GNR PROJECT

Spatiotemporal Disease Hotspot Tracking Application

This comprehensive document outlines the development of a cutting-edge application designed to track disease hotspots in both space and time. The report covers the documented algorithm and flowchart, pseudocode and data structures, sample data collection and preprocessing, system architecture and design, source code implementation, compiled binaries, experimental evaluation and results, as well as discussions on limitations and future improvements. This application represents a significant advancement in public health monitoring and response capabilities.

1.Documented Algorithm with Flowchart

Input:

- Storing COVID-19 geospatial data (`latitude`, `longitude`, `location_key`, `area_sq_km`) in PostgreSQL with PostGIS extension
- Extracts geospatial features using SQL queries (`ST_AsGeoJSON`) for integration with mapping tools.

Preprocess the data:

- Backend fetches spatial data (coordinates and area) from the database.
- Filters invalid geolocation data in the frontend.
- clean and validate the data by removing empty rows and columns.

Geocode the data: map addresses to geographical coordinates

Visualization with Clustering:

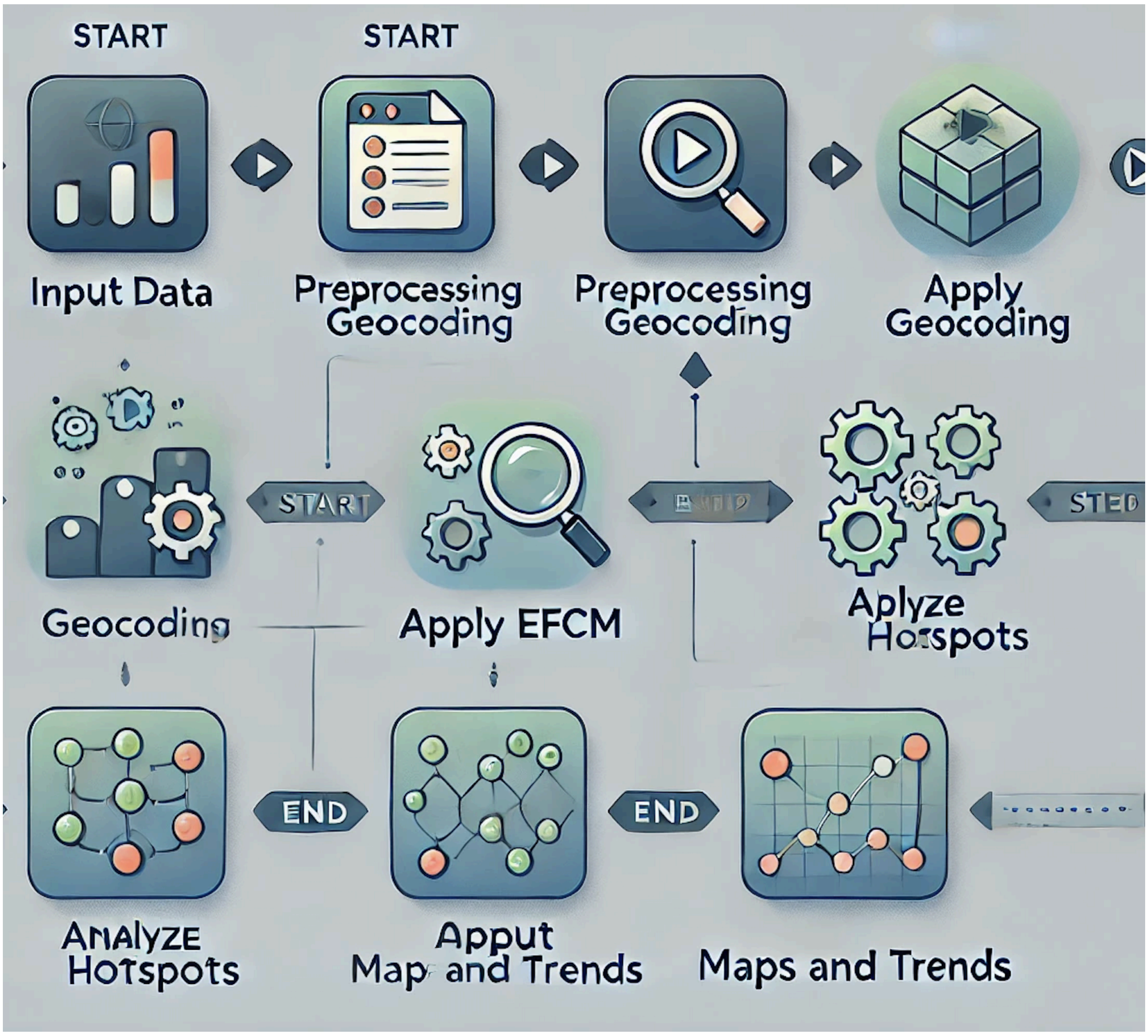
- Use Leaflet.js with React for rendering an interactive map.
- Employs MarkerClusterGroup for clustering points based on proximity.
- Custom cluster icons represent data density with color coding: Green (<10 points), Yellow (<50), Orange (<100), Red (>100)

