes directly via satellite.

### 📱 **Tourist Side (In-App Features)**

* Your app can add an **“Offline SOS Mode”**:
  + If no network → the app guides the tourist:
    - “Use your Garmin/iPhone Satellite SOS to send alert.”
  + App can connect to Garmin devices via Bluetooth using:
    - react-native-bluetooth-serial

### ⚠️ **Challenges**

* Expensive (per SMS/MB charges).
* Requires partnerships with **satellite providers** (Garmin, Iridium, Starlink).
* Not every tourist will have a satellite-enabled phone/device.
* But with **government collaborations (NDMA, State Tourism Dept)** → feasible at large scale.

### 🔮 **Future (2025 & Beyond)**

* **Starlink + Direct-to-Phone**: Elon Musk’s Starlink is working on connecting normal **4G phones directly to satellites**.
* In future, even normal tourists will receive **satellite SMS/alerts without special devices**.
* In India, **BSNL + OneWeb project** is also working to provide satellite connectivity in hill stations.

✅ **Summary:**

* Short Term → Push notifications, SMS, mesh networking, offline maps.
* Long Term → Direct-to-phone satellite alerts (Starlink, OneWeb, iPhone SOS).

# 🌍 Smart Tourist Safety Monitoring & Incident Response System

## **P1: Core App (Frontend with React Native)**

📱 The mobile app tourists will actually use.

**Why React Native?**

* You already know JavaScript → easier for you.
* Single code → works on **Android & iOS**.
* Big ecosystem + support.

**Key Libraries:**

* react-native-paper → ready-made UI components (buttons, forms, cards).
* react-navigation → move between pages (Home, Alerts, Profile, Map).
* react-native-maps → embed Google Maps in the app.
* expo-location → get user’s live GPS location.
* react-native-background-geolocation → keep tracking location even when app is closed.

**Features in this phase:**

* Register/Login (later connected with blockchain).
* Map screen (shows tourist’s current location).
* Alerts tab (list of warnings & safe shelters).
* Profile tab (user’s info stored on blockchain).

## **P2: Backend (FastAPI – Python)**

⚙️ This is the “brain” that connects app + AI model + database.

**Why FastAPI?**

* Very fast, beginner-friendly, modern Python framework.
* Easy to connect AI models (since Python is best for ML).

**Key Libraries:**

* fastapi + uvicorn → run backend server.
* sqlalchemy → talk to relational databases (like PostgreSQL).
* pydantic → check data (example: no wrong format for GPS coordinates).
* websockets → real-time communication (instant alerts).
* firebase-admin → send push notifications.

**Features in this phase:**

* Receive data from app (location, user info).
* Connect with database to store users + disaster logs.
* Call AI model API to check if disaster is predicted.
* Send alerts to app when risk detected.

## **P3: Database (Storage System)**

💾 We need a reliable place to store all info.

**Options:**

1. **PostgreSQL** (best for structured data)
   * Example tables: Users, Alerts, Predictions, Safe Routes.
2. **MongoDB** (best for unstructured data/logs)
   * Example collections: Sensor logs, AI input history.

**Key Libraries:**

* sqlalchemy + psycopg2 → for PostgreSQL.
* pymongo → for MongoDB.

**Data Stored:**

* Users (linked with blockchain ID).
* Location logs of tourists.
* Prediction model results.
* Alert messages (who received what).

## **P4: Blockchain-based Digital ID**

🔐 Tourist identity must be **secure + tamper-proof**.

**Why Blockchain?**

* Digital ID is **immutable** (cannot be faked).
* Government/Hotels/Rescue teams can trust it.

**Tech Stack:**

* **Polygon / Ethereum Blockchain** (Polygon is cheaper).
* **Smart Contracts** → code deployed on blockchain for ID storage.
* **Wallets:** MetaMask / Web3 wallets.

**Libraries:**

* web3.js (React Native side) → connect app with blockchain.
* ethers.js → handle wallet & transactions.
* solidity → to write smart contracts for ID.

**Features:**

* Each tourist gets a **Digital Safety ID** when registering.
* ID contains: Name, National ID, Emergency Contact.
* Stored on blockchain → secure, not editable by anyone.

## **P5: AI Disaster Prediction Model**

🤖 Predicting disasters early.

**Step 1 – Collect Data:**

* Weather (rainfall, temperature, wind) → via APIs like OpenWeatherMap.
* River/Water level → government sensors, satellites.
* Seismic activity → earthquake monitoring APIs.
* Historical data of past disasters.

**Step 2 – Preprocess Data:**

* pandas → handle datasets.
* numpy → calculations.
* scikit-learn → normalize/clean data.

**Step 3 – Train Model:**

* **Machine Learning:**
  + Random Forest, Decision Trees (simple models).
  + Library: scikit-learn.
* **Deep Learning (for accuracy):**
  + LSTM (time-series like rainfall patterns).
  + CNN (satellite image analysis).
  + Libraries: tensorflow or pytorch.

**Step 4 – Deploy Model:**

* Save model (joblib or .h5).
* Load into FastAPI → expose endpoint like /predict-disaster.

## **P6: Geo-Fencing & Location Safety**

🗺️ Prevent tourists from entering unsafe zones.

**Geo-fencing:** A virtual “circle” around a dangerous area. If tourist enters → instant alert.

**Libraries:**

* geopy (Python) → calculate distance between points.
* react-native-background-geolocation → monitor user’s movement.

**How it works:**

* AI predicts “danger zone” → backend creates geo-fence.
* App checks: “Is user inside this circle?”
* If yes → app vibrates + shows red warning on map.

## **P7: Disaster Alert & Communication**

📢 How to alert users when disaster occurs.

**Case 1: Network Available**

* **WebSocket (FastAPI)** → instant real-time alerts.
* **Firebase Cloud Messaging (FCM)** → push notifications.

**Case 2: No Network (Hill Station Issue)**

* **Satellite SMS APIs** (Iridium, Starlink, Garmin).
* **Bluetooth Mesh Networking** → nearby phones share alert (using react-native-ble-plx).
* **Offline Maps + Pre-downloaded Safety Data** → still guide users if no internet.

## **P8: Safe Route Finder**

🛣️ Suggest safest path to escape disaster.

**Tech:**

* **Google Maps Directions API** → get multiple routes.
* **AI + Algorithms:**
  + Filter out routes crossing danger zones.
  + Use **Dijkstra’s Algorithm** or **A**\* → find shortest safe path.
* **Libraries:**
  + networkx (Python) → pathfinding.
  + react-native-maps → display safe route to user.

## **P9: Implementation Plan (Phases)**

### **P1 – Core System**

* React Native app UI + FastAPI backend + database.

### **P2 – Blockchain ID**

* Implement secure digital ID for users.

### **P3 – AI Model**

* Collect data → Train model → Deploy API.

### **P4 – Geo-Fencing**

* Add virtual danger zones with real-time alerts.

### **P5 – Alert System**

* WebSocket + Firebase + Satellite integration.

### **P6 – Safe Routes**

* Google Maps + AI route filtering.

### **P7 – Final Integration**

* Combine **App + AI + Blockchain + Alerts + Routes**.
* Test in simulation of disaster scenarios.

✅ This roadmap now covers **everything end-to-end**:

* **Frontend (React Native)**
* **Backend (FastAPI + WebSockets)**
* **Database (Postgres/MongoDB)**
* **Blockchain (Ethereum/Polygon)**
* **AI Model (TensorFlow/Sklearn)**
* **Geo-fencing (Geopy)**
* **Alerts (Firebase, Satellite, Mesh)**
* **Safe Routes (Google Maps + NetworkX)**