```
import { useEffect, useRef } from "react";
import mapboxgl from "mapbox-gl";

const roleConfig = {
  police: { color: "#3B82F6", icon: "🚔", label: "Police" },
  ambulance: { color: "#10B981", icon: "🚑", label: "Ambulance" },
  authority: { color: "#6366F1", icon: "🛡️", label: "Authority" },
  volunteer: { color: "#8B5CF6", icon: "🙋", label: "Volunteer" },
};

const statusConfig = {
  en_route: { pulse: true, opacity: 1, label: "En Route" },
  on_scene: { pulse: false, opacity: 1, label: "On Scene" },
  available: { pulse: false, opacity: 0.6, label: "Available" },
  offline: { pulse: false, opacity: 0.3, label: "Offline" },
};

export function useAuthorityMarkers({ map, responders, mapLoaded }) {
  const markersRef = useRef(new Map());

  useEffect(() => {
    if (!map || !mapLoaded) return;

    // Remove stale markers
    markersRef.current.forEach((marker, id) => {
      if (!responders.some(r => r.id === id)) {
        marker.remove();
        markersRef.current.delete(id);
      }
    });

    // Add or update markers
    responders.forEach((responder) => {
      const existing = markersRef.current.get(responder.id);
      const config = roleConfig[responder.role] || roleConfig.authority;
      const status = statusConfig[responder.status] || statusConfig.available;

      if (existing) {
        existing.setLngLat([responder.longitude, responder.latitude]);
      } else {
        const el = document.createElement("div");
        el.className = "authority-marker";

        el.innerHTML = `
        <div class="relative cursor-pointer transform hover:scale-110 transition-transform
duration-300" style="opacity: ${status.opacity}">
          ${status.pulse ? `<div class="absolute -inset-3 rounded-full animate-ping"
style="background:${config.color}40"></div>` : ""}
          <div class="w-11 h-11 rounded-full flex items-center justify-center shadow-xl border-3"
style="background: linear-gradient(135deg, ${config.color}, ${config.color}dd); border-color:
white;">
            <span class="text-xl">${config.icon}</span>
          </div>
          ${responder.status === "en_route" && responder.eta_minutes ? `<div class="absolute
-bottom-1 -right-1 bg-white rounded-full px-2 py-0.5 text-xs font-bold shadow-lg" style="color:
${config.color}">${responder.eta_minutes}m</div>` : ""}
          ${responder.status === "on_scene" ? `<div class="absolute -top-1 -right-1 w-4 h-4
bg-emerald-500 rounded-full border-2 border-white flex items-center justify-center"><span
class="text-white text-[8px]">✓</span></div>` : ""}
        </div>
      `;

      const popup = new mapboxgl.Popup({ offset: 30, closeButton: false, className:
"responder-popup" }).setHTML(`
        <div class="p-3 text-sm min-w-[160px]">
          <div class="flex items-center gap-2 mb-2">
            <span class="text-xl">${config.icon}</span>
            <strong style="color: ${config.color}">${config.label}</strong>
          </div>
          <div class="space-y-1 text-xs text-gray-600">
            <p class="flex items-center gap-1">
              <span class="w-2 h-2 rounded-full" style="background: ${responder.status ===
"on_scene" ? "#22c55e" : responder.status === "en_route" ? "#f59e0b" : "#9ca3af"}"></span>
              ${status.label}
            </p>
            ${responder.eta_minutes ? `<p>ETA: ${responder.eta_minutes} min</p>` : ""}
            ${responder.name ? `<p class="text-gray-400">${responder.name}</p>` : ""}
          </div>
        </div>
      `);

      const marker = new mapboxgl.Marker({ element: el })
        .setLngLat([responder.longitude, responder.latitude])
        .setPopup(popup)
        .addTo(map);

      markersRef.current.set(responder.id, marker);
    }
    });

    return () => {
      markersRef.current.forEach((m) => m.remove());
      markersRef.current.clear();
    };
  }, [map, mapLoaded, responders]);
}


import { useEffect } from "react";
import mapboxgl from "mapbox-gl";

function incidentsToGeoJSON(incidents) {
  const now = Date.now();
  return {
    type: "FeatureCollection",
    features: incidents
      .filter(i => {
        if (i.status === "resolved") return false;
        if (!i.created_at) return true;
        return now - new Date(i.created_at).getTime() < 24 * 60 * 60 * 1000;
      })
      .map((incident) => ({
        type: "Feature",
        id: incident.id,
        geometry: {
          type: "Point",
          coordinates: [incident.longitude, incident.latitude],
        },
        properties: {
          id: incident.id,
          type: incident.type || "other",
          status: incident.status || "active",
          verified: incident.verified || false,
          confidence: incident.confidence_score || 50,
          created_at: incident.created_at,
          description: incident.description,
          isRecent: Date.now() - new Date(incident.created_at).getTime() < 30 * 60 * 1000,
        },
      })),
  };
}
```

```
export function useIncidentLayers({ map, incidents, mapLoaded, onSelect }) {
  useEffect(() => {
    if (!map || !mapLoaded) return;

    const sourceId = "incidents";

    if (!map.getSource(sourceId)) {
      map.addSource(sourceId, { type: "geojson", data: incidentsToGeoJSON([]), cluster: true,
clusterRadius: 50, clusterMaxZoom: 14 });
    }

    if (!map.getLayer("incident-clusters")) {
      // Clusters
      map.addLayer({
        id: "incident-clusters",
        type: "circle",
        source: sourceId,
        filter: ["has", "point_count"],
        paint: {
          "circle-color": [
            "step",
            ["get", "point_count"],
            "#ef4444", 5,
            "#dc2626", 10,
            "#b91c1c",
          ],
          "circle-radius": [
            "step",
            ["get", "point_count"],
            20, 5, 25, 10, 30,
          ],
          "circle-opacity": 0.85,
          "circle-stroke-width": 2,
          "circle-stroke-color": "#fff",
        },
      });

      // Cluster count
      map.addLayer({
        id: "incident-cluster-count",
        type: "symbol",
        source: sourceId,
        filter: ["has", "point_count"],
        layout: {
```

```
    "text-field": "{point_count_abbreviated}",
    "text-size": 14,
    "text-font": ["DIN Pro Medium", "Arial Unicode MS Bold"],
  },
  paint: {
    "text-color": "#fff",
  },
});

// Unclustered points
map.addLayer({
  id: "incident-points",
  type: "circle",
  source: sourceId,
  filter: ["!", ["has", "point_count"]],
  paint: {
    "circle-radius": [
      "case",
      ["==", ["get", "status"], "resolved"], 6,
      ["==", ["get", "isRecent"], true], 12,
      10,
    ],
    "circle-color": [
      "case",
      ["==", ["get", "type"], "panic"], "#ef4444",
      ["==", ["get", "type"], "amber"], "#f59e0b",
      ["==", ["get", "type"], "robbery"], "#ef4444",
      ["==", ["get", "type"], "assault"], "#dc2626",
      ["==", ["get", "type"], "kidnapping"], "#b91c1c",
      ["==", ["get", "type"], "accident"], "#f59e0b",
      ["==", ["get", "type"], "suspicious"], "#8b5cf6",
      "#6b7280",
    ],
    "circle-opacity": [
      "case",
      ["==", ["get", "status"], "resolved"], 0.5,
      0.9,
    ],
    "circle-stroke-width": [
      "case",
      ["==", ["get", "verified"], true], 3,
      2,
    ],
    "circle-stroke-color": [
          "case",
          ["==", ["get", "verified"], true], "#22c55e",
          ["==", ["get", "status"], "resolved"], "#6b7280",
          "#fff",
        ],
      },
    });

    // Pulse layer for recent/active incidents
    map.addLayer({
      id: "incident-pulse",
      type: "circle",
      source: sourceId,
      filter: [
        "all",
        ["!", ["has", "point_count"]],
        ["==", ["get", "isRecent"], true],
        ["!=", ["get", "status"], "resolved"],
      ],
      paint: {
        "circle-radius": 18,
        "circle-color": [
          "case",
          ["==", ["get", "type"], "panic"], "#ef4444",
          ["==", ["get", "type"], "amber"], "#f59e0b",
          "#ef4444",
        ],
        "circle-opacity": 0.3,
      },
    });
  }

  // Update data
  const source = map.getSource(sourceId);
  if (source) source.setData(incidentsToGeoJSON(incidents));

  // Layer visibility
  map.setLayoutProperty("incident-clusters", "visibility", "visible");
  map.setLayoutProperty("incident-points", "visibility", "visible");
 }, [map, mapLoaded, incidents, onSelect]);
}
```

```
import { useEffect } from "react";

function incidentsToHeatmapGeoJSON(incidents) {
  const now = Date.now();
  const recentIncidents = incidents.filter(i => {
    if (i.status === "resolved") return false;
    if (!i.created_at) return true;
    return now - new Date(i.created_at).getTime() < 24 * 60 * 60 * 1000; // last 24h
  });
  return {
    type: "FeatureCollection",
    features: recentIncidents.map((incident) => ({
      type: "Feature",
      geometry: {
        type: "Point",
        coordinates: [incident.longitude, incident.latitude],
      },
      properties: {
        confidence: incident.confidence_score || 50,
      },
    })),
  };
}

export function useHeatmapLayers({ map, incidents, mapLoaded, visible }) {
