COMP7503A Multimedia Technologies Programming Assignment - Project report

Student: Liu Junqi;Wang Wenhan
University number: 3036657445;3036655019

Introduction

This project implements a small-scale **smart city data engine** and **multimedia dashboard** using Node-RED, Docker and MongoDB. The goal is to continuously collect open data from Hong Kong public APIs, store and analyse them, and then present actionable insights for city stakeholders through an interactive dashboard. The system focuses on three main domains:

1. urban micro-climate,
2. environmental health risk, and
3. urban mobility and parking availability.

Smart city use cases

Urban micro-climate monitoring

The first use case is fine-grained monitoring of air temperature across Hong Kong. Real-time observations are retrieved from the Hong Kong Observatory (HKO) “rhrread” API, which contains station-level temperature and rainfall measurements. By storing time series for all stations and visualising them on the “Historical Analysis – Micro-Climate Trends” chart, the system allows city managers to study how temperatures vary across districts over the last 24 hours and to identify local hot-spots or cooler coastal areas.

Environmental health risk: Temperature vs Air Quality (AQHI)

The second use case looks at **heat–air-quality coupling**. Air quality data comes from the government AQHI (Air Quality Health Index) API, and the system focuses on the **Central/Western** station as a representative dense urban district. At the same time, HKO temperature data from the **“Hong Kong Observatory”** station is collected.

An analytics engine aligns both time series and computes a **Pearson correlation coefficient** over a rolling 24-hour window. Mathematically, if we have aligned pairs
$(x_i, y_i) = (\text{temperature}_i,\ \text{AQHI}_i), \quad i = 1, \dots, n$

the Pearson index is

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\ \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}},$$

where $\bar{x}$ and $\bar{y}$ are the sample means of temperature and AQHI respectively.

Equivalently, in a form more convenient for implementation:

$$r = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{\left[n\sum x_i^2 - (\sum x_i)^2\right]\left[n\sum y_i^2 - (\sum y_i)^2\right]}}.$$

The **“Real-time Overview”** dashboard shows:

- the latest temperature and AQHI,
- the Pearson index r and an English interpretation (“No significant correlation detected...”, or an alert when strong coupling is found),
- a historical **“Temperature vs AQHI”** chart on the **“Historical Analysis”** tab.

This provides an intuitive way to answer questions such as *“Are hotter periods associated with worse air quality in the past day?”* and could be extended for heat-health early warning.

Urban mobility and parking availability

The second use case looks at **heat–air-quality coupling**. Air quality data comes from the government AQHI (Air Quality Health Index) API, and the system focuses on the **Central/Western** station as a representative dense urban district. At the same time, HKO temperature data from the **“Hong Kong Observatory”** station is collected.

An analytics engine aligns both time series and computes a **Pearson correlation coefficient** over a rolling 24-hour window. Mathematically, if we have aligned pairs

$$(x_i, y_i) = (\text{temperature}_i,\ \text{AQHI}_i), \quad i = 1, \ldots, n$$

the Pearson index is$r = \dfrac{\displaystyle\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\displaystyle\sum_{i=1}^{n}(x_i - \bar{x})^2}\ \sqrt{\displaystyle\sum_{i=1}^{n}(y_i - \bar{y})^2}},$

where $\bar{x}$ and $\bar{y}$ are the sample means of temperature and AQHI respectively.

Equivalently, in a form more convenient for implementation:

$$r = \frac{n\sum x_i y_i - (\sum x_i)(\sum y_i)}{\sqrt{\left[n\sum x_i^2 - (\sum x_i)^2\right]\left[n\sum y_i^2 - (\sum y_i)^2\right]}}.$$

The **“Real-time Overview”** dashboard shows:

- the latest temperature and AQHI,
- the Pearson index r and an English interpretation (“No significant correlation detected...”, or an alert when strong coupling is found),
- a historical **“Temperature vs AQHI”** chart on the **“Historical Analysis”** tab.

This provides an intuitive way to answer questions such as *“Are hotter periods associated with worse air quality in the past day?”* and could be extended for heat-health early warning.

System implementation details

Overall architecture

The system runs completely in Docker containers: one MongoDB container (“Mymongo”) and one Node-RED container. Node-RED hosts three main flow tabs:

- “Data Ingestion” – pulls AQHI and traffic data.
- “HKO Temperature” – pulls HKO real-time weather data and prepares micro-climate charts.
- “Analytics Engine” – performs cross-domain analytics (temperature vs AQHI, traffic trends) and feeds the dashboard widgets.

Node-RED Dashboard is used as the presentation layer, with two tabs: **Real-time Overview** and **Historical Analysis**, using a dark theme for better visual contrast.

Environmental data ingestion (HKO Temperature flow)

Every 5 minutes an inject node triggers the HKO API:

- A MongoDB node first retrieves the **last stored weather record** and reads its updateTime.
- An HTTP Request node calls https://data.weather.gov.hk/weatherAPI/opendata/weather.php?dataType=rhrread&lang=en.
- A function node compares the new updateTime with the last one; only if the new timestamp is later, the JSON is inserted into the Weather Report collection. This de-duplication avoids storing repeated snapshots when the upstream API has not yet updated.

For micro-climate trend analysis, another flow periodically constructs a **24-hour time window**, queries all weather records in that range, and reformats them into the structure expected by the ui_chart node. Each weather station becomes a separate series, producing the multi-line chart “Station Temperature Trend (°C)” on the Historical Analysis tab.

Air quality data ingestion (AQHI flow)

Every 15 minutes the data ingestion flow triggers the AQHI API:

- HTTP Request to https://dashboard.data.gov.hk/api/aqhi-individual?format=json.
- A function node Extract AQHI Data searches the returned array for the record whose station field includes **“Central/Western”**. It then builds a simplified object containing station, numeric aqhi, health_risk, and a timestamp.
- This object is inserted into the aqhi_data collection via a MongoDB node.
- In parallel, the AQHI value is converted into a simple number and pushed directly into a ui_gauge, creating the green-yellow-red AQHI gauge on the Overview dashboard.

Traffic and parking data ingestion (Mobility flow)

Another 15-minute inject node triggers:

- HTTP Request to https://resource.data.one.gov.hk/td/carpark/vacancy_all.json.

- Because this API sometimes returns a BOM-prefixed string, a Clean & Parse JSON function strips BOM characters and calls JSON.parse.
- The Aggregate Vacancy function then loops over all car parks, filters to vehicle type "P" (private cars), and sums the non-negative vacancies across all service categories, ignoring error codes (–1). It also counts how many valid vacancy records contributed to the total and calculates an average.
- The result message has fields metric, total_vacancy, count_carparks, average_vacancy, and timestamp. It is stored in the mobility_data collection and simultaneously sent to the “Parking Availability” gauge.

An error-handling catch node is attached to the HTTP request to log any API failures, increasing robustness of the data engine.

Analytics Engine

An hourly inject node drives the cross-domain analytics:

- A function Set 24h Query computes a [now – 24h, now] window and stores it in msg.query.
- Two MongoDB nodes retrieve matching documents from Weather Report and aqhi_data respectively. A join node merges the two result arrays into msg.payload for further processing.

Two main analyses are implemented:

1. **Temperature vs AQHI time series –** the Build Temp vs AQHI Series function creates two series:
    - For each weather record, it extracts the temperature at “Hong Kong Observatory” (or falls back to the first station) and converts updateTime to a timestamp.
    - For each AQHI record it uses aqhi and its timestamp.

      Both arrays are sorted and packed into the structure required by ui_chart, which renders the “Temperature vs AQHI (°C / Index)” line chart.
2. **Pearson correlation computation –** the Calc Pearson Correlation function:
    - Aligns temperature and AQHI records via a **30-minute sliding window**, pairing each temperature observation with the closest AQHI sample in time.
    - Extracts numerical pairs (x = temperature, y = aqhi) and computes the standard Pearson coefficient r using sums of x, y, xy, x^2, y^2.
    - Classifies the relationship:
        - if |r| is small → “No significant correlation detected”
        - if r > 0.5 → “High Temp correlates with Poor Air Quality”
        - if r < –0.5 → “Negative correlation detected”
    - Also finds the latest temperature and AQHI, and packs everything into msg.payload with fields correlation_coefficient, insight_text, latest_temperature, latest_aqhi and data_points.

This result feeds several UI text nodes and a ui_toast node: when r > 0.6 a red toast alert is shown on the dashboard to highlight strong coupling between heat and air pollution.

A separate flow queries mobility_data over the last 24 hours, formats timestamps and total_vacancy into chart data and produces the “Traffic Vacancy Trend (Spaces)” line chart.

Data storing approach

All data is stored in **MongoDB**, accessed via the node-red-contrib-mongodb3 nodes.

1. Weather Report collection
    - Stores near-raw HKO JSON for each rhrread snapshot, including temperature and rainfall arrays.
    - Each document contains updateTime as an ISO string, which is used both for de-duplication at ingestion time and for time-window queries in analytics.
2. aqhi_data collection
    - Stores one simplified object per AQHI fetch for Central/Western station: { station, aqhi, health_risk, timestamp }.
    - timestamp is a native Date object to support efficient time-range queries ($gt,$lt).
3. mobility_data collection
    - Stores aggregated city-wide metrics: { metric: "city_wide_vacancy", total_vacancy, count_carparks, average_vacancy, timestamp }.
    - Again, timestamp enables rolling 24-hour mobility analysis.

A schemaless NoSQL store is appropriate here because upstream APIs may evolve and the nested structures (e.g. arrays of stations, vehicle types) are more naturally represented as JSON documents. Rolling windows are implemented purely in queries rather than by physically deleting data, so the database also preserves full historical archives for deeper offline analysis.

Dashboard design and smart city insights

Real-time Overview tab

This tab is intended for **operational monitoring**.

- The **Temperature–AQHI Correlation** panel (left in the screenshot) shows:
    - A textual “Pearson Index” summary that embeds both an interpretation and the numeric r, e.g. “No significant correlation detected (r = 0.25)”.
    - “Latest Temperature” and “Latest AQHI” fields, extracted from the most recent records.
    - A “Methodology” box explaining in plain language the algorithm (Pearson correlation), 24-hour rolling window, and 30-minute alignment logic.

      Together, this gives operators a concise but transparent view of how environmental health risk is being quantified.
- The **Real-time Gauge** group (right) holds two gauges:
    - “AQHI (Central/Western)” – a green/yellow/red semi-circular gauge for the current AQHI index (0–10). This immediately communicates short-term respiratory risk level.
    - “Parking Availability” – another gauge showing total available parking spaces (0–10,000) across the city, colour-coded from red (low availability) to green (high). This gives a quick sense of overall congestion and can support decisions such as parking guidance or demand-responsive

pricing.

Historical Analysis tab

This tab provides **analytical and planning insights** over the last 24 hours.

- **Cross-Domain Analysis: Temperature vs AQHI**

  A dual-line chart overlays temperature (°C) and AQHI index. City analysts can visually inspect whether peaks in air pollution tend to coincide with high or low temperature, complementing the numeric Pearson index on the overview page.

- **Micro-Climate Trends: Station Temperature Trend (°C)**

  Multi-series line chart where each line represents a weather station. This reveals spatial patterns such as inland vs coastal differences, urban heat island effects, and local micro-climate anomalies, supporting planning of green infrastructure or heat-mitigation interventions.

- **Traffic Flow Trends: Traffic Vacancy Trend (Spaces)**

  A line chart of total parking vacancies across car parks over time. Morning and evening peaks, night-time low demand, and special-event surges are all visible. This insight can inform transport planners on where and when to adjust supply, encourage modal shifts, or deploy real-time traveller information.

Together, the two dashboard tabs cover both **real-time situational awareness** and **short-term historical analysis** across environmental and mobility dimensions of the smart city.

Conclusion

The implemented system demonstrates how open government data, a lightweight Node-RED pipeline and a MongoDB backend can be combined into a functional smart city analytics platform. It delivers three concrete use cases—micro-climate monitoring, environmental health risk assessment, and mobility/parking analytics—linked to intuitive dashboard visualisations. The architecture is modular and extensible: additional data streams (noise, energy, public transport, etc.) could be ingested using the same pattern, and new cross-domain analytics (e.g. traffic vs air quality) could be added to enrich the city-scale insights provided to decision-makers and citizens.