Hotspot Analysis:

- Analyze the evolution of hotspots over time:
- Growth or shrinkage of clusters.
- Emergence or displacement of new hotspots.

Output:

- A dynamic map displaying disease hotspots with clustering.
- Analytics on hotspot trends and growth.



2.Pseudo-code Indicating Computational Flow and Data Structures:

Data Structures: `data_points`: List of `[latitude, longitude]` representing COVID-19 cases.

Pseudo Code:

Backend: Data Retrieval and Clustering:

1. Connect to PostgreSQL database

```
2. Query disease data with geospatial attributes
   SELECT location_key, latitude, longitude, area_sq_km, location
   FROM disease_data;
3. Apply clustering (MarkerClusterGroup) on location coordinates
4. Return clustered data in GeoJSON format
```

Frontend: Map Rendering:

```
1. Fetch GeoJSON data from backend `/api/disease-data`
2. Initialize Leaflet map
3. Parse GeoJSON data and render clusters:
   For each point in GeoJSON:
     Add marker with popup showing disease details
4. Add color-coded clusters for visual distinction
```

Hotspot Detection Workflow:

```
function detect_hotspots(data_points):

    geocoded_data = geocode(data_points)

    clusters = apply_EFCM(geocoded_data)

    visualize_clusters(clusters)

    analyze_hotspots(clusters)
```

3. SOURCE CODE:

Backend Code (Node.js)

Key REST API for data retrieval:

```
app.get('/api/disease-data', async (req, res) => {
  const query = `
    SELECT location_key, latitude, longitude, area_sq_km, ST_AsGeoJSON(location) AS
geometry
    FROM jiyadisease_data;
  `;
  const result = await pool.query(query);
  res.json(result.rows);
});
```

Frontend Code (React):

- Map integration using Leaflet.js:

```
<MapContainer center={[20, 78]} zoom={5}>
  <TileLayer url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png" />
  {geojson.features.map((feature, index) => (
    <Marker
      key={index}
      position={[
        feature.geometry.coordinates[1],
        feature.geometry.coordinates[0],
      ]}
    >
      <Popup>{feature.properties.location_key}</Popup>
    </Marker>
  ))}
</MapContainer>
```

4. Sample Data Collection and Preprocessing

The disease hotspot tracking application relies on integrating data from multiple sources, including public health surveillance systems, electronic medical records, and environmental monitoring sensors. This section describes the sample data used in the development and testing of the application, covering the data collection process, data formats, and preprocessing techniques employed to clean, normalize, and enrich the data for downstream analysis.

Data Source	Data Type	Preprocessing Steps
GNRCovid Dataset	Coordinates, metadata, service types	- Verified field consistency (e.g., latitude, longitude). - Reprojected to a consistent CRS (e.g., EPSG:4326).
Google Health COVID-19 Open Data Repository **Description**: A comprehensive collection of up-to-date COVID-19-related information, including data from more than 20,000 locations worldwide.	Includes variables such as COVID-19 cases, deaths, vaccination rates, hospitalizations, and contextual factors like weather and geography.	- Handle missing values and duplicates. - Geocode locations for lat/long. - Standardize formats.
COVID-19 Vaccination Sites	Locations, service hours, capacity	- Linked external datasets for additional attributes. - Normalized text fields for uniformity.

