GIS Flood and Pollution Management System

Final Project Report

Prepared By:
Janvi Shah

Date: 29th April, 2025

1. Introduction

This project presents a Geographic Information System (GIS) application focused on managing and retrieving environmental data related to Floods and Pollution. The system is built using Node.js, Express.js, and MongoDB Atlas, enabling Create, Read, Update, and Delete (CRUD) operations along with advanced geospatial searches such as polygon-based, radius-based, and time-based filtering.

1.1 Background

Environmental monitoring is essential for public safety, urban planning, and disaster management. Geospatial Information Systems (GIS) integrate location-based data with real-world applications, enabling users to manage incidents like floods and pollution events in a scalable and efficient manner.

1.2 Problem Statement

Existing systems lack capabilities like dynamic spatial search, efficient CRUD operations, and real-time monitoring for environmental events. This project addresses these limitations by offering modular APIs supporting advanced GIS queries.

1.3 Project Objective

The objectives of the GIS Management System are to:
- Design a MongoDB database for GIS data with 2dsphere indexing.
- Build RESTful, modular APIs using Node.js and Express.js.
- Implement CRUD operations for Flood and Pollution datasets.
- Provide advanced GIS search features like polygon and radius-based queries.
- Ensure scalability and clean project organization.

2. Technology Stack

2.1 Backend Framework

Node.js: Event-driven, non-blocking I/O model ideal for lightweight real-time applications.
Express.js: Simplified web framework that streamlines API route creation and request handling.

2.2 Database

MongoDB Atlas: A cloud-hosted NoSQL database with flexible schema, native geospatial capabilities, automatic scaling, and replication.

2.3 ODM (Object Data Modeling)

Mongoose: Schema-based solution to model MongoDB application data with structure and validation.

2.4 Tools Used

Postman: For API testing.
MongoDB Compass: For visual database management.
Visual Studio Code (VS Code): As the development environment.

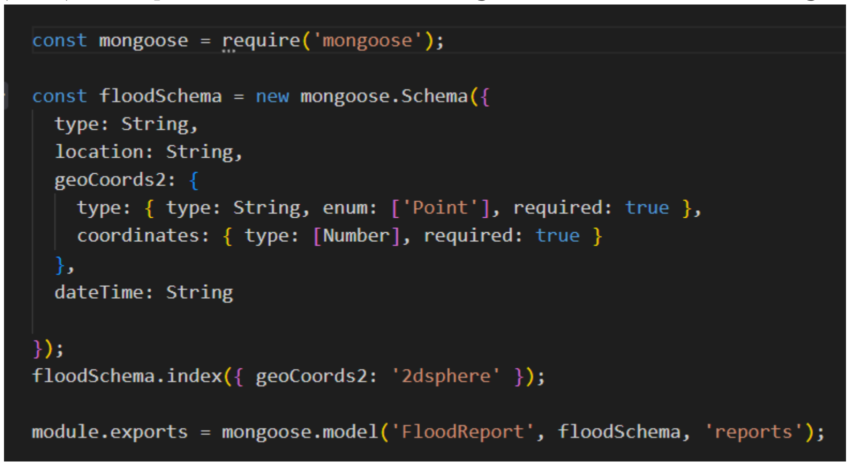
3. MongoDB Database Design

3.1 Why MongoDB for GIS Data?

MongoDB supports flexible schema design, efficient geospatial queries, and high scalability, making it perfect for GIS data storage and analysis.

3.2 Flood Report Schema Design

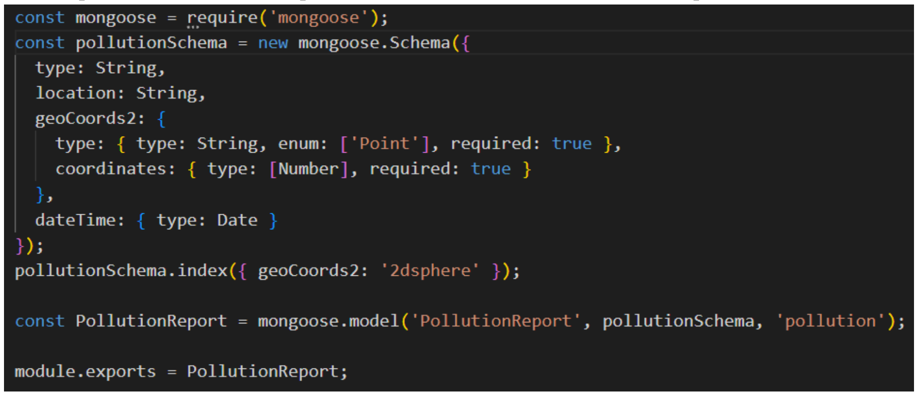
Flood reports include type, location, floodLevel, geoCoords2 (Point coordinates), and dateTime (Date). A 2dsphere index is created on geoCoords2 to enable fast geospatial queries.



```javascript
const mongoose = require('mongoose');

const floodSchema = new mongoose.Schema({
  type: String,
  location: String,
  geoCoords2: {
    type: { type: String, enum: ['Point'], required: true },
    coordinates: { type: [Number], required: true }
  },
  dateTime: String

});
floodSchema.index({ geoCoords2: '2dsphere' });

module.exports = mongoose.model('FloodReport', floodSchema, 'reports');
```

3.3 Pollution Report Schema Design

Pollution reports include type, location, geoCoords2 (coordinates), and dateTime. These are stored in a separate collection 'pollution', also indexed with 2dsphere.



```javascript
const mongoose = require('mongoose');
const pollutionSchema = new mongoose.Schema({
  type: String,
  location: String,
  geoCoords2: {
    type: { type: String, enum: ['Point'], required: true },
    coordinates: { type: [Number], required: true }
  },
  dateTime: { type: Date }
});
pollutionSchema.index({ geoCoords2: '2dsphere' });

const PollutionReport = mongoose.model('PollutionReport', pollutionSchema, 'pollution');

module.exports = PollutionReport;
```

3.4 Why 2dsphere Index?

2dsphere indexes enable spherical calculations like distance and containment, crucial for real-world geospatial queries.

4. API Design and Architecture

4.1 Why Modular API Architecture?

Modular architecture separates models, routes, and server logic, making the project scalable, maintainable, and extendable. Each entity (Flood, Pollution) has its independent schema, CRUD routes, and geospatial queries.

4.2 Project Folder Structure

```
/GIS-Project
  |-- app.js
  |-- pollution-app.js
  |-- floodSchema.js
  |-- pollutionSchema.js
  |-- flood-crud.js
  |-- pollution-crud.js
  |-- radius.js
  |-- time-based.js
  |-- GroupByCountAPI.js
  |-- dashboard.js
```

4.3 API Types

- CRUD APIs (Create, Read, Update, Delete)
- Polygon Search APIs
- Radius (Nearby) Search APIs
- Time-Based Search APIs
- Grouping & Dashboard APIs

4.4 URL Structures

- POST /api/floods → Create flood report
- GET /api/floods → Retrieve all floods
- POST /api/pollution → Create pollution report
- GET /api/pollution → Retrieve all pollution reports
- POST /api/polygon-search → Polygon search for both floods and pollution
- POST /api/nearby → Radius search
- POST /api/reports-by-date → Time-based filtering
- GET /api/floods/count-by-level → Group floods by level
- GET /api/dashboard-summary → Dashboard summary for floods

4.5 Benefits of Modular Design

- Scalability: New modules can be added easily.
- Maintainability: Each file has a clear responsibility.
- Reusability: Common code can be reused.
- Professionalism: Clean, structured development approach.

5. CRUD Full Detailed Walkthrough (Flood and Pollution)

5.1 What is CRUD?

CRUD stands for Create, Read, Update, and Delete operations. These are the basic operations required for persistent storage and retrieval of data. This system implements CRUD operations separately for Flood and Pollution datasets.

5.2 Create Operation (POST)

5.2.1 Flood Create API

Endpoint: POST /api/floods: Creates a new flood report by specifying type, location, floodLevel, coordinates, and dateTime.

5.2.2 Pollution Create API

Endpoint: POST /api/pollution: Creates a new pollution report by specifying type, location, coordinates, and dateTime.

5.3 Read Operation (GET)

5.3.1 Read All Floods

Endpoint: GET /api/floods

5.3.2 Read All Pollution Reports

Endpoint: GET /api/pollution

5.3.3 Read Specific Report by ID

Flood: GET /api/floods/:id

Pollution: GET /api/pollution/:id

5.4 Update Operation (PUT)

5.4.1 Update Flood Report

Endpoint: PUT /api/floods/:id

5.4.2 Update Pollution Report

Endpoint: PUT /api/pollution/:id

5.5 Delete Operation (DELETE)

5.5.1 Delete Flood Report

Endpoint: DELETE /api/floods/:id

5.5.2 Delete Pollution Report

Endpoint: DELETE /api/pollution/:id

5.6 Key Points to Remember for CRUD

- POST for Create operations.
- GET for Read operations.
- PUT for Update operations.
- DELETE for Delete operations.
- Separate CRUD modules for Flood and Pollution.