  useEffect(() => {
    if (!map || !mapLoaded) return;

    const sourceId = "heatmap-source";
    const layerId = "incident-heat";

    if (!map.getSource(sourceId)) {
      map.addSource(sourceId, {
        type: "geojson",
        data: incidentsToHeatmapGeoJSON([]),
      });
    }

    if (!map.getLayer(layerId)) {
      map.addLayer({
        id: layerId,
        type: "heatmap",
        source: sourceId,
        maxzoom: 15,
    paint: {
      "heatmap-weight": [
        "interpolate",
        ["linear"],
        ["get", "confidence"],
        0, 0,
        100, 1,
      ],
      "heatmap-intensity": [
        "interpolate",
        ["linear"],
        ["zoom"],
        0, 1,
        15, 3,
      ],
      "heatmap-color": [
        "interpolate",
        ["linear"],
        ["heatmap-density"],
        0, "rgba(0,0,0,0)",
        0.2, "rgba(255,200,0,0.3)",
        0.4, "rgba(255,140,0,0.5)",
        0.6, "rgba(255,80,0,0.7)",
        0.8, "rgba(255,0,0,0.8)",
        1, "rgba(200,0,0,0.9)",
      ],
      "heatmap-radius": [
        "interpolate",
        ["linear"],
        ["zoom"],
        0, 5,
        15, 30,
      ],
      "heatmap-opacity": [
        "interpolate",
        ["linear"],
        ["zoom"],
        12, 0.8,
        15, 0,
      ],
    },
  });
}
```

```
    // Update data
    const source = map.getSource(sourceId);
    if (source) source.setData(incidentsToHeatmapGeoJSON(incidents));

    // Toggle visibility
    map.setLayoutProperty(layerId, "visibility", visible ? "visible" : "none");
  }, [map, incidents, mapLoaded, visible]);
}



import { motion } from "framer-motion";
import { MapPin, Clock, ThumbsUp, MessageSquare, Navigation, X, Shield, CheckCircle,
AlertTriangle } from "lucide-react";
import { formatDistanceToNow } from "date-fns";

const typeLabels = {
  panic: "Emergency",
  amber: "Amber Alert",
  robbery: "Robbery",
  assault: "Assault",
  kidnapping: "Kidnapping",
  accident: "Accident",
  suspicious: "Suspicious",
  other: "Incident",
};

export function IncidentPreviewCard({ incident, userLocation, onClose, onViewDetails,
onNavigate }) {
  const typeLabel = typeLabels[incident.type] || "Incident";

  const distance =
    userLocation &&
    Math.hypot(incident.latitude - userLocation.lat, incident.longitude - userLocation.lng);

  const timeAgo = incident.created_at
    ? formatDistanceToNow(new Date(incident.created_at), { addSuffix: true })
    : "Just now";

  const isResolved = incident.status === "resolved";
  const isVerified = (incident.verified_count || 0) >= 3;

  return (
    <motion.div
```

```
      initial={{ opacity: 0, y: 20 }}
      animate={{ opacity: 1, y: 0 }}
      exit={{ opacity: 0, y: 20 }}
      className="fixed bottom-4 left-4 right-4 max-w-md mx-auto z-50"
    >
      <div className="bg-white rounded-xl shadow-lg overflow-hidden">
        {/* Header */}
        <div className="flex justify-between items-center px-4 py-3 bg-gray-100">
          <div className="flex items-center gap-2">
            <div className="w-8 h-8 bg-blue-500 flex items-center justify-center rounded-full
text-white">
              <AlertTriangle className="w-4 h-4" />
            </div>
            <h3 className="font-semibold text-gray-800">{typeLabel}</h3>
          </div>
          {/* Close Button */}
          <button onClick={onClose} className="p-1 rounded-full hover:bg-gray-200">
            <X className="w-4 h-4" />
          </button>
        </div>

        {/* Content */}
        <div className="p-4 space-y-3">
          {/* Status Badges */}
          <div className="flex items-center gap-2">
            {isVerified && (
              <span className="flex items-center gap-1 px-2 py-0.5 bg-green-200 text-green-800
text-xs rounded-full">
                <Shield className="w-3 h-3" /> Verified
              </span>
            )}
            {isResolved && (
              <span className="flex items-center gap-1 px-2 py-0.5 bg-gray-300 text-gray-700
text-xs rounded-full">
                <CheckCircle className="w-3 h-3" /> Resolved
              </span>
            )}
          </div>

          {/* Meta info */}
          <div className="flex items-center gap-4 text-xs text-gray-500">
            {distance !== null && (
              <div className="flex items-center gap-1">
                <MapPin className="w-3 h-3" /> {formatDistance(distance)} away
              </div>
            )}
            <div className="flex items-center gap-1">
              <Clock className="w-3 h-3" /> {timeAgo}
            </div>
          </div>

          {/* Description */}
          {incident.description && (
            <p className="text-sm text-gray-700 line-clamp-2">{incident.description}</p>
          )}

          {/* Stats */}
          <div className="flex justify-between mt-2">
            <div className="flex items-center gap-2 text-sm text-gray-600">
              <ThumbsUp className="w-4 h-4" /> {incident.verified_count || 0}
            </div>
            <div className="flex items-center gap-2 text-sm text-gray-600">
              <MessageSquare className="w-4 h-4" /> {incident.comment_count || 0}
            </div>
          </div>

          {/* Buttons */}
          <div className="flex gap-2 mt-4">
            <button
              onClick={onNavigate}
              className="flex-1 bg-blue-600 text-white px-4 py-2 rounded-lg hover:bg-blue-700
transition flex items-center justify-center gap-2"
            >
              <Navigation className="w-4 h-4" /> Navigate
            </button>
            <button
              onClick={onViewDetails}
              className="flex-1 border border-gray-300 px-4 py-2 rounded-lg hover:bg-gray-100
transition"
            >
              View Details
            </button>
          </div>
        </div>
      </div>
    </motion.div>
    );
}
```

```
import { motion } from "framer-motion";
import { AlertTriangle, X, MapPin, Clock, ChevronRight } from "lucide-react";
import { formatDistanceToNow } from "date-fns";

const typeLabels = {
  panic: "Emergency Alert",
  amber: "Amber Alert",
  robbery: "Robbery Reported",
  assault: "Assault Reported",
  kidnapping: "Kidnapping Alert",
  accident: "Accident Reported",
  suspicious: "Suspicious Activity",
  other: "Incident Reported",
};

export function NearYouStrip({ alert, isHighPriority, onDismiss, onViewOnMap }) {
  const timeAgo = alert.created_at
    ? formatDistanceToNow(new Date(alert.created_at), { addSuffix: true })
    : "Just now";

  return (
    <motion.div
      initial={{ opacity: 0, y: -20, scale: 0.95 }}
      animate={{ opacity: 1, y: 0, scale: 1 }}
      exit={{ opacity: 0, y: -20, scale: 0.95 }}
      className={`mx-4 rounded-2xl shadow-2xl overflow-hidden ${
        isHighPriority ? "bg-destructive text-destructive-foreground shadow-panic" : "bg-warning
text-warning-foreground"
      }`}
    >
      {isHighPriority && (
        <div className="absolute inset-0 bg-destructive animate-pulse opacity-30" />
      )}
      <div className="relative p-4">
        {/* Icon and Label */}
        <div className="flex items-start gap-3">
          <div className={`w-10 h-10 rounded-xl flex items-center justify-center flex-shrink-0
${isHighPriority ? "bg-white/20" : "bg-black/10"}`}>
            <AlertTriangle className="w-5 h-5 text-white" />
          </div>
          {/* Content */}
          <div className="flex-1 min-w-0">
            <div className="flex items-center gap-2 mb-1">
              <span className="font-bold text-sm">{typeLabels[alert.type] || "Alert"}</span>
              {isHighPriority && (
                <span className="px-2 py-0.5 bg-white/20 rounded-full text-xs font-medium
animate-pulse">URGENT</span>
              )}
            </div>
            {/* Location & Time */}
            <div className="flex items-center gap-3 text-xs opacity-90">
              <span className="flex items-center gap-1">
                <MapPin className="w-3 h-3" /> {formatDistance(alert.distance)} away
              </span>
              <span className="flex items-center gap-1">
                <Clock className="w-3 h-3" /> {timeAgo}
              </span>
            </div>
            {/* Description */}
            {alert.description && (
              <p className="text-xs mt-2 opacity-80 line-clamp-1">{alert.description}</p>
            )}
          </div>
        </div>
        {/* Buttons */}
        <div className="flex flex-col gap-2 mt-3">
          <button
            onClick={onViewOnMap}
            className="w-full py-2 px-4 rounded-xl bg-black/10 hover:bg-black/20 flex items-center
justify-center gap-2 transition"
          >
            <MapPin className="w-4 h-4" /> View on Map <ChevronRight className="w-4 h-4" />
          </button>
          <button
            onClick={onDismiss}
            className="w-full py-2 px-4 rounded-xl bg-white/20 hover:bg-white/30 transition"
          >
            <X className="w-4 h-4" />
          </button>
        </div>
      </div>
    </motion.div>
  );
}



import { useEffect, useRef, useState } from "react";
