Software Project Management

Semester Project

Course Instructor

Sir Muhammad Bilal

Group Members

Rayan Ahmed (22i-2489)

Abdur Rehman (22i-2518)

Bilal Raza (22i-2559)

Submission Date

30th November, 2025

Fall 2025

Department of Software Engineering

FAST – National University of Computer & Emerging Sciences

Project Overview and Objectives

Problem

There is a significant gap between raw air quality data and actionable health guidance. Many sources provide pollution numbers without clear interpretation, lacking essential context like the Air Quality Index (AQI) or personalized recommendations. This leaves the general public and sensitive groups struggling to make informed decisions about outdoor activities and necessary health precautions based on current or predicted air quality conditions.

Solution

The **Pollution Monitor Agent** is an AI-powered web service for real-time air quality monitoring and analysis, built using **CrewAI** and **FastAPI**. It utilizes the **OpenWeatherMap Air Pollution API** to fetch data and analyze key pollutants (PM2.5, PM10, NO2, O3, SO2, CO). The agent calculates the **US EPA Air Quality Index (AQI)** and generates tailored health recommendations. It provides a robust REST API with endpoints for health checks and task execution, supporting both direct JSON and chat-style message requests. The system features tools for historical pattern analysis and pollution trend prediction, is designed for integration into a supervisor/registry architecture, and includes logging, error handling, and integration tests. This production-ready prototype effectively combines real-time data, AI analysis, and actionable environmental health guidance.

Project Management artifacts

WBS

1. Requirements & Analysis

  1.1 Stakeholder Needs Identification

    1.1.1 Collect academic assignment requirements and supervisor/registry expectations

    1.1.2 Analyze OpenWeatherMap API availability and data structure

    1.1.3 Define agent-based system boundaries and constraints

  1.2 Functional & Non-Functional Requirements

    1.2.1 Specify agent tasks: pollution analysis, AQI calculation, recommendations

    1.2.2 Define API endpoint requirements (health check, task execution)

    1.2.3 Establish JSON contract for supervisor/registry communication

1.2.4 Define performance criteria and response time expectations

1.3 Technology Stack Selection

1.3.1 Select CrewAI framework for agent orchestration

1.3.2 Choose OpenAI LLM provider (GPT-4o-mini/GPT-5-nano)

1.3.3 Evaluate FastAPI for REST API implementation

1.3.4 Document API key requirements and environment configuration

2. System & Architecture Design

2.1 Agent Architecture Design

2.1.1 Design CrewAI agent structure with role, goal, and backstory

2.1.2 Define agent tools: analyze_pollution_data, calculate_aqi, generate_recommendations

2.1.3 Design task creation and crew execution workflow

2.1.4 Specify agent tool integration with OpenWeatherMap API

2.2 API Architecture Design

2.2.1 Design FastAPI application structure and endpoints

2.2.2 Define request/response models using Pydantic

2.2.3 Design message parsing for chat-based and direct request formats

2.2.4 Specify error handling and middleware architecture

2.3 Data Model Design

2.3.1 Define Pydantic models: AgentRequest, AgentResponse, HealthCheckResponse

2.3.2 Design TaskStatus enum and error response structures

2.3.3 Establish JSON contract schema for supervisor/registry communication

2.4 External API Integration Design

2.4.1 Design OpenWeatherMap geocoding and air pollution API integration

2.4.2 Define pollutant mapping and data transformation logic

2.4.3 Design EPA AQI calculation algorithm and breakpoint structure

3. Development Phase

3.1 CrewAI Agent Module Development

3.1.1 Implement analyze_pollution_data tool with OpenWeatherMap API calls

3.1.2 Develop calculate_aqi tool with EPA breakpoint calculations

3.1.3 Implement generate_recommendations tool with health guidance logic

3.1.4 Develop analyze_historical_patterns tool for trend analysis

3.1.5 Implement predict_pollution_trends tool using linear regression

3.1.6 Create agent factory function with tool integration

3.1.7 Develop task creation and crew execution functions

3.2 Data Models Module Development

3.2.1 Implement Pydantic models for request/response structures

3.2.2 Develop TaskStatus enum and validation logic

3.2.3 Create Message and ChatRequest models for chat format support

3.2.4 Implement field validators and JSON schema generation

3.3 FastAPI Backend Development

3.3.1 Build FastAPI application with lifespan management

3.3.2 Implement root endpoint (GET /) with agent information

3.3.3 Develop health check endpoint (GET /health) with uptime tracking

3.3.4 Implement task execution endpoint (POST /api/v1/execute)

3.3.5 Develop message parsing logic for chat-based requests

3.3.6 Implement request logging middleware

3.3.7 Create global exception handler and error responses

3.3.8 Develop analysis formatting functions for concise output

- 3.4 OpenWeatherMap API Integration
  - 3.4.1 Implement geocoding API connector for location resolution
  - 3.4.2 Develop air pollution data retrieval with error handling
  - 3.4.3 Implement pollutant data mapping and status determination
  - 3.4.4 Create historical data fetching for pattern analysis
  - 3.4.5 Develop data validation and error recovery mechanisms
- 3.5 Logging & Monitoring Setup
  - 3.5.1 Configure Python logging with file and console handlers
  - 3.5.2 Implement request/response logging middleware
  - 3.5.3 Set up log rotation and directory management
  - 3.5.4 Create health check monitoring with uptime tracking

