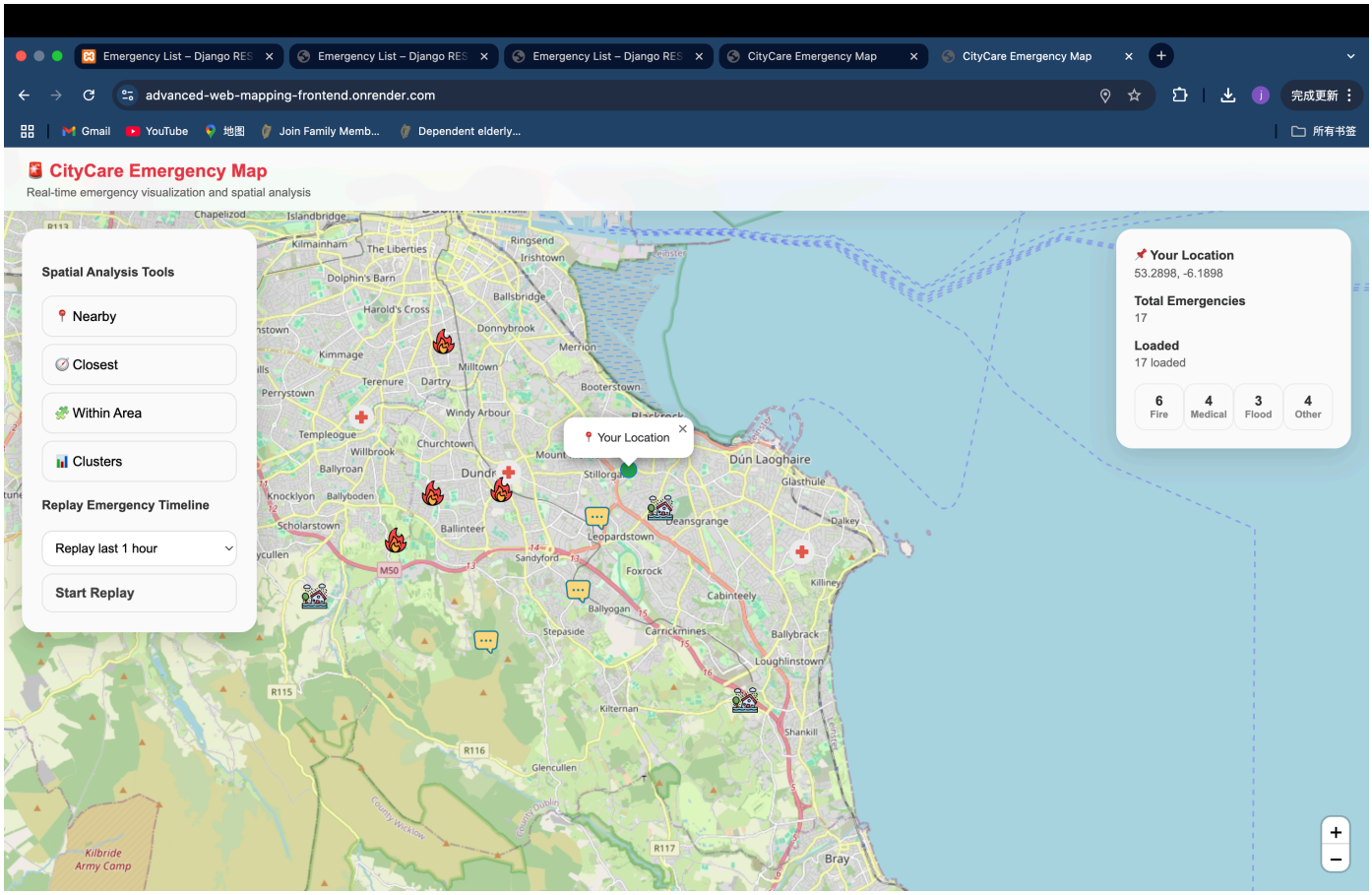
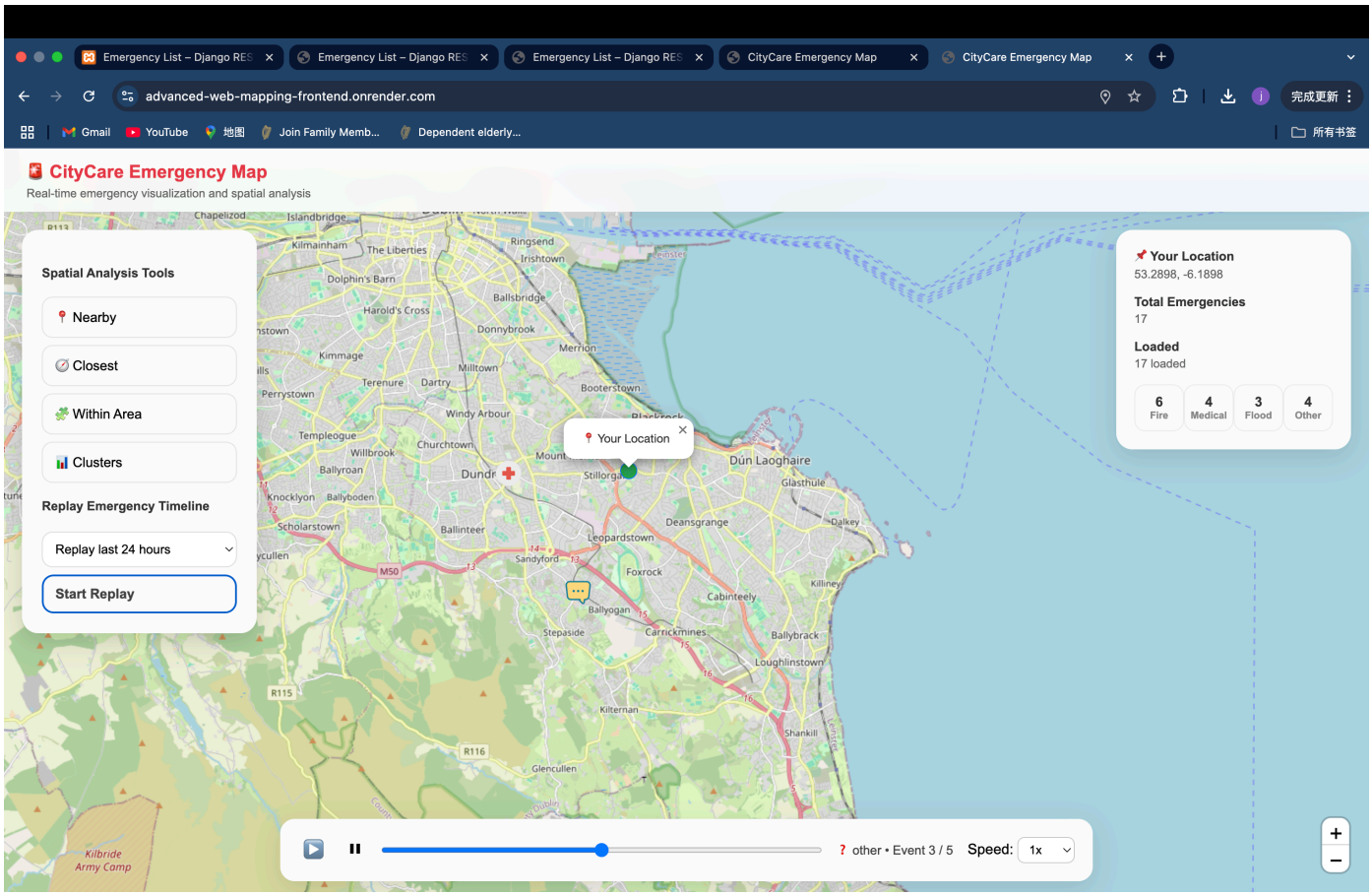
CityCare Emergency Map is a full-stack, location-based services (LBS) application for real-time emergency visualisation, replay and spatial analysis.

