DATA ANALYTICS AND VISUALIZATION REPORT

Submitted by
Cenex Consult Limited
to
Clean Air Fund

Project Title: Nairobi City-Owned Air Quality Data Management System (AQDMS) & Public Data Portal

DOCUMENT NUMBER	RELEASE/REVISION NUMBER	RELEASE/REVISION DATE
4	V:01	31st October 2025
4	V:02	06 November 2025

Executive Summary

The Data Analytics and Visualization components of the Nairobi City-Owned Air Quality Data Management System (AQDMS) represent a pivotal advancement in Nairobi's environmental intelligence and civic data infrastructure. Developed by Cenex Consult Limited under the Clean Air Fund's Breathe Cities initiative, this system transforms raw sensor data into actionable insights for government officials, researchers, and the public.

At its core, the AQDMS analytics engine is a modular, scalable platform built on open-source technologies including Node.js, Express.js, PostgreSQL, and React. It ingests real-time air quality readings from low-cost sensors deployed across Nairobi City County, enabling automated data validation, aggregation, and visualization. The system supports both technical and non-technical users through two tailored interfaces: a County Management Dashboard for strategic planning and a Public Data Portal for simplified, mobile-friendly access.

The development process followed a phased, stakeholder-driven approach—beginning with system design and consultations, followed by backend and frontend implementation, API development, and iterative testing. County ICT teams, environmental experts, and community representatives provided critical feedback on pollutant labelling, spatial analysis tools, and user experience enhancements.

Key analytics features include:

- ✓ Real-time pollutant monitoring and anomaly detection
- ✓ Historical trend analysis with customizable date ranges
- ✓ Geospatial mapping of pollution hotspots using Leaflet
- ✓ Threshold-based alerting and exportable reports (CSV, PDF)
- ✓ Role-based access control and secure API endpoints

Performance testing confirmed the system's ability to handle high-frequency sensor data while maintaining responsive dashboards. Security protocols such as JWT authentication and SSL encryption safeguard data integrity and user privacy. The API layer ensures interoperability with external platforms, enabling future integration and regional data sharing.

Together, these analytics and visualization tools empower Nairobi City County to monitor air quality in real time, inform policy with reliable data, and engage the public in environmental awareness. The AQDMS lays the groundwork for a more transparent, data-driven, and resilient urban future.

Acknowledgement

The successful design and implementation of the Data Analytics and Visualization components within the Nairobi City-Owned Air Quality Data Management System (AQDMS) and Public Data Portal were made possible through the collective expertise, commitment, and collaboration of diverse stakeholders.

We extend our sincere appreciation to the Nairobi City County Government ICT and Environment Departments, whose technical guidance and institutional support were instrumental throughout the analytics lifecycle. Their dedication to digital innovation and environmental stewardship laid the groundwork for a system that transforms raw sensor data into actionable insights—advancing public health, transparency, and data-driven governance.

We especially acknowledge the Clean Air Fund for their strategic partnership and unwavering advocacy under the Breathe Cities initiative. Their leadership in convening stakeholders, facilitating technical dialogue, and championing open data principles was central to shaping the analytics framework. The collaboration with C40 Cities and Bloomberg Philanthropies enriched the system with global best practices and policy alignment.

Our gratitude also goes to environmental experts and community stakeholders who contributed domain knowledge and local perspectives during the validation and usability testing of the analytics dashboards. Their input on pollutant labelling, sensor calibration, and visualization logic helped ensure the platform is both technically robust and socially inclusive.

We recognize the citizen testers and public users whose engagement during the feedback and optimization phases helped refine the user experience. Their participation ensured that the analytics tools remain intuitive, accessible, and responsive to community needs—reflecting a growing civic commitment to environmental awareness and digital empowerment.

The shared dedication to a cleaner, healthier Nairobi was evident in the collaborative spirit, technical rigor, and forward-thinking contributions that guided the development of the AQDMS analytics system. As the platform continues to evolve, these insights will remain vital in fostering inclusive governance, environmental resilience, and sustainable urban development.

Table of Contents

1. Introduction

This report outlines the design and implementation of the data analytics and visualization components within the Nairobi City-Owned Air Quality Data Management System (AQDMS) and its Public Data Portal. Developed in close collaboration with the Nairobi City County Government, the AQDMS serves as a cornerstone of the city's digital infrastructure for environmental monitoring and public health awareness.