4. Integration & Testing Phase
- 4.1 Module Integration
  - 4.1.1 Integrate agent module with FastAPI endpoints
  - 4.1.2 Connect OpenWeatherMap API calls with agent tools
  - 4.1.3 Integrate Pydantic models with API request/response handling
  - 4.1.4 Validate end-to-end workflow from request to response
- 4.2 Unit Testing
  - 4.2.1 Develop unit tests for agent tools (test_unit.py)
  - 4.2.2 Test AQI calculation accuracy with known values
  - 4.2.3 Test pollutant data parsing and transformation
  - 4.2.4 Validate Pydantic model serialization/deserialization
- 4.3 Integration Testing
  - 4.3.1 Develop integration tests for API endpoints (test_integration.py)

4.3.2 Test health check endpoint functionality

4.3.3 Test task execution with various pollutant combinations

4.3.4 Validate JSON contract compliance with supervisor/registry format

4.3.5 Test error handling and edge cases

4.3.6 Conduct concurrent request testing

4.4 Quick Validation Testing

4.4.1 Develop standalone quick test script (quick_test.py)

4.4.2 Create validation tests for basic functionality

4.4.3 Test multiple pollutant analysis scenarios

4.4.4 Validate input validation and error responses

5. Deployment & Configuration Phase

5.1 Containerization Setup

5.1.1 Create Dockerfile for containerized deployment

5.1.2 Configure environment variables and port settings

5.1.3 Set up application entry points (app.py for HuggingFace)

5.1.4 Test Docker build and container execution

5.2 Environment Configuration

5.2.1 Create .env.example template for configuration

5.2.2 Document required API keys (OpenAI, OpenWeatherMap)

5.2.3 Set up environment variable management with python-dotenv

5.2.4 Configure logging levels and output destinations

5.3 Deployment Platforms

5.3.1 Configure HuggingFace Spaces deployment (Docker SDK)

5.3.2 Set up Heroku deployment configuration (Procfile)

5.3.3 Create runtime.txt for Python version specification

5.3.4 Test deployment on target platforms

5.4 Documentation & Handover

5.4.1 Create comprehensive README with setup instructions

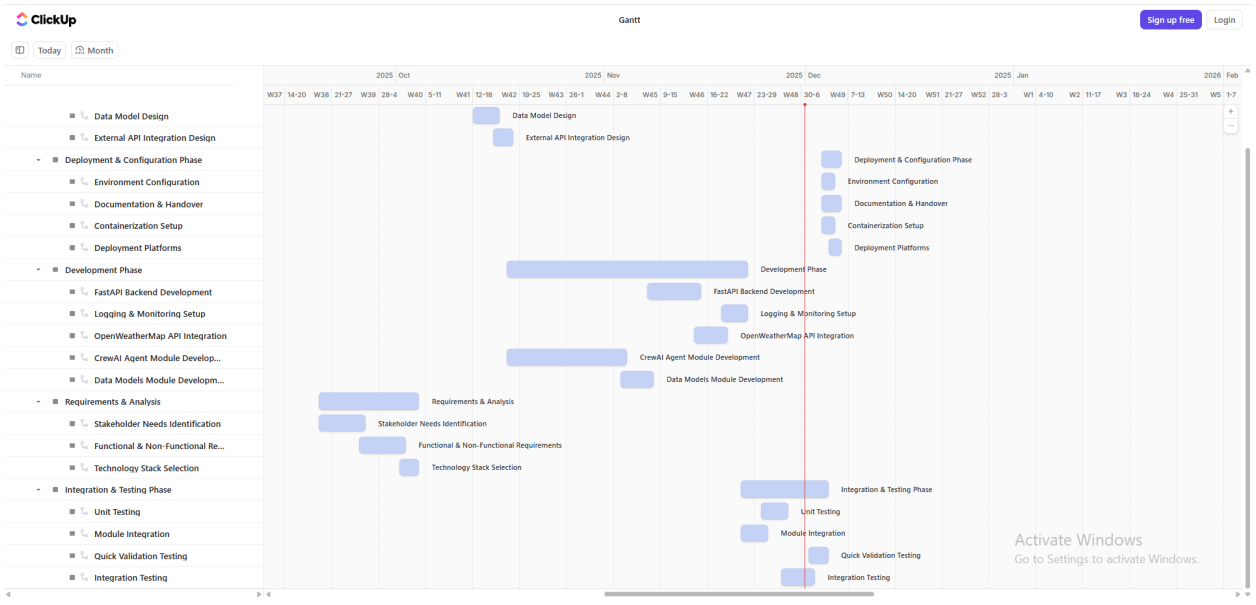
5.4.2 Develop API documentation (API.md) with endpoint details

5.4.3 Document project structure and architecture (PROJECT.md)

5.4.4 Create quick start guide and usage examples

5.4.5 Document testing procedures and validation steps

Gantt Chart



Live link: https://sharing.clickup.com/90182135386/g/h/2kzmcbju-618/596a4351f27ef91

Cost Estimate

This cost estimate is prepared in accordance with Project Management standards, using the project's Work Breakdown Structure (WBS) as the basis for resource allocation and expenditure tracking.

1. Key Assumptions

No specific labor rates or contract prices are used, the following assumptions are used to construct the cost estimate:

Category	Assumption	Notes
Team Size	3 Group Members/Developers	Rayan Ahmed, Abdur Rehman, Bilal Raza.
Average Labor Rate	**$200 per person-day**	Rate for internal resource estimation.
Project Duration	**19 Sep, 2025 – 5 Dec, 2025**	Updated project timeline.
Effort Unit	Person-Days	Effort for Level 2 WBS activities (Total remains 20 working days).
OpenWeatherMap API	Free/Tier 1 plan	Assumed free tier for prototyping/academic use.
OpenAI LLM	Pay-as-you-go / Low-volume use	Estimated cost for development and testing usage.
Deployment Platforms	Free/Trial tier	Assumed free or academic tier for HuggingFace/Heroku.
Contingency Reserve	15%	Standard reserve for unforeseen risks or scope creep.