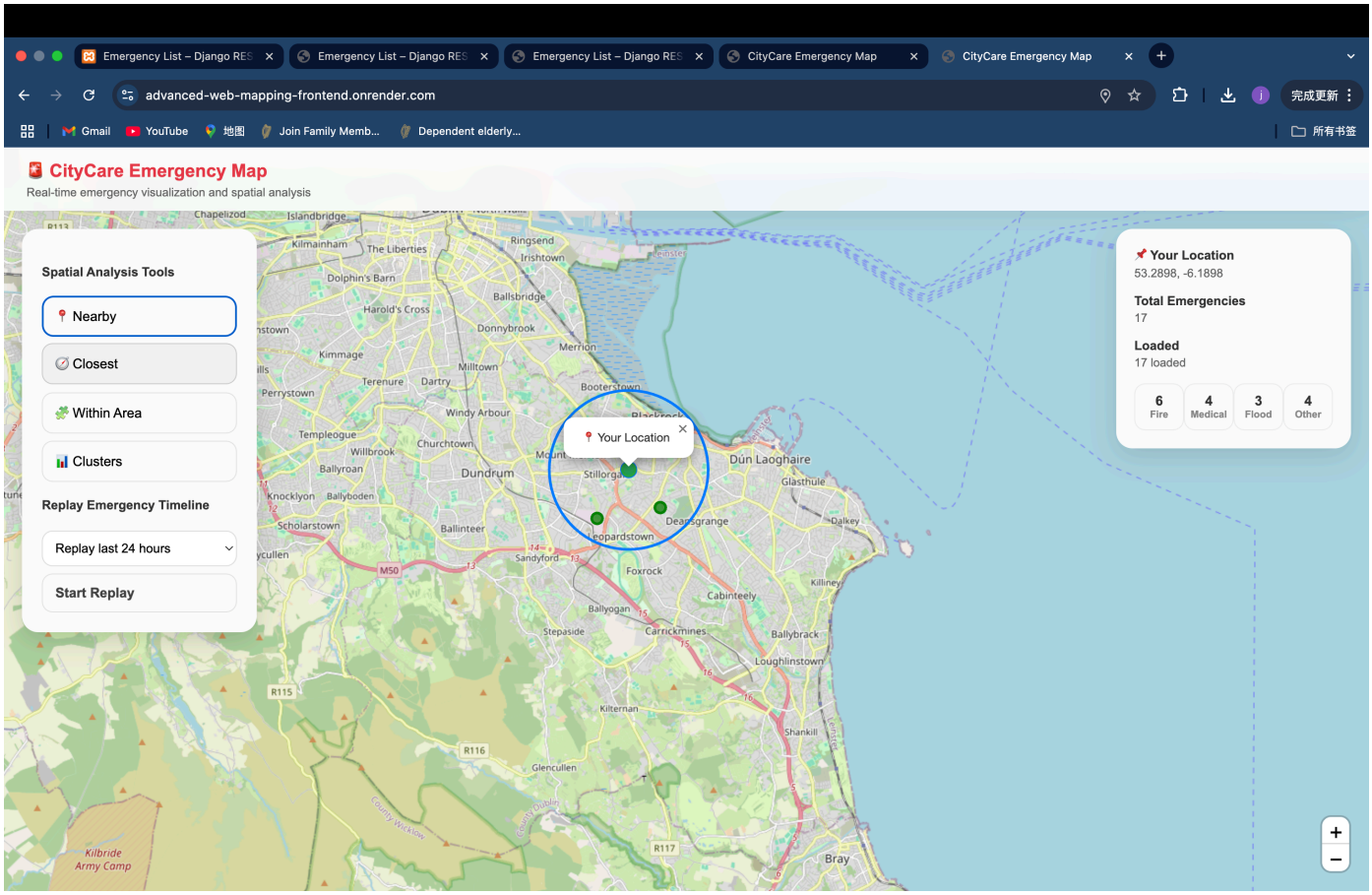
Overview – Map with emergencies:

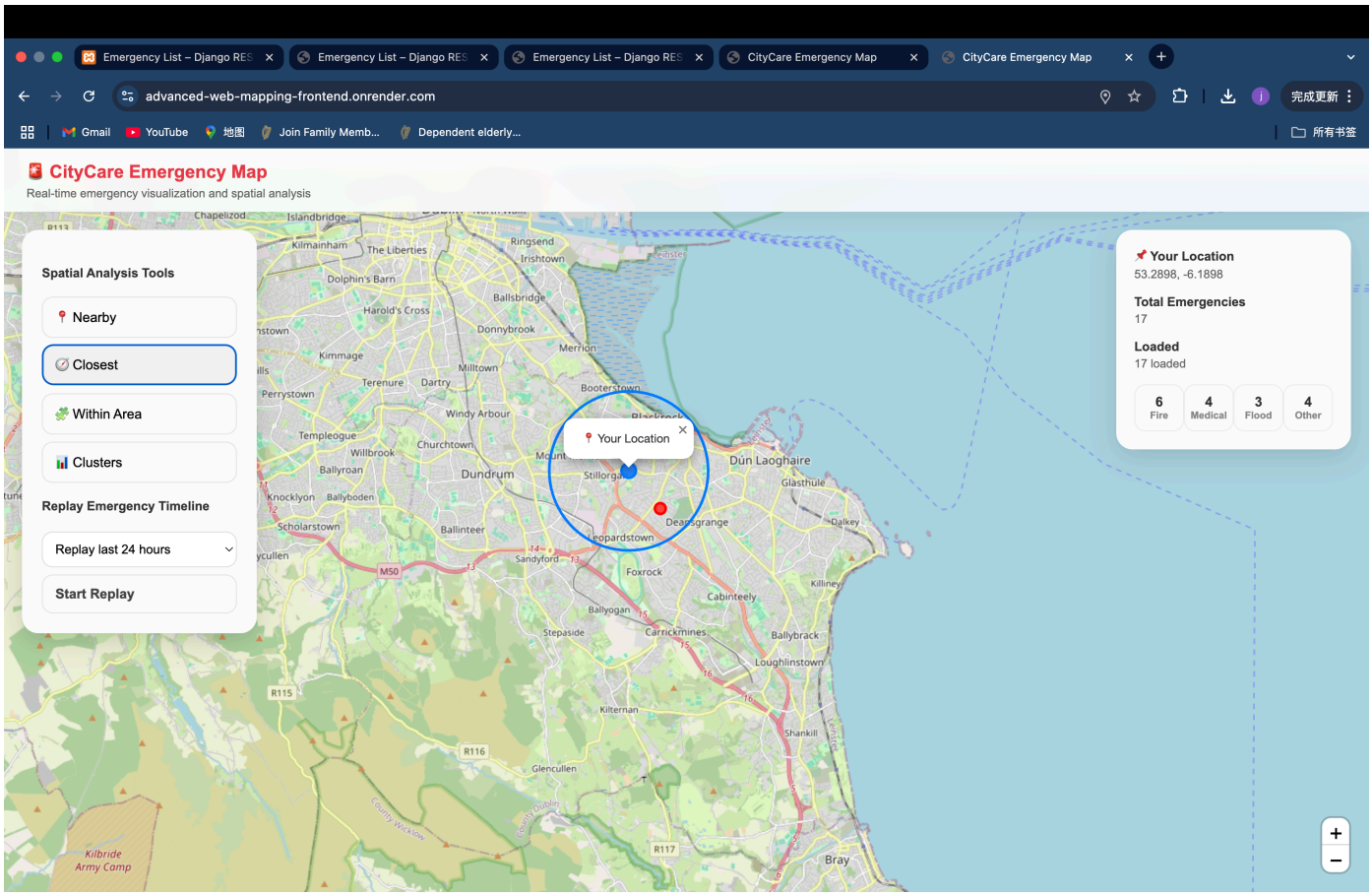


Replay timeline:

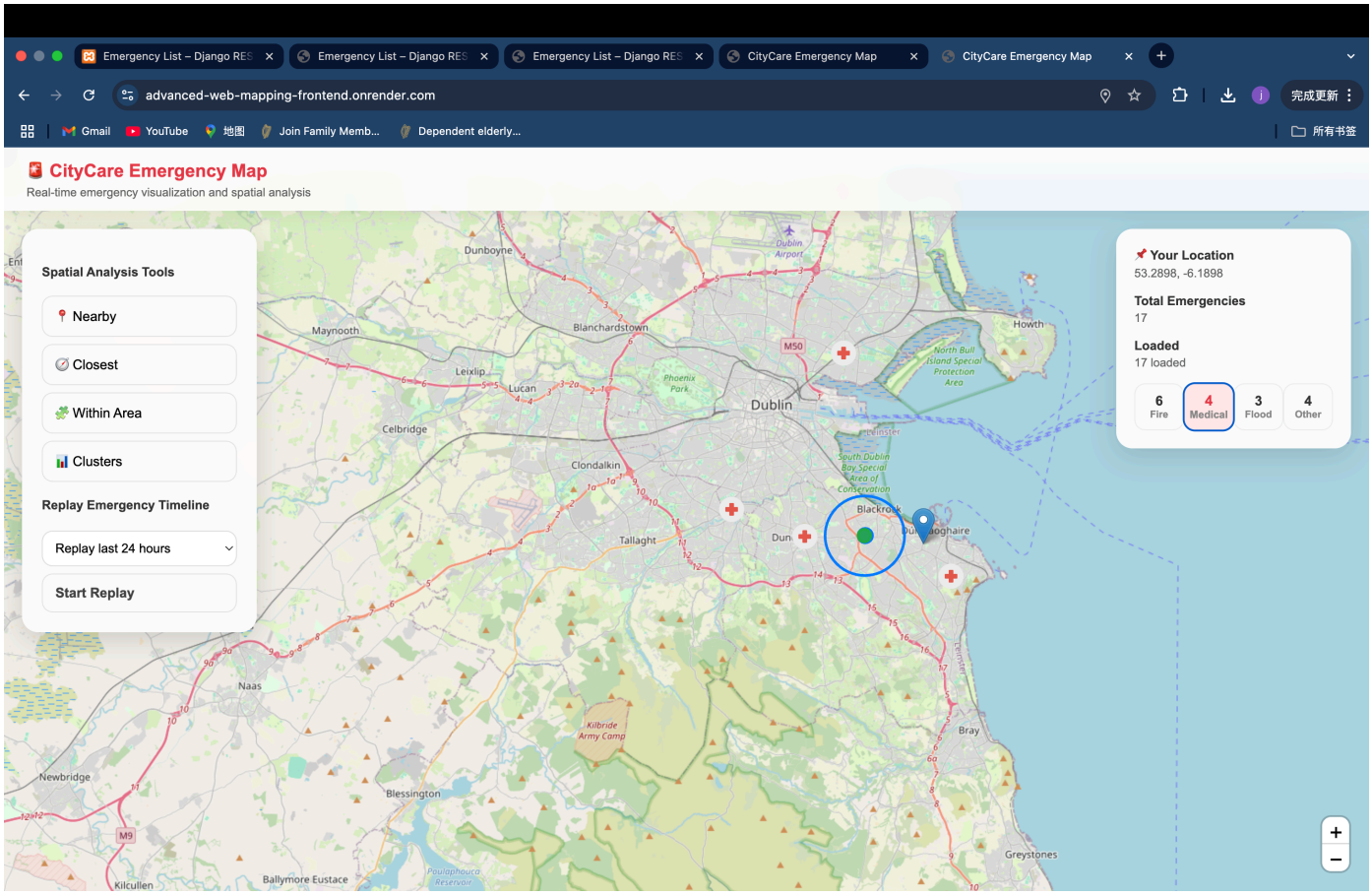


Spatial tools (Nearby / Closest / Within Area / Clusters):