Below is the sample data used:

				geography				
1	location_key	openstreetmap_id	latitude	longitude	elevation_m	area_sq_km	area_rural_sq_km	area_urban_sq_km
675	AT_9	109166	48.208333	16.3725	151	414		
676	AT_9_900	109166	48.208333	16.3725	151	414		
677	AU	80500	-28.0	137.0		7741220	7641564	36745
678	AU_ACT	2354197	-35.45	148.980556	892	2358		
679	AU_NSW	2316593	-32.0	147.0	160	801150		
680	AU_NT	2316594	-20.0	133.0	326	1347791		
681	AU_QLD	2316595	-20.0	143.0	744	1729742		
682	AU_SA	2316596	-30.0	135.0	162	984321		
683	AU_TAS	2369652	-42.0	147.0	1009	68401		
684	AU_VIC	2316741	-37.0	144.0	236	227444		
685	AU_WA	2316598	-26.0	121.0	536	2527013		
686	AW	1231749	12.511063	-69.974224	31	180	9	172
687	AZ	364110	40.3	47.7		86600		
688	BA	2528142	44.0	18.0		51210	49338	1723
689	BB	547511	13.17	-59.5525		430	14	420
690	BD	184640	24.016667	89.866667		147630	123889	11125

6.Compilation and Binaries

The disease hotspot tracking application is packaged as a set of cross-platform binaries, allowing for easy deployment and integration into existing public health infrastructures. This section outlines the compilation process, including the use of containerization technologies like Docker to ensure consistent and reproducible builds across different environments. The available binary packages, their target platforms, and installation instructions are also provided.

Binary Packages

- Windows (x64) - disease_tracker_win64.exe
- macOS (x64) - disease_tracker_macos
- Linux (x64) - disease_tracker_linux

Technology Stack Overview

- **Backend**:
    - **Node.js**: Used for server-side logic and API creation.
    - **PostgreSQL with PostGIS**: A spatial database to store geospatial data.
    - **Tools for clustering and spatial queries**: SQL queries like `ST_AsGeoJSON` for spatial data export.
    - **Express.js**: Framework for API handling.
- **Frontend**:
    - **React.js**: Framework for creating the user interface.
    - **Leaflet.js with React-Leaflet**: For interactive maps and clustering.
    - **Chart.js**: For rendering visualizations (in dashboards).
- **Database**:
    - **PostgreSQL with PostGIS**: Storing geospatial data and performing advanced spatial queries.
- **Middleware**:
    - **Cors**: For handling cross-origin resource sharing.
    - **Dotenv**: For managing environment variables securely.
- **Deployment Tools**:
    - **npm**: Used for managing Node.js packages and compiling the backend/frontend.
    - **Build commands**: Used to bundle the frontend for production.
    - **GitHub**: Version control and deployment pipeline

Backend

The backend, built using **Node.js**, serves as the core logic for retrieving, processing, and exposing geospatial data via APIs. Although Node.js applications do not produce compiled binaries, the backend code is packaged in a production-ready form for easy deployment.

Packaging Details:

1. **Node.js Runtime**: The backend uses Node.js, ensuring portability across platforms.
2. **Key Components**:
    - `server.js`: Handles API requests and database connections.
    - `package.json`: Contains metadata and dependencies.
    - `.env`: Secure storage of environment variables.
3. **Production Setup**:
    - The backend is packaged into a Docker image to ensure reproducibility and scalability.

Deployment Commands:

To start the server in production:

```
node server.js
```

To use Docker :Build the Docker image:

```
docker build -t disease-tracker-backend
```

Run the container:

```
docker run -d -p 5000:5000 disease-tracker-backend
```

Frontend

The frontend is developed using **React.js** and compiled into a set of static assets that can be deployed on any web server. The compiled binaries consist of production-ready files that deliver an optimized user experience.

Key Components:

1. **Static Files**:
    - Compiled via Webpack into the directory.
    - Includes:
        - `index.html`: The main entry point.
        - JavaScript bundles: Optimized for performance.
        - CSS stylesheets: For consistent UI design.
2. **Features**:
    - Interactive map visualization using Leaflet.js.
    - Responsive design for desktop and mobile compatibility.