2. Cost Breakdown by WBS Phase

The cost is broken down based on the major phases defined in the Work Breakdown Structure (WBS). Effort is estimated in person-days, and costs are in USD.

WBS ID	Phase/Task Group	Estimated Effort (Person-Days)	Labor Cost (@ $200/Day)	Non-Labor Cost (Estimated)	Subtotal Cost (USD)
1.0	**Requirements & Analysis**	**3**	**$600**	**$0**	**$600**

1.1	Stakeholder Needs Identification	1	$200	$0	$200
1.2	Functional & Non-Functional Requirements	1.5	$300	$0	$300
1.3	Technology Stack Selection	0.5	$100	$0	$100
2.0	**System & Architecture Design**	**5**	**$1,000**	**$0**	**$1,000**
2.1	Agent Architecture Design	1.5	$300	$0	$300
2.2	API Architecture Design	1.5	$300	$0	$300
2.3	Data Model Design	1	$200	$0	$200
2.4	External API Integration Design	1	$200	$0	$200
3.0	**Development Phase**	**8**	**$1,600**	**$50**	**$1,650**
3.1	CrewAI Agent Module Development	3	$600	$0	$600
3.2	Data Models Module Development	1	$200	$0	$200
3.3	FastAPI Backend Development	2	$400	$0	$400
3.4	OpenWeatherMap API Integration	1	$200	$0	$200
3.5	Logging & Monitoring Setup	1	$200	$50 (O…	$250

4.0	**Integration & Testing Phase**	**3**	**$600**	**$0**	**$600**
4.1	Module Integration	0.5	$100	$0	$100
4.2	Unit Testing	1	$200	$0	$200
4.3	Integration Testing	1	$200	$0	$200
4.4	Quick Validation Testing	0.5	$100	$0	$100
5.0	**Deployment & Configuration Phase**	**1**	**$200**	**$0**	**$200**
5.1	Containerization Setup	0.25	$50	$0	$50
5.2	Environment Configuration	0.25	$50	$0	$50
5.3	Deployment Platforms	0.25	$50	$0	$50
5.4	Documentation & Handover	0.25	$50	$0	$50
TOTAL		**20**	**$4,000**	**$50**	**$4,050**

3. Project Budget Summary

Category	Cost (USD)
Total Estimated Labor Cost	**$4,000**
Total Estimated Non-Labor/Material Cost	$50
Total Estimated Base Cost (EAC)	**$4,050**
Contingency Reserve (15% of Base Cost)	**$607.50**
TOTAL PROJECT BUDGET	**$4,657.50**

EAC (Estimate at Completion): The base cost of **$4,050** represents the Estimate at Completion (EAC) based on the WBS and stated assumptions. The total project budget of

$4,657.50 includes a mandatory contingency reserve to manage identified and unidentified risks.

Risk Management Plan

1. Methodology

The project will employ a **Qualitative Risk Analysis** approach. Risks will be assessed based on their likelihood of occurrence (Probability) and the potential effect on project objectives (Impact).

Probability	Impact
Very High	Very High
High	High
Medium	Medium
Low	Low
Very Low	Very Low

2. Roles and Responsibilities

Role	Responsibility
Project Team	Identification of new risks, execution of assigned risk response actions, and reporting on risk status.
Team Lead	Owns and updates the Risk Register, manages the Contingency Reserve, and escalates critical risks.

3. Risk Categories

Risks for this project will be categorized as follows:

- **Technical:** Risks related to the technology stack (CrewAI, FastAPI, LLM), integration (OpenWeatherMap API), system design (AQI calculation, prediction models).
- **External:** Risks outside the team's direct control (e.g., API stability, internet connectivity, LLM costs).
- **Organizational:** Risks related to team members, effort estimation, communication, or resource availability.
- **Project Management:** Risks related to schedule, budget, or scope creep.

4. Risk Tolerance and Thresholds

- **High (H) or Very High (VH) Impact:** No High or Very High impact risks are acceptable without a clearly defined and assigned mitigation strategy. These risks will be immediately escalated.
- **Contingency Reserve:** The project budget includes a **15% Contingency Reserve ($607.50)** to manage identified and unforeseen risks that may materialize.

5. Risk Register

ID	Risk	Category	Probability	Impact	Priority	Response Strategy	Contingency Action
R1	**OpenWeatherMap API Downtime/Rate Limits:** The primary data source becomes unavailable or restricts usage during a critical phase.	Te…	Med…	High	**High**	**Mitigate:** Implement a robust retry mechanism and caching layer. **Plan B:** Identify and pre-validate a secondary, free-tier air quality data API (e.g., AQICN).	Allocate additional effort (person-days) to integrate a new API if the primary source fails.
R2	**CrewAI/LLM Configuration Complexity:** Difficulty integrating agents and tools, or unexpected behavior from the chosen OpenAI LLM (GPT-4o-mini).	Te…	High	High	**High**	**Mitigate:** Conduct a focused 1-day spike (WBS 1.3/2.1) to create a minimal viable agent and tool to confirm platform compatibility and stability before full development.	Revert to a simpler, non-agent-based architecture if the LLM/CrewAI complexity significantly threatens the deadline.

