HACKATHON PROBLEM STATEMENT

Project Aegis: The Offline-First Disaster Response System

1. The Scenario: Disaster in Ratnapura

Engineers,

A major natural disaster has struck the **Ratnapura District**. Heavy monsoon rains have triggered severe landslides and widespread flooding, cutting off communities and destroying critical infrastructure.

Power lines are down. Cell towers have failed. The entire region is a digital dead zone.

First responders are on the ground right now, working in chaotic conditions to save lives and assess damage. They need to report critical information back to headquarters—locations of trapped civilians, blocked roads, and urgent supply needs—but their current digital tools are useless without an internet connection. They are resorting to pen and paper, which is slow, error-prone, and impossible to coordinate in real-time.

2. Your Mission: Build the Digital Lifeline

Your challenge over the next 24 hours is to build a disaster management system that is **resilient by design**. You must build a solution that assumes the internet is broken and works perfectly without it.

You are building a system with two connected parts:

1. **The Field Responder App:** A mobile tool for responders to log incidents in the "dead zone."
2. **The Command Dashboard:** A web interface for HQ to visualize incoming data once connectivity is restored.

The core engineering challenge is to shift your mindset from "Online-First" to **"Offline-First."** Your app must save data locally on the device and automatically synchronize it with the server only when a network connection becomes available. **Zero data loss is tolerated.**

---

3. Scope of Work (The 24-Hour MVP)

To succeed, your team must deliver the following minimal viable product (MVP).

Component A: The Field Responder App (Mobile/PWA)

This is the tool used in the mud. It must be robust and simple.

- **Feature 1: Offline-First Data Collection**
  - Responders must be able to fill out and save an incident report form completely offline (in Airplane Mode).
  - The data must be stored securely in a local on-device database (e.g., SQLite, Room, IndexedDB).
  - **Required Data Fields:**
    - **Incident Type:** (Dropdown: Landslide, Flood, Road Block, Power Line Down)
    - **Severity:** (Scale of 1-Critical to 5-Low)
    - **GPS Location:** (Auto-captured Latitude & Longitude. *Note: Even offline, phones can get a GPS signal.*)
    - **Timestamp:** (Date & Time of report creation)
    - **Photo:** (Optional but recommended).
- **Feature 2: The Sync Engine**
  - The app must automatically detect when network connectivity returns.
  - Upon reconnection, it must upload all locally stored reports to the central server without user intervention.
- **Feature 3: Persistent, Offline Authentication**
  - A responder logs in *once* while online (e.g., at HQ before deployment) to establish their session.
  - **The Challenge:** The app must securely cache this session. If the user closes the app completely and re-opens it hours later in a dead zone, they must still be logged in and ready to file a report. They should not see a login screen when offline.

Component B: The Command Dashboard (Web)

This is the view for decision-makers at Headquarters.

- **Feature 1: Live Map Visualization**
  - Display a map that plots incident reports as pins based on their GPS coordinates.
- **Feature 2: Real-Time List View**

- A list of incoming reports that automatically updates as new data syncs from the field.

What is Out of Scope (Don't Waste Time On These):

- **Payment Gateways:** Strictly forbidden.
- **Complex Offline Maps:** You do not need to download gigabytes of map tiles for offline use. Capturing the raw Lat/Long coordinate is sufficient for the offline requirement. The map only needs to load when online.

---

4. Technical Guidelines & Constraints

We know teams have different hardware. To ensure a fair playing field, here are the rules of engagement.

Platform Selection & "Safety"

- **Recommended: Progressive Web App (PWA):** This is often the fastest path. A well-built PWA works offline in the browser on both Android and iOS, has access to GPS, and uses IndexedDB for storage.
- **Flutter / React Native:** Great choices if your team has mobile development experience.
- **Native Android:** Powerful, but ensure you have an Android device to demo on.

The "iOS Safety Rule" (Foreground Sync)

We recognize that iOS is aggressive about killing background tasks, making true background syncing difficult. We are not testing your ability to hack iOS limitations.

- **The Rule: "Foreground Sync" is acceptable.** It is a pass if your app only syncs data when the user **re-opens the app** or brings it back to the screen. You do not need to implement complex background fetch tasks for iOS.

---

5. Evaluation & Marking Scheme (Total: 100 Points)

5.1 Submission & The Hard Deadline

- All project code must be maintained in a version control repository (e.g., GitHub, GitLab).
- **The Deadline is Final:** Judging will be based **strictly** on the latest commit pushed to your repository timestamped *before* the 24-hour timer ends.

- **Any commits, pushes, or changes made *after* the timer stops will be ignored**. If a feature is not in the repo at the deadline, it does not exist for grading purposes.

Phase 1: The "Airplane Mode" Test (Pass/Fail Gate)

Before grading anything else, judges will perform this physical test on your device. **If you fail this test, your maximum possible score is capped at 50/100.**

1. The Judge turns on **Airplane Mode** on your demo device.
2. They fill out a report and click **Submit**. (It must save successfully, no crashes or endless spinners).
3. They **kill the app** completely (remove it from memory).
4. They **re-open the app** (still offline). You must still be logged in, and the data must still be present locally (e.g., in a "Pending" list).
5. They turn the **Internet back on**.
6. They watch the Dashboard. The data must appear automatically within 30 seconds.

Phase 2: Scoring Rubric

Category	Points	Criteria Details
A. Technical Engineering	**40 pts**	
Offline Robustness	20 pts	Does it pass the Airplane Mode test flawlessly? Does it handle app restarts without losing data?
Sync Logic	10 pts	Is the sync reliable? Does it avoid creating duplicate records on the server? Does it retry failed uploads?
Security & Auth	10 pts	Is the auth token securely cached? Does the user remain logged in offline? Are secrets handled properly?
B. Feature Implementation	**25 pts**	
Map Integration	15 pts	Does the dashboard show correct pins? Does the mobile app capture accurate GPS coordinates even when offline?

Dashboard Utility	10 pts	Is the dashboard auto-refreshing and easy to read? Is the data sortable?
C. UI/UX & Usability	**20 pts**	*Focus on high-fidelity usability for a crisis, not fancy animations.*
Crisis UX	10 pts	Are buttons large and easy to tap? Is there a clear, unmistakable indicator of "Offline" vs. "Online" status?
Feedback Loops	10 pts	Does the user have absolute confidence their data is safe? (e.g., A clear "Saved Locally" confirmation).
D. Demonstration & Pitch	**15 pts**	
The Flow & Narrative	10 pts	Did you clearly demonstrate the problem and solution? Was the demo smooth from mobile to web?
Technical Q&A	5 pts	Can every team member clearly explain their part of the architecture (database choice, sync logic, API design)?

6. Recommended Resources & Tools

Don't reinvent the wheel. Use established tools to move faster.

- **Free Maps (Highly Recommended): Leaflet JS** with OpenStreetMap tiles. Requires no credit card and no API key.
- **Paid Maps:** Google Maps Platform, Mapbox (be mindful of API limits and billing).
- **Authentication:** Firebase Auth (fastest to implement), or custom JWT via a backend framework (Node.js, Laravel, Django, etc.).
- **Local Storage Options:**
  - **Web/PWA:** IndexedDB (use a wrapper library like idb or Dexie.js for easier use).
  - **React Native:** WatermelonDB or Realm.
  - **Android:** Room Database.
  - **iOS:** CoreData.

Good luck. The clock starts now.