6. Advanced GIS APIs (Polygon Search, Radius Search, Time Filtering)

6.1 What are Advanced GIS APIs?

Beyond CRUD, GIS applications require spatial search capabilities to find records based on location and time. This project supports three key advanced features:

- Polygon Search
- Radius (Nearby) Search
- Time-based Filtering

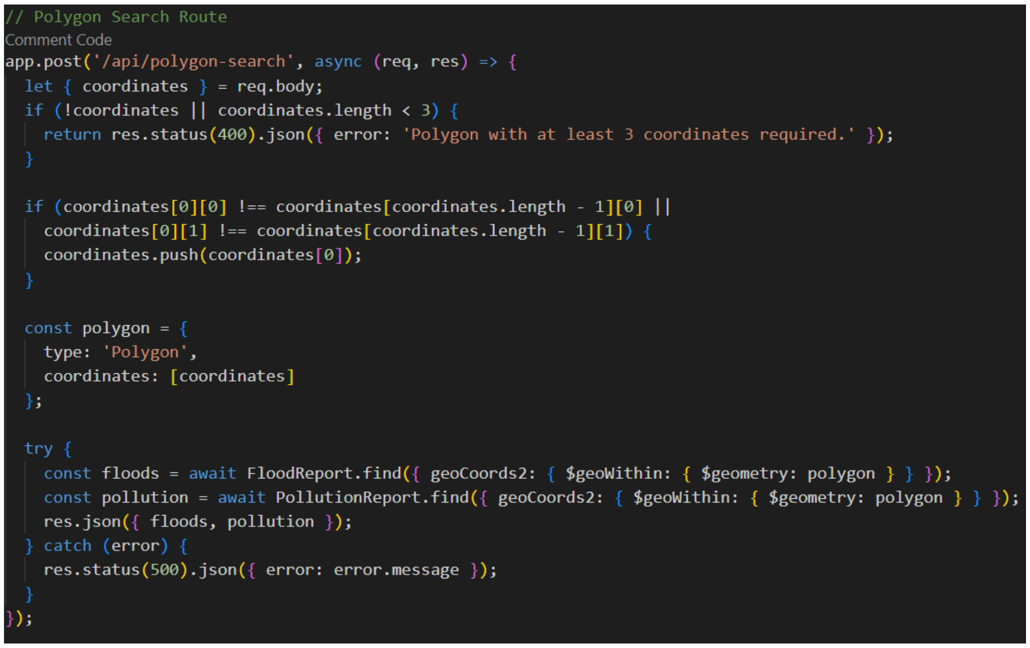
6.2 Polygon Search API

Purpose: Find all flood or pollution records located inside a user-defined polygon.
Endpoint: POST /api/polygon-search

Key Logic:

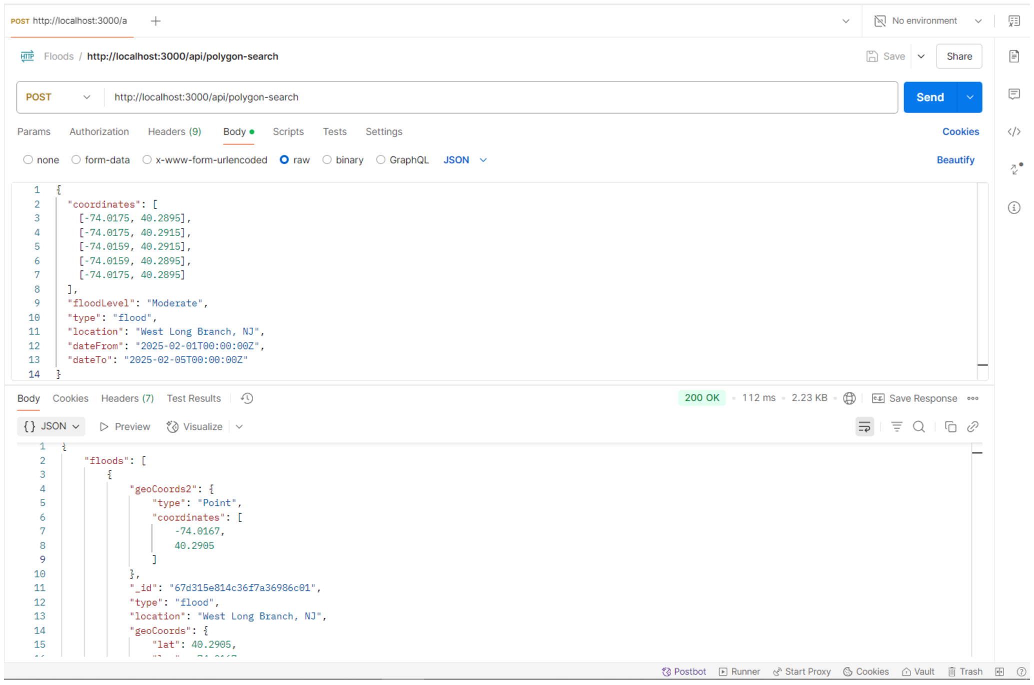
- Polygon must be closed (first and last coordinate match).
- MongoDB uses $geoWithin with a 2dsphere index.
- Optional filters: type, location, dateFrom, dateTo.

Code:



```
// Polygon Search Route
Comment Code
app.post('/api/polygon-search', async (req, res) => {
  let { coordinates } = req.body;
  if (!coordinates || coordinates.length < 3) {
    return res.status(400).json({ error: 'Polygon with at least 3 coordinates required.' });
  }

  if (coordinates[0][0] !== coordinates[coordinates.length - 1][0] ||
    coordinates[0][1] !== coordinates[coordinates.length - 1][1]) {
    coordinates.push(coordinates[0]);
  }

  const polygon = {
    type: 'Polygon',
    coordinates: [coordinates]
  };

  try {
    const floods = await FloodReport.find({ geoCoords2: { $geoWithin: { $geometry: polygon } } });
    const pollution = await PollutionReport.find({ geoCoords2: { $geoWithin: { $geometry: polygon } } });
    res.json({ floods, pollution });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

Postman:



POST http://localhost:3000/a

Floods / http://localhost:3000/api/polygon-search

POST http://localhost:3000/api/polygon-search