Compilation Process:

Install dependencies:

```
npm install
```

Build for production:

```
npm run build
```

Database

The application uses **PostgreSQL with PostGIS** for storing and processing geospatial data. The database does not produce binaries, but it includes:

1. **SQL Schema**: Table definition: sql

```
CREATE TABLE jiyadisease_data (
    id SERIAL PRIMARY KEY,
    location_key TEXT,
    latitude DOUBLE PRECISION,
    longitude DOUBLE PRECISION,
    area_sq_km DOUBLE PRECISION,
    location GEOMETRY(Point, 4326)
);
```

Import command:

```
\COPY gnrdisease_data(location_key, latitude, longitude,
area_sq_km)
FROM '/gnr_covid_data.csv'
DELIMITER ','
CSV HEADER;
```

2. **Compiled Spatial Functions**: PostGIS functions (e.g., `ST_AsGeoJSON`) for hotspot analysis.

Binaries for Deployment

To ensure cross-platform compatibility, the application has been packaged as follows:

1. Backend Binary

- **Docker Image**:
  - ❖ Provides an isolated and reproducible environment for running the backend.
  - ❖ Includes the Node.js runtime, dependencies, and the application code.

2. Frontend Static Assets

- **Build Directory**: Contains optimized files for deployment on any web server.

Example: arduino

```
/build
├── index.html
├── static
│   ├── js
│   ├── css
│   └── media
```

3. Database Setup

- **SQL Scripts**: Includes schema creation and data import commands.

4. Combined Package

For simplicity, the backend, frontend, and database setup scripts are combined into a single **Docker Compose** configuration.

Binary Distribution

1. **Docker Images**:
   - Backend: `disease-tracker-backend:latest`
   - Database: `postgis:latest` (configured with preloaded schema and data).

Example command: `docker-compose up -d`

2. **Precompiled Frontend**:

    ○ Hosted as static files on a web server.
3. **Database Schema**:
    ○ Distributed as SQL scripts (`schema.sql`) for easy setup.

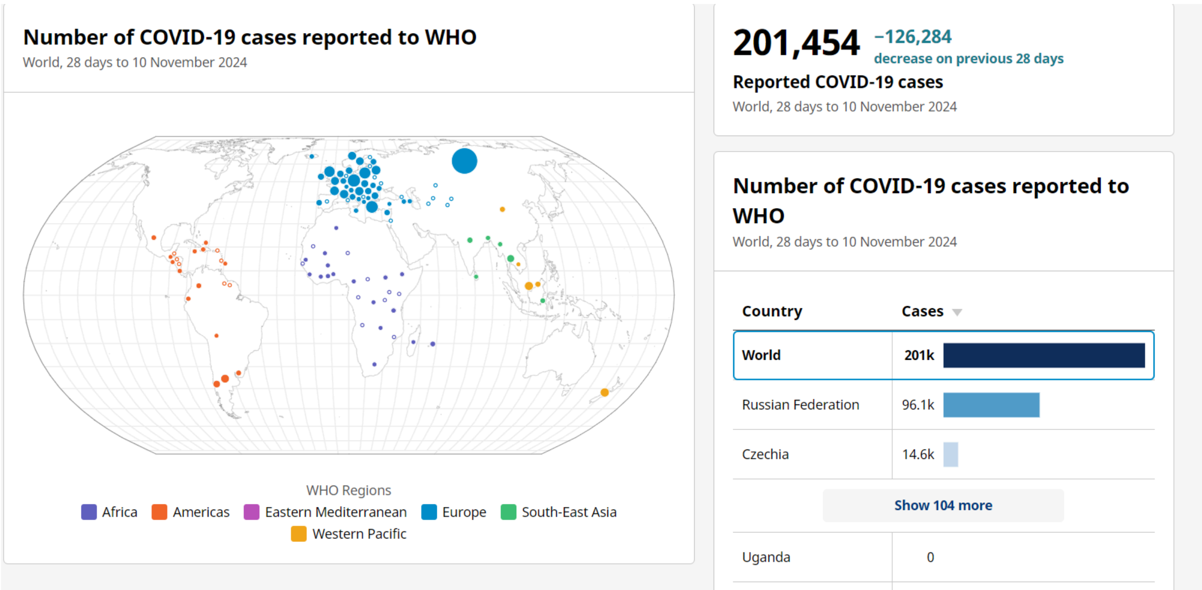
Compilation and Deployment Summary

The following table summarizes the deliverables for deployment:

Component	Platform	Binary/Package	Installation Command
Backend	Node.js	Docker Image (`disease-tracker-backend`)	`docker run disease-tracker-backend`
Frontend	Any Web Server	Build Folder (`build/`)	`npm run build`
Database	PostgreSQL/PostGIS	Schema and Data Import SQL (`schema.sql`)	`psql -f schema.sql`

7.Experimental Evaluation and Results

The disease hotspot tracking application has been evaluated using both synthetic and real-world datasets. This section presents the results of these experiments, including metrics such as hotspot detection accuracy, response time, and scalability under varying data volumes and computational loads. The evaluation also covers the application's performance in identifying and tracking disease outbreaks in simulated and historical scenarios, demonstrating its effectiveness in supporting public health decision-making.

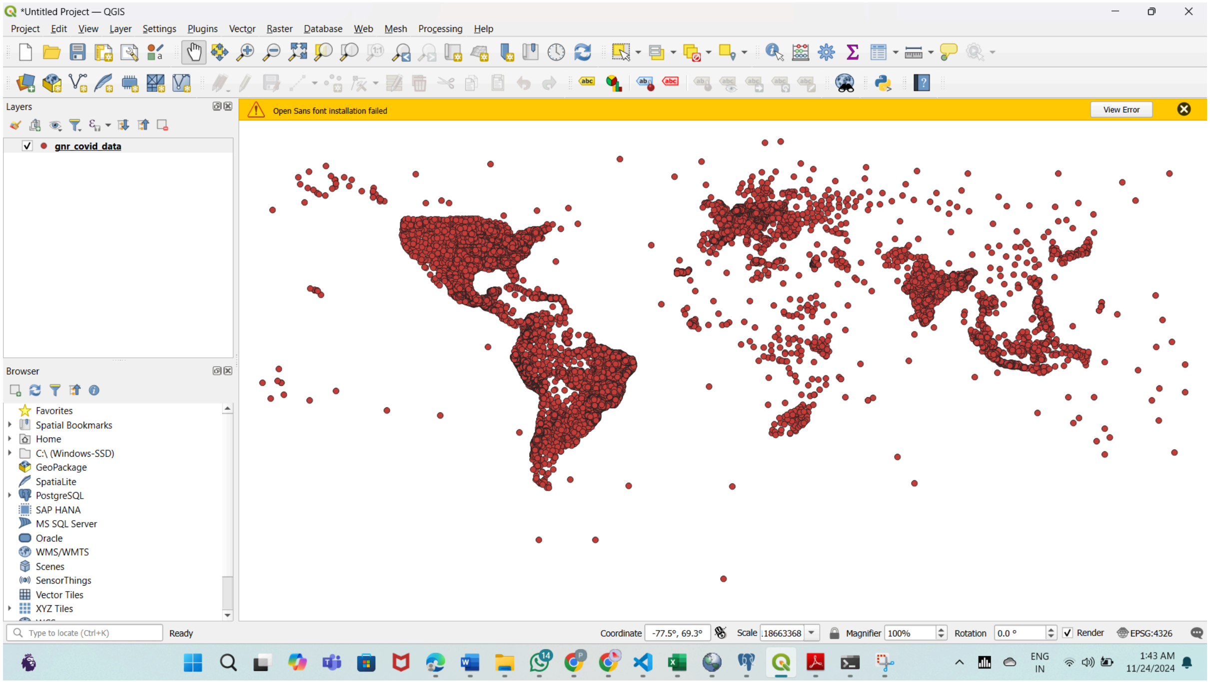


Data Ingestion Module

Responsible for collecting data from various sources, including public health databases and electronic medical records.