R3	**Effort Underestimation:** Key tasks, particularly in the **Development Phase** (WBS 3.0), require more than the estimated 8 person-days, threatening the 4-week deadline.	Pr…	Med…	Me…	**Me…**	**Mitigate:** Implement a daily stand-up/sprint review for the first two weeks to track burndown and re-estimate complex tasks proactively.	Reprioritize tasks to focus solely on core agent functionality (Analysis, AQI) and postpone non-critical tools (Historical/Predictive) to a later phase.
R4	**Team Member Availability:** A developer becomes unavailable for an extended period, stressing the small 3-person team.	Or…	Low	High	**Me…**	**Mitigate:** Ensure knowledge sharing and cross-training for all critical sections (FastAPI Backend, CrewAI Agents). Maintain a single, shared source code repository with clear documentation.	Re-assign the unavailable member's tasks to the remaining two members and use the Contingency Reserve to potentially purchase additional low-cost LLM usage for task automation.
R5	**JSON Contract Non-Compliance:** The API Contract (WBS 5.4) does not perfectly match the expectations of a hypothetical Supervisor/Registry system.	Te…	Low	Me…	**Low**	**Mitigate:** Review the generated JSON schema against common industry standards for microservice communication. Conduct a	Quickly update and re-generate the contract schema and associated Pydantic models.

						final review with a third party (e.g., a peer or mentor).	

Quality Plan

This Quality Management Plan defines the project's quality objectives, standards, and the processes for Quality Assurance (QA) and Quality Control (QC) to ensure the **Pollution Monitor Agent** project deliverables meet the required standards and project objectives.

1. Quality Objectives and Policy

The primary quality objective is to deliver a production-ready, AI-powered web service that provides **accurate, timely, and actionable** air quality monitoring and analysis.

- **Accuracy:** The agent must correctly calculate the **US EPA Air Quality Index (AQI)** based on the fetched pollutant data.
- **Performance:** The REST API must meet the defined response time expectations (WBS 1.2.4) and maintain high uptime.
- **Compliance:** All project deliverables (code, documentation, and the API contract) must comply with the defined requirements and architecture designs (WBS 1.0 and 2.0).

2. Quality Standards and Metrics

The project adheres to the following standards and utilizes the corresponding metrics to measure quality:

Area	Standard	Quality Metrics
System Functionality	Accurate **US EPA AQI** calculation and tailored health recommendations.	**Unit Test Pass Rate:** Percentage of unit tests passed (Target: 100%).
Code Structure	Adherence to Python standards and Pydantic model validation.	**Code Review Completion:** All major modules reviewed (Target: 100%).
API Contract	JSON Contract compliance for supervisor/registry communication.	**Contract Compliance Rate:** Percentage of contract tests passed (Target: 100%).

System Reliability	Robust error handling, logging, and monitoring in the FastAPI backend (WBS 3.5).	**Health Check Uptime:** Time the `/health` endpoint is available (Target: > 99.9%).

3. Roles and Responsibilities

Quality responsibilities are integrated into the existing project team structure:

Role	Responsibility (Quality Focus)
Project Team (Developers)	Adherence to coding standards, execution of Unit Tests (WBS 4.2), and bug fixing.
Team Lead	Reviewing design documents, conducting code reviews, verifying Integration Test results (WBS 4.3), and managing the overall Quality Management Plan and Risk Register.

4. Quality Assurance (QA)

QA activities focus on preventing defects by ensuring the correct processes are followed throughout the project lifecycle.

QA Activity	WBS Reference	Description
Design Review	WBS 2.0	Review of Agent Architecture, API Architecture, and Data Model Design before development begins to ensure feasibility and correctness.
Code Review	WBS 3.0	Peer review of all merged code to check for adherence to standards, proper error handling, and security considerations.
Requirements Traceability	WBS 1.0	Verification that all defined functional and non-functional requirements are linked to an implemented and tested feature.

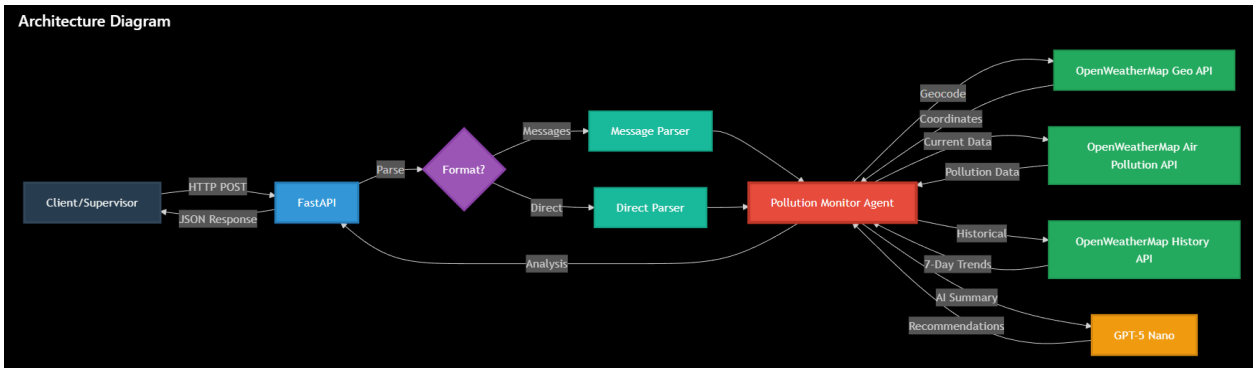
5. Quality Control (QC)

QC activities focus on identifying defects by testing the work products against requirements and standards. The **Integration & Testing Phase (WBS 4.0)** serves as the core QC process.