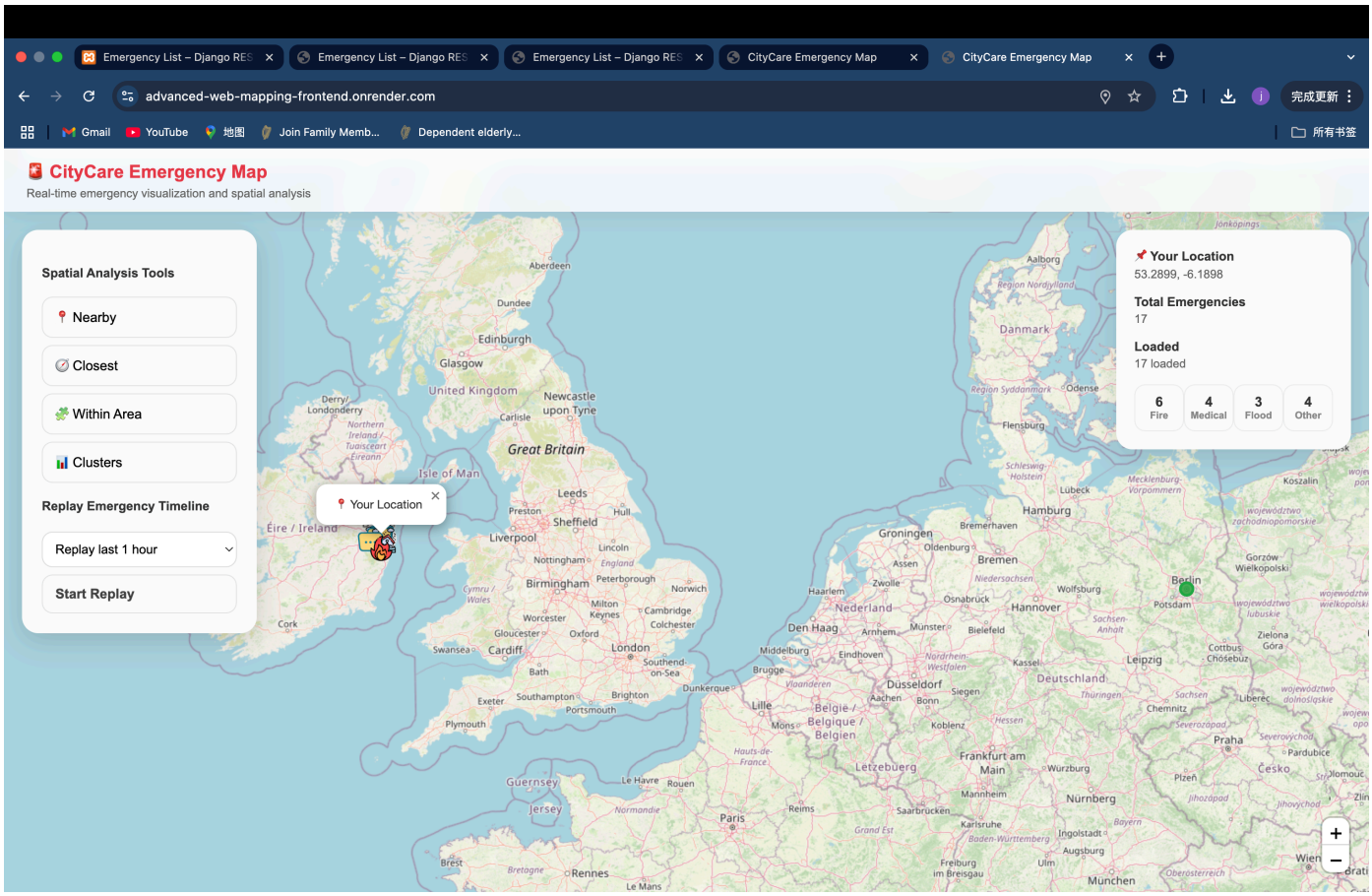




Queries for each type of emergencies:



Multi-user presence (online users):



Project Goals:

The app demonstrates a complete LBS stack:

Create / store / manipulate spatial data

Emergencies and user locations stored in PostgreSQL + PostGIS.

Middle layer (MVC):

Django + Django REST Framework provide an API for spatial queries.

Hybrid / mobile-friendly frontend:

A Leaflet JS web app (PWA-ready) optimised for both desktop and mobile.

Cloud deployment:

Backend and frontend deployed on Render

Database deployed on neon

Tech Stack:

Database: PostgreSQL + PostGIS (geometry / geography types, spatial indexes)

Backend: Django,Django REST Framework,PostGIS integration (django.contrib.gis)

Frontend: Plain HTML + CSS, Leaflet JS, JavaScript

Mapping: Leaflet + OpenStreetMap tiles

Key Features

1. Emergency Management

Report emergencies by clicking on the map, filling:

Title

Description

Type: fire, medical, flood, other

Emergencies are sent via POST /api/emergencies/ with GeoJSON:

```
{
  "title": "Fire in city centre",
  "description": "Smoke reported near bridge",
  "type": "fire",
  "location": {
    "type": "Point",
    "coordinates": [-6.2603, 53.3498]
  }
}
```

Emergencies are displayed as custom icons on the map.

Any emergency can be deleted via the popup Delete button (DELETE /api/emergencies/{id}/).

2. Real-time Statistics & Filtering

Right-hand info card shows:

User location

Total emergencies

Per-type counters: Fire / Medical / Flood / Other

Filter buttons (.filter-btn) allow type-based filtering:

Clicking a type applies the filter.

Clicking again resets and reloads all emergencies.