The AQDMS was conceived to address the urgent need for real-time and historical access to air quality data, empowering a wide range of stakeholders—including policymakers, environmental researchers, and the general public—to make informed decisions grounded in reliable data.

This initiative is part of the broader Breathe Cities program, delivered by the Clean Air Fund, C40 Cities, and Bloomberg Philanthropies, which supports cities globally in reducing air pollution and greenhouse gas emissions. In Kenya, Breathe Cities is actively partnering with Nairobi City County to develop a localized AQDMS that reflects the city's unique environmental challenges while aligning with both national and international air quality standards.

2. Data Analytics Tools

2.1. Objectives

The data analytics layer of the Air Quality Data Management System (AQDMS) was developed to:

- Transform raw sensor data into actionable insights for both technical and non-technical users.
- Support time-based and spatial aggregation of air quality readings across Nairobi's sensor network.
- Enable visual exploration of trends, anomalies, and pollution patterns to inform public awareness and policy decisions.

2.2. Key Features

The analytics engine integrates real-time processing, automated aggregation, and interactive visualization through the following core features:

- Automated Data Ingestion: Continuously collects air quality readings from NCCG-owned sensors via secure API endpoints.
- Data Aggregation: Organizes data by station, pollutant type (e.g., PM2.5, PM10, NO_2), and time intervals (hourly, daily, weekly) to support flexible analysis.
- Real-Time Processing: Background jobs compute rolling averages, detect anomalies, and flag threshold exceedances for alert generation.
- Visualization Dashboards: Interactive dashboards display time-series graphs, pollutant breakdowns, and geospatial maps for trend monitoring and hotspot identification.

2.3. Tools & Technologies

Component	Technology Stack
Data Processing	**Node.js (Express), Prisma ORM for optimized queries**
Database	**On-premise PostgreSQL with structured schema**
Visualization	**Chart.js and React-based dashboards**
Scripting & Automation	**Cron jobs and API integrations for sensor data retrieval**

These tools were selected for their performance, scalability, and compatibility with open-source standards and Nairobi City County ICT infrastructure.

2.4. Data Analytics Implementation

The backend analytics are powered by a robust Node.js and Express API service layer. The implementation ensures efficient, reliable, and secure data flow from sensors to the visualization frontend.

2.4.1 API Service Layer

The system uses a centralized API service to handle all data communication between the frontend and backend, ensuring consistent error handling and JSON parsing.

```
// lib/api.ts

const BASE_URL = "https://xp-backend.sytes.net/api/v1";

//  ◇  Common helper to handle fetch + JSON parsing + errors

const fetchJSON = async (url: string, options: RequestInit = {}) => {

 const response = await fetch(url, {

  ...options,

  headers: {

   "Content-Type": "application/json",

   ...(options.headers || {}),

  },

});


 if (!response.ok) {

  const text = await response.text();
```

```
  throw new Error(`Request failed: ${response.status} ${response.statusText} - ${text}`);
 }

 return response.json();
};
```

Fig 1. Centralized API Communication Helper This reusable function standardizes all API interactions across the AQDMS frontend. It ensures consistent request formatting, header management, and JSON parsing, while providing robust error handling. By throwing descriptive errors on failed requests, it enables graceful UI recovery and improves debugging efficiency during data fetch operations.

2.4.2 Real-Time Station Data Fetching

This function retrieves current air quality readings from all monitoring stations, normalizing the data for consistent display.

typescript

```
// lib/api.ts - Fetch all stations
export const getStations = async (): Promise<Station[]> => {
 const data = await fetchJSON(`${BASE_URL}/stations`);
 const stations = data.data;

 return Array.isArray(stations)
  ? stations.map((station: any) => ({
    ...station,
```

```
      pm10: station.pm10 ? Number(station.pm10.toFixed(2)) : null,
      pm25: station.pm25 ? Number(station.pm25.toFixed(2)) : null,
    }))
  : stations;
};
```

Fig 2. Real-Time Station Data Fetching: *This function retrieves current air quality readings from all monitoring stations. It normalizes pollutant values (PM2.5, PM10) for consistent display and handles both array and single-object responses, ensuring frontend resilience.*

.

2.4.3 Historical Data Aggregation

This function fetches time-series data for trend analysis, supporting the analytics dashboard's ability to display pollution patterns over time.

typescript

```
// lib/api.ts - Fetch historical readings
// ◇ Get historical readings for a given sensor
export const getHistoricalData = async (sensorId: string, period = 24): Promise<HistoricalData[]> => {
 const data = await fetchJSON(`${BASE_URL}/stations/${sensorId}/readings?range=${period}&direction=asc&sort=timeStamp`);
 return data.data;
};
```

Fig 3. Historical Data Aggregation Function: *Fetches time-series pollutant readings for a given station over a configurable period. The data is sorted chronologically to support trend visualization and comparative analysis across stations.*

2.5. Stakeholder Feedback

Stakeholder engagement played a critical role in shaping the analytics experience. Feedback from county officials, environmental experts, and community testers led to several refinements:

- Improved pollutant labelling and truncation (e.g., PM2.5, PM10) for clarity.
- Trend comparison tools across multiple city stations to support spatial analysis.
- CSV and PDF export options for offline reporting and data sharing.
- Timestamp overlays on geospatial maps for temporal context.
- Enhanced threshold labelling to highlight health-relevant pollution levels.
- Date-range filters on all charts to support custom time-series exploration.

These enhancements ensure the analytics tools are not only technically robust but also user-friendly, interpretable, and aligned with real-world decision-making needs.

3. Public Data Portal

3.1. Objectives

The Public Data Portal is a core component of the AQDMS, designed to:

- Provide transparent, real-time access to air quality data for Nairobi residents.
- Enable users to view, compare, and subscribe to alerts from stations across the city.
- Support integration with third-party applications and research platforms via a public API.

This portal empowers citizens with actionable environmental insights while reinforcing government transparency and civic engagement.

3.2. Key Features

- Interactive Data Views: Displays pollutant readings by time, location, and station, with dynamic charts and maps.
- Search & Filters: Users can filter data by location, pollutant type (e.g., PM2.5, NO_2), and custom date ranges.
- Feedback Mechanism: Allows users to submit comments, suggestions, or concerns directly through the portal.
- Alert Subscription: Enables users to subscribe to pollution alerts for specific stations or zones of interest.
- Mobile Responsiveness: Optimized for accessibility across smartphones, tablets, and desktops.

3.3. Architecture

Layer	Technology Stack
Frontend	**Next.js (React-based framework)**

Backend	**Node.js with Express API, Prisma ORM**
Database	**PostgreSQL (on-premise deployment)**

This architecture ensures modularity, performance, and scalability while aligning with Nairobi City County ICT standards.

3.4. Public Portal Implementation

The portal is built as a responsive Single-Page Application (SPA) using Next.js 14. The main dashboard component centralizes state management for stations, user selections, and UI state.

3.4.1 Main Dashboard Component & State Management

```
const [stations, setStations] = useState<Station[]>([]);

 const [loading, setLoading] = useState(false);

 const [trendsStation, setTrendsStation] = useState<Station | null>(null);

 const [selectedStation, setSelectedStation] = useState<Station | null>(null);

 const [isTrendsDropdownOpen, setIsTrendsDropdownOpen] = useState(false);


 const [selectedPollutant, setSelectedPollutant] =

   useState<PollutantType>("aqi");


 // const intervalRef = useRef<NodeJS.Timeout | null>(null);
```

```
const [searchTerm, setSearchTerm] = useState("");

const [currentTime, setCurrentTime] = useState<string>("");
```

```
// Effect #1: Fetch station

// Effect #1: Fetch stations on mount and every 60 seconds

useEffect(() => {

  const fetchStations = async () => {

    try {

      const stationsData = await getStations();

      setStations(stationsData);

      setTrendsStation(stationsData[0]);

    } catch (err: any) {

      console.log(err.message || "Failed to load stations");

    } finally {

      setLoading(false);

    }

  };


  // Initial fetch

  fetchStations();

}, []);
```

***Fig 4. Main Dashboard State Management**: React hooks manage core application state—station data, user selections, loading status, and pollutant filters. This centralized state logic ensures synchronized updates across dashboard components*

3.4.2 Interactive Map Visualization with Leaflet

The portal features an interactive map that provides a geospatial view of air quality across Nairobi.

```
useEffect(() => {
  if (mapRef.current && !leafletMapRef.current) {
    // Initialize the map
    leafletMapRef.current = L.map(mapRef.current).setView(
      [-1.2921, 36.8719],
      12
    );

    // Add tile layer
    L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {
      attribution:
        '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors',
    }).addTo(leafletMapRef.current);

    markerGroupRef.current = L.layerGroup().addTo(leafletMapRef.current);
```

```
  }

  // Clear and add new markers
  if (leafletMapRef.current && markerGroupRef.current) {
    markerGroupRef.current.clearLayers();

    stations.forEach((station) => {
      const marker = L.marker([station.lat, station.lng], {
        icon: createCustomIcon(
          station,
          selectedStation?.id === station.id
        ),
      });

      const value = getPollutantValue(station, selectedPollutant);
      const unit = selectedPollutant === "aqi" ? "" : "µg/m³";

          marker.bindPopup(`
            <div class="p-2">
              <div class="text-sm text-[#667085] mb-1">
                <span class="font-semibold text-[#101828]">${station.name}</span>
```

```
            </div>
            <div class="text-xs text-[#667085] mb-2">
             ${
               station.timeStamp
                 ? new Date(station.timeStamp).toLocaleString()
                 : "No data available"
             }
            </div>
            <div class="text-sm text-[#667085]">
              ${selectedPollutant.toUpperCase()}:          ${value}${unit}          -
${getPollutantLevel(
            value,
            selectedPollutant
          )}
            </div>
            <div class="mt-2 text-xs text-[#667085]">
              <div>AQI: ${station.aqi}</div>
              <div>PM2.5: ${station.pm25} µg/m³</div>
              <div>PM10: ${station.pm10} µg/m³</div>
            </div>
          </div>
        `);
```

```
    marker.on("click", () => {
      onStationSelect(station);
    });

    marker.addTo(markerGroupRef.current!);
  });

  if (selectedStation) {
    leafletMapRef.current.setView(
      [selectedStation.lat, selectedStation.lng],
      12,
      { animate: true }
    );
  }
}

return () => {
  // Clean up map on unmount
  if (leafletMapRef.current) {
    leafletMapRef.current.remove();
```

```
      leafletMapRef.current = null;
    }
  };
}, [
  stations,
  selectedStation,
  selectedPollutant,
  getPollutantValue,
  getPollutantColor,
  getPollutantLevel,
  onStationSelect,
]);

return <div ref={mapRef} style={{ height: "100%", width: "100%" }} />;
}
```

Fig 5. Interactive Map Initialization with Leaflet*: Initializes a geospatial map centered on Nairobi, adds OpenStreetMap tiles, and prepares a marker layer for dynamic station rendering. This forms the foundation for hotspot visualization and station-level interaction.*

3.4.3 Station Ranking System

The portal displays side-by-side rankings of the cleanest and most polluted stations.

```
const value = getPollutantValue(station, selectedPollutant);
```

```
  const color = getPollutantColor(value, selectedPollutant);

  return divIcon({

    html: `

      <div style="

        width: 2rem;

        height: 2rem;

        background-color: ${color};

        border: 2px solid white;

        border-radius: 50%;

        display: flex;

        align-items: center;

        justify-content: center;

        font-size: 12px;

        font-weight: 600;

        color: #101828;

        box-shadow: 0 4px 6px -1px rgba(0, 0, 0, 0.1);

        cursor: pointer;

        transition: all 0.2s;

        ${isSelected ? "transform: scale(1.6); box-shadow: 0 8px 25px -5px
rgba(0, 0, 0, 0.2);" : ""}

      ">

        ${value}
```

```
      </div>
    `,
    className: "custom-marker",
    iconSize: [27, 27],
    iconAnchor: [16, 16],
  });
};


useEffect(() => {
  if (mapRef.current && !leafletMapRef.current) {
    // Initialize the map
    leafletMapRef.current = L.map(mapRef.current).setView(
      [-1.2921, 36.8719],
      12
    );

    // Add tile layer
    L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {
      attribution:
        '&copy;                                                    <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors',
```

```
  }).addTo(leafletMapRef.current);


  markerGroupRef.current = L.layerGroup().addTo(leafletMapRef.current);
}


// Clear and add new markers
if (leafletMapRef.current && markerGroupRef.current) {
  markerGroupRef.current.clearLayers();


  stations.forEach((station) => {
    const marker = L.marker([station.lat, station.lng], {
      icon: createCustomIcon(
        station,
        selectedStation?.id === station.id
      ),
    });
```

Fig 6. Custom Map Marker Rendering: Generates color-coded circular markers based on pollutant severity using WHO AQI thresholds. Selected markers scale up visually, enhancing user feedback and spatial awareness of pollution levels.

3.4.4 Alert Subscription System

The system allows users to subscribe to email alerts based on air quality thresholds.

typescript

// lib/api.ts - Subscribe to alerts

```
// ◇ Subscribe to alerts

export const subscribeToAlerts = async (payload: any): Promise<any> =>
{

 const data = await fetchJSON(`${BASE_URL}/alerts/users`, {

   method: "POST",

   body: JSON.stringify(payload),

 });

 return data.data;

};
```

// components/email-alert-section.tsx - Alert form submission

```
const AQI_LEVELS: Record<string, number> = {

 "Unhealthy for Sensitive Groups": 101,

 "Unhealthy": 151,

 "Very Unhealthy": 201,

 "Hazardous": 301,

};
```

```
const onSubmit = async (data: EmailAlertData) => {

  try {
```

```
    setServerError("");

    // 🧠 Derive AQI based on alert level
    const aqiValue = AQI_LEVELS[data.alertLevel] || 0;

    const payload = {
      ...data,
      aqi: aqiValue, // 👈 include AQI in the payload
    };

    console.log("Submitting:", payload);

    await subscribeToAlerts(payload);
    reset();
    setSelectedIds([]);
    setIsOpen(false);
  } catch (err) {
    console.error(err);
    setServerError("Something went wrong. Try again.");
  }
};
```

```
```

Fig 7. Alert Subscription Payload Construction: *Converts human-readable alert levels into AQI thresholds before submitting user preferences to the backend. This abstraction simplifies user interaction while maintaining backend compatibility.*

3.4.5 Feedback System Implementation

An integrated feedback system allows users to rate the portal and submit comments.

typescript

```
// lib/api.ts - Submit feedback

export const submitFeedback = async (payload: any): Promise<any> =>
{
 const data = await fetchJSON(`${BASE_URL}/feedback`, {
   method: "POST",
   body: JSON.stringify(payload),
 });
 return data.data;
};
```

```
// components/feedbacksection.tsx - Star rating state management

const FeedbackSection = () => {
 const [rating, setRating] = useState(0);
 const [hoveredRating, setHoveredRating] = useState(0);
 const [isSubmitted, setIsSubmitted] = useState(false);
```

```
const [isLoading, setIsLoading] = useState(false);


const {
  register,
  handleSubmit,
  reset,
  setError,
  clearErrors,
  formState: { errors },
} = useForm<FeedbackData>();


const onSubmit = async (data: FeedbackData) => {
  if (rating === 0) {
    setError("rating", { type: "manual", message: "Please select a rating." });
    return;
  }


  setIsLoading(true);
  try {
    //feedback → message
    await submitFeedback({
```

```
      name: data.name,
      email: data.email,
      message: data.feedback,
      rating,
    });

    setIsSubmitted(true);
    setTimeout(() => {
      setIsSubmitted(false);
      reset();
      setRating(0);
    }, 3000);
  } catch (error) {
    console.error("Error submitting feedback:", error);
    setError("root", {
      type: "server",
      message: "Something went wrong. Please try again later.",
    });
  } finally {
    setIsLoading(false);
  }
```

```
};


const getRatingText = (r: number) =>
 ({
   1: "Poor",
   2: "Fair",
   3: "Good",
   4: "Very Good",
   5: "Excellent",
 }[r] || "");
```

Fig 8. Feedback Submission with Star Rating*: Implements a star-based rating system with hover effects and descriptive labels. Captures user feedback and sentiment, contributing to iterative portal improvements and civic engagement.*

3.5. Access & Security

- Role-Based Access Control: County officials and authenticated users have tiered access to administrative features and data exports.
- HTTPS Enforcement: All endpoints are secured with SSL/TLS encryption to protect data in transit.
- Backup & Migration Tracking: Regular automated backups and schema migration tracking via Prisma ORM ensure data integrity and disaster recovery readiness.

4. Dashboard Analytics System

4.1. Overview & Architecture

The Dashboard Analytics System provides Nairobi City County administrators and researchers with comprehensive air quality monitoring and management capabilities through a role-based access control system. The system serves two primary user roles with differentiated permissions and functionalities.

4.1.1. User Roles & Permissions

Role	Access Level	Features
Researcher	Limited	Overview, Sensors, Analytics pages
Administrator	Full	All features + User Management

4.1.2. System Architecture

```
frontend: React.js + TypeScript + Material-UI;
authentication: JWT + Role-Based Access Control;
dataLayer: Axios API Client + Real-time Updates;
components: Modular React Components;
routing: Protected Routes + Role Guards;
```

4.2. Authentication & Security Implementation

4.2.1. JWT Authentication System

4.2.2. Route Protection & Role Guards

4.3. Core Dashboard Pages Implementation

4.3.1. Overview Page - Real-time Monitoring

```
import { useEffect, useState } from 'react';


// material-ui

import { useTheme } from '@mui/material/styles';

import { Grid, SelectChangeEvent, Typography } from '@mui/material';


// project import

import MainCard from 'components/MainCard';


import WelcomeBanner from 'sections/dashboard/analytics/WelcomeBanner';


// assets

import { DownloadOutlined, BarChartOutlined, CalendarOutlined,
FileTextOutlined } from '@ant-design/icons';

import ReportCard from 'components/cards/statistics/ReportCard';

import MapComponent from 'sections/map/maps-component';
```

```
import { getStations, getStationsCron, Station } from 'api/maps-api';

import Legend from 'components/Legend';

import CurrentReadingTable from 'sections/data-
tables/CurrentReadingsTabletable';


import SensorsOutlinedIcon from '@mui/icons-material/SensorsOutlined';

import SensorsOffOutlinedIcon from '@mui/icons-material/SensorsOffOutlined';


type PollutantType = 'aqi' | 'pm25' | 'pm10';


// ==============================|| DASHBOARD - OVERVIEW
||============================== //


const DashboardAnalytics = () => {

 const theme = useTheme();


 const [loading, setLoading] = useState(false);

 const [stations, setStations] = useState<Station[]>([]);

 const [selectedStation, setSelectedStation] = useState<Station | null>(null);


 const [currentAlerts, setCurrentAlerts] = useState<number>(0);
```

```
const [averageAqi, setAverageAqi] = useState<number>(0);

const [offlineSensors, setOfflineSensors ] = useState<number>(0)


useEffect(() => {

  const fetchStations = async () => {

    try {

      const stationsData = await getStations();

      setStations(stationsData);


      // Calculate average AQI

      const totalAqi = stationsData.reduce((sum: number, station: Station) => sum
+ (station.aqi || 0), 0);

      const avgAqi = stationsData.length > 0 ? totalAqi / stationsData.length : 0;

      setAverageAqi(avgAqi);


      // Calculate current alerts (e.g. AQI > 100 or alertLevel !== "Good")

      const alertsCount = stationsData.filter((station: Station) => station.aqi >
100).length;

      setCurrentAlerts(alertsCount);


      //offline sensors
```

```
      const offlineCount = stationsData.filter((station: Station) => station.status
=== 'offline').length

      setOfflineSensors(offlineCount)

    } catch (err: any) {

      console.log(err.message || 'Failed to load stations');

    } finally {

      setLoading(false);

    }

  };

  // Initial fetch

  fetchStations();

}, []);


// useEffect for AQS

useEffect(() => {

  const fetchStationsCron = async () => {

    try {

      await getStationsCron();

    } catch (err: any) {

      console.log(err.message || 'Failed to load stations');

    }

  };
```

```
  // Initial fetch
  fetchStationsCron();
}, []);


function getPollutantValue(station: Station, pollutant: PollutantType): number {
  switch (pollutant) {
    case 'aqi':
      return station.aqi;
    case 'pm25':
      return station.pm25;
    case 'pm10':
      return station.pm10;
    default:
      return station.aqi;
  }
}
function getPollutantLevel(value: number, pollutant: PollutantType): string {
  if (pollutant === 'aqi') {
    if (value <= 50) return 'Good';
    if (value <= 100) return 'Moderate';
    if (value <= 150) return 'Unhealthy for Sensitive Groups';
```

```
  if (value <= 200) return 'Unhealthy';
  if (value <= 300) return 'Very Unhealthy';
  return 'Hazardous';
}

if (pollutant === 'pm25') {
  if (value <= 12) return 'Good';
  if (value <= 35.4) return 'Moderate';
  if (value <= 55.4) return 'Unhealthy for Sensitive Groups';
  if (value <= 150.4) return 'Unhealthy';
  if (value <= 250.4) return 'Very Unhealthy';
  return 'Hazardous';
}

if (pollutant === 'pm10') {
  if (value <= 54) return 'Good';
  if (value <= 154) return 'Moderate';
  if (value <= 254) return 'Unhealthy for Sensitive Groups';
  if (value <= 354) return 'Unhealthy';
  if (value <= 424) return 'Very Unhealthy';
  return 'Hazardous';
```

```
  }   return 'Good';
}
function getPollutantColor(value: number, pollutant: PollutantType): string {
  const colorByAQI = (aqi: number): string => {
    if (aqi <= 50) return '#00e400'; // Good – Green
    if (aqi <= 100) return '#ffff00'; // Moderate – Yellow
    if (aqi <= 150) return '#ff7e00'; // Unhealthy for sensitive – Orange
    if (aqi <= 200) return '#ff0000'; // Unhealthy – Red
    if (aqi <= 300) return '#8f3f97'; // Very unhealthy – Purple
    return '#7e0023'; // Hazardous – Maroon
  };

  if (pollutant === 'aqi') {
    return colorByAQI(value);
  } else if (pollutant === 'pm25') {
    // assume value already given as AQI-equivalent scale
    return colorByAQI(value);
  } else if (pollutant === 'pm10') {
    // assume value is pm10 broken into equivalent AQI ranges
    return colorByAQI(value);
  }
```

```
  return '#00e400'; // default – Good
}
```

***Fig 09. Overview Page Metrics Calculation**: Calculates average AQI, active alerts, and offline sensors from live station data. These metrics power summary cards and provide administrators with a quick snapshot of city-wide air quality status.*

4.3.2. Sensors Page - Equipment Management

```
// material-ui
import { Container, Grid } from '@mui/material';
import { getStations, Station } from 'api/maps-api';
import { useEffect, useState } from 'react';
import SensorsTable from 'sections/data-tables/SensorsTable';



function Sensors() {
 const [stations, setStations] = useState<Station[]>([]);

 useEffect(() => {
   const fetchStations = async () => {
```

```
    try {
      const stationsData = await getStations();
      setStations(stationsData);
    } catch (err: any) {
      console.log(err.message || "Failed to load stations");
    }
  };

  // Initial fetch
  fetchStations();
}, []);
 return (
  <Grid justifyContent="center" alignItems="center" sx={{ mb: 12 }}>
    <Grid item xs={12} sm={10} lg={9}>
        <SensorsTable />
    </Grid>
  </Grid>
 );
}
export default Sensors;
```

***Fig 10. Sensors Page Data Fetching**: Retrieves and displays sensor metadata including status, location, and last reading timestamp. Supports equipment management and operational monitoring for county administrators.*

4.3.3. Analytics Page - Historical Data Analysis

```
import { getStations, Station, updateUserReportStations } from 'api/maps-api';

import { openSnackbar } from 'api/snackbar';

import FullReport from 'components/FullReport';

import useAuth from 'hooks/useAuth';

import { useEffect, useState } from 'react';

import AlertsSummary from 'sections/trends/alerts-chart';

import AnalyticsTimeSeries from 'sections/trends/analytics-timeseries-chart';

import AqiDistributionPieChart from 'sections/trends/api-pie-chart';

import ComparisonChart from 'sections/trends/comparison-chart';

import { toast, Toaster } from 'sonner';

import { ThemeMode } from 'types/config';

import { SnackbarProps } from 'types/snackbar';


function Analytics() {

 const { user } = useAuth();


 const [stations, setStations] = useState<Station[]>([]);

 const theme = useTheme();
```

```
useEffect(() => {
  const fetchStations = async () => {
    try {
      const stationsData = await getStations();
      setStations(stationsData);
    } catch (err: any) {
      console.log(err.message || 'Failed to load stations');
    }
  };

  fetchStations();
}, []);

const [selectedStations, setSelectedStations] = useState<string[]>([]);
const [openSubscribeDialog, setOpenSubscribeDialog] = useState(false);
const [openReportDialog, setOpenReportDialog] = useState(false);

const [reportStart, setReportStart] = useState('');
const [reportEnd, setReportEnd] = useState('');
const [reportStation, setReportStation] = useState<Station | null>(null);
```

```
const [showPreview, setShowPreview] = useState(false);


// --- Populate selected stations when user or stations load ---
useEffect(() => {
  if (!user || stations.length === 0) return;

  const local = JSON.parse(localStorage.getItem('reportStations') || '[]');
  const fromUser =
    Array.isArray(user.reportStations) && user.reportStations.length > 0
      ? user.reportStations.map((s: any) => (typeof s === 'string' ? s : s.id))
      : [];

  const chosen = fromUser.length > 0 ? fromUser : local;

  // Validate IDs
  const validIds = stations.map((s) => s.id);
  const filtered = chosen.filter((id: string) => validIds.includes(id));

  setSelectedStations(filtered);
}, [user, stations]);
```

```
useEffect(() => {
  if (stations.length > 0 && !reportStation) {
    setReportStation(stations[0]);
  }
}, [stations]);


const handleSubmitSubscribe = async () => {
  const toastId = toast.loading('Updating stations...');
  try {
    await updateUserReportStations(user?.id, selectedStations);
    toast.success('Stations updated!', { id: toastId });
  } catch (err) {
    console.error(err);
    toast.error('Failed to update stations!', { id: toastId });
  } finally {
    setOpenSubscribeDialog(false);
  }
};


const handleGenerateReport = () => {
  if (!reportStation || !reportStart || !reportEnd) return;
```

```
  setShowPreview(true);

  setOpenReportDialog(false);

};
```

Fig 11. Analytics Page Visualization Components: *Combines time-series charts, alert summaries, pie charts, and comparison graphs to provide historical insights. Enables researchers and officials to explore pollution patterns and station performance over time.*

.

4.3.4. Feedback Page - User Engagement

```
// material-ui

import { Container, Grid } from '@mui/material';

import { getStations, Station } from 'api/maps-api';

import { useEffect, useState } from 'react';

import FeedbackTable from 'sections/data-tables/feedbackTable';

function Feedback() {

 const [stations, setStations] = useState<Station[]>([]);

 useEffect(() => {

  const fetchStations = async () => {

    try {

      const stationsData = await getStations();

      setStations(stationsData);

    } catch (err: any) {

      console.log(err.message || "Failed to load stations");
```

```
    }
  };
  // Initial fetch
  fetchStations();
}, []);
export default Feedback;
```

Fig 12. Feedback Page – Civic Engagement and Administrative Oversight: This dual-access interface fosters user engagement through a public feedback form available to all portal users. Submissions include ratings, comments, and contact details, contributing to iterative platform improvement. Administrative users are granted access to a secure management dashboard where they can view, filter, and respond to submitted feedback—supporting transparency, responsiveness, and continuous system refinement.

4.4. API Service Layer & Data Management

4.4.1. Centralized API Configuration

Explanation: This centralized Axios instance provides consistent API configuration across the application, including automatic JWT token attachment to requests and global error handling for authentication failures.

4.5. Key Features & Benefits

4.5.1. Role-Based Dashboard Access

- Researchers: Access to monitoring data and analytics tools
- Administrators: Full system control including user management
- Secure Routing: Automatic redirects for unauthorized access attempts

4.5.2. Real-time Data Processing

- Auto-refreshing station data every 5 minutes
- Live map updates with color-coded air quality indicators
- Immediate visual feedback for pollution level changes

4.5.3. Advanced Analytics Capabilities

- Customizable date ranges for historical analysis
- Multi-parameter trend visualization (AQI, PM2.5, PM10)
- Statistical distribution analysis
- Export-ready chart configurations

4.5.4. User Experience Features

- Responsive design for desktop and mobile access
- Intuitive navigation with Material-UI components
- Loading states and error handling
- Accessible color schemes and typography
-

5. Conclusion and Next Steps

The Nairobi AQDMS and Public Data Portal successfully deliver a robust, scalable, and user-centric platform for air quality data analytics and visualization. The system transforms raw sensor data into actionable insights, empowering both county officials and the public to understand and respond to air pollution.

The implementation leverages a modern, open-source technology stack to ensure sustainability and alignment with the County's ICT infrastructure. Continuous stakeholder feedback has been integral to shaping a tool that is both technically sound and practically useful.

Next Steps and Future Enhancements:

- Expanded Sensor Network: Integration of data from additional sensors and partner networks.
- Advanced Analytics: Implementation of predictive modeling and source apportionment analysis.
- Enhanced Public Engagement: Development of multi-language support and more granular SMS alert systems.
- API Expansion: Providing more comprehensive public API endpoints for researchers and developers.

6. Appendices

Appendix A: Live Portal Links

- Public-Facing Data Portal: **https://air-quality-portal.vercel.app**
- County Admin Dashboard: **https://dash-blush-ten.vercel.app/**