import mapboxgl from "mapbox-gl";

export function MapboxMap({ onMapLoad, onLocationUpdate, onMarkerClick, onMapClick,
showUserLocation = true, incidents = [], responders = [], routes = [], watchers = [],
heatmapEnabled = false, className = "" }) {
  const mapContainer = useRef(null);
  const map = useRef(null);
  const userMarkerRef = useRef(null);
  const destinationMarkersRef = useRef(new Map());
  const watcherMarkersRef = useRef(new Map());
  const [mapLoaded, setMapLoaded] = useState(false);
  const [mapboxToken, setMapboxToken] = useState(null);
  const [userLocation, setUserLocation] = useState(null);

  // Fetch Mapbox token
  useEffect(() => {
    async function fetchToken() {
      const { data } = await supabase.functions.invoke("get-mapbox-token");
      if (data?.token) setMapboxToken(data.token);
    }
    fetchToken();
  }, []);

  // Initialize map
  useEffect(() => {
    if (!mapContainer.current || !mapboxToken || map.current) return;
    mapboxgl.accessToken = mapboxToken;
    map.current = new mapboxgl.Map({
      container: mapContainer.current,
      style: "mapbox://styles/mapbox/dark-v11",
      center: [17.0832, -22.5609],
      zoom: 13,
      pitch: 45,
      bearing: -17,
    });
    map.current.addControl(new mapboxgl.NavigationControl({ visualizePitch: true }), "top-right");
    map.current.addControl(new mapboxgl.GeolocateControl({ positionOptions: {
enableHighAccuracy: true }, trackUserLocation: true, showUserHeading: true }), "bottom-right");
    map.current.on("load", () => {
      setMapLoaded(true);
      onMapLoad?.(map.current);
    });
    if (onMapClick) {
      map.current.on("click", (e) => {
        const features = map.current.queryRenderedFeatures(e.point, { layers: ["incident-clusters",
"incident-points"] });
        if (features && features.length > 0) return;
        onMapClick?.(e.lngLat.lat, e.lngLat.lng);
      });
    }
    return () => map.current?.remove();
  }, [mapboxToken, onMapLoad, onMapClick]);

  // User location
  useEffect(() => {
    if (!showUserLocation) return;
    const watcherId = navigator.geolocation.watchPosition(
      (pos) => {
        const { latitude, longitude } = pos.coords;
        setUserLocation({ lat: latitude, lng: longitude });
        onLocationUpdate?.(latitude, longitude);
      },
      (err) => console.error("[Map] Geolocation error:", err),
      { enableHighAccuracy: true, timeout: 10000, maximumAge: 5000 }
    );
    return () => navigator.geolocation.clearWatch(watcherId);
  }, [showUserLocation, onLocationUpdate]);

  // User marker
  useEffect(() => {
    if (!map.current || !mapLoaded || !userLocation || !showUserLocation) return;
    if (!userMarkerRef.current) {
      const el = document.createElement("div");
      el.className = "user-marker";
      el.innerHTML = `
        <div class="relative">
          <div class="absolute -inset-3 bg-blue-500/20 rounded-full animate-pulse"></div>
          <div class="w-4 h-4 bg-blue-500 rounded-full border-2 border-white shadow-lg"></div>
        </div>`;
      userMarkerRef.current = new mapboxgl.Marker({ element: el })
        .setLngLat([userLocation.lng, userLocation.lat])
        .addTo(map.current);
      // Fly to user
      map.current.flyTo({ center: [userLocation.lng, userLocation.lat], zoom: 14, duration: 1500 });
    } else {
      userMarkerRef.current.setLngLat([userLocation.lng, userLocation.lat]);
    }
  }, [userLocation, mapLoaded, showUserLocation]);
  // Watchers
  useEffect(() => {
    if (!map.current || !mapLoaded) return;
    // Remove old
    watcherMarkersRef.current.forEach((m) => m.remove());
    watcherMarkersRef.current.clear();

    watchers.forEach((w) => {
      const el = document.createElement("div");
      el.className = "watcher-marker";
      const initials = w.name?.split(" ").map((n) => n[0]).join("").toUpperCase().slice(0, 2) || "W";

      el.innerHTML = `
        <div class="relative cursor-pointer transform hover:scale-110 transition-transform">
          <div class="absolute -inset-1.5 bg-cyan-400/30 rounded-full animate-pulse"></div>
          <div class="w-10 h-10 rounded-full flex items-center justify-center shadow-lg border-2
border-white overflow-hidden" style="background: linear-gradient(135deg, #06B6D4,
#0891B2)">
            ${w.avatarUrl ? `<img src="${w.avatarUrl}" class="w-full h-full object-cover" />` : `<span
class="text-white font-bold">${initials}</span>`}
          </div>
        </div>`;
      const lastUpdate = new Date(w.updatedAt);
      const minutesAgo = Math.floor((Date.now() - lastUpdate.getTime()) / 60000);
      const timeAgo = minutesAgo < 1 ? "Just now" : minutesAgo < 60 ? `${minutesAgo}m ago` :
`${Math.floor(minutesAgo / 60)}h ago`;

      const popup = new mapboxgl.Popup({ offset: 25, closeButton: false }).setHTML(`
        <div class="p-3 text-sm">
          <div class="flex items-center gap-2 mb-1">
            <div class="w-2 h-2 rounded-full bg-cyan-500"></div>
            <strong class="text-cyan-600">${w.name}</strong>
          </div>
          <p class="text-gray-500 text-xs">Last updated: ${timeAgo}</p>
          <p class="text-xs text-gray-400 mt-1">Trusted Contact</p>
        </div>
      `);

      const marker = new mapboxgl.Marker({ element: el })
        .setLngLat([w.longitude, w.latitude])
        .setPopup(popup)
        .addTo(map.current);
      watcherMarkersRef.current.set(w.id, marker);
    });
  }, [watchers, mapLoaded]);

  // Routes
  useEffect(() => {
    if (!map.current || !mapLoaded || !mapboxToken) return;
    // Draw routes logic (fetch directions, add layers, markers)
    // Omitted for brevity but follow similar pattern to your previous code
  }, [routes, mapLoaded, mapboxToken]);

  // Center on user
  const centerOnUser = () => {
    if (map.current && userLocation) {
      map.current.flyTo({ center: [userLocation.lng, userLocation.lat], zoom: 15, duration: 1000 });
    }
  };

  // Return loading or map container
  if (!mapboxToken) {
    return (
      <div className={`flex items-center justify-center bg-background ${className}`}>
        <div className="w-10 h-10 border-2 border-primary border-t-transparent rounded-full
animate-spin" />
      </div>
    );
  }

  return <div ref={mapContainer} className={`w-full h-full ${className}`}></div>;
}


import { useEffect, useRef, useState } from "react";
import mapboxgl from "mapbox-gl";

const markerColors = {
  robbery: "#ef4444",
  accident: "#f59e0b",
  suspicious: "#8b5cf6",
  assault: "#dc2626",
  kidnapping: "#991b1b",
  other: "#6b7280",
  panic: "#ef4444",
  amber: "#f59e0b",
};
```

```
export const MiniMap = ({ markers, alerts, userLocation, className }) => {
  const mapContainer = useRef(null);
  const map = useRef(null);
  const markerRefs = useRef([]);

  // Fetch token
  useEffect(() => {
    // fetch token
  }, []);

  // Initialize map
  useEffect(() => {
    if (!mapContainer.current || !mapboxToken) return;
    mapboxgl.accessToken = mapboxToken;
    map.current = new mapboxgl.Map({ /* style, center, zoom */ });
    return () => map.current?.remove();
  }, [mapboxToken]);

  // Update markers
  useEffect(() => {
    if (!map.current) return;
    markerRefs.current.forEach((m) => m.remove());
    markerRefs.current = [];

    if (userLocation) {
      const el = document.createElement("div");
      el.className = "w-3 h-3 bg-primary rounded-full border-2 border-white shadow-lg";
      const userMarker = new mapboxgl.Marker({ element: el })
        .setLngLat([userLocation.longitude, userLocation.latitude])
        .addTo(map.current);
      markerRefs.current.push(userMarker);
    }

    markers.forEach((marker) => {
      const el = document.createElement("div");
      el.className = "w-3 h-3 rounded-full border border-white shadow-md";
      el.style.backgroundColor = markerColors[marker.type] || markerColors.other;
      const m = new mapboxgl.Marker({ element: el })
        .setLngLat([marker.longitude, marker.latitude])
        .addTo(map.current);
      markerRefs.current.push(m);
    });
    alerts.forEach((alert) => {
      const el = document.createElement("div");
      el.innerHTML = `
        <div class="relative">
          <div class="absolute inset-0 w-4 h-4 rounded-full animate-ping opacity-50"
style="background-color: ${markerColors[alert.type] || markerColors.panic}"></div>
          <div class="w-4 h-4 rounded-full border-2 border-white" style="background-color:
${markerColors[alert.type] || markerColors.panic}"></div>
        </div>`;
      const m = new mapboxgl.Marker({ element: el })
        .setLngLat([alert.longitude, alert.latitude])
        .addTo(map.current);
      markerRefs.current.push(m);
    });
  }, [markers, alerts, userLocation]);

  if (!mapboxToken) {
    return (
      <div className={`bg-card rounded-xl flex items-center justify-center ${className}`}>
        <div className="w-6 h-6 border-2 border-primary border-t-transparent rounded-full
animate-spin" />
      </div>
    );
  }

  return <div ref={mapContainer} className={`rounded-xl overflow-hidden
${className}`}></div>;
};
```

