The Final Test Plan

Date: 24 November 2024

MONTCLAIR STATE UNIVERSITY

ClimateView

The Final Test Plan
(VERSION - 1.0)

By:
TEAM SPAM-HA! (SH)

Course: CSIT 515 Software Engineering & Reliability – Fall 2024

UNDER THE GUIDANCE OF

Professor **Dr. Hubert Johnson**

BS, MS, EDM, EdD

Team Members:

ABHISHEK REDDY ADUNOORI
PRATIK BUDHATHOKI
SIVA PRASANTH SIVAKUMAR
KOPPULA SAI HRUSHIK
MIKKO PALIS

Revision History

Date	Version	Description	Author
11/21/2024	1	The Final Test Plan	TEAM - SPAM-HA!

Table Of Contents

List of Tables

1. Frontend UI Testing Plan

1.1. Test Objective

Verify the functionality, performance, and reliability of the Frontend UI by focusing on UI interactions, input validation, and communication with the backend.

Objective ID	Test Objective	Priority	Completion Criteria

OBJ-FE-001	Verify that UI elements are displayed correctly and respond as expected to user interactions.	High	All UI elements are visible, functional, and aligned correctly. The UI should be responsive to different screen sizes and resolutions.
OBJ-FE-002	Ensure data input fields (region selection, time frame, climate variables) function correctly.	High	Input validation is effective, and user inputs are sanitized to prevent security risks.
OBJ-FE-003	Validate that the UI communicates effectively with the backend, sending requests and receiving responses without errors.	High	API requests and responses are handled seamlessly.
OBJ-FE-004	Confirm that the front end displays clear error messages for issues (e.g., API errors, invalid inputs).	Medium	Error messages are user-friendly and informative.

Table No 1.1.1 Frontend UI Test Objective Id's

1.2. Test Procedures and Expected Outputs

1.2.1. UI Elements Functionality Test

- **Objective**: Ensure all UI components are displayed and operate correctly.
- **Procedure**: Interact with each UI element in a browser (buttons, dropdowns, input fields).
- **Expected Output**: UI elements are functional, responsive, and perform as expected.

1.2.2. User Input Validation Test

- **Objective**: Verify input handling and validation.
- **Procedure**: Test with valid and invalid inputs for fields (region, time frame, variables).
- **Expected Output**:
    - Valid inputs are accepted.

- Invalid inputs trigger error messages.

1.2.3. API Communication Test

- **Objective**: Confirm effective communication with the backend.
- **Procedure**: Monitor network requests via browser tools during UI actions.
- **Expected Output**: API calls succeed, and errors are handled gracefully.

1.2.4. Error Message Display Test

- **Objective**: Validate clarity of error messages.
- **Procedure**: Simulate various error scenarios and observe error displays.
- **Expected Output**: Messages describe the issue clearly and offer guidance.

1.3. High-Level Code for Input Validation

1.3.1. Frontend UI - Input Validation

function ValidateRegion(regionInput):
Check if the regionInput is valid based on a predefined list of allowed regions.
If valid:
Return "Valid".
Otherwise:
Display an error message: "Invalid region selected. Please choose from the list."
Return "Invalid".

function ValidateTimeFrame(startDate, endDate):
Compare startDate and endDate to ensure the startDate is earlier than the endDate.
If valid:
Return "Valid".
Otherwise:
Display an error message: "Invalid time frame. The start date must be before the end date."
Return "Invalid".

2. Backend Server Test Plan

2.1. Introduction

2.1.1. Purpose

The purpose of this test plan is to define the objectives, scope, and testing methodologies for validating the backend server of the ClimateView system. The backend server acts as the core processing layer, integrating with:

- The **Data Fetching Module** to retrieve and process climate data.
- The **AI Prediction Module** to generate climate trend predictions.
- The **Visualization Engine** to serve data for display.

2.1.2. Scope

The backend server testing will validate:

- Provide APIs for accessing data (historical, real-time, and predictions).
- Normalize and store data retrieved from external APIs (NASA, NOAA).
- Facilitate communication between modules, ensuring seamless integration.
- Handle concurrent requests and maintain scalability.
- Implement robust error handling and security mechanisms.

