METAR & TAF Real-Time Monitoring and Consistency System

1. Project Overview :

An automated system that continuously monitors METAR and TAF aviation weather data from multiple official sources. It runs every 15 minutes, compares data consistency, detects updates or stale states, and maintains a complete historical record for audit and analysis. The system supports Airline Operations, Dispatch, and Monitoring teams by ensuring data reliability and synchronization visibility.

---

2. Key Objectives Achieved :

Objective	Status
Automated scraping	☑ Implemented
Multi-source comparison	☑ Implemented
Data normalization (whitespace-safe)	☑ Implemented
Match / Partial / No-data classification	☑ Implemented
Live vs stale update detection	☑ Implemented
Continuous historical logging	☑ Implemented
Clear operational status indicators	☑ Implemented

---

3. High-Level System Flow

Scheduler (every 15 minutes)

↓

Scrape METAR & TAF from Base + Comparison Sources

↓

Normalize & Parse METAR strings

↓

Compare Base vs All Sources

↓

Classify Status (match / partial_match / no_data)

↓

Detect Live vs Stale updates using METAR timestamps

↓

Append results to metar.json & taf.json (history preserved)

---

4. Output Files :

File	Purpose
metar.json	Full historical METAR log (appends every run)
taf.json	Full historical TAF log (appends every run)

Each run adds a new timestamped entry. No data is overwritten.

---

5. Output Structure (Example) :


```
{
  "timestamp": "2025-11-07T15:21:33Z"
  "data": {
    "VILH": {
      "base": "071500Z 00000KT 6000 SKC M01/M13 Q1027 NOSIG=",
      "comparisons": {
        "olbs_amssdelhi_gov_in": "...",
        "olbs_amsschennai_gov_in": "..."
      },
      "status": "match",
      "update_status": {
        "overall_condition": "□ fully_live"
      }
    }
  }
}
```


---

6. Status Definitions :

- **match** — All sources identical.

  Meaning: ✔ Data consistent across platforms.

- **partial_match** — Some sources updated, others pending.
  Meaning: 🔄 Update propagating across systems.
- **no_data** — Base METAR unavailable.
  Meaning: ⚠ Base source has not published data yet.

---

7. Live / Stale Detection Logic :

- METAR timestamp format: DDHHMMZ (e.g., 071500Z)
- If a site's METAR timestamp is newer than the base → **live**
- If unchanged → **stable (no new update)**

---

8. Overall Conditions

Condition	Appears When	Meaning
□ fully_live	status == match AND all sites updated	All sources synchronized
🔄 updating	status == partial_match	Update in progress
□ stable_no_new_update	status == match but no timestamp change	Data valid but unchanged
⬤ no_data	Base METAR missing	No report available

---

9. Stable / No New Update Example :

"status": "match",
"update_status": {
  "overall_condition": "□ stable_no_new_update"
}

Meaning:

✔ Data valid

✔ No new METAR published

✔ System healthy

---

10. Timestamp Handling :

Each run updates the top-level timestamp to reflect the latest monitoring cycle, even

if no new data is published. This ensures continuous tracking and operational transparency.

---

11. Design Strengths :

- Avoids false mismatches caused by whitespace
- Differentiates fresh vs stable data
- Maintains complete audit history
- Scalable to more stations or sources
- Ideal for dashboards, alerts, and reporting

---

12. Business Value :

- Enhances confidence in weather data reliability
- Detects delayed or missing updates early
- Supports operational decision-making with verified, synchronized data
- Provides traceable historical records for compliance and analysis

Project 2:

Flight Data MCP API – Comprehensive Testing Documentation

Executive Summary :

This document outlines the complete testing strategy and implementation for the **Flight Data MCP API** project. The testing framework ensures the system's reliability, maintainability, and performance through extensive **unit**, **integration**, and **regression** testing. The suite provides rapid feedback, high coverage, and confidence in production readiness.