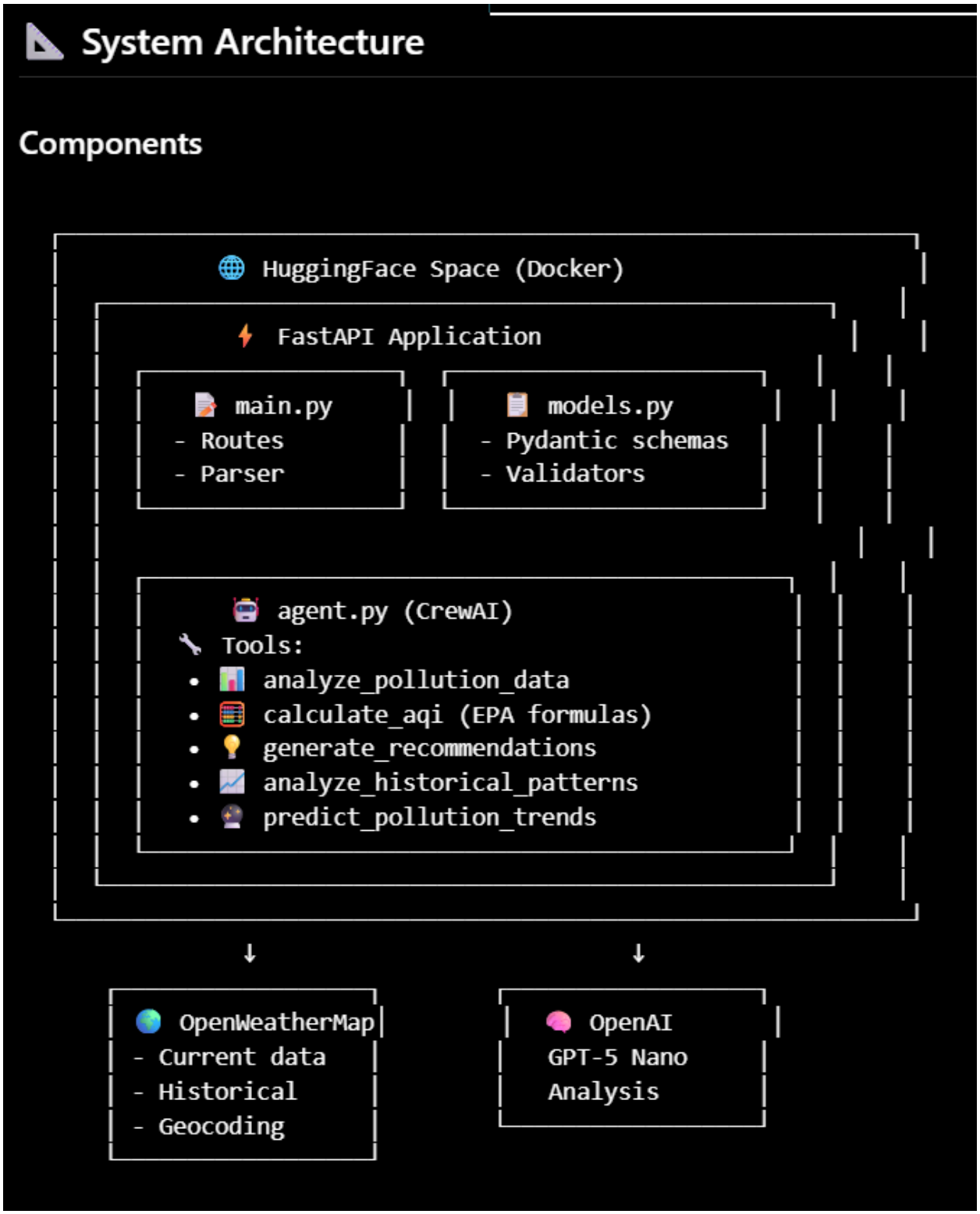
QC Activity	WBS Reference	Description
Unit Testing	WBS 4.2	Testing individual components (agent tools, Pydantic models) to ensure they function correctly in isolation. Includes testing AQI calculation accuracy.
Integration Testing	WBS 4.3	Testing the end-to-end workflow: agent module integration with FastAPI endpoints and OpenWeatherMap API calls. Includes concurrent request testing.
Quick Validation Testing	WBS 4.4	Standalone script to perform validation tests for basic functionality and multiple pollutant analysis scenarios before deployment.
Acceptance Testing	N/A	Final validation against the Project Overview and Objectives to ensure the system delivers on the core problem statement (actionable health guidance).

