Disaster Response Coordination Platform Assignment

Duration: 1 day (8-10 hours, due by [insert deadline])
Objective: Build a backend-heavy MERN stack app for a disaster response platform that aggregates real-time data to aid disaster management. Use Google Gemini API to extract location names from disaster descriptions, then convert them to lat/lng coordinates using a mapping service (Google Maps, Mapbox, or OpenStreetMap). Use Supabase geospatial queries for location-based lookups, a mock Twitter API (or alternatives like Twitter API/Bluesky) for social media reports, Browse Page for official updates, and Gemini API for image verification. Use Supabase for data storage and caching. The frontend is a minimal interface of the candidate's choice to test backend functionality. Use Cursor or Windsurf (or similar AI coding tools) to ship fast, focusing on complex backend logic.

Features

1. **Disaster Data Management**: Robust CRUD for disaster records (title, location name, description, tags like "flood," "earthquake"), with ownership and audit trail tracking.
2. **Location Extraction and Geocoding**:
    - Use Google Gemini API to extract location names (e.g., "Manhattan, NYC") from disaster descriptions or user inputs.
    - Convert location names to lat/lng coordinates using a mapping service (Google Maps, Mapbox, or OpenStreetMap).
3. **Real-Time Social Media Monitoring**: Fetch and process social media reports using a mock Twitter API, the real Twitter API (if accessible), or an alternative like Bluesky to identify needs, offers, or alerts; update in real-time.
4. **Geospatial Resource Mapping**: Use Supabase geospatial queries to locate affected areas, shelters, and resources based on lat/lng coordinates; support queries for nearby resources.
5. **Official Updates Aggregation**: Use Browse Page to fetch updates from government or relief websites (e.g., FEMA, Red Cross).
6. **Image Verification**: Use Google Gemini API to analyze user-uploaded disaster images for authenticity (e.g., detect manipulated content or verify context).
7. **Backend Optimization**: Use:
    - Supabase for data storage and caching API responses (using a dedicated table).
    - Geospatial indexes in Supabase for fast location-based queries.
    - Structured logging (e.g., "Report processed: Flood Alert").
    - Rate limiting and error handling for external APIs.

Requirements

1. **Backend (Node.js, Express.js)**:

- Build REST APIs:
  - Disasters: `POST /disasters`, `GET /disasters?tag=flood`, `PUT /disasters/:id`, `DELETE /disasters/:id`
  - Social Media: `GET /disasters/:id/social-media` (mock Twitter API, Twitter API, or Bluesky)
  - Resources: `GET /disasters/:id/resources?lat=...&lon=...` (Supabase geospatial lookup)
  - Updates: `GET /disasters/:id/official-updates` (Browse Page data)
  - Verification: `POST /disasters/:id/verify-image` (Gemini API)
  - Geocoding: `POST /geocode` (extract location with Gemini, convert to lat/lng with mapping service)
- Implement real-time updates via WebSockets (Socket.IO):
  - Emit `disaster_updated` on create/update/delete.
  - Emit `social_media_updated` on new social media results.
  - Broadcast `resources_updated` on new geospatial data.
- Mock authentication with hard-coded users (e.g., `netrunnerX`, `reliefAdmin`) and roles (admin, contributor).
- Use Supabase for caching:
  - Create a `cache` table (`key`, `value` [JSONB], `expires_at`) to store social media, mapping service, Browse Page, and Gemini API responses (TTL: 1 hour).
  - Implement cache logic to check `expires_at` before fetching from external APIs.
- Implement geospatial query logic using Supabase/PostgreSQL (e.g., `ST_DWithin` to find resources within 10km).
- Log actions in a structured format (e.g., “Resource mapped: Shelter at Manhattan, NYC”).
- Use Cursor/Windsurf for API routes, caching, and geospatial logic (e.g., “Generate a Node.js route for Gemini location extraction”).

2. **Database (Supabase)**:

   - Use Supabase (PostgreSQL) with tables:
     - `disasters`: (id, title, location_name [TEXT], location [GEOGRAPHY], description, tags [TEXT[]], owner_id, created_at, audit_trail [JSONB])
     - `reports`: (id, disaster_id, user_id, content, image_url, verification_status, created_at)
     - `resources`: (id, disaster_id, name, location_name [TEXT], location [GEOGRAPHY], type, created_at)
     - `cache`: (key, value [JSONB], expires_at)

- Create geospatial indexes on `location` columns (e.g., `CREATE INDEX disasters_location_idx ON disasters USING GIST (location)`) for fast queries.
- Create indexes on `tags` (GIN index) and `owner_id` for efficient filtering.
- Store audit trails as JSONB (e.g., `{ action: "update", user_id: "netrunnerX", timestamp: "2025-06-17T17:16:00Z" }`).
- Use Supabase JavaScript SDK for queries (e.g., `supabase.from('disasters').select('*')`).
- Optimize geospatial queries (e.g., `SELECT * FROM resources WHERE ST_DWithin(location, ST_SetSRID(ST_Point(-74.0060, 40.7128), 4326), 10000)`).
- Use Cursor/Windsurf for Supabase queries (e.g., “Generate a Supabase geospatial query”).

