Major System Components

The proposed AI-Powered Emergency Dispatch Command Center consists of several major components that work together to ensure real-time incident management, intelligent decision making, and automated dispatching. The components are categorized into AI Subsystem, Dashboard, Incident Management, Mapping, User Interface, and Utility Libraries.

A. AI Subsystem

The AI subsystem is responsible for intelligent decision-making tasks, leveraging a Large Language Model (LLM) integrated via Google Genkit. It is implemented in the directory `src/ai/flows/` and `src/ai/genkit.ts`.

1. **Incident Report Analysis**
   The module `analyze-report.ts` performs classification of caller reports into predefined incident types (e.g., "Cardiac Arrest", "Fire") using few-shot prompting. It also extracts key entities influencing the classification to support further decision processes.

2. **Dispatch Package Recommendation**
   The `get-dispatch-package.ts` module recommends the optimal number and types of responder vehicles based on incident complexity and severity.

3. **Hospital Recommendation**
   Implemented in `recommend-hospital.ts`, this component suggests the most suitable hospital by evaluating medical capabilities, bed availability, distance, and live traffic conditions.

4. **Protocol Generation and Traffic Report**
   The system provides actionable checklists via `get-protocol.ts` and analyzes live traffic conditions using `get-traffic-report.ts` to support routing and hospital selection.

5. **Incident Summarization and Debriefing**
   The modules `summarize-incident.ts` and `debrief-incident.ts` generate comprehensive post-incident reports, allowing performance evaluation and process improvement.

6. **Genkit Integration**
   The `genkit.ts` helper manages communication with Google's Genkit service, executing AI model inference calls.

B. Dashboard Components

The dashboard, located in `src/components/dashboard/`, provides a comprehensive interface for emergency management.

1. **Analytics Dashboard**
   Displays system performance statistics, incident trends, and response times.

2. **Dispatch Dashboard**
   Central interface for monitoring active incidents, responder availability, and dispatch actions.

3. **Fleet Status Monitor**
   Displays real-time location and operational status of all emergency units.

4. **Incident Summary and Logging**
   Provides summaries of ongoing and past incidents along with a full action log for auditing purposes.

5. **New Incident Form**
   Allows manual creation of new incidents or editing automatically analyzed data prior to dispatch.

C. Incident Management Components

Located in `src/components/incident/`, these components handle the full lifecycle of incidents.

1. **Incident Card and List**
   Displays individual and aggregated incident details in a user-friendly card or list format.

2. **Incident Details and Debrief**
   Provides detailed views and AI-generated post-incident debrief reports.

D. Mapping Components

Mapping functionalities are implemented in `src/components/map/`.

1. **Map View and Layout**
   Uses MapLibre GL JS and React Map GL to provide an interactive map showing incidents, hospitals, and responders.

2. **Route Visualization**
   `map-route.tsx` visualizes computed routes based on data from the Open Source Routing Machine (OSRM).

3. **Routing Algorithm Integration**
   The `map-layout.tsx` component integrates OSRM via API calls to compute and display the fastest driving route, employing Contraction Hierarchies for optimal performance.

E. User Interface Components

The `src/components/ui/` directory contains reusable UI elements, including buttons, tables, forms, modals, and alerts, following consistent design principles (ShadCN UI and Tailwind CSS).

F. Theme & Utility Libraries

1. **Theme Provider and Toggle**
   Supports global theming (e.g., light/dark mode).

2. **Utility Hooks**
   Includes device detection (`use-mobile.tsx`) and toast notification support (`use-toast.ts`).

3. **Helper Libraries**
   Type definitions (`types.ts`), static test data (`data.ts`), and general utility functions (`utils.ts`) facilitate consistent development and testing.