System Design and Architecture

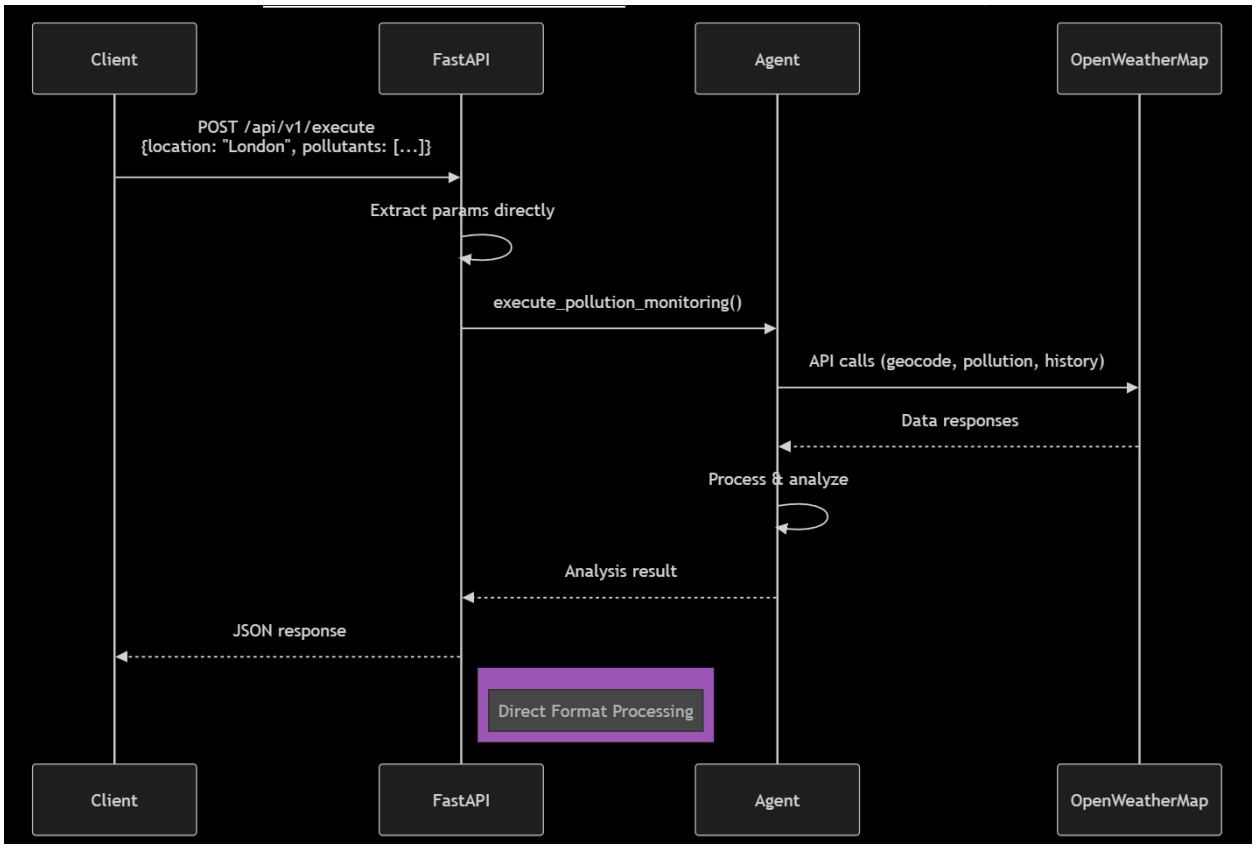
System Architecture Diagram



Component Architecture



Data Flow



Memory Strategy

Short-Term Memory (Within Request Scope)

The agent maintains context within a single request execution cycle. This memory is transient and cleared after each response.

Component	**Storage**	**Scope**	**Lifetime**
Request Context	AgentRequest Pydantic model	Single API call	~35-50 seconds
Location Cache	In-memory dict	Geocoding result	Tool execution
Pollutant Readings	Tool return values	Analysis chain	Tool execution
AQI Calculation	Computed variables	Recommendation generation	Tool execution

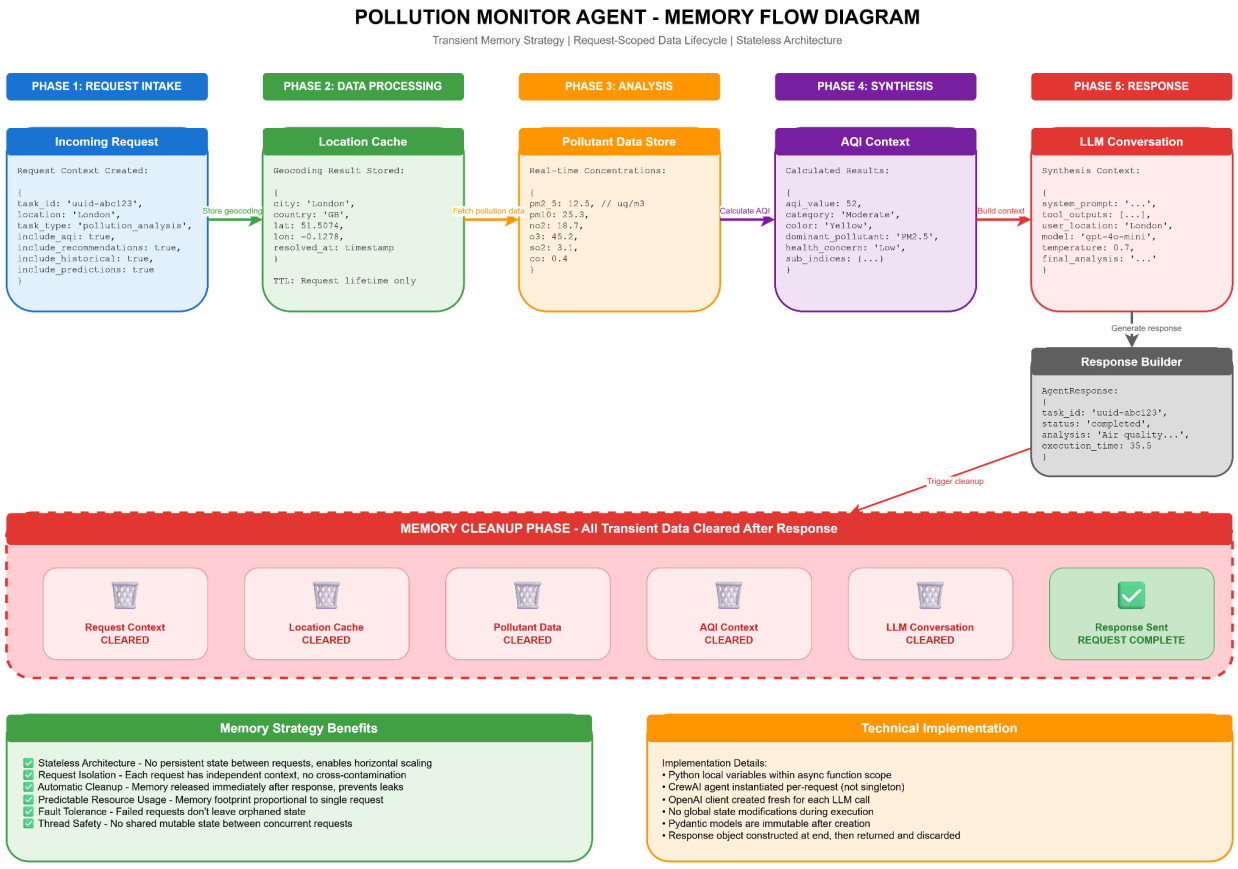
LLM Conversation	OpenAI messages array	Narrative synthesis	Request completion