Implementation (frontend):

```
document.querySelectorAll(".filter-btn").forEach(btn => {
  btn.addEventListener("click", () => {
    const type = btn.getAttribute("data-type");

    if (activeFilter === type) {
      activeFilter = null;
      document.querySelectorAll(".filter-btn")
        .forEach(b => b.classList.remove("active-filter"));
      loadEmergencies();
      return;
    }

    activeFilter = type;
    document.querySelectorAll(".filter-btn")
      .forEach(b => b.classList.remove("active-filter"));
    btn.classList.add("active-filter");

    applyTypeFilter(type);

  });
});
```

3. Multi-User Live Locations

Each browser gets its own anonymous user ID:

```
const USER_ID = (() => {
  try {
    const key = "citycare_user_id";
    let id = localStorage.getItem(key);
    if (!id) {
      id = "browser" + Math.random().toString(36).slice(2, 10);
      localStorage.setItem(key, id);
    }
    return id;
  } catch {
    return "browser" + Math.random().toString(36).slice(2, 10);
  }
})();
```

On map load, the browser:

Gets current location via navigator.geolocation

Sends a heartbeat every 10 seconds:

```
async function sendLocationHeartbeat(lat, lon) {
  await fetch(`${API_BASE}/api/users/update_location/`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      user_id: USER_ID,
      location: {
        type: "Point",
        coordinates: [lon, lat]   // GeoJSON = [lng, lat]
      },
    }),
  });
}
```

Backend stores the latest location + last_seen timestamp in PostGIS.

/api/users/active/ returns recently active users (as GeoJSON or WKT).

loadActiveUsers() renders other users as green dots:

```
L.circleMarker([lat, lng], {
  radius: 7,
  color: "#28a745",
  fillColor: "#28a745",
  fillOpacity: 0.9,
})
.addTo(activeUsersLayer)
.bindPopup(`👤 <b>${uid}</b><br>
  Last Seen: ${new Date(lastSeen).toLocaleString()}`);
```

Your own user ID is filtered out so you only see other users.

4. Spatial Tools (PostGIS)

Four spatial tools are available on the left panel:

Nearby – /api/emergencies/nearby/

Input: current user lat, lng, and radius (e.g. 2000m)

Draws a circle around user and highlights emergencies in that radius.

Closest – /api/emergencies/closest/

Finds the nearest emergency to the user.

Within Area – /api/emergencies/within_area/

Uses the current map view bounding box as a polygon.

Backend returns all emergencies within that polygon.

Clusters – /api/emergencies/cluster_summary/

Returns cluster centroids with a count.

Frontend draws proportional circles based on severity / density.

Example frontend call for Nearby:

const url = `${API_BASE}/api/emergencies/nearby/?lat=${lat}&lng=${lon}&radius=2000`;
renderGeoData(url, "green");

5. Replay Timeline

The Replay feature allows you to visualise how emergencies evolved over time.

User selects a time window (e.g. last 1 / 3 / 6 / 12 / 24 hours).

Frontend calls: /api/emergencies/replay/?hours=...

Backend returns emergencies ordered by reported_at.

A bottom timeline player controls:

Play / Pause

Playback speed

Manual scrubbing with a slider

Core playback logic:

```
function replayEmergencies(events) {
  replayEvents = events;
  replayIndex = 0;
  replayPaused = false;

  showTimeline();
  resetTimelineUI();
  emergencyLayer.clearLayers();

  startReplayEngine();
}
```

updateReplayFrame(i) uses emergencyLayer.clearLayers() to redraw the map with all events up to the current time step.

6. Responsive & Mobile-Friendly UI

Desktop:

Left: Spatial Tools card

Right: Info / stats card

Bottom: Timeline controller

Mobile:

Map stays full-screen.

Panels can be hidden or toggled to avoid covering the map.

Styling is controlled by CSS media queries in css/style.css.

Backend API – Overview

Emergency Endpoints

GET /api/emergencies/
List all emergencies (GeoJSON FeatureCollection).

POST /api/emergencies/
Create a new emergency.

DELETE /api/emergencies/{id}/
Delete an emergency.

GET /api/emergencies/nearby/?lat=..&lng=..&radius=..
Emergencies within radius.

GET /api/emergencies/closest/?lat=..&lng=..
Single closest emergency.

POST /api/emergencies/within_area/
Body: GeoJSON Polygon of map bounds.

GET /api/emergencies/cluster_summary/
Cluster centroids with counts.

GET /api/emergencies/replay/?hours=..
All emergencies in the time window, ordered by reported_at.

User Location Endpoints

POST /api/users/update_location/
Body: { user_id, location: GeoJSON Point }.

GET /api/users/active/
Recently active users as spatial features.

Data Model:

Emergency

id – integer, primary key

title – short text

description – text

type – fire / medical / flood / other

location – PointField (SRID 4326, PostGIS)

reported_at – DateTimeField (auto-now-add)

UserLocation

user_id – string (e.g. browser_xxxxxxxx)

location – PointField

last_seen – DateTimeField