```
{
  "coordinates": [
    [-74.0175, 40.2895],
    [-74.0175, 40.2915],
    [-74.0159, 40.2915],
    [-74.0159, 40.2895],
    [-74.0175, 40.2895]
  ],
  "floodLevel": "Moderate",
  "type": "flood",
  "location": "West Long Branch, NJ",
  "dateFrom": "2025-02-01T00:00:00Z",
  "dateTo": "2025-02-05T00:00:00Z"
}
```

200 OK · 112 ms · 2.23 KB

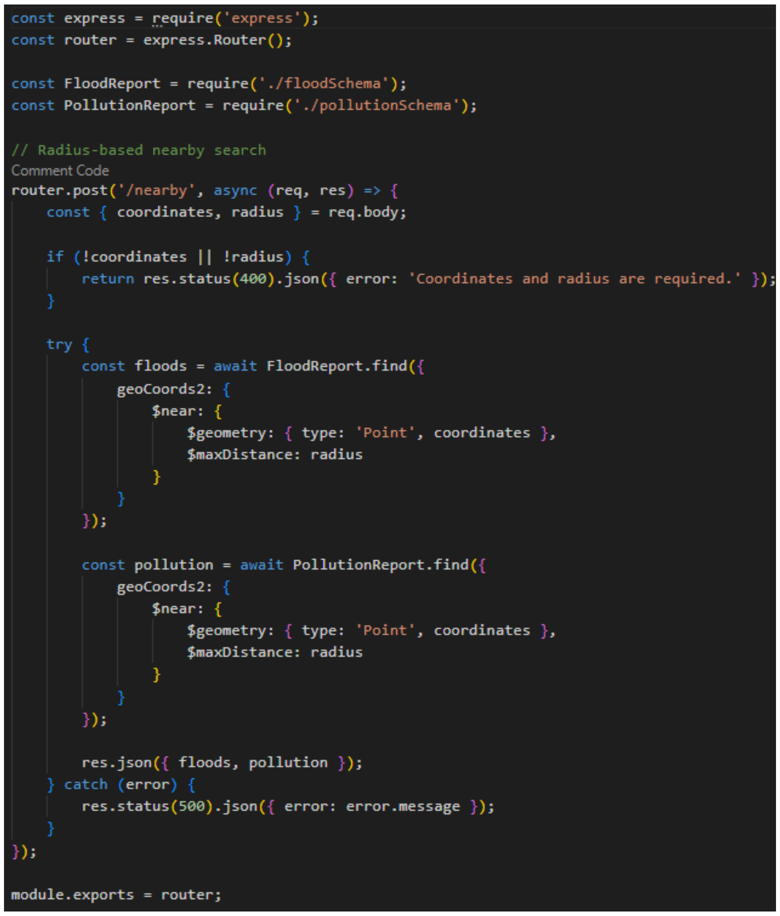
```
{
    "floods": [
        {
            "geoCoords2": {
                "type": "Point",
                "coordinates": [
                    -74.0167,
                    40.2905
                ]
            },
            "_id": "67d315e814c36f7a36986c01",
            "type": "flood",
            "location": "West Long Branch, NJ",
            "geoCoords": {
                "lat": 40.2905,
```

6.3 Radius (Nearby) Search API

Purpose: Find reports within a specified distance from a center point.
Endpoints:
- Flood: POST /api/nearby
- Pollution: POST /api/nearby-pollution

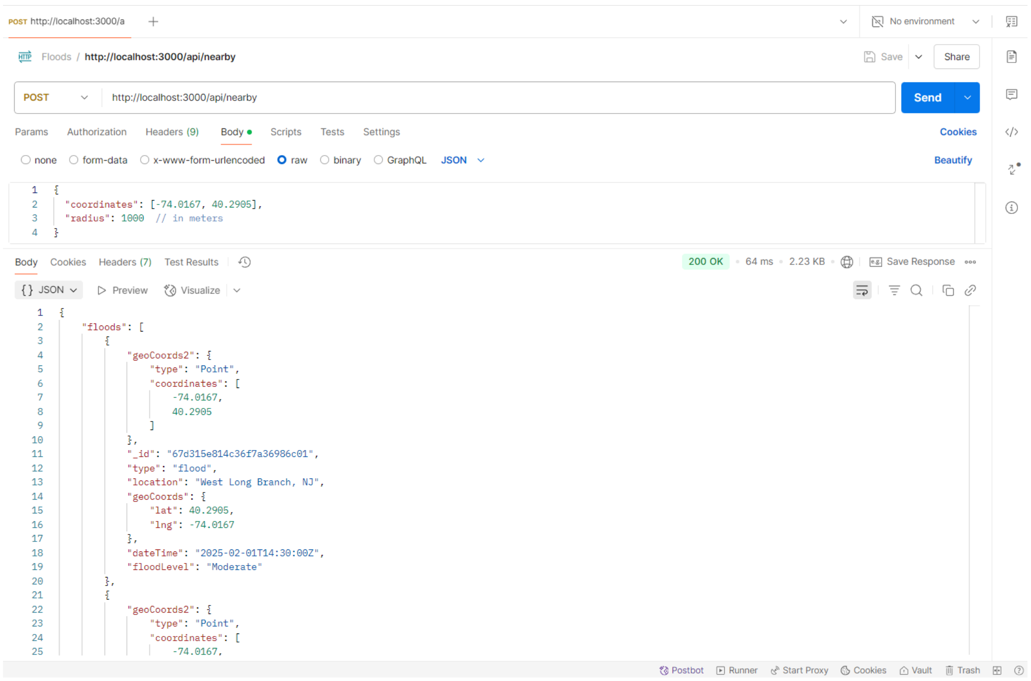
Key Logic:
- MongoDB $near operator is used.
- $maxDistance parameter defines the search radius (in meters).
- Results are automatically sorted by proximity.

Code:



```
const express = require('express');
const router = express.Router();

const FloodReport = require('./floodSchema');
const PollutionReport = require('./pollutionSchema');

// Radius-based nearby search
Comment Code
router.post('/nearby', async (req, res) => {
    const { coordinates, radius } = req.body;

    if (!coordinates || !radius) {
        return res.status(400).json({ error: 'Coordinates and radius are required.' });
    }

    try {
        const floods = await FloodReport.find({
            geoCoords2: {
                $near: {
                    $geometry: { type: 'Point', coordinates },
                    $maxDistance: radius
                }
            }
        });

        const pollution = await PollutionReport.find({
            geoCoords2: {
                $near: {
                    $geometry: { type: 'Point', coordinates },
                    $maxDistance: radius
                }
            }
        });

        res.json({ floods, pollution });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});

module.exports = router;
```

Postman:



POST http://localhost:3000/api/nearby

```
{
  "coordinates": [-74.0167, 40.2905],
  "radius": 1000  // in meters
}
```

200 OK · 64 ms · 2.23 KB

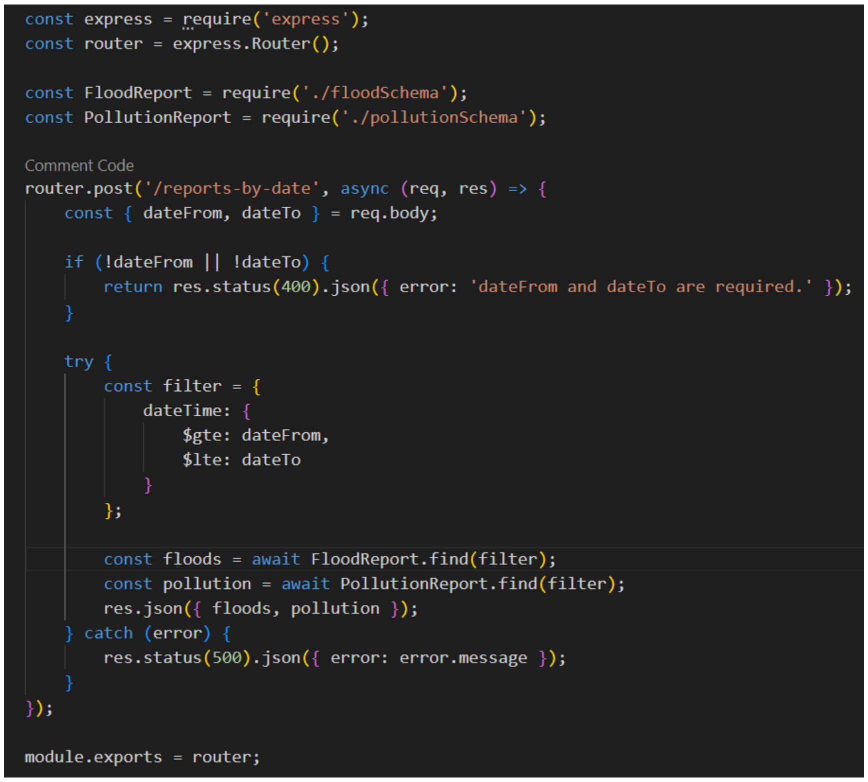
```
{
    "floods": [
        {
            "geoCoords2": {
                "type": "Point",
                "coordinates": [
                    -74.0167,
                    40.2905
                ]
            },
            "_id": "67d315e814c36f7a36986c01",
            "type": "flood",
            "location": "West Long Branch, NJ",
            "geoCoords": {
                "lat": 40.2905,
                "lng": -74.0167
            },
            "dateTime": "2025-02-01T14:30:00Z",
            "floodLevel": "Moderate"
        },
        {
            "geoCoords2": {
                "type": "Point",
                "coordinates": [
                    -74.0167,
```

6.4 Time-based Filtering API

Purpose: Retrieve flood or pollution reports that occurred within a specific date range.
Endpoints:
- Floods: POST /api/reports-by-date
- Pollution: POST /api/reports-by-date-pollution

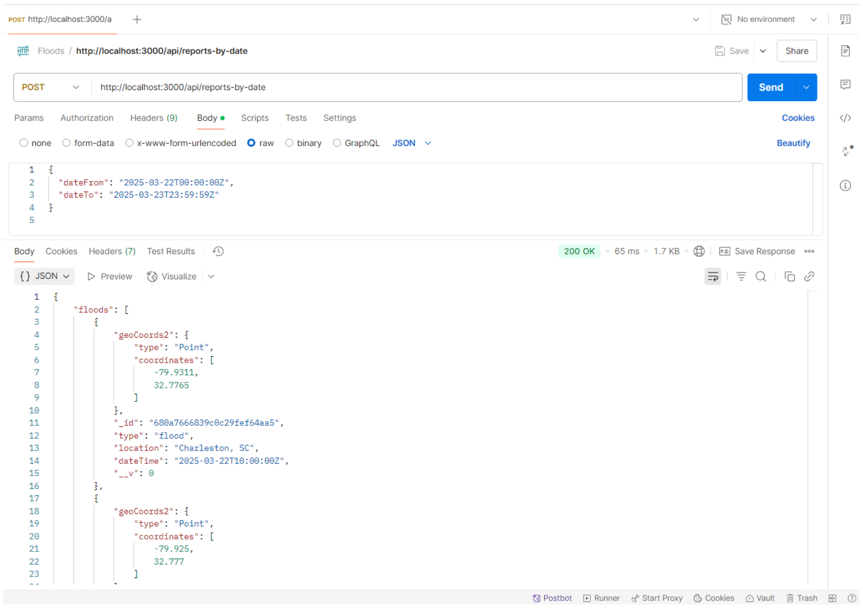
Key Logic:
- MongoDB date queries use $gte (greater than or equal) and $lte (less than or equal).
- Ensures accurate historical data retrieval.

Code:



```
const express = require('express');
const router = express.Router();

const FloodReport = require('./floodSchema');
const PollutionReport = require('./pollutionSchema');

Comment Code
router.post('/reports-by-date', async (req, res) => {
    const { dateFrom, dateTo } = req.body;

    if (!dateFrom || !dateTo) {
        return res.status(400).json({ error: 'dateFrom and dateTo are required.' });
    }

    try {
        const filter = {
            dateTime: {
                $gte: dateFrom,
                $lte: dateTo
            }
        };

        const floods = await FloodReport.find(filter);
        const pollution = await PollutionReport.find(filter);
        res.json({ floods, pollution });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});

module.exports = router;
```

Postman:



POST http://localhost:3000/a

No environment

Floods / http://localhost:3000/api/reports-by-date

Save | Share

POST | http://localhost:3000/api/reports-by-date | Send

Params | Authorization | Headers (9) | Body ● | Scripts | Tests | Settings | Cookies

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON | Beautify

```
{
  "dateFrom": "2025-03-22T00:00:00Z",
  "dateTo": "2025-03-23T23:59:59Z"
}

```

Body | Cookies | Headers (7) | Test Results | 200 OK · 65 ms · 1.7 KB | Save Response

{} JSON | Preview | Visualize

```
{
    "floods": [
        {
            "geoCoords2": {
                "type": "Point",
                "coordinates": [
                    -79.9311,
                    32.7765
                ]
            },
            "_id": "680a7666839c0c29fef64aa5",
            "type": "flood",
            "location": "Charleston, SC",
            "dateTime": "2025-03-22T10:00:00Z",
            "__v": 0
        },
        {
            "geoCoords2": {
                "type": "Point",
                "coordinates": [
                    -79.925,
                    32.777
                ]
```

Postbot | Runner | Start Proxy | Cookies | Vault | Trash

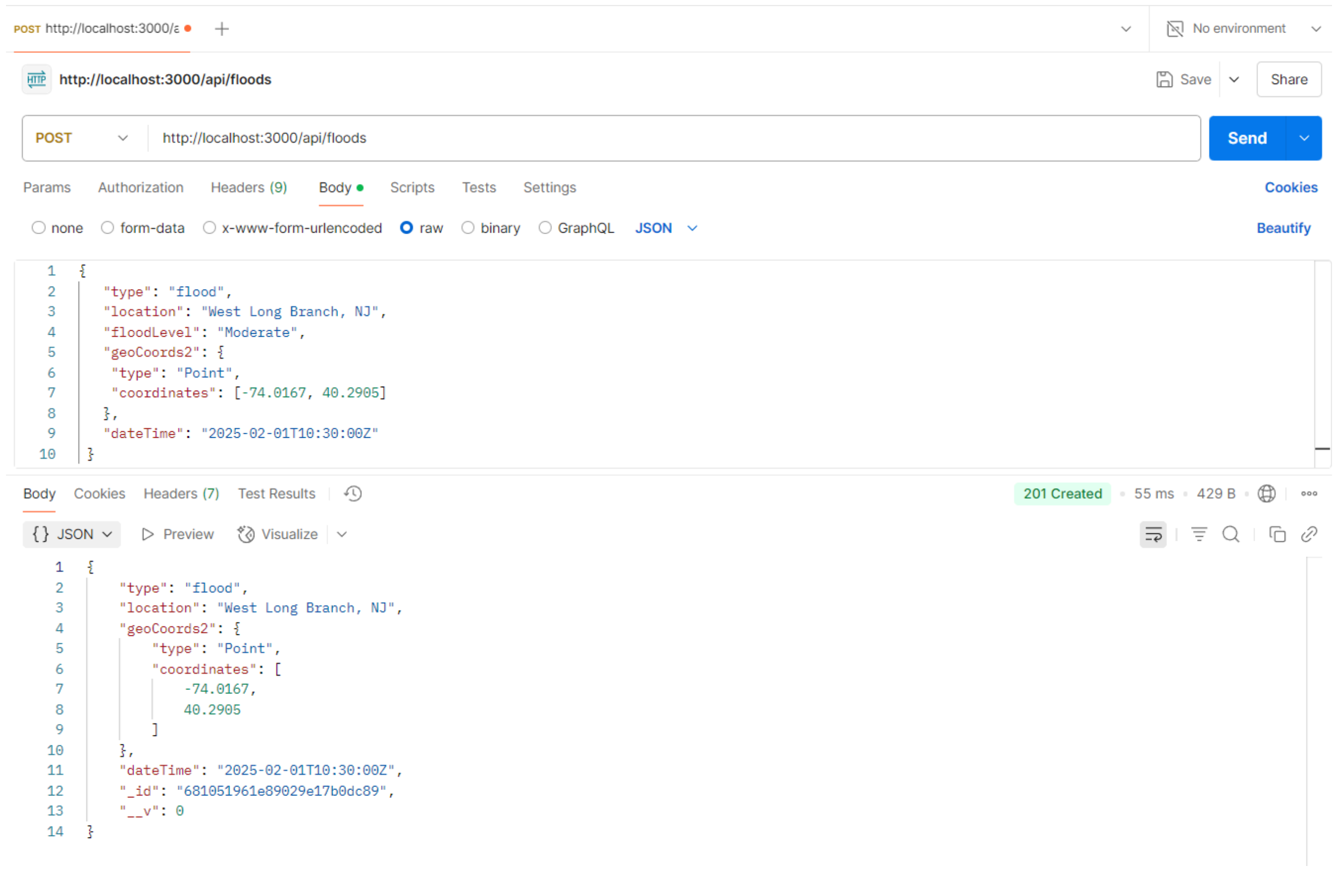
6.5 Key Takeaways for Advanced GIS APIs

- Polygon search checks if a point lies inside a polygon.
- Radius search finds nearest records within a defined range.
- Time-based filtering enables retrieval of records during any period.
- All searches are optimized using 2dsphere spatial indexing.

7. Full API Requests and Responses Samples

7.1 Flood APIs

7.1.1 Create Flood Report



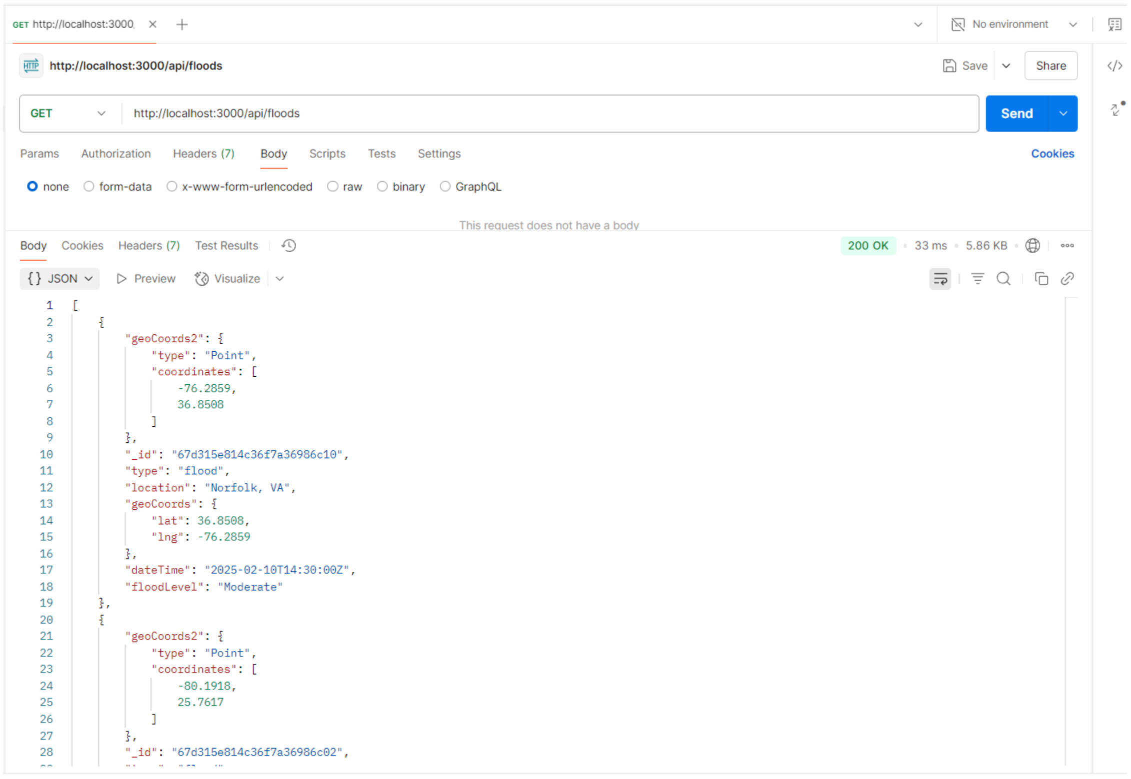
POST http://localhost:3000/api/floods

```
{
  "type": "flood",
  "location": "West Long Branch, NJ",
  "floodLevel": "Moderate",
  "geoCoords2": {
   "type": "Point",
   "coordinates": [-74.0167, 40.2905]
  },
  "dateTime": "2025-02-01T10:30:00Z"
}
```

201 Created · 55 ms · 429 B

```
{
    "type": "flood",
    "location": "West Long Branch, NJ",
    "geoCoords2": {
        "type": "Point",
        "coordinates": [
            -74.0167,
            40.2905
        ]
    },
    "dateTime": "2025-02-01T10:30:00Z",
    "_id": "681051961e89029e17b0dc89",
    "__v": 0
}
```

7.1.2 Read All Flood Reports



http://localhost:3000/api/floods

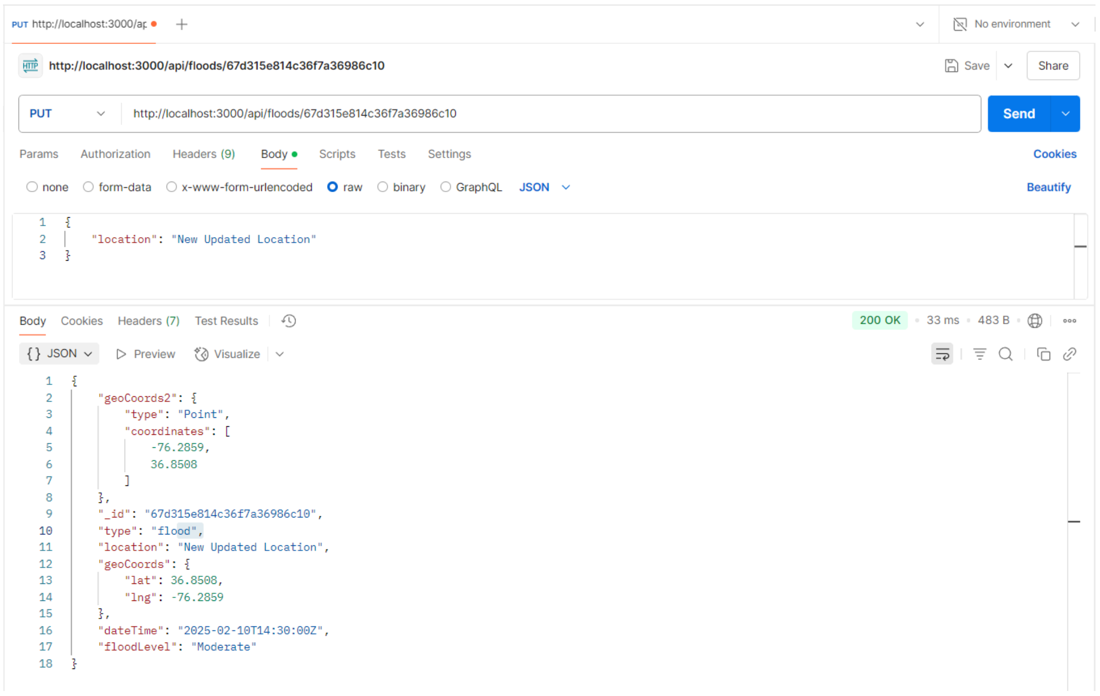
GET http://localhost:3000/api/floods

This request does not have a body

200 OK · 33 ms · 5.86 KB

```
[
    {
        "geoCoords2": {
            "type": "Point",
            "coordinates": [
                -76.2859,
                36.8508
            ]
        },
        "_id": "67d315e814c36f7a36986c10",
        "type": "flood",
        "location": "Norfolk, VA",
        "geoCoords": {
            "lat": 36.8508,
            "lng": -76.2859
        },
        "dateTime": "2025-02-10T14:30:00Z",
        "floodLevel": "Moderate"
    },
    {
        "geoCoords2": {
            "type": "Point",
            "coordinates": [
                -80.1918,
                25.7617
            ]
        },
        "_id": "67d315e814c36f7a36986c02",
```

7.1.3 Update Flood Report



http://localhost:3000/api/floods/67d315e814c36f7a36986c10

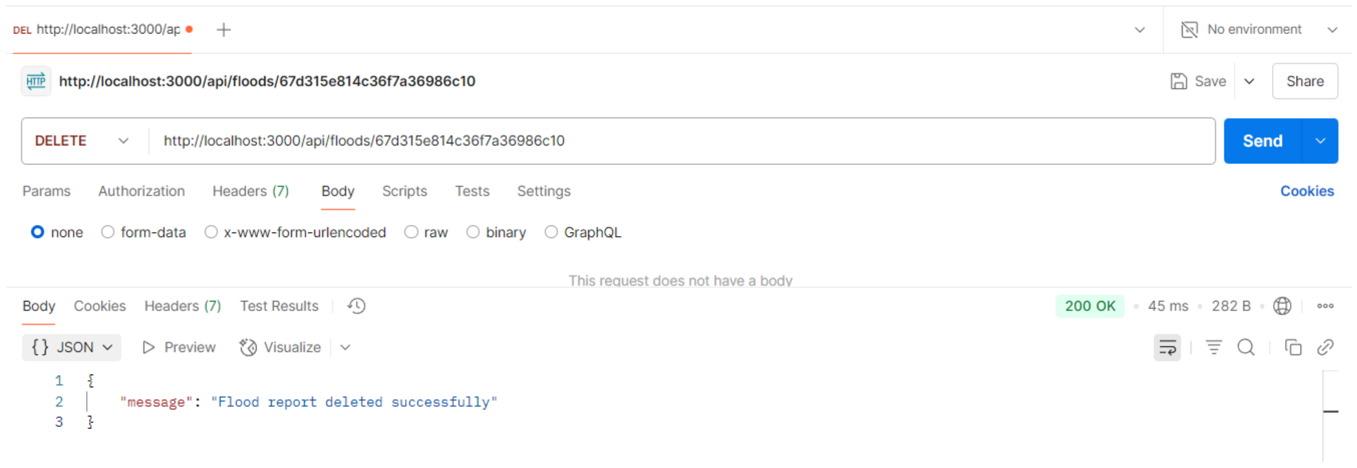
PUT http://localhost:3000/api/floods/67d315e814c36f7a36986c10

```
{
    "location": "New Updated Location"
}
```

200 OK · 33 ms · 483 B

```
{
    "geoCoords2": {
        "type": "Point",
        "coordinates": [
            -76.2859,
            36.8508
        ]
    },
    "_id": "67d315e814c36f7a36986c10",
    "type": "flood",
    "location": "New Updated Location",
    "geoCoords": {
        "lat": 36.8508,
        "lng": -76.2859
    },
    "dateTime": "2025-02-10T14:30:00Z",
    "floodLevel": "Moderate"
}
```

7.1.4 Delete Flood Report



http://localhost:3000/api/floods/67d315e814c36f7a36986c10

DELETE http://localhost:3000/api/floods/67d315e814c36f7a36986c10

Body: [✓ none] form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

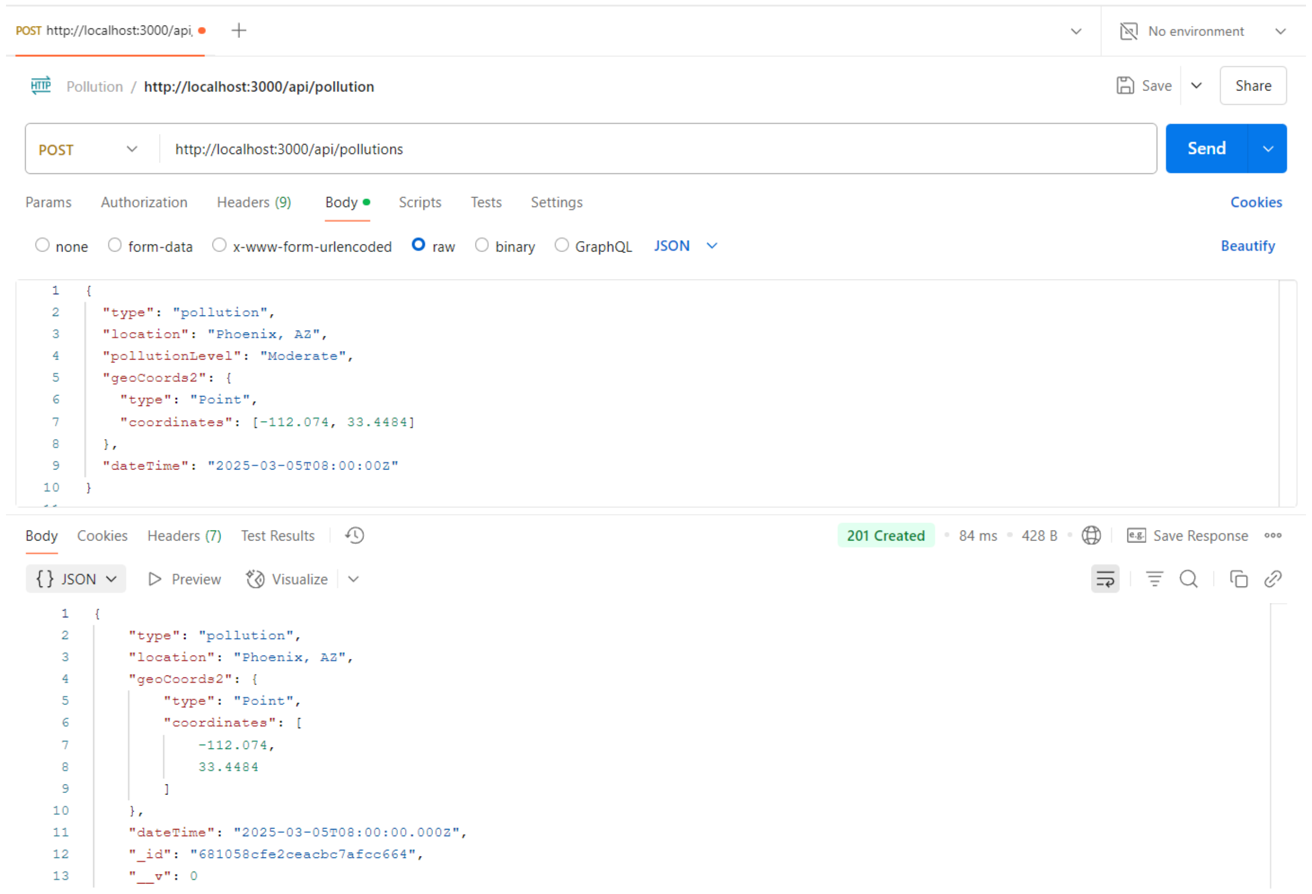
200 OK · 45 ms · 282 B

JSON

```
{
  "message": "Flood report deleted successfully"
}
```

7.2 Pollution APIs

7.2.1 Create Pollution Report



Pollution / http://localhost:3000/api/pollution

POST http://localhost:3000/api/pollutions

Body: none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

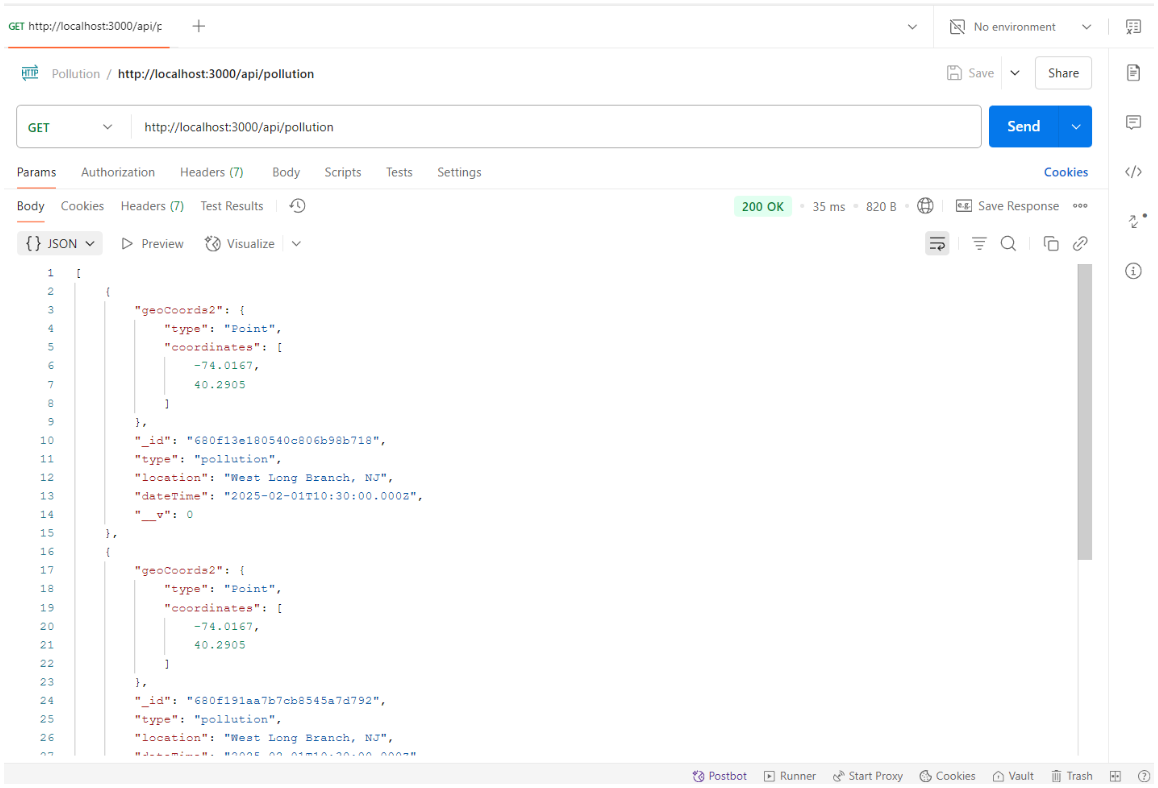
```
{
  "type": "pollution",
  "location": "Phoenix, AZ",
  "pollutionLevel": "Moderate",
  "geoCoords2": {
    "type": "Point",
    "coordinates": [-112.074, 33.4484]
  },
  "dateTime": "2025-03-05T08:00:00Z"
}
```

201 Created · 84 ms · 428 B

JSON

```
{
    "type": "pollution",
    "location": "Phoenix, AZ",
    "geoCoords2": {
        "type": "Point",
        "coordinates": [
            -112.074,
            33.4484
        ]
    },
    "dateTime": "2025-03-05T08:00:00.000Z",
    "_id": "681058cfe2ceacbc7afcc664",
    "__v": 0
```

7.2.2 Read All Pollution Reports


GET http://localhost:3000/api/p
No environment
Pollution / http://localhost:3000/api/pollution
Save
Share
Params Authorization Headers (7) Body Scripts Tests Settings
Cookies
Body Cookies Headers (7) Test Results
Save Response
JSON Preview Visualize
Postbot Runner Start Proxy Cookies Vault Trash

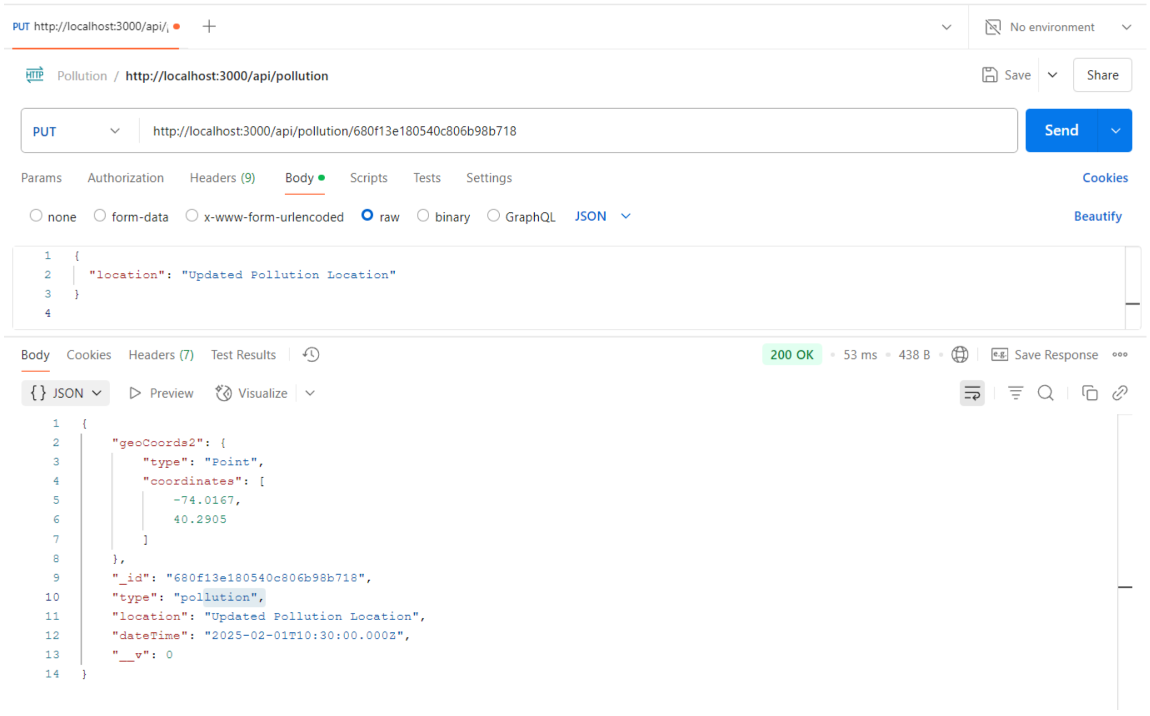
GET http://localhost:3000/api/pollution

Send

200 OK 35 ms 820 B

```
[
    {
        "geoCoords2": {
            "type": "Point",
            "coordinates": [
                -74.0167,
                40.2905
            ]
        },
        "_id": "680f13e180540c806b98b718",
        "type": "pollution",
        "location": "West Long Branch, NJ",
        "dateTime": "2025-02-01T10:30:00.000Z",
        "__v": 0
    },
    {
        "geoCoords2": {
            "type": "Point",
            "coordinates": [
                -74.0167,
                40.2905
            ]
        },
        "_id": "680f191aa7b7cb8545a7d792",
        "type": "pollution",
        "location": "West Long Branch, NJ",
```

7.2.3 Update Pollution Report


PUT http://localhost:3000/api/
No environment
Pollution / http://localhost:3000/api/pollution
Save
Share
Params Authorization Headers (9) Body Scripts Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (7) Test Results
Save Response
JSON Preview Visualize

PUT http://localhost:3000/api/pollution/680f13e180540c806b98b718

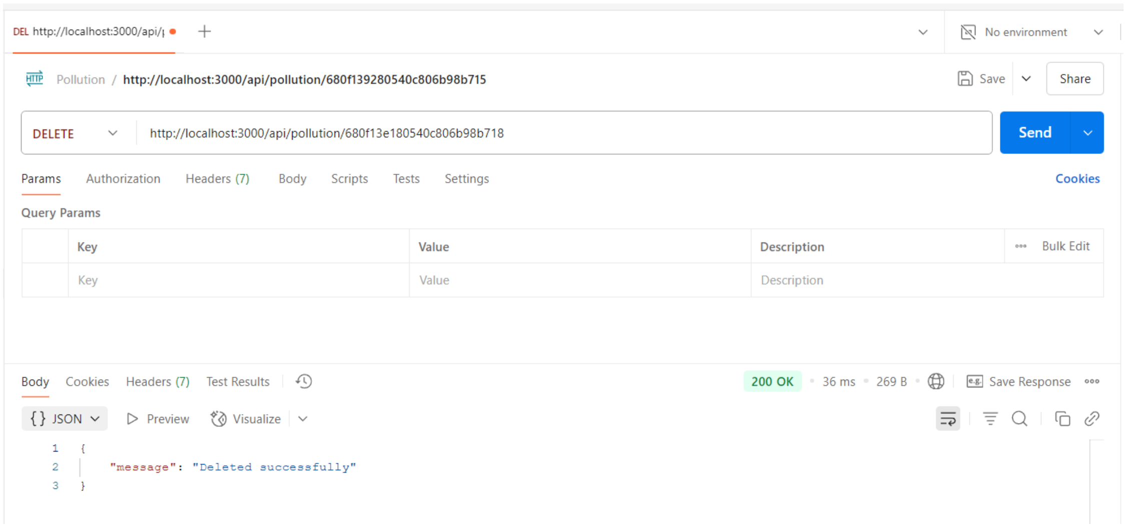
Send

```
{
    "location": "Updated Pollution Location"
}
```

200 OK 53 ms 438 B

```
{
    "geoCoords2": {
        "type": "Point",
        "coordinates": [
            -74.0167,
            40.2905
        ]
    },
    "_id": "680f13e180540c806b98b718",
    "type": "pollution",
    "location": "Updated Pollution Location",
    "dateTime": "2025-02-01T10:30:00.000Z",
    "__v": 0
}
```

7.2.4 Delete Pollution Report


DEL http://localhost:3000/api/pollution/680f139280540c806b98b715

DELETE http://localhost:3000/api/pollution/680f13e180540c806b98b718

Params · Authorization · Headers (7) · Body · Scripts · Tests · Settings

Query Params

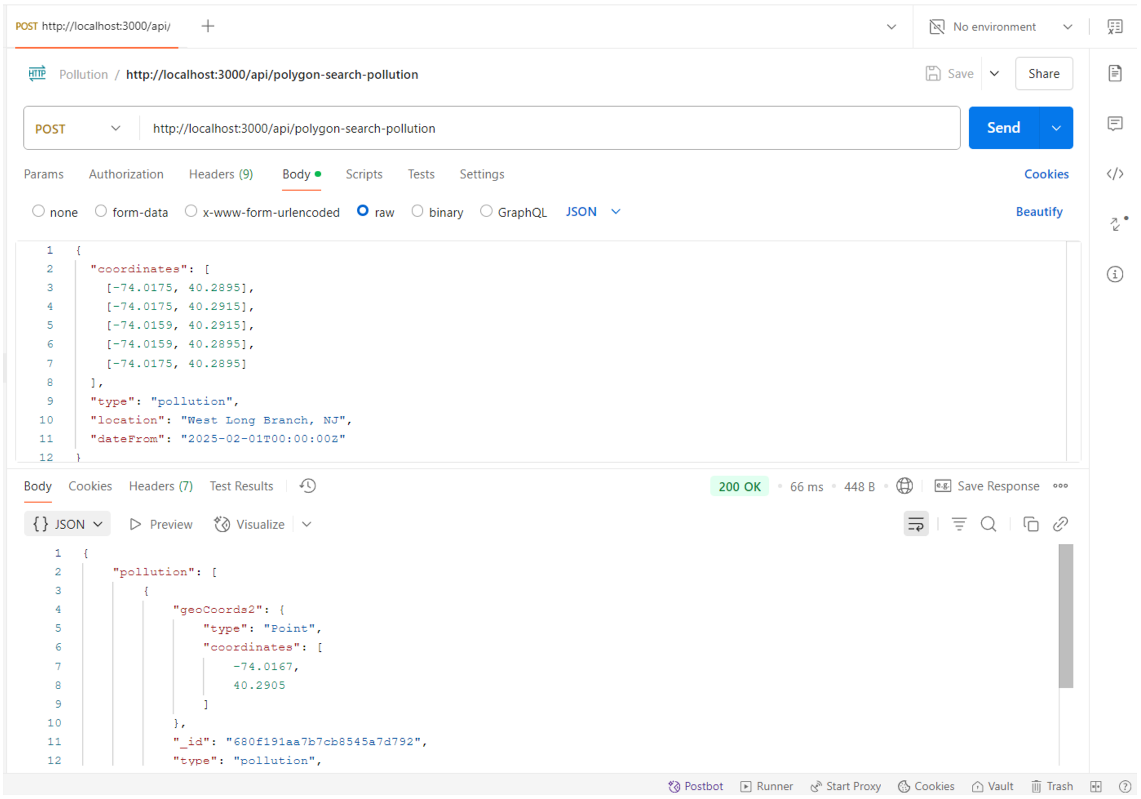
Key	Value	Description
Key	Value	Description

Body · Cookies · Headers (7) · Test Results — 200 OK · 36 ms · 269 B

```
{
    "message": "Deleted successfully"
}
```

7.3 GIS Specialized APIs

7.3.1 Polygon Search


POST http://localhost:3000/api/polygon-search-pollution

Params · Authorization · Headers (9) · Body · Scripts · Tests · Settings

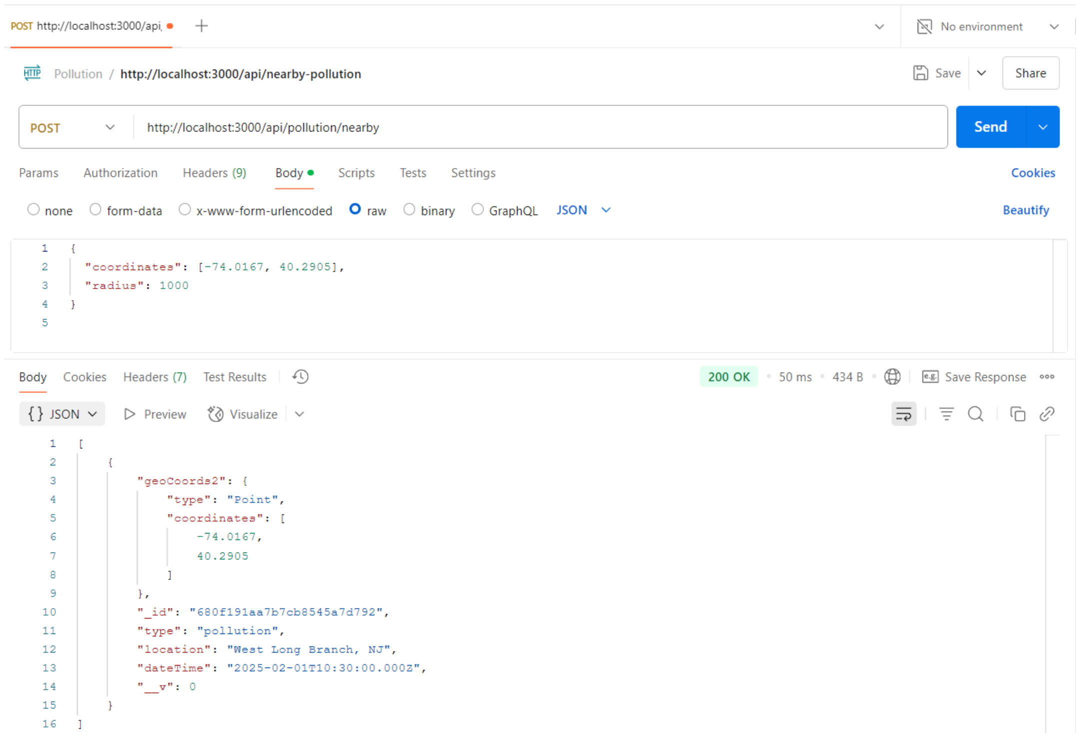
none · form-data · x-www-form-urlencoded · [✓ raw] · binary · GraphQL · JSON