📊 Testing Overview

Metric	Value	Coverage
Total Test Files	15 files	100% of modules

Total Test Cases	50+ individual tests	All critical functions
Test Categories	3 types	Unit, Integration, Regression
Code Coverage	95%+	Server, Client, UI modules
Execution Time	< 5 seconds	Fast feedback loop

🎯 Testing Strategy :

1. **Unit Testing (Individual Function Testing)**
   - Focuses on isolated functions with mocked dependencies.
   - Ensures correctness of logic and error handling.
1. **Integration Testing (Multi-Component Testing)**
   - Validates interactions between APIs, databases, and external services.
   - Confirms data flow and component interoperability.
1. **Regression Testing (Change Safety)**
   - Protects against unintended side effects from new code changes.
   - Ensures consistent behavior across releases.

🗀 Test Suite Structure :

A. Client Module Testing (test_client/)

Test File	**Purpose**	**Test Count**	**Coverage**
test_azure.py	Azure OpenAI API integration	4	API calls, error handling
test_connection.py	MCP server connections	6	Connection logic, timeouts
test_plan.py	Tool planning algorithms	5	Planning logic, tool selection
test_run_query.py	Query execution	4	Query processing, results
test_utils.py	Utility functions	4	Tool prompts, registries

Key Test Examples:

- Mocked Azure API responses for latency and error handling.
- Connection retry logic validation under simulated network failures.
- Planning algorithm correctness with varied tool configurations.

B. Server Module Testing (test_server/) :

Test File	Purpose	Test Count	Coverage
test_run_agg_query.py	MongoDB aggregation queries	4	Success, error, empty, multiple
test_raw_query.py	Raw MongoDB queries	3	Query execution, results
test_health_check.py	Health monitoring	3	Status checks, dependencies
test_db.py	Database connections	4	Connection, authentication
test_tools_basic.py	Tool implementations	6	Individual tool functions
test_utils.py	Server utilities	3	Helper functions

Critical Test Examples:

- Aggregation query validation with mock MongoDB collections.
- Health check endpoint verification under dependency failures.
- Authentication and connection resilience testing.

C. UI Adapter Testing (test_ug_ai/)

Test File	Purpose	Test Count	Coverage
test_ag_ui_adapter_basic.py	Basic API endpoints	3	Health, root, status
test_ag_ui_errors.py	Error scenarios	3	Invalid input, connection errors
test_ag_ui_streaming.py	Streaming responses	2	Event streaming, real-time

Streaming Test Example:

- Simulated event stream validation for real-time updates.
- Error handling for interrupted or malformed streams.

☑ Test Quality Assurance :

Mock Strategy

- **External APIs:** Azure OpenAI and MongoDB fully mocked.
- **Database Calls:** MockCursor and MagicMock ensure isolation.
- **Network Requests:** HTTP clients mocked for deterministic results.

Assertion Quality

- Validates both success and failure paths.
- Ensures consistent response formats and error messages.

Error Testing Coverage

- Network timeouts
- Database connection failures
- Invalid input validation
- API rate limiting
- Malformed responses

🌓 Regression Testing Benefits :

Scenario	**Protection**
Bug fixes	Old functionality remains intact
Performance optimizations	Response formats stay consistent
Refactoring	Behavior is preserved
New features	Existing features don’t break
Library updates	API compatibility maintained

Continuous Integration Ready:
All tests are automated and integrated into the CI pipeline for consistent validation during every build.

📈 Business Value Delivered :

1. **Reliability Assurance**
   - 50+ test cases validate critical business logic.
   - Comprehensive error and edge case coverage.
1. **Development Velocity**
   - Rapid feedback (< 5 seconds).
   - Safe refactoring and confident deployments.
1. **Maintainability**
   - Clear documentation and consistent test patterns.
   - Structured mock strategy for reproducible results.
1. **Cost Savings**
   - Early bug detection reduces production incidents.
   - Automated regression testing minimizes manual QA effort.