2.2. Test Objectives

The Test Objectives define the primary goals for testing the backend server to ensure functionality, integration, performance, and security. Each objective aligns with the core modules and features of the backend.

Number	Test Objective	Priority	Completion Criteria
OBJ - 001	Validate data fetching and processing from the **Data Fetching Module**.	High	Fetched data is processed and stored successfully.

OBJ - 002	Ensure accurate API responses for data queries.	High	API endpoints return data in the correct format.
OBJ - 003	Test backend integration with the AI Prediction Module.	High	Predictions align with expected trends and formats.
OBJ - 004	Verify backend communication with the Visualization Engine.	Medium	Data is served accurately and efficiently to the frontend.
OBJ - 005	Ensure error handling and logging mechanisms are robust.	High	Errors are logged, and descriptive messages are provided.
OBJ - 006	Test backend performance under high concurrency.	Medium	Response time ≤ 2 seconds for 95% of requests.
OBJ - 007	Perform security tests to prevent unauthorized access.	Medium	Unauthorized access attempts are blocked.

Table No. 2.2.1 Back end Test Objectives

2.3. Test Matrix

The Test Matrix provides an overview of the testing coverage across the different backend modules, indicating which types of tests will be conducted for each module. It maps software functionalities to test types (Unit, Integration, Performance, and Security). Here's a detailed explanation of the Test Matrix:

Software Function	**Unit Test**	**Integration Test**	**Performance Test**	**Security Test**
Data Fetching Module	✓	✓	✓	✗

AI Prediction Module	✓	✓	✓	✓
Backend API Endpoints	✓	✓	✓	✓
Visualization Integration	✓	✓	✓	✓
Error Handling	✓	✓	✗	✓

Table No. 2.3.1 Test Matrix

2.4. Software Modules

The Software Modules describe the core components of the backend server, their responsibilities, and how they are evaluated. Each module corresponds to a key feature or functionality of the backend system.

Number	**Module Name**	**Description**	**Evaluation Criteria**
MOD - 001	Data Fetching	Retrieves and processes data from NASA/NOAA APIs.	Data matches API structure and is normalized.
MOD - 002	API Management	Provides endpoints for accessing processed data.	Endpoints respond with correct data and error codes.
MOD - 003	AI Integration	Connects to the AI module for predictions.	Predictions are formatted correctly for the frontend.
MOD - 004	Database Operations	Manages CRUD operations in the database.	CRUD operations are consistent and secure.

Table No. 2.4.1 Software Modules

2.5. Test Specifications

2.5.1. Functional Testing.

2.5.1.1. TC-001: Validate Data Fetching

- **Objective**: Ensure the backend retrieves and processes data correctly from external APIs (e.g., NASA, NOAA).
- **Precondition**: The Data Fetching Module is configured with valid API endpoints.
- **Input**:
  - Valid API keys and sample queries.
  - Invalid API keys or incorrect query parameters.
- **Steps**:
  - Send requests to the external APIs with valid credentials.
  - Test with invalid API keys or malformed requests.
- **Expected Output**:
  - Data is successfully fetched and logged for valid requests.
  - Error messages are returned for invalid requests.
  - Logs contain timestamps and error details for debugging.

2.5.1.2. TC-002: Verify API Endpoints

- **Objective:** Validate API endpoints respond correctly to client queries
- **Precondition:** API endpoints are deployed and accessible.
- **Input:**
  - Query for historical data: /api/historical?region=us&date=2024-01-01.
  - Query for real-time data: /api/realtime?region=us.
  - Invalid query: /api/historical?invalid_param=xyz.
- **Steps:**
  - Submit valid queries to the API endpoints.
  - Submit invalid or malformed queries to test error handling.
- **Expected Output**:
  - Correct data is returned for valid queries in JSON format.

    - 400 Bad Request error for invalid queries with descriptive error messages.