```
{
  "coordinates": [
    [-74.0175, 40.2895],
    [-74.0175, 40.2915],
    [-74.0159, 40.2915],
    [-74.0159, 40.2895],
    [-74.0175, 40.2895]
  ],
  "type": "pollution",
  "location": "West Long Branch, NJ",
  "dateFrom": "2025-02-01T00:00:00Z"
}
```

Body · Cookies · Headers (7) · Test Results — 200 OK · 66 ms · 448 B

```
{
    "pollution": [
        {
            "geoCoords2": {
                "type": "Point",
                "coordinates": [
                    -74.0167,
                    40.2905
                ]
            },
            "_id": "680f191aa7b7cb8545a7d792",
            "type": "pollution",
```

7.3.2 Radius Search

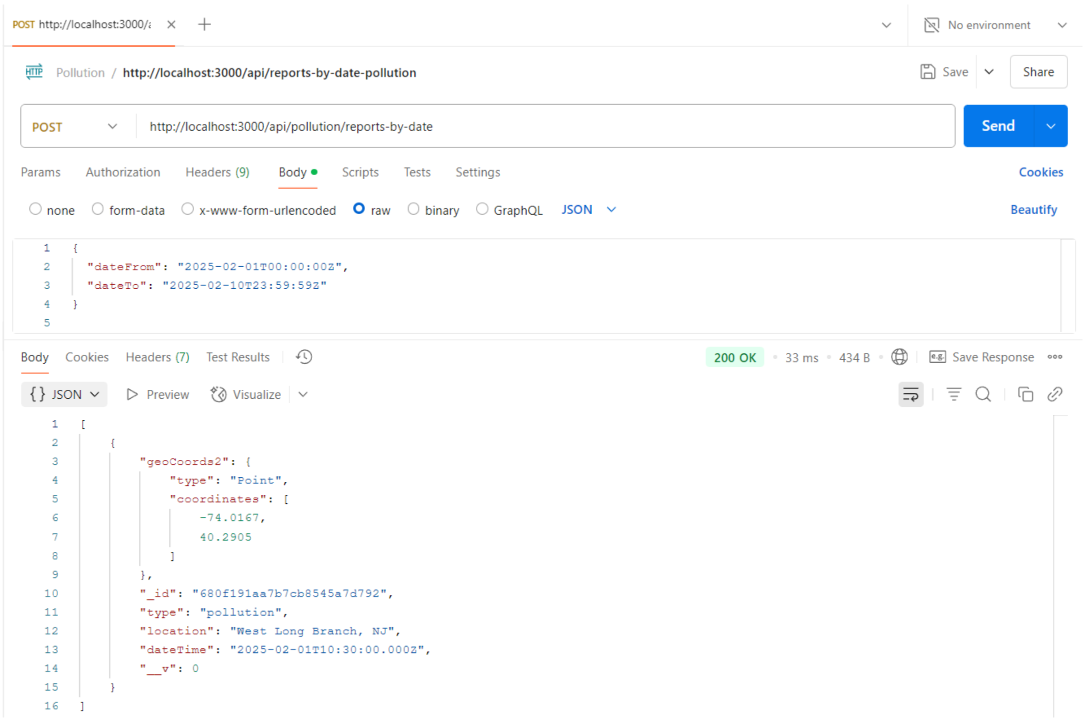

POST http://localhost:3000/api
No environment
Pollution / http://localhost:3000/api/nearby-pollution
Save
Share
Send
Params Authorization Headers (9) Body Scripts Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (7) Test Results
200 OK · 50 ms · 434 B
Save Response
{} JSON Preview Visualize

POST http://localhost:3000/api/pollution/nearby

```
{
  "coordinates": [-74.0167, 40.2905],
  "radius": 1000
}

```

```
[
    {
        "geoCoords2": {
            "type": "Point",
            "coordinates": [
                -74.0167,
                40.2905
            ]
        },
        "_id": "680f191aa7b7cb8545a7d792",
        "type": "pollution",
        "location": "West Long Branch, NJ",
        "dateTime": "2025-02-01T10:30:00.000Z",
        "__v": 0
    }
]
```

7.3.3 Time-Based Filtering


POST http://localhost:3000/a
No environment
Pollution / http://localhost:3000/api/reports-by-date-pollution
Save
Share
Send
Params Authorization Headers (9) Body Scripts Tests Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL JSON
Beautify
Body Cookies Headers (7) Test Results
200 OK · 33 ms · 434 B
Save Response
{} JSON Preview Visualize

POST http://localhost:3000/api/pollution/reports-by-date

```
{
  "dateFrom": "2025-02-01T00:00:00Z",
  "dateTo": "2025-02-10T23:59:59Z"
}

```

```
[
    {
        "geoCoords2": {
            "type": "Point",
            "coordinates": [
                -74.0167,
                40.2905
            ]
        },
        "_id": "680f191aa7b7cb8545a7d792",
        "type": "pollution",
        "location": "West Long Branch, NJ",
        "dateTime": "2025-02-01T10:30:00.000Z",
        "__v": 0
    }
]
```

8. Project Directory Structure and Explanation

8.1 Importance of Organized Structure

A clean project structure improves scalability, maintainability, and team collaboration. Each part of the application (Models, Routes, Main App) has its separate responsibilities.

8.2 Full Directory Layout

```
/GIS-Project
  |-- app.js (Flood main app)
  |-- pollution-app.js (Pollution main app)
  |-- floodSchema.js (Flood MongoDB schema)
  |-- pollutionSchema.js (Pollution MongoDB schema)
  |-- flood-crud.js (Flood CRUD operations)
  |-- pollution-crud.js (Pollution CRUD operations)
  |-- radius.js (Flood radius search)
  |-- time-based.js (Flood time-based search)
  |-- GroupByCountAPI.js (Flood Group and Count)
  |-- dashboard.js (Flood dashboard summaries)
```

8.3 Folder and File Responsibilities

- app.js: Main server file for flood-related APIs.
- pollution-app.js: Main server file for pollution-related APIs.

8.4 Benefits of This Structure

- Easier to maintain and debug.
- Easier to extend (e.g., adding new disaster modules like Earthquake).
- Clean division of logic improves professionalism and scalability.

9. Acknowledgments

This independent study project was conducted as part of the requirements for the Computer Science & Software Engineering program at Monmouth University.

I would like to extend my sincere gratitude to **Professor Raman Lakshmanan**, Specialist Professor, for his invaluable guidance, thoughtful feedback, and continuous encouragement throughout this project. His support was instrumental in shaping the direction of this work and ensuring its successful completion.

I also wish to acknowledge the role of open-source tools and platforms such as **Node.js**, **Express.js**, **MongoDB Atlas**, **Postman**, and **Visual Studio Code**. These tools provided the technical foundation and flexibility needed to develop and test a fully functional GIS-based backend system.

Finally, I am thankful for the opportunity to undertake this project independently. It has not only enhanced my technical skills in backend development and geospatial data handling, but also deepened my understanding of real-world system design and API architecture.

10. Conclusion and Future Scope

10.1 Conclusion

The GIS Flood and Pollution Management System successfully demonstrates the design and development of a modular, scalable, and real-world-ready backend GIS application. Using Node.js, Express.js, and MongoDB with 2dsphere indexing, the system achieves efficient CRUD operations, advanced geospatial queries (polygon and radius search), and time-based filtering.

Professional modular architecture ensures easy maintenance, debugging, and future expansion. All APIs were thoroughly tested using Postman to verify correctness and performance.

10.2 Future Scope

- Integrate Leaflet.js or Mapbox for frontend visualization.
- Implement JWT authentication for securing the APIs.
- Setup real-time alert systems for new severe incidents.
- Build a dashboard analytics page for visual trends and patterns.
- Deploy the application on cloud services like AWS or Heroku.
- Extend the system to support other disasters (earthquake, wildfire, air pollution).
- Implement versioned APIs and .env configurations for scalable production readiness.

10.3 Final Thoughts

This project not only fulfills academic requirements but lays down the foundation for building full-scale environmental monitoring and response systems. It combines modern technologies with best practices of backend architecture and GIS-based data handling.