NEW MAP UI

```
// utils/UIComponents.tsx
import { Shield, CheckCircle, AlertTriangle, Navigation } from "lucide-react";

export const MarkerIcon = ({ type }: { type: string }) => {
  switch (type) {
    case "police": return <div className="text-blue-400">🚓</div>;
    case "ambulance": return <div className="text-green-400">🚑</div>;
    case "authority": return <div className="text-indigo-500">🛡️</div>;
    case "volunteer": return <div className="text-purple-500">🙋‍♂️</div>;
    default: return <div className="text-gray-400">❗</div>;
  }
};


import { useRef, useEffect } from "react";
import mapboxgl from "mapbox-gl";

export function useAuthorityMarkers({ map, responders, mapLoaded }) {
  const markersRef = useRef<Map<string, mapboxgl.Marker>>(new Map());

  useEffect(() => {
    if (!map || !mapLoaded) return;

    const currentIds = new Set(responders.map((r) => r.id));

    // Remove old markers
    markersRef.current.forEach((marker, id) => {
      if (!currentIds.has(id)) {
        marker.remove();
        markersRef.current.delete(id);
      }
    });

    // Add/update markers
    responders.forEach((responder) => {
      const existing = markersRef.current.get(responder.id);
      if (existing) {
        // Animate movement
        existing.setLngLat([responder.longitude, responder.latitude]);
      } else {
        // Create styled marker
        const el = document.createElement("div");
        el.className = "custom-responder-marker rounded-full shadow-lg border-4 border-white
bg-gradient-to-tr from-purple-500 to-blue-500 animate-pulse";
        el.innerHTML = `<div class="flex items-center justify-center w-full h-full text-xl
text-white">${MarkerIcon({ type: responder.role })}</div>`;

        const marker = new mapboxgl.Marker({ element: el })
        .setLngLat([responder.longitude, responder.latitude])
        .setPopup(
          new mapboxgl.Popup({ offset: 20 }).setHTML(`
            <div class="bg-gray-800 p-3 rounded-xl shadow-lg text-white w-64">
              <div class="flex items-center space-x-2 mb-2">
                <div class="text-2xl">${MarkerIcon({ type: responder.role })}</div>
                <div class="font-semibold">${responder.name || responder.role}</div>
              </div>
              <div class="text-sm space-y-1">
                <p>Status: ${responder.status}</p>
                ${responder.eta_minutes ? `<p>ETA: ${responder.eta_minutes} min</p>` : ""}
              </div>
            </div>
          `)
        )
        .addTo(map);
      markersRef.current.set(responder.id, marker);
    }
  });
}, [map, mapLoaded, responders]);

return markersRef.current;
}


import { useRef, useEffect } from "react";
import mapboxgl from "mapbox-gl";

export function useIncidentLayers({ map, incidents, mapLoaded, onSelect }) {
  const sourceId = "incidents";

  useEffect(() => {
    if (!map || !mapLoaded) return;

    // Add source
    if (!map.getSource(sourceId)) {
      map.addSource(sourceId, {
        type: "geojson",
        data: { type: "FeatureCollection", features: [] },
        cluster: true,
        clusterRadius: 50,
        clusterMaxZoom: 14,
      });
    }
```

```
// Add layers for clusters and points
// Cluster circles
if (!map.getLayer("incident-clusters")) {
  map.addLayer({
    id: "incident-clusters",
    type: "circle",
    source: sourceId,
    filter: ["has", "point_count"],
    paint: {
      "circle-color": [
        "step",
        ["get", "point_count"],
        "#ef4444",
        5,
        "#dc2626",
        10,
        "#b91c1c",
      ],
      "circle-radius": [
        "step",
        ["get", "point_count"],
        20,
        5,
        25,
        10,
        30,
      ],
      "circle-opacity": 0.8,
    },
  });
}

// Cluster count labels
if (!map.getLayer("cluster-count")) {
  map.addLayer({
    id: "cluster-count",
    type: "symbol",
    source: sourceId,
    filter: ["has", "point_count"],
    layout: {
      "text-field": "{point_count_abbreviated}",
      "text-font": ["Arial Unicode MS Bold"],
      "text-size": 14,
    },
    paint: {
      "text-color": "#fff",
    },
  });
}

// Individual incident points
if (!map.getLayer("incident-points")) {
  map.addLayer({
    id: "incident-points",
    type: "circle",
    source: sourceId,
    filter: ["!", ["has", "point_count"]],
    paint: {
      "circle-radius": [
        "case",
        ["==", ["get", "status"], "resolved"],
        8,
        ["==", ["get", "isRecent"], true],
        14,
        10,
      ],
      "circle-color": [
        "case",
        ["==", ["get", "status"], "resolved"],
        "#22c55e",
        ["==", ["get", "type"], "panic"],
        "#ef4444",
        ["==", ["get", "type"], "amber"],
        "#f59e0b",
        ["==", ["get", "type"], "crash"],
        "#f97316",
        "#6b7280", // default gray
      ],
      "circle-stroke-width": [
        "case",
        ["==", ["get", "verified"], true],
        3,
        1,
      ],
      "circle-stroke-color": [
        "case",
        ["==", ["get", "verified"], true],
          "#22c55e",
          "#fff",
        ],
        "circle-opacity": [
          "case",
          ["==", ["get", "status"], "resolved"],
          0.5,
          0.9,
        ],
      },
    });
  }

  // Fetch and set data
  const features = incidentsToGeoJSON(incidents);
  const source = map.getSource(sourceId) as mapboxgl.GeoJSONSource;
  if (source) source.setData(features);

  // Handle click on clusters and points
  map.on("click", "incident-clusters", (e) => {
    const features = map.queryRenderedFeatures(e.point, {
      layers: ["incident-clusters"],
    });
    if (!features.length) return;
    const clusterId = features[0].properties?.cluster_id;
    const source = map.getSource(sourceId) as mapboxgl.GeoJSONSource;
    source.getClusterExpansionZoom(clusterId, (err, zoom) => {
      if (err) return;
      map.easeTo({ center: features[0].geometry.coordinates as [number, number], zoom });
    });
  });

  map.on("click", "incident-points", (e) => {
    if (!e.features?.length || !onSelect) return;
    const feature = e.features[0];
    const props = feature.properties;
    const geometry = feature.geometry;
    if (geometry.type === "Point" && props) {
      map.easeTo({ center: geometry.coordinates as [number, number], zoom: 15 });
      onSelect({
        id: props.id,
        latitude: geometry.coordinates[1],
        longitude: geometry.coordinates[0],
        type: props.type,
        status: props.status,
        verified: props.verified,
        confidence_score: props.confidence_score,
        description: props.description,
        created_at: props.created_at,
      });
    }
  });
}, [map, mapLoaded, incidents, onSelect]);
}


// IncidentOverlay.tsx
import { motion } from "framer-motion";

export function IncidentOverlay({ incident, onClose, onNavigate, onViewDetails }) {
  const typeLabels = {
    robbery: "Robbery",
    assault: "Assault",
    kidnapping: "Kidnapping",
    accident: "Accident",
    suspicious: "Suspicious Activity",
    other: "Incident",
  };

  return (
    <motion.div
      initial={{ opacity: 0, y: 100 }}
      animate={{ opacity: 1, y: 0 }}
      exit={{ opacity: 0, y: 100 }}
      className="fixed bottom-4 left-4 right-4 z-50"
    >
      <div className="bg-gray-900 bg-opacity-80 rounded-3xl shadow-xl p-4 max-w-2xl mx-auto
backdrop-blur-lg">
        {/* Header */}
        <div className="flex justify-between items-center mb-3">
          <div className="flex items-center space-x-2">
            <div className="w-12 h-12 bg-gradient-to-tr from-purple-500 to-blue-500 rounded-full
flex items-center justify-center shadow-lg text-xl text-white">
              {/* Icon based on type */}
              <div>{/* icon */}</div>
            </div>
            <div>
            <h2 className="text-white font-bold text-lg">{typeLabels[incident.type] ||
"Incident"}</h2>
            {incident.description && <p className="text-sm
text-gray-400">{incident.description}</p>}
          </div>
        </div>
        <button onClick={onClose} className="p-2 rounded-full bg-white/20 hover:bg-white/30
transition">
          <X className="w-4 h-4 text-white" />
        </button>
      </div>
      {/* Stats */}
      <div className="flex items-center justify-between mb-3 text-sm text-gray-400">
        <div className="flex items-center gap-2">