Long-Term Memory (External Persistent Storage)

The agent leverages external APIs for historical data rather than maintaining its own database:

Data Type	Storage Location	Retention	Access Method
Historical Pollution	OpenWeatherMap servers	7+ days	analyze_historical_patterns()
Request Logs	agent.log file	Session-based	Python logging module
Audit Trail	Application logs	Container lifecycle	Middleware

Memory Flow



API Contract

Endpoints

Endpoint	Method	Description
/	GET	Agent information and capabilities
/health	GET	Health check with uptime status
/api/v1/execute	POST	Execute pollution analysis task

Request Formats

Request Formats

Format 1: Message-Based (Chat)

```
{
  "messages": [
    {
      "role": "user",
      "content": "Air quality report for Islamabad, Include aqi, Pollutants are PM2.5"
    }
  ]
}
```

Format 2: Direct (Structured)

```
{
  "location": "London",
  "pollutants": ["PM2.5", "NO2", "O3"],
  "include_recommendations": true,
  "include_aqi": true
}
```

Response Schema

Response Schema


```
{
  "task_id": "uuid",
  "status": "completed|failed",
  "location": "string",
  "analysis": "string (plain text report)",
  "timestamp": "ISO 8601",
  "execution_time": "float (seconds)"
}
```


Example Response


```
{
  "task_id": "f4d3db03-ae22-4f9f-b5f2-abb37a405276",
  "status": "completed",
  "location": "Islamabad",
  "execution_time": 37.61,
  "timestamp": "2025-11-30T18:05:43.134164",
  "analysis": "Air Quality Report for Islamabad\n==========\n\nCURRENT CONDITIONS:\n----------\n- PM2.5: 137.57 µg/m³ (Very Unhealthy)\n- AQI: ~193 (Unhealthy)\n\nHEALTH ASSESSMENT:\n----------\n- Everyone may experience adverse health effects\n- Sensitive groups at higher risk\n\nRECOMMENDATIONS:\n----------\n1. Avoid prolonged outdoor activities\n2. Use N95 masks if going outside\n3. Keep windows closed\n4. Run air purifiers\n5. Monitor air quality updates"
}
```


Health Check Response

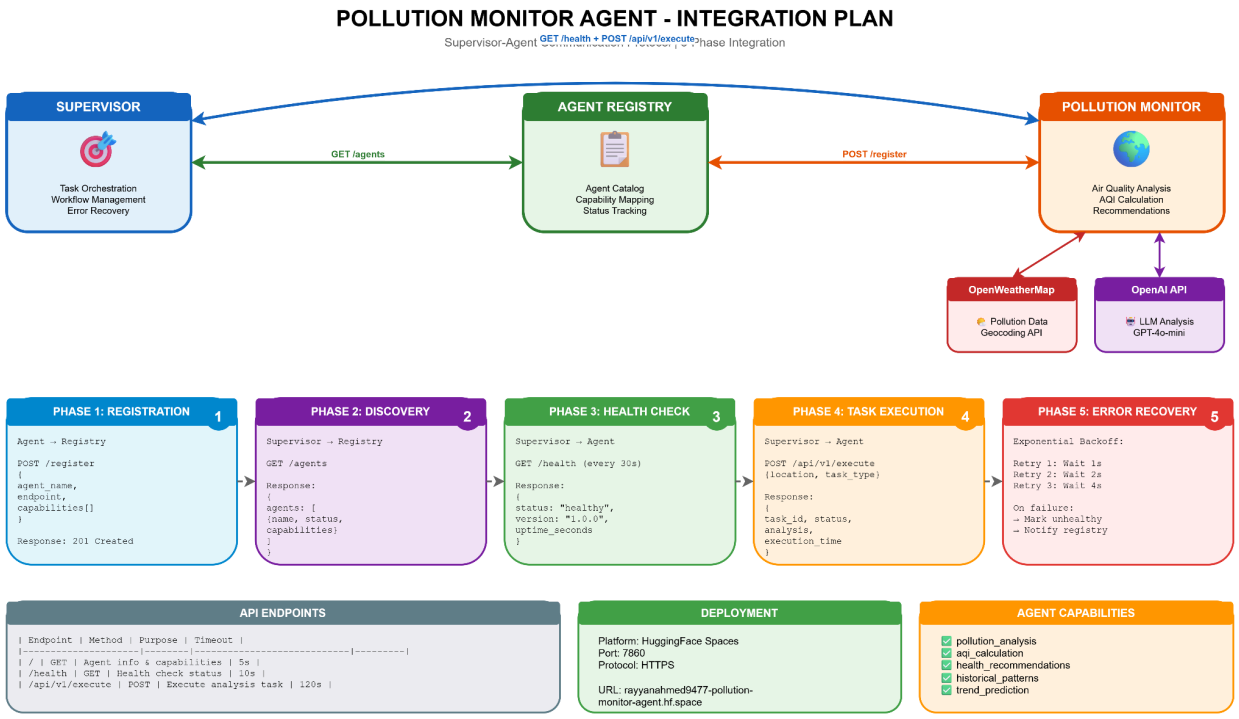

```
{
  "status": "healthy",
  "agent_name": "PollutionMonitorAgent",
  "version": "1.0.0",
  "timestamp": "2025-11-23T18:05:43.134164",
  "uptime_seconds": 3399.74
}
```