2.5.1.3. TC-003: Test AI Module Integration

- **Objective**: Ensure predictions generated by the AI module are accurate and formatted for frontend use.
- **Precondition**: The AI Prediction Module is integrated with the backend.
- **Input**:
    - Historical climate datasets.
- **Steps**:
    - Send historical data to the AI module via the backend.
    - Retrieve predictions and validate their format.
    - Test with incomplete or noisy historical data to check robustness.
- Expected Output:
    - Predictions align with expected trends based on historical data.
    - Output is returned in the required format (e.g., JSON).

2.5.1.4. TC-004: Test Error Handling and Logging

- **Objective**: Validate the backend handles errors gracefully and logs them appropriately.
- **Precondition**: Logging functionality is active.

- **Input**:
    - Invalid API requests.
    - Database connectivity issues.
- **Steps**:
    - Simulate invalid API requests (e.g., malformed URLs, missing tokens).
    - Temporarily disconnect the database and observe backend behavior.
- **Expected Output**:
    - Errors are logged with timestamps and relevant details.
    - Users receive meaningful error messages without exposing sensitive information.

2.5.2. Performance Testing

2.5.2.1. TC-BE-005: Simulate High Traffic

- **Objective**: Test backend response time and stability under concurrent user load.
- **Precondition**: Backend is deployed in a staging environment.
- **Input**:
  - 100+ concurrent API requests.
- **Steps**:
  - Simulate high traffic using JMeter or Locust.
  - Measure response times and error rates.
- **Expected Output**:
  - Response time ≤ 2 seconds for 95% of requests.
  - No server crashes or bottlenecks.

2.5.2.2. TC-BE-006: Test Backend with Large Datasets

- **Objective:** Validate backend handling of large datasets.
- **Precondition**: Database contains datasets with >1M records.
- **Input**:
  - Query for large datasets: /api/historical?region=global&date_range=2010-2024.
- **Steps**:
  - Submit queries that return large datasets.
  - Monitor memory and CPU usage during processing.
- **Expected Output**:
  - Backend processes queries without crashes.
  - Memory usage remains within acceptable limits.

2.5.3. Security Testing

2.5.3.1. TC-BE-007: Test Unauthorized Access

- **Objective:** Ensure sensitive API endpoints are protected from unauthorized access.
- **Precondition:** Authentication and authorization mechanisms are in place.
- **Input:**
  - Requests without valid authentication tokens.
- **Steps:**
  - Attempt to access protected endpoints without a token.

    - Attempt to access endpoints with expired or invalid tokens.
- **Expected Output**:
    - 401 Unauthorized error for requests without valid authentication.
    - Logs capture unauthorized access attempts.

2.5.3.2. TC-BE-008: Validate Input Sanitization

- **Objective**: Protect the backend from malicious payloads (e.g., SQL injection).
- **Precondition**: Input validation and sanitization mechanisms are active.
- **Input**:
    - Malicious SQL payload: /api/data?query="SELECT * FROM users;".
- **Steps**:
    - Submit malicious payloads to various endpoints.
    - Monitor system response and logs.
- **Expected Output**:
    - Malicious inputs are sanitized and do not affect the backend.
    - Logs record the attack attempt without exposing sensitive data.

2.5.4. Integration Testing

2.5.4.1. TC-BE-009: Test Data Flow Between Modules

- **Objective**: Validate seamless integration between the Data Fetching Module, AI Prediction Module, and APIs.
- **Precondition**: All modules are deployed in a staging environment.
- **Input:**
    - Query historical data, process it via the AI module, and serve it through APIs.
- **Steps:**
    - Submit data-fetching requests.
    - Pass the fetched data to the AI module for predictions.
    - Retrieve results via API endpoints.
- **Expected Output**:
    - Data flows correctly through all modules.
    - Predictions are served accurately to clients.

2.5.4.2. TC-BE-010: Test Frontend Integration

- **Objective**: Ensure backend data is accessible by the frontend visualization engine.
- **Precondition**: Backend and frontend are integrated in a test environment.
- **Input**:
  - API queries from the frontend.
- **Steps**:
  - Submit data requests via the frontend.
  - Observe backend responses and frontend rendering.
- **Expected Output**:
  - Data is served efficiently to the frontend for visualization.