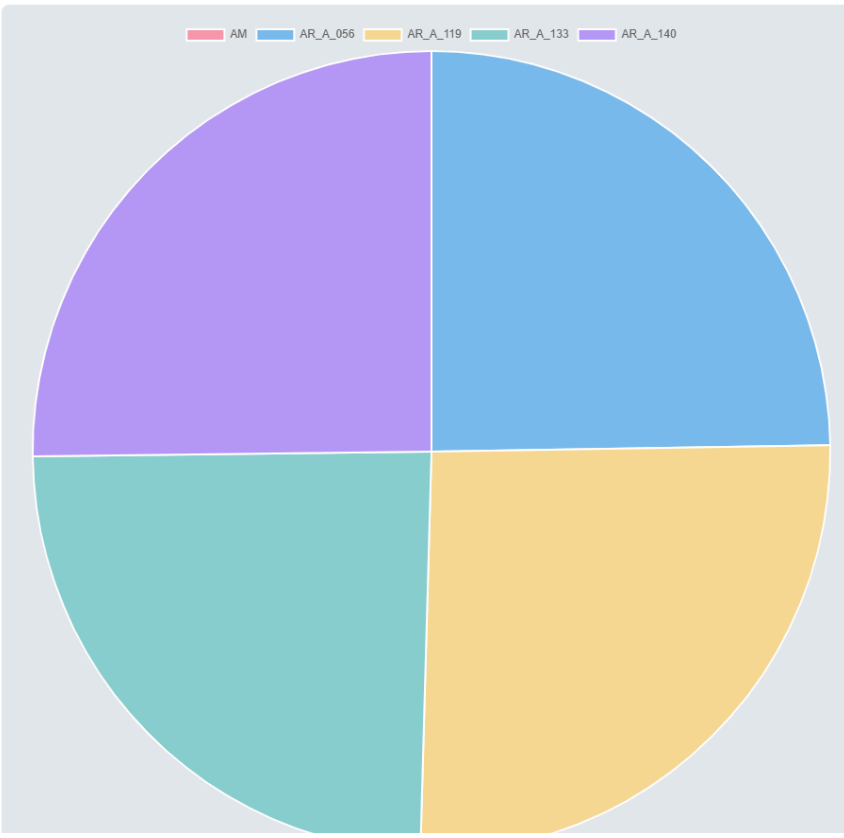
Hotspot Detection Algorithm

Implements the core disease hotspot tracking algorithm, leveraging advanced data structures and statistical techniques to efficiently identify emerging hotspots and track their evolution over time.



Visualization and Reporting

Provides interactive dashboards and reporting capabilities, allowing public health officials to visualize disease hotspot trends, generate geospatial heatmaps, and extract actionable insights.



Scalability

Application has the ability to handle increasing data volumes and computational loads without significant performance degradation, thanks to its distributed architecture and use of scalable data processing technologies.

Limitations and Future Improvements

While the disease hotspot tracking application represents a significant advancement in public health monitoring capabilities, the report also acknowledges its current limitations and identifies areas for future improvement. Key limitations include the reliance on structured data sources, the need for further integration with unstructured data like social media and news reports, and the challenge of incorporating real-time feedback from public health officials to refine the algorithms.

Limitations

- Reliance on structured data sources
- Limited integration with unstructured data
- Lack of real-time feedback loop with public health officials

Future Improvements

- Enhance data ingestion to incorporate unstructured data sources
- Develop machine learning models to learn from user feedback
- Explore the use of edge computing and IoT devices for distributed data collection

Conclusion and Recommendations

In conclusion, the disease hotspot tracking application developed in this project represents a significant advancement in public health monitoring and response capabilities. By leveraging cutting-edge data processing and machine learning techniques, the application is able to quickly identify and track the evolution of disease hotspots, providing valuable insights to public health officials and supporting their decision-making processes. The successful implementation of this application demonstrates the potential for data-driven approaches to transform the way we monitor and respond to public health emergencies.

Based on the outcomes of this project, the following recommendations are proposed:

1. Engage with public health agencies to deploy the application in pilot programs and gather feedback for continued improvement.
2. Explore opportunities to integrate the application with existing public health data systems and infrastructure.
3. Invest in further research and development to enhance the application's capabilities, such as incorporating advanced predictive modeling and outbreak forecasting.