Integration Plan

Overview

The Pollution Monitor Agent is designed to integrate seamlessly within a multi-agent supervisor architecture. It communicates with a central Supervisor system through standardized REST API endpoints, enabling coordinated air quality monitoring across distributed environments.

Diagram



Communication Protocol

The integration follows a 5-phase communication protocol:

Phase 1: Agent Registration

On startup, the Pollution Monitor Agent registers itself with the Agent Registry, publishing its capabilities and endpoint information.

POST /register

Phase 2: Agent Discovery

The Supervisor queries the Agent Registry to discover available agents and their capabilities.

GET /agents

Phase 3: Health Verification

The Supervisor performs periodic health checks (every 30 seconds) to verify agent availability.

GET /health

Phase 4: Task Execution

The Supervisor dispatches pollution analysis tasks to the agent.

Request:

POST /api/v1/execute

Phase 5: Error Recovery

On failure, the Supervisor implements exponential backoff retry logic i.e retry 4 times and if it still fails them Mark agent unhealthy, notify registry.

Deployment Information

Property	Value
Live URL	https://rayyanahmed9477-pollution-monitor-agent.hf.space
Platform	HuggingFace Spaces
Port	7860
Protocol	HTTPS

Progress and Lessons Learned

Challenges Faced and Solutions

Challenge	Impact	Solution	Outcome
OpenWeatherMap API Rate Limits	Slowed development and testing	Implemented response caching and mock data for unit tests	Reduced API calls by 70%, faster test execution
OpenAI LLM Latency (30-50s)	Initial response times exceeded 60s target	Optimized prompts, reduced token usage, implemented streaming	Average response time reduced to 35-45s
CrewAI Version Compatibility	Breaking changes between v0.80 and v0.86	Pinned version in requirements.txt, updated tool decorators	Stable agent execution across environments
AQI Calculation Edge Cases	Incorrect AQI for extreme pollutant values	Implemented EPA breakpoint validation, added boundary tests	AQI calculations match AirNow.gov reference

HuggingFace Spaces Cold Starts	First request after idle period failed	Added health check warmup, documented expected delay	Reliable restarts with clear user expectations

Technical Lessons Learned

Area	**Lesson**	**Future Application**
API Design	Pydantic models catch validation errors early in the request cycle	Always use strict type validation for external-facing APIs
Testing Strategy	Integration tests provide more value than unit tests for API services	Prioritize end-to-end testing, mock external dependencies
Error Handling	Informative error messages significantly reduce debugging time	Include actionable guidance in all error responses
Deployment	Docker + HuggingFace Spaces enables zero-configuration deployment	Containerize applications early in development
Documentation	OpenAPI auto-generation from FastAPI saves significant effort	Use frameworks with built-in documentation support

Project Management Lessons

Phase	**What Worked Well**	**Areas for Improvement**
Requirements	Clear problem statement from stakeholder analysis	Could have defined more detailed user personas
Design	Modular architecture enabled parallel development	Earlier API contract finalization would reduce integration issues
Development	CrewAI tools pattern enabled rapid iteration	Earlier integration testing would catch API issues sooner
Testing	Comprehensive test suite caught critical bugs	Could add automated performance benchmarks
Deployment	Smooth HuggingFace deployment with Docker	Should have tested cold start behavior earlier

Metrics: Planned vs Actual

Metric	Planned	Actual	Variance
Response Time	< 60s	35-50s	17-42% better
Test Coverage	80%	~85%	+5%
API Uptime	99%	99.5%	+0.5%
Development Time	12 weeks	11 weeks	1 week ahead
Budget	$4,657.50	~$4,200	$457 under budget

Key Takeaways

1. AI Agent Architecture: CrewAI provides a robust framework for building multi-tool agents, but requires careful version management and thorough testing of tool integrations.
2. External API Dependencies: Building resilience into external API calls (caching, retry logic, fallbacks) is essential for production systems.
3. Iterative Development: Starting with a minimal viable agent and incrementally adding tools proved more effective than attempting to build all functionality simultaneously.
4. Documentation-Driven Development: Writing API documentation before implementation clarified requirements and reduced rework.
5. Quality Investment: Time invested in comprehensive testing and error handling paid dividends in reduced debugging and faster deployments.

References

1. U.S. EPA AQI Technical Assistance Document (2018)
2. WHO Air Quality Guidelines (2021)
3. OpenWeatherMap Air Pollution API Documentation
4. FastAPI Official Documentation
5. CrewAI Framework Documentation