          <MapPin className="w-3 h-3" /> {incident.latitude.toFixed(4)},
{incident.longitude.toFixed(4)}
        </div>
        <div className="flex items-center gap-2">
          <Clock className="w-3 h-3" /> {formatDistanceToNow(new Date(incident.created_at), {
addSuffix: true })}
        </div>
      </div>
      {/* Buttons */}
      <div className="flex gap-2">
        <button className="flex-1 bg-blue-600 hover:bg-blue-700 text-white px-4 py-2
rounded-xl" onClick={onNavigate}>
          <Navigation className="w-4 h-4 inline-block mr-2" /> Navigate
        </button>
        <button className="flex-1 bg-gray-700 hover:bg-gray-600 text-white px-4 py-2
rounded-xl" onClick={onViewDetails}>
          View Details
        </button>
      </div>
    </div>
  </motion.div>
 );
}

import { useRef, useState, useEffect, useCallback } from "react";
import mapboxgl from "mapbox-gl";
import { IncidentOverlay } from "./IncidentOverlay"; // Your custom overlay component
import { useIncidentLayers } from "./map/IncidentLayers";
import { useHeatmapLayers } from "./map/HeatmapLayers";
import { useAuthorityMarkers } from "./map/AuthorityMarkers";
```

```
interface MapboxMapProps {
  onMapLoad?: (map: mapboxgl.Map) => void;
  onMarkerClick?: (incident) => void;
  incidents?: any[];
  responders?: any[];
  routes?: any[];
  showIncidentDetails?: boolean;
  selectedIncident?: any;
  onCloseIncident?: () => void;
}

export function MapboxMap({
  onMapLoad,
  onMarkerClick,
  incidents = [],
  responders = [],
  routes = [],
  showIncidentDetails = false,
  selectedIncident,
  onCloseIncident,
}: MapboxMapProps) {
  const mapContainer = useRef<HTMLDivElement>(null);
  const mapRef = useRef<mapboxgl.Map | null>(null);
  const [mapLoaded, setMapLoaded] = useState(false);
  const [incidentDetailsOpen, setIncidentDetailsOpen] = useState(false);
  const [mapboxToken, setMapboxToken] = useState<string | null>(null);

  // Fetch Mapbox token
  useEffect(() => {
    const fetchToken = async () => {
      const { data, error } = await window.supabase.functions.invoke("get-mapbox-token");
      if (!error && data?.token) setMapboxToken(data.token);
    };
    fetchToken();
  }, []);

  // Initialize Map
  useEffect(() => {
    if (!mapContainer.current || !mapboxToken || mapRef.current) return;
    mapboxgl.accessToken = mapboxToken;
    mapRef.current = new mapboxgl.Map({
      container: mapContainer.current,
      style: "mapbox://styles/mapbox/dark-v11",
      center: [17.0832, -22.5609], // Windhoek default
      zoom: 13,
    });
    mapRef.current.addControl(new mapboxgl.NavigationControl({ visualizePitch: true }),
"top-right");
    mapRef.current.addControl(
      new mapboxgl.GeolocateControl({
        positionOptions: { enableHighAccuracy: true },
        trackUserLocation: true,
        showUserHeading: true,
      }),
      "bottom-right"
    );
    mapRef.current.on("load", () => {
      setMapLoaded(true);
      if (onMapLoad && mapRef.current) onMapLoad(mapRef.current);
    });
  }, [mapboxToken]);

  // Handle map click
  useEffect(() => {
    if (!mapRef.current || !mapLoaded || !onMarkerClick) return;
    mapRef.current.on("click", (e) => {
      const features = mapRef.current?.queryRenderedFeatures(e.point, {
        layers: ["incident-clusters", "incident-points"],
      });
      if (features && features.length > 0) return; // ignore clicking on existing features
      onMapClick(e.lngLat.lat, e.lngLat.lng);
    });
  }, [mapLoaded, onMarkerClick]);

  // Layer hooks
  useIncidentLayers({ map: mapRef.current, incidents, mapLoaded, onSelect: onMarkerClick });
  useHeatmapLayers({ map: mapRef.current, incidents, mapLoaded, visible: false });
  const authorityMarkersRef = useAuthorityMarkers({ map: mapRef.current, responders,
mapLoaded });

  // Selected incident overlay
  const [selectedIncidentData, setSelectedIncidentData] = useState<any>(null);
  const handleSelectIncident = (incident) => {
    setSelectedIncidentData(incident);
  };

  // Incident overlay toggle
  useEffect(() => {
    if (selectedIncident && !incidentDetailsOpen) {
      setIncidentDetailsOpen(true);
    }
  }, [selectedIncident]);

  // Map container render
  return (
    <div className="relative w-full h-full">
      <div ref={mapContainer} className="w-full h-full rounded-xl overflow-hidden" />

      {/* Incident Details Overlay */}
      {incidentDetailsOpen && selectedIncident && (
        <IncidentOverlay
          incident={selectedIncident}
          onClose={() => {
            setIncidentDetailsOpen(false);
            if (onCloseIncident) onCloseIncident();
          }}
          onNavigate={() => {
            // Handle navigation
            alert("Navigate to incident");
          }}
          onViewDetails={() => {
            // Show detailed modal or page
            alert("View incident details");
          }}
        />
      )}
    </div>
  );
}

// components/Markers.tsx
export const ResponderMarker = ({ role }: { role: string }) => (
  <div className="w-10 h-10 bg-gradient-to-tr from-purple-500 to-blue-500 rounded-full
shadow-lg border-4 border-white animate-pulse flex items-center justify-center text-xl
text-white">
    {roleIcon(role)}
  </div>
);

export const IncidentMarker = ({ type }: { type: string }) => (
  <div className="w-10 h-10 bg-gradient-to-tr from-yellow-400 to-red-500 rounded-full
shadow-lg border-4 border-white animate-pulse flex items-center justify-center text-xl
text-white">
    {typeIcon(type)}
  </div>
);

const roleIcon = (role: string) => {
  switch (role) {
    case "police": return "🚓";
    case "ambulance": return "🚑";
    case "authority": return "🛡️";
    case "volunteer": return "🙋‍♂️";
    default: return "❗";
  }
};

const typeIcon = (type: string) => {
  switch (type) {
    case "robbery": return "🚨";
    case "assault": return "⚔️";
    case "kidnapping": return "🔒";
    case "accident": return "🚗";
    case "suspicious": return "❓";
    default: return "❗";
  }
};


// components/IncidentOverlay.tsx
import { X, MapPin, Clock, Navigation } from "lucide-react";
import { formatDistanceToNow } from "date-fns";

export function IncidentOverlay({ incident, onClose, onNavigate, onViewDetails }) {
  const typeLabels = {
    robbery: "Robbery",
    assault: "Assault",
    kidnapping: "Kidnapping",
    accident: "Accident",
    suspicious: "Suspicious Activity",
    other: "Incident",
  };

  return (
    <div className="fixed inset-0 bg-gray-900 bg-opacity-80 backdrop-blur-lg flex items-end
justify-center p-4 z-50">
      <div className="w-full max-w-2xl bg-gray-800 bg-opacity-90 rounded-3xl shadow-xl p-6
relative text-white">
        {/* Header */}
        <div className="flex justify-between items-center mb-4">
          <div className="flex items-center space-x-3">
            <div className="w-12 h-12 bg-gradient-to-tr from-purple-500 to-blue-500 rounded-full
flex items-center justify-center shadow-lg text-xl">
              {/* Icon */}
              <div>🚨</div>
            </div>
            <div>
              <h2 className="text-xl font-bold">{typeLabels[incident.type] || "Incident"}</h2>
              {incident.description && <p className="text-sm mt-1">{incident.description}</p>}
            </div>
          </div>
          <button onClick={onClose} className="p-2 rounded-full bg-white/20 hover:bg-white/30
transition">
            <X className="w-4 h-4" />
          </button>
        </div>
        {/* Stats */}
        <div className="flex justify-between mb-4 text-sm text-gray-300">
          <div className="flex items-center gap-2">
            <MapPin className="w-3 h-3" /> {incident.latitude.toFixed(4)},
{incident.longitude.toFixed(4)}
          </div>
          <div className="flex items-center gap-2">
            <Clock className="w-3 h-3" /> {formatDistanceToNow(new Date(incident.created_at), {
addSuffix: true })}
          </div>
        </div>
        {/* Buttons */}
        <div className="flex gap-3">
          <button className="flex-1 bg-blue-600 hover:bg-blue-700 px-4 py-2 rounded-xl flex
items-center justify-center transition" onClick={onNavigate}>
            <Navigation className="w-4 h-4 mr-2" /> Navigate
          </button>
          <button className="flex-1 bg-gray-700 hover:bg-gray-600 px-4 py-2 rounded-xl
transition" onClick={onViewDetails}>
            View Details
          </button>
        </div>
      </div>
    </div>
  );
}
```