3. Data Fetching Module Test Plan

3.1. Validation of Data Retrieval and Error Handling for External APIs

Software Project: ClimateView
Software Module Name: Data Fetching Module
Test Name: Validation of Data Retrieval and Error Handling for External APIs
Test Number: DF-001
Date: 11/20/24

3.1.1. Test Objective:

To validate the accuracy, error handling, and performance of the Data Fetching Module when retrieving climate data from external APIs such as NASA and NOAA.

Objective ID	Test Objective	Priority	Completion Criteria
OBJ-001	Validate that climate data is retrieved accurately from external APIs (e.g., NASA, NOAA).	High	Retrieved data matches the API response with 99% accuracy.

OBJ-002	Verify that the module handles API timeouts by using cached data.	High	Cached data is used, and a fallback notification is shown to the user.
OBJ-003	Ensure the system logs errors and halts processing for invalid or corrupted API responses.	High	Error logs are created with appropriate timestamps, and data processing stops.
OBJ-004	Test that the API request response time meets the performance benchmark of 2 seconds or less.	Medium	Average response time is ≤ 2 seconds under normal load conditions.
OBJ-005	Confirm that the module handles unreachable APIs gracefully by falling back to cached data.	High	Cached data is displayed, and an error is logged: "Endpoint unreachable."

Table No 3.1.1.1 Data Fetching Module Objective Id's

3.1.2. Test Input:

Prepare the following input types:

- **Valid API Data:** Ensure the API endpoint returns valid and properly formatted data.
- **Invalid API Data:** Simulate an API returning incorrect or malformed data.
- **API Timeout:** Simulate a timeout scenario where the server doesn't respond.
- **Unavailable API:** Provide an unreachable or incorrect API endpoint.
- **High Traffic Load:** Simulate multiple concurrent API requests to test performance under load.
- **Empty Cache:** Test a scenario where no cached data exists for fallback during API timeouts or unavailability.

Input Type	**Description**	**Example**
Valid API Data	Well-formatted, complete data returned from a valid API endpoint.	{ "temperature": 72, "humidity": 40, "wind_speed": 10 }
Malformed Data	API response missing fields or containing	{ "temperature": null, "humidity": "N/A" }

	invalid values.	
API Timeout	Simulated delayed or no response from the API.	API request exceeds a 2-second timeout or returns HTTP 504.
Unavailable API	Incorrect or unreachable endpoint simulating an offline server.	API URL: https://invalid-api.com/data
High Traffic Load	Simulate multiple concurrent API requests to test performance under load.	100 parallel API requests per second.
Empty Cache	Empty Cache	Cache data: { } (empty).

Table No 3.1.2.1 Data Fetching Module Test Input

3.2. Test Procedures

3.2.1. Connect to a valid API endpoint and retrieve climate data.

- **Precondition:** Ensure the API endpoint is live and accessible.
- **Steps:**
    - Send a request to the valid API endpoint (https://example-api.com/data).
    - Verify that the returned data matches the expected format and values.
- **Expected Output:** Retrieved data matches the API response.
- **Example:**

```
{
    “temperature”: 72,
    “humidity": 40,
    "wind_speed": 10
}
```

3.2.2. Simulate an API timeout and verify fallback to cached data.

- **Precondition:** Preload the cache with valid data.
- **Steps:**
    - Simulate an API timeout by delaying the server response beyond 2 seconds.

    - Verify that the system retrieves and displays data from the cache.
- **Expected Output:** Cached data is displayed, and a timeout error is logged: "TimeoutError: API request failed after 2 seconds."
- **Cached data example:**

```
{
    "temperature": 68,
    "humidity": 35
}
```

3.2.3. Provide an invalid API response and confirm the system logs errors.

- **Precondition:** Configure the API to return malformed data.
- **Steps:**
    - Replace the API response with malformed data:

```
{
    "temperature": null,
    "humidity": "N/A"
}
```

- Check the system logs for the error and confirm no data is processed.
- **Expected Output:**
    - Error log: