Project Design Document: SkyAware

Air Quality Companion Application (NASA Space Apps Challenge 2025)

Metadata	Detail
Project Title	SkyAware: Cloud Computing with Earth Observation Data for Predicting Cleaner, Safer Skies
Document Version	1.0 (Pre-Hackathon Design Freeze)
Date	October 2, 2025
Target Platform	Google Cloud Platform (GCP)
Status	Approved for Development

I. Executive Summary

The objective of **SkyAware** is to deliver a scalable, web-based application that leverages NASA's **Tropospheric Emissions: Monitoring of Pollution (TEMPO)** mission data alongside ground-based Air Quality Index (AQI) and meteorological data to provide granular, real-time air quality monitoring and 72-hour forecasting across North America. The solution is designed with a mobile-first user interface to provide actionable public health advice and educational content.

The core technical differentiator is the utilization of **Google Cloud Platform (GCP)** services to handle the processing and serving of large, complex TEMPO data files, aligning directly with the "Cloud Computing" challenge theme.

II. Technical Architecture & Stack

A. Core Technology Stack

Component	Technology	Rationale
Cloud Platform	Google Cloud Platform (GCP)	Provides scalable, integrated services (Cloud Run, Cloud Functions, Cloud Storage) essential for processing large Earth Observation datasets.
Frontend	**Next.js (React)**	Server-side rendering (SSR) for performance, optimized routing, and rapid component development.
Backend API	**Node.js (Express/Hapi)**	High-throughput, asynchronous I/O ideal for serving data endpoints to the frontend application.
Data Processing/ML	**Python (Pandas, xarray, scikit-learn/ XGBoost)**	Industry standard for NetCDF4/HDF data handling (TEMPO) and Machine Learning model training/inference.
Mapping & Viz	**Mapbox GL JS / Deck.gl**	Highly performant libraries for rendering large custom geospatial data layers (TEMPO raster/GeoJSON) in the browser.

B. GCP Component Architecture

The application implements a decoupled, event-driven architecture utilizing GCP services:

Component	GCP Service	Purpose
API Layer	**Cloud Run**	Hosts the Node.js API, serving current AQI, forecast, and location data to the frontend. Scales down to zero when idle.
TEMPO Processor	**Cloud Functions (Python)**	Executes the TEMPO data ingestion, pollutant-to-AQI conversion, and format transformation (NetCDF4 to GeoJSON/Raster). Triggered hourly.
Data Cache	**Cloud Storage (Bucket)**	Stores the pre-processed, browser-friendly map data (e.g., GeoJSON, compressed GeoTIFFs) generated by the Python Cloud Function.
Scheduler	**Cloud Scheduler**	Triggers the **Cloud Function** hourly to ensure the TEMPO data is always current.

III. Project Scope & Phasing (Roadmap)

The project execution is divided into three distinct phases for parallel team development.

Phase	Duration (Estimate)	Goal	Key Deliverables
Phase 1: Foundation (Data Ingestion)	≈6\approx Hours	Establish all data pipelines (Ground, Weather, TEMPO access) and the core Next.js/Node.js structure.	Working Node.js API with a populated /api/current_aqi endpoint. Next.js application shell with functional Mapbox component.
Phase 2: Integration & Core Features	≈12\approx Hours	Integrate the ML forecasting model, display TEMPO satellite data on the map, and merge frontend/backend components.	Functional 72-hour AQI forecast displayed. TEMPO GeoJSON/Raster overlay displayed on map. Draft **Educational Mode** complete.
Phase 3: Polish & Deployment	≈6\approx Hours	Implement notifications, optimize performance, finalize documentation, and prepare the pitch presentation.	Fully deployed application on GCP. Notification simulation logic implemented. Final presentation and README documentation.

IV. Team Roles & Assignment Matrix

The team is structured to maximize parallel development streams (Frontend, Backend/API, AI/ML, and Support/Content).

Team Member	Role	Primary Focus Area	Key Responsibilities
Ebrima S. Jallow	**Team Lead / Support**	Project Management, Pitch Narrative, Content/UX	Define scope, manage timeline, own final presentation, co-write **Educational Mode** content, ensure data citation compliance.
Saul Zayn	**Senior Frontend Dev**	Next.js Architecture, Map Visualization	Initialize Next.js, implement Mapbox/Deck.gl, display TEMPO map layer, optimize mobile responsiveness.
Sawaneh	**Senior Backend Dev**	GCP Architecture, Node.js API	Set up GCP services (Cloud Run, Cloud Functions), define and own all API endpoints (JSON contract), implement alert logic.
Hassan	**Full Stack Dev**	Data Ingestion, Frontend-Backend Bridge	Implement API calls for **EPA AirNow** & **OpenWeatherMap**. Fetch historical data for ML. Connect frontend components to backend APIs.
Omar	**AI/ML Engineer**	TEMPO Processing, Forecasting Model	Access and convert TEMPO NetCDF4 data to GeoJSON/Raster using Python. Train and serve the XGBoost/Statistical AQI forecasting model on GCP.
Hawa Cham	**Support / Content**	QA Testing, Content Creation, Documentation	Conduct comprehensive QA testing, co-write/finalize **Educational Mode** content, manage API keys/credentials, finalize project documentation.

V. Prerequisites & Critical Dependencies (Go/No-Go Checklist)

The following items are **Critical Path Items (CPIs)** that must be confirmed before the start of the execution phase.

Item	Status	Responsible	Action Required
GCP Project & Billing	☐	Ebrima	Project created; billing enabled (to use Cloud Run/Functions).
NASA Earthdata Login (EDL)	☐	Omar	Account created to facilitate programmatic TEMPO data access.
API Keys (EPA AirNow & OpenWeatherMap)	☐	Hassan	Keys obtained and stored securely for real-time data ingestion.
Mapbox API Key	☐	Saul	API key created for map rendering.
TEMPO Product Lock	☐	Omar	Specific TEMPO NRT Level 2/3 product for NO2 or O3 identified.
AQI Conversion Formulas	☐	Ebrima/Hawa	US EPA AQI breakpoint table and conversion equations documented for implementation.
Code Repository Setup	☐	Ebrima	Next.js/Node.js shell committed to GitHub/GitLab repository.

VI. Technical Contracts & Interface Definition

A. API Contract (Schema Agreement)

Endpoint	Method	Purpose	Critical Data Fields Agreed Upon
/api/current_aqi	GET	Real-time AQI at user's location.	current_aqi_epa, current_aqi_tempo_derived, primary_pollutant, last_updated_time.
/api/forecast	GET	72-hour AQI prediction.	forecast_days[] array containing date, aqi_max, and health_advice_text.
/api/tempo_grid	GET	Map overlay data.	data_type (GeoJSON/Raster URL), data_url (GCP Cloud Storage link), min_val, max_val (for color scaling).

B. TEMPO Processor Contract

The Python **Cloud Function** must perform the following transformation:

- **Input:** TEMPO NetCDF4/HDF file (via NASA Harmony API).
- **Process:** Extract pollutant NO2 or O3 column amount →to→Regrid to ~ 1km → to→ Convert mol/cm2 to AQI using EPA formula.
- **Output: Simplified GeoJSON file** (for vector display) **OR** a **compressed GeoTIFF/Raster Tile** (for raster display).
- **Storage:** Output file must be written to the designated **GCP Cloud Storage** bucket, with the link served by the /api/tempo_grid endpoint.

C. ML Model Input Contract

The forecasting model (managed by Omar) is a supervised learning model (XGBoost/LSTM) that will consume the following features sourced by Hassan:

- **Target Variable: next_24hr_aqi_max** (from historical EPA data).
- **Input Features:** previous_aqi_24hr, current_temp, wind_speed, current_tempo_pollutant_concentration.

VII. Known Risks & Mitigation Strategies

Risk (Plan Deficiency)	Description	Impact Severity	Mitigation Strategy
R-1: TEMPO Data Processing Lag	Python processing of large NetCDF4 files on a serverless function may be slow, causing outdated map data.	High	**Pre-Process/Cache:** Implement the Python function to process data and save the **simplified GeoJSON/Raster** to **GCP Cloud Storage**. The API will serve the *cached link*, not the processing result.
R-2: AQI Conversion Failure	Incorrect implementation of the EPA pollutant-to-AQI formula due to unit or breakpoint errors.	High	**Support-Driven Validation:** Ebrima/Hawa will provide the exact EPA documentation. Omar will build a small unit test to validate the conversion function against known values.
R-3: Cloud Cost Overruns	Excessive API calls or resource utilization on GCP (especially Cloud Functions/Run) during development.	Medium	**Budget Controls:** Implement GCP budget alerts. **Caching:** Hassan/Sawaneh must implement aggressive caching for all external APIs (EPA/Weather) to minimize redundant calls.
R-4: Map Layer Performance	The frontend may struggle to render complex GeoJSON data for the map overlay smoothly on mobile devices.	Medium	**Layer Optimization:** Saul will prioritize using **Mapbox/Deck.gl Vector/Raster Tile Sources** over large GeoJSON files to leverage browser GPU rendering. Omar's GeoJSON output must be highly simplified.