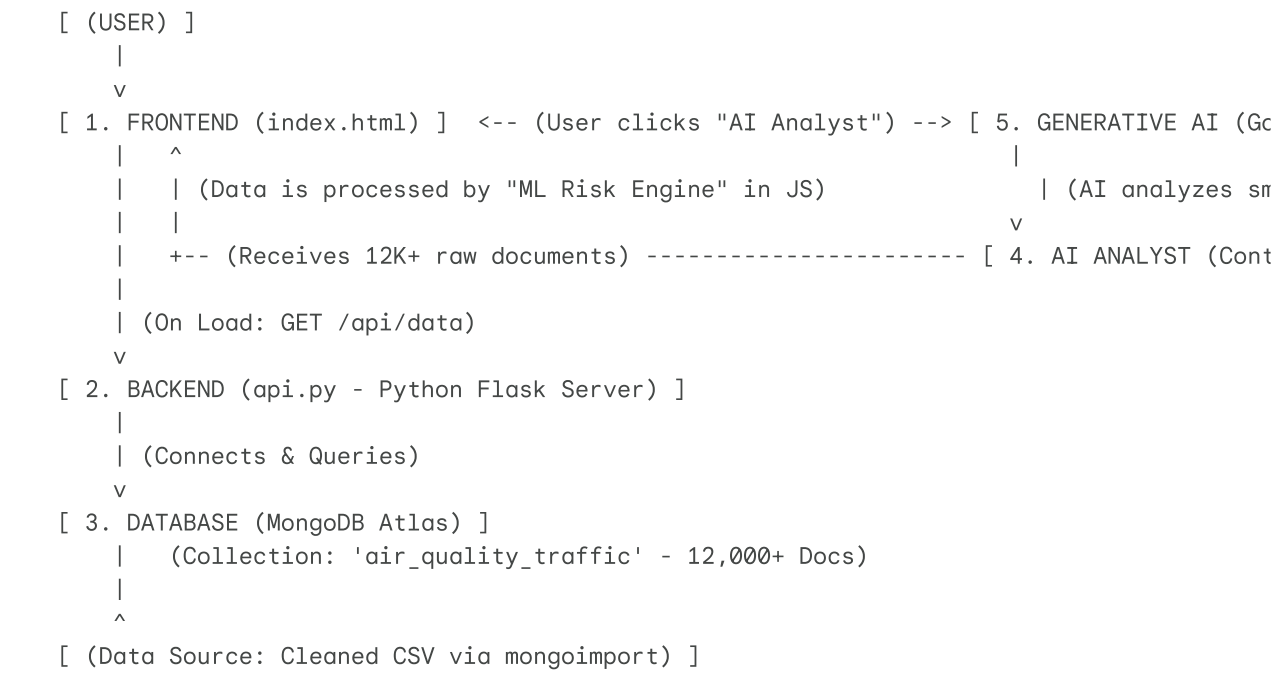
System Architecture: Unified AI Dashboard

This document outlines the full-stack architecture of the **NYC Environmental Risk Dashboard**, a real-time, AI-powered tool designed to move the DEP from a reactive to a proactive model.

The system is built on a modern, decoupled three-tier architecture, augmented by a fourth Generative AI service layer.

High-Level Architectural Flow

Here is the end-to-end flow of data, from the database to the user's screen:



Component Breakdown

1. Data & Storage Layer (The "MongoDB" Foundation)

This layer is the "single source of truth" for all city data.

- **Database: MongoDB Atlas (Cloud-Hosted)**
- **Database Name:** `myDatabase`
- **Collection:** `air_quality_traffic`
- **Schema:** The system leverages MongoDB's flexible, **polymorphic schema** to store the 12,000+ documents of raw, combined data (Air Quality, Traffic Volume, and Geospatial info) in a single, high-performance collection.

- **Ingestion:** Data was loaded from the team's cleaned CSV using the standard `mongoimport` tool, which directly maps CSV headers to JSON document keys.

2. Backend (Service Layer)

This is a lightweight "bridge" that securely connects our cloud database to the user's browser.

- **Framework: Python (Flask)**
- **Role:** This microservice (`api.py`) has one job: data provision. It decouples our frontend from the database, preventing our database credentials from ever being exposed to the user.
- **Endpoint:** `GET http://127.0.0.1:5001/api/data`
- **Function:** When called by the dashboard, this server connects to MongoDB Atlas, queries the `air_quality_traffic` collection, retrieves *all* documents, and serves them as a single, large JSON payload.

3. Frontend & "ML" Logic Layer (The "Spotify" Experience)

This is where all the "magic" happens. The `index.html` file is a powerful, single-page application (SPA).

- **Technology:** Vanilla JavaScript, Tailwind CSS, Chart.js, and Leaflet.js.
- **Role 1: Data Fetching:** On page load, it calls the `/api/data` endpoint to get the 12,000+ raw documents.
- **Role 2: The "ML Risk Engine" (`runRiskEngine()`):**
  - This is the *core* of our "proactive" model. It's a client-side JavaScript function that acts as our **ML aggregation engine**.
  - It loops through the 12,000+ raw documents and **aggregates them** by 'Geo Place Name' (e.g., "Brooklyn").
  - It calculates new, actionable metrics: `avg_pm25` and `avg_traffic` .
  - It runs our **predictive risk algorithm** (60% PM2.5, 40% Traffic) to create the final `riskScore` for each zone.
  - This engine transforms **Big Data (12K docs)** into **Smart Data (10-15 zones)**.
- **Role 3: The "Live Dashboard":** The application renders this new, aggregated "Smart Data" into all components:
  - **Priority Queue:** A "Spotify-style" list of the highest-risk zones.
  - **Geospatial Hotspots:** A Leaflet map with risk-scored markers.
  - **Meters & Charts:** Real-time visualizations of the aggregated data.
- **Role 4: "Simulate Alert" (Event-Driven UX):** The simulation button demonstrates the engine's power by modifying the raw data and re-running the *entire* aggregation and rendering pipeline in real-time, proving the system is dynamic and not static.

4. Generative AI Service (The "AI Analyst")

This layer solves Officer Sridevi's "fragmented tools" pain point by building an AI analyst *directly into* the dashboard.

- **Service: Google Gemini API** (External).
- **Data Flow:**
  1. The user opens the "AI Analyst" modal and asks a question (e.g., "Which 3 zones are at highest risk for next week?").
  2. The frontend bundles the user's question with the **processed "Smart Data"** **(** `globalRiskData` **)** as context.
  3. This small, relevant package (10-15 JSON objects) is sent to the Gemini API.
  4. Gemini's LLM performs a **descriptive or predictive analysis** on the *live data* and returns a plain-English text answer.
  5. The dashboard displays the answer, making the data conversational and actionable.