3. **External Service Integrations**:

- **Google Gemini API**:
  - **Location Extraction**: Extract location names from descriptions (key from https://aistudio.google.com/app/apikey). Prompt example: “Extract location from: [description].”
  - **Image Verification**: Verify image authenticity. Prompt example: “Analyze image at [image_url] for signs of manipulation or disaster context.”
  - Cache responses in Supabase `cache` table.
- **Mapping Service** (choose one):
  - **Google Maps**: Use Geocoding API to convert location names to lat/lng (key from https://console.cloud.google.com).
  - **Mapbox**: Use Geocoding API for coordinates (key from https://www.mapbox.com).
  - **OpenStreetMap**: Use Nominatim for geocoding (https://nominatim.org).
- **Social Media (Mock Twitter API or Alternative)**:
  - If Twitter API access is unavailable (https://developer.twitter.com), implement a mock endpoint (`GET /mock-social-media`) returning sample data (e.g., JSON with posts like `{ "post": "#floodrelief Need food in NYC", "user": "citizen1" }`).
  - If accessible, use Twitter API (free tier) for real-time posts with disaster keywords (e.g., “#floodrelief”).
  - Alternatively, use Bluesky API (https://docs.bsky.app) for social media posts if available.
- **Browse Page**: Fetch official updates from government/relief websites (e.g., FEMA, Red Cross) using a web scraping library (e.g., Cheerio in Node.js).
- Cache responses in Supabase `cache` table to handle rate limits (TTL: 1 hour).

- Use Cursor/Windsurf for API integrations (e.g., “Generate a Node.js Gemini geocoding endpoint”).

4. **Frontend (Candidate’s Choice)**:

    - Create a minimal frontend (e.g., single `index.html` or framework-based) with:
        - Form to create/update disasters (title, location name or description, description, tags).
        - Form to submit reports (content, image URL).
        - Display for disasters, social media reports, resources, and verification statuses.
        - Real-time updates for social media and resource data via WebSockets.
    - UI design and technology (e.g., plain JS, React, CSS) are up to the candidate.
    - Ensure functionality to test all backend APIs.
    - Use Cursor/Windsurf for frontend code (e.g., “Generate a fetch call for POST /disasters”).

5. **Vibe Coding**:

    - Use Cursor’s Composer or Windsurf’s Cascade (or similar AI tools) for:
        - Generating API routes (e.g., “Create a Node.js route for geospatial queries”).
        - Writing Supabase queries (e.g., “Generate a query for nearby resources”).
        - Implementing mock social media or Supabase caching logic (e.g., “Generate Supabase caching logic”).
    - Note AI tool usage in submission (e.g., “Cursor generated WebSocket logic”).

Bonus (Optional)

- Add a “priority alert” system to flag urgent social media reports (e.g., based on keywords like “urgent” or “SOS”).
- Implement a basic keyword-based classifier to prioritize reports.
- Integrate a mapping service to fetch additional resources (e.g., hospitals near a disaster) beyond sample data.
- Enhance the frontend with a custom feature (e.g., interactive map for resources).

Submission Instructions

- **Code**: Push to a GitHub repo (public or shared with [insert email]).
- **Demo**: Deploy on **Vercel** (frontend) and **Render** (backend); provide the live URL.
- **Files**: Submit a zip file with code and a note on how you used Cursor/Windsurf (e.g., “Windsurf generated mock social media logic”).
- **Submit**: Email [insert email] with repo link, live URL, and zip file by [insert deadline].

Evaluation

- **Functionality (50%)**: APIs, external integrations, WebSockets, and geospatial queries work.
- **Backend Complexity (30%)**: Effective use of Supabase caching, geospatial indexes, rate limiting, and error handling.
- **External Integrations (15%)**: Creative handling of Gemini location extraction, mapping service geocoding, mock Twitter API or alternatives, and Browse Page.
- **Vibe Coding (5%)**: Cursor/Windsurf usage is effective, noted in submission.

Notes

- Use mock data for testing (e.g., sample disaster locations: `{ title: "NYC Flood", location_name: "Manhattan, NYC", description: "Heavy flooding in Manhattan", tags: ["flood"] }`).
- Handle API rate limits with Supabase caching and fallback mock responses (e.g., “No new social media reports”).
- Supabase setup: Create a free project at https://supabase.com, use JavaScript SDK.
- Candidates choose the frontend approach, but it must test all backend functionality.
- Note shortcuts or assumptions in submission (e.g., “Used mock Twitter API due to access limits”).
- Use Cursor/Windsurf aggressively; mention their impact in the submission note.

Sample Data:

- Disaster: `{ title: "NYC Flood", location_name: "Manhattan, NYC", description: "Heavy flooding in Manhattan", tags: ["flood", "urgent"], owner_id: "netrunnerX" }`
- Report: `{ disaster_id: "123", user_id: "citizen1", content: "Need food in Lower East Side", image_url: "http://example.com/flood.jpg", verification_status: "pending" }`
- Resource: `{ disaster_id: "123", name: "Red Cross Shelter", location_name: "Lower East Side, NYC", type: "shelter" }`

Build fast, test thoroughly, and help coordinate disaster response! 🚀