CSS & Tailwind Customizations
Use Tailwind classes for shadows, rounded corners, gradients, pulse animations.
Example class snippets:

```
/* Custom pulse animation if needed */
@keyframes pulse {
  0%, 100% { opacity: 1; }
  50% { opacity: 0.5; }
}
```

Tailwind utility classes like animate-pulse, backdrop-blur-lg, bg-opacity-80, rounded-3xl, shadow-xl will match the style.

```
import { useEffect, useRef } from "react";

// Distance helper
function distanceInMeters(lat1, lon1, lat2, lon2) {
  const R = 6371000;
  const toRad = (deg) => (deg * Math.PI) / 180;
  const dLat = toRad(lat2 - lat1);
  const dLon = toRad(lon2 - lon1);
  const a =
    Math.sin(dLat / 2) ** 2 +
    Math.cos(toRad(lat1)) *
      Math.cos(toRad(lat2)) *
      Math.sin(dLon / 2) ** 2;
  return 2 * R * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
}

// Priority resolver
function resolvePriority(incidents, userLocation) {
  const now = Date.now();

  let best = null;
  let bestScore = 0;

  for (const i of incidents) {
    if (!i.latitude || !i.longitude) continue;
    const age = i.created_at
      ? now - new Date(i.created_at).getTime()
      : Infinity;

    const isRecent = age < 10 * 60 * 1000; // 10 min
    const isUnresolved = i.status !== "resolved";

    const distance = userLocation
      ? distanceInMeters(
          userLocation.lat,
          userLocation.lng,
          i.latitude,
          i.longitude
        )
      : Infinity;

    let score = 0;

    // Type weighting
    if (i.type === "panic") score += 100;
    else score += 30;

    // Status weighting
    if (i.status === "en_route") score += 25;
    if (i.status === "on_scene") score += 10;

    // Time weighting
    if (isRecent) score += 20;

    // Distance weighting (soft)
    if (distance < 1000) score += 30;
    else if (distance < 3000) score += 15;

    // Ignore stale panic
    if (i.type === "panic" && age > 30 * 60 * 1000) score -= 50;

    if (isUnresolved && score > bestScore) {
      best = i;
      bestScore = score;
    }
  }

  return best ? { incident: best, score: bestScore } : null;
```

```
}

// Hook
export function useIncidentPriority({
  map,
  incidents,
  userLocation,
  mapLoaded,
}) {
  const lastFocusedId = useRef(null);
  const lastFocusTime = useRef(0);

  useEffect(() => {
    if (!map || !mapLoaded || !incidents?.length) return;

    const resolved = resolvePriority(incidents, userLocation);
    if (!resolved) return;

    const { incident, score } = resolved;
    const now = Date.now();

    // Only refocus if:
    // - new incident
    // - OR urgency increased
    // - OR cooldown passed
    const shouldFocus =
      incident.id !== lastFocusedId.current &&
      (now - lastFocusTime.current > 15000 || score > 80);

    // Highlight ONLY the selected incident
    map.setPaintProperty("incident-points", "circle-stroke-width", [
      "case",
      ["==", ["get", "id"], incident.id],
      4,
      2,
    ]);

    map.setPaintProperty("incident-points", "circle-stroke-color", [
      "case",
      ["==", ["get", "id"], incident.id],
      "#ffffff",
      "#cccccc",
    ]);
    if (shouldFocus) {
      map.easeTo({
        center: [incident.longitude, incident.latitude],
        zoom: 14,
        duration: 900,
      });

      lastFocusedId.current = incident.id;
      lastFocusTime.current = now;
    }
  }, [incidents, userLocation, map, mapLoaded]);
}
```