🚀 Deployment Readiness :

Production Readiness Checklist

- ☑ Unit Tests: All individual functions validated
- ☑ Integration Tests: Component interactions verified
- ☑ Error Handling: Failure scenarios comprehensively tested

Status: *Ready for production deployment*

Time Zone Conversion Tool Implementation - Documentation

Executive Summary :

Successfully implemented and integrated a **UTC to Local Time Conversion Tool** across the **Flight Data MCP API system**. This enhancement enables real-time

timezone conversions for flight scheduling, operations, and user interfaces, specifically optimized for **Indian Standard Time (IST)** operations.

🎯 Feature Overview :

Business Value

- Real-time timezone conversion for flight operations
- IST (Asia/Kolkata) optimization for Indian aviation market
- Standardized time handling across all flight data systems
- User-friendly time display in local timezones

Technical Implementation

- **MCP Tool Integration** – Server-side conversion logic
- **Client-side Planning** – Intelligent tool selection
- **Tool Registry** – Centralized tool documentation
- **Comprehensive Testing** – Validation and error handling

Implementation Details :

A. Server-side Tool Implementation (server.py)

@mcp.tool()
async def convert_utc_to_local_time(utc_time_str: str, timezone_str: str = "Asia/Kolkata") -> str:

"""

Convert UTC datetime to local time with +5:30 IST offset

Features:

☑ Multiple input formats (ISO, standard)

☑ Manual IST conversion (+5:30)

☑ Error handling and validation

☑ JSON response format:

"""

Key Capabilities

Feature	Implementation	Business Benefit

Format Flexibility	Supports ISO and standard datetime formats	Works with various flight data sources
IST Optimization	Default +5:30 offset for Indian operations	Automatic local time for Indian airports
Error Handling	Graceful parsing and validation	Reliable operations even with bad data
JSON Response	Structured output with metadata	Easy integration with UI components

---

B. Client-side Integration (client.py) :

Enhanced Planning System

SYSTEM_PROMPT_PLAN = f"""

Rules for time conversion:

3. If user asks about current time or converting UTC to local time,

   use "convert_utc_to_local_time" with arguments:

   - utc_time_str: UTC datetime string

   - timezone_str: Target timezone (defaults to Asia/Kolkata)

"""

Summarization Enhancement

SYSTEM_PROMPT_SUMMARIZE = """

When summarizing time conversions:

- Clearly mention UTC time

- Show +5:30 hour offset applied

- Display resulting local (IST) time

- Focus on clarity and readability

"""

C. Tool Registry Update (tool_registry.py) :

"convert_utc_to_local_time": {

   "args": ["utc_time_str", "timezone_str"],

   "desc": "Convert UTC datetime to local time with +5 hours 30 minutes IST offset.

Timezone defaults to 'Asia/Kolkata'."

}

Registry Benefits

- Centralized Documentation – All tools in one place
- Argument Specification – Clear parameter requirements
- Description Standards – Consistent tool documentation

Quality Assurance Implementation :

Test Function (client.py)

async def test_convert_time():

"""

Comprehensive time conversion testing:

1. Manual calculation verification

2. MCP tool validation

3. Response format checking

4. Error handling validation

"""

Test Coverage

Test Scenario	Validation	Expected Result
Valid UTC Input	"2024-06-23 13:25"	"2024-06-23 18:55" (+5:30)
ISO Format	"2024-06-23T13:25:00"	Parsed and converted correctly
Invalid Format	"invalid-date"	Graceful error response
Response Structure	JSON validation	ok: true, data: {...}

📊 Integration Flow

End-to-End User Journey

User Query: "Convert 13:25 UTC to Indian time

↓

Client Planning System

↓

Tool Selection: convert_utc_to_local_time

↓

MCP Server Execution

↓

UTC + 5:30 = IST Calculation

↓

JSON Response Generation

↓

Client Summarization

↓

User-Friendly Output: "13:25 UTC becomes 18:55 IST"

🕐 **Original UTC Time:** 2024-06-23 13:25

🇮🇳 **Simulated Local Time (+5:30):** 2024-06-23 18:55

🚀 **Implementation Verification:**

All test cases passed successfully, confirming accurate UTC-to-IST conversion, robust error handling, and seamless integration across the MCP API system.