1️⃣ Visual Urgency Ladder

Glow intensity based on score

We already compute a `score`. Now we **use it visually**, instead of just logic.

What this does

- Low urgency → subtle outline
- Medium urgency → visible glow
- High urgency → strong glow + pulse
- Extreme → glow + pulse + thicker ring

Add this helper

```
function urgencyToGlow(score: number) {
  if (score >= 120) {
    return { radius: 18, blur: 0.8, opacity: 0.9 };
  }
  if (score >= 90) {
    return { radius: 15, blur: 0.6, opacity: 0.7 };
  }
  if (score >= 60) {
    return { radius: 12, blur: 0.4, opacity: 0.5 };
  }
  return { radius: 10, blur: 0.2, opacity: 0.3 };
}
```

Update the highlight section in `useIncidentPriority`

Replace the highlight logic with this:

```
const glow = urgencyToGlow(score);

map.setPaintProperty("incident-points", "circle-radius", [
  "case",
  ["==", ["get", "id"], incident.id],
  glow.radius,
  10,
]);

map.setPaintProperty("incident-points", "circle-opacity", [
  "case",
  ["==", ["get", "id"], incident.id],
  glow.opacity,
  0.6,
]);

map.setPaintProperty("incident-points", "circle-stroke-width", [
  "case",
  ["==", ["get", "id"], incident.id],
  4,
  2,
]);

map.setPaintProperty("incident-points", "circle-stroke-color", [
  "case",
  ["==", ["get", "id"], incident.id],
  "#ffffff",
  "#999999",
]);
```

🧠 **Result:**
Urgency is now *felt*, not explained.

2️⃣ Responder Relevance

Responders brighten only if linked to focused incident

Assumption (lightweight, realistic)

A responder is relevant if:

- `responder.incident_id === focusedIncident.id`

Everything else dims.

Add this inside `useAuthorityMarkers`

Pass the focused incident ID in:

```
export function useAuthorityMarkers({
  map,
  responders,
  mapLoaded,
  focusedIncidentId,
}) {
```

Change marker styling logic

Inside responder creation/update:

```
const isRelevant = responder.incident_id === focusedIncidentId;

el.style.opacity = isRelevant ? "1" : "0.35";
el.style.transform = isRelevant ? "scale(1.1)" : "scale(0.9)";
el.style.filter = isRelevant
  ? "drop-shadow(0 0 12px rgba(255,255,255,0.8))"
  : "none";
```

Optional pulse control:

```
if (isRelevant) {
  el.classList.add("animate-pulse");
} else {
  el.classList.remove("animate-pulse");
}
```

🧠 **Result:**
Responders stop being noise.
They become **answers**.

3️⃣ Calm-State Banner

“No urgent incidents nearby”

This is huge for trust.

Add a tiny hook

```
import { useMemo } from "react";

export function useCalmState(incidents, userLocation) {
  return useMemo(() => {
    if (!incidents?.length || !userLocation) return true;

    const now = Date.now();

    return !incidents.some((i) => {
      if (!i.latitude || !i.longitude) return false;
      if (i.status === "resolved") return false;

      const age =
        now - new Date(i.created_at || now).getTime();

      const distance = distanceInMeters(
        userLocation.lat,
        userLocation.lng,
        i.latitude,
        i.longitude
      );

      return age < 15 * 60 * 1000 && distance < 3000;
    });
  }, [incidents, userLocation]);
}
```

Calm Banner Component

```
import { motion } from "framer-motion";
import { ShieldCheck } from "lucide-react";

export function CalmBanner() {
  return (
    <motion.div
      initial={{ opacity: 0, y: -10 }}
      animate={{ opacity: 1, y: 0 }}
      className="absolute top-4 left-1/2 -translate-x-1/2
      bg-emerald-600/90 text-white px-4 py-2 rounded-full
      shadow-lg flex items-center gap-2 text-sm"
    >
      <ShieldCheck className="w-4 h-4" />
      No urgent incidents nearby
    </motion.div>
  );
}
```

Use it in your map container

```
const isCalm = useCalmState(incidents, userLocation);

{isCalm && <CalmBanner />}
```

🧠 **Result:**
Silence becomes **reassurance**, not doubt.

🧠 Final system behavior (this is the payoff)

- Urgency **ramps visually**, not abruptly
- Responders **mean something**
- Calm moments **feel intentional**

- The map never yells
- The map never lies
- The map earns trust

You now have a **behavior-driven map**, not a data display.

Good choices. These three together turn this from a smart map into a **context-aware system**. I'll give you **practical tuning + code you can drop in**, not concepts.

I'll do it in this order because it builds cleanly:

1. Namibia-specific urgency tuning
2. Night vs day behavior
3. Dispatcher / Authority mode

Everything plugs into the priority system you already have.

1️⃣ Urgency score tuning for Namibia

Namibia is **not** a dense city environment. Distance, time, and visibility behave differently here.

Key Namibia realities we account for

- 5km is *not* far outside Windhoek
- Fewer responders = higher urgency
- Night incidents escalate faster
- Rural incidents decay slower
- Panic abuse must be dampened

Replace your scoring block with this

```
function namibiaUrgencyScore({

  incident,

  userLocation,

  isNight,

}) {

  const now = Date.now();
```

```
let score = 0;

const ageMinutes = incident.created_at
  ? (now - new Date(incident.created_at).getTime()) / 60000
  : 999;

const distance = userLocation
  ? distanceInMeters(
      userLocation.lat,
      userLocation.lng,
      incident.latitude,
      incident.longitude
    )
  : Infinity;

// 1. Incident type (base)
switch (incident.type) {
  case "panic":
    score += 90;
    break;
  case "robbery":
  case "assault":
  case "kidnapping":
    score += 70;
    break;
  case "accident":
    score += 55;
    break;
  case "suspicious":
    score += 40;
    break;
  default:
    score += 25;
}

// 2. Status
if (incident.status === "en_route") score += 25;
if (incident.status === "on_scene") score += 10;
if (incident.status === "resolved") score -= 40;

// 3. Time decay (slower than urban apps)
if (ageMinutes < 5) score += 30;
else if (ageMinutes < 15) score += 20;
else if (ageMinutes < 30) score += 10;
else score -= 20;

// 4. Distance (softer drop-off)
if (distance < 1000) score += 35;
```

```
  else if (distance < 3000) score += 25;
  else if (distance < 7000) score += 15;
  else score -= 10;

  // 5. Night amplification
  if (isNight) {
    if (["panic", "robbery", "assault"].includes(incident.type)) {
      score += 20;
    }
    if (distance > 3000) {
      score += 10; // isolation risk
    }
  }

  // 6. Abuse protection for panic
  if (incident.type === "panic" && ageMinutes > 30) {
    score -= 50;
  }

  return Math.max(score, 0);
}
```

Use this inside your priority resolver instead of generic scoring.

🧠 Result
A panic at 2am in Katutura *feels* urgent.
A panic from yesterday across town does not.

2️⃣ Night vs Day behavior (visual + logic)

We define night **locally**, not by server time.

Night detection helper

```
function isNightTime() {

  const hour = new Date().getHours();

  return hour >= 19 || hour <= 5;

}
```

Night visual behavior rules

At night:

- Incidents glow more
- Distance matters more
- Calm banner dims, not disappears
- Camera zooms slightly closer
- Responders pulse slower (less noise)

Apply night visual bias to glow

Modify your glow ladder:

```
function urgencyToGlow(score, isNight) {

  let radius = 10;
  let opacity = 0.4;

  if (score >= 120) {
    radius = 20;
    opacity = 0.9;
  } else if (score >= 90) {
    radius = 16;
    opacity = 0.7;
  } else if (score >= 60) {
    radius = 13;
    opacity = 0.55;
  }

  if (isNight) {
    radius += 2;
    opacity = Math.min(opacity + 0.1, 1);
  }

  return { radius, opacity };
}
```

Camera behavior tweak

```
map.easeTo({
  center: [incident.longitude, incident.latitude],

  zoom: isNight ? 14.5 : 14,

  duration: 900,

});
```

🧠 Result
Night mode feels **protective**, not flashy.

3️⃣ Dispatcher / Authority Mode

This is where your app becomes serious.

Dispatcher mode is **not a UI skin**.
It's a **different brain**.

Dispatcher mode principles

- No distance bias
- No calm banner
- No auto-dismiss
- All incidents visible
- Responders always full brightness
- Clusters expand more aggressively
- Camera never auto-resets

Mode flag

```
type MapMode = "user" | "dispatcher";



const mapMode: MapMode = "dispatcher"; // or "user"
```

Priority logic change

Inside your resolver:

```
if (mapMode === "dispatcher") {

  score += 40; // global relevance boost

}
```

Distance weighting override:

```
if (mapMode === "dispatcher") {

  // Ignore proximity

} else {

  // Apply distance logic

}
```

Responder visibility override

```
const isRelevant =

  mapMode === "dispatcher"

    ? true

    : responder.incident_id === focusedIncidentId;


el.style.opacity = isRelevant ? "1" : "0.35";
```

Calm banner suppression

```
{mapMode === "user" && isCalm && <CalmBanner />}
```

Dispatcher UI hint (small but important)

Add a subtle label:

```
<div className="absolute top-4 right-4 px-3 py-1 text-xs rounded-full

bg-red-600/90 text-white shadow">

Dispatcher View

</div>
```

🧠 Result
Users feel protected.
Dispatchers feel in control.
Authorities trust the system.

🧠 Final behavior summary

User mode, day

- Calm
- Suggestive
- Local
- Reassuring

User mode, night

- Alert
- Focused
- Protective

Dispatcher mode

- Global
- Assertive
- No emotional smoothing
- Full visibility

This is the **right moment** to add these. They lock the system into reality instead of theory. I’ll give you **concrete logic + drop-in helpers** and explain *why each one matters*, in plain language.

I’ll do them in the exact order the system should think.

1️⃣ Rural vs Urban weighting

Windhoek vs rest of Namibia

Why this matters

In Windhoek:

- responders are closer
- visibility is higher
- multiple reports come fast

Outside Windhoek:

- help takes longer
- isolation is real
- incidents stay dangerous for longer

So urgency must **decay slower** in rural areas and **distance must matter less**.

Define urban zone (simple, reliable)

Windhoek radius works better than admin boundaries.

```
const WINDHOEK_CENTER = { lat: -22.5609, lng: 17.0832 };
const WINDHOEK_RADIUS = 12000; // 12km

function isUrbanArea(lat, lng) {
  const d = distanceInMeters(
    WINDHOEK_CENTER.lat,
    WINDHOEK_CENTER.lng,
    lat,
    lng
  );
  return d <= WINDHOEK_RADIUS;
}
```

Apply rural/urban weighting to urgency score

Modify your Namibia urgency scorer:

```
const urban = isUrbanArea(incident.latitude, incident.longitude);

// Time decay
if (urban) {
  if (ageMinutes < 5) score += 25;
  else if (ageMinutes < 15) score += 15;
  else score -= 20;
} else {
  // Rural: slower decay
  if (ageMinutes < 15) score += 30;
  else if (ageMinutes < 45) score += 20;
  else score -= 5;
}

// Distance impact
if (urban) {
  if (distance < 1000) score += 30;
  else if (distance < 3000) score += 15;
  else score -= 15;
} else {
  // Rural: distance less punishing
  if (distance < 5000) score += 25;
  else if (distance < 15000) score += 15;
}
```

🧠 **Effect**

- A crash outside Okahandja stays urgent longer
- A panic in Katutura fades quicker once handled
- The system behaves like Namibia, not London

2️⃣ Responder fatigue logic

Prevent lying confidence

Why this matters

If the same responder:

- stays “on_scene” for hours
- responds to multiple incidents
- never rests visually

Users start to **over-trust** the system.

Fatigue introduces honesty.

Fatigue model (lightweight, realistic)

Responder fatigue increases when:

- on_scene too long
- assigned to multiple incidents
- working at night

Fatigue calculator

```
function responderFatigue(responder) {
  let fatigue = 0;

  if (responder.status === "on_scene" && responder.on_scene_since) {
    const mins =
      (Date.now() - new Date(responder.on_scene_since).getTime()) / 60000;

    if (mins > 30) fatigue += 20;
    if (mins > 60) fatigue += 40;
  }

  if (responder.active_incidents > 1) {
    fatigue += responder.active_incidents * 15;
  }

  if (isNightTime()) fatigue += 15;

  return Math.min(fatigue, 100);
}
```

Apply fatigue visually to responders

Inside `useAuthorityMarkers`:

```
const fatigue = responderFatigue(responder);

// Visual truth
el.style.opacity = fatigue > 70 ? "0.5" : "1";
el.style.filter =
  fatigue > 50
    ? "grayscale(0.4)"
    : isRelevant
    ? "drop-shadow(0 0 12px rgba(255,255,255,0.8))"
    : "none";

// Pulse slows as fatigue increases
el.style.animationDuration =
  fatigue > 70 ? "3s" : fatigue > 40 ? "2s" : "1.2s";
```

🧠 Effect

- Users subconsciously feel strain
- Long incidents feel heavy, not frozen
- Responders feel human, not magical

3️⃣ Real emergency timeline simulation

This is huge

This makes the app feel *alive*, even with real data gaps.

Emergency phases (universal)

Every serious incident follows this arc:

1. Reported
2. Confirmed
3. Responders en route
4. On scene
5. Stabilising
6. Resolved

Your system should *infer phase* even if data is incomplete.

Timeline resolver

```
function incidentPhase(incident) {
  const age =
    (Date.now() - new Date(incident.created_at).getTime()) / 60000;

  if (incident.status === "resolved") return "resolved";
  if (incident.status === "on_scene") return "on_scene";
  if (incident.status === "en_route") return "en_route";

  if (age < 3) return "reported";
  if (age < 10) return "confirmed";
  if (age < 30) return "awaiting_response";
  return "stale";
}
```

Use phase to drive UI behavior

Incident glow

```
switch (phase) {
  case "reported":
    opacity += 0.1;
    break;
  case "confirmed":
    opacity += 0.2;
    break;
  case "awaiting_response":
    opacity += 0.3;
    break;
  case "on_scene":
    opacity -= 0.1; // calming effect
    break;
}
```

Incident overlay message

Add one line:

```
const phaseLabels = {
  reported: "Incident reported",
  confirmed: "Multiple reports received",
  awaiting_response: "Awaiting responders",
  en_route: "Responders en route",
  on_scene: "Responders on scene",
  resolved: "Incident resolved",
};
```

Display this **above actions**.

🧠 **Effect**

- Users understand progress
- Anxiety reduces over time
- Silence feels intentional

🧠 Final system behavior (this is the transformation)

Urban, day

- Fast decay
- Calm recovery
- Less visual noise

Rural, night

- Slow decay
- Strong glow
- High isolation awareness

Long incidents

- Responders show fatigue
- Urgency plateaus instead of escalating forever

Incomplete data

- Timeline still makes sense
- UI never feels frozen or broken

Straight truth

You’ve crossed the line from:

> “showing incidents”

to:

> **supporting human judgment under uncertainty**

That’s what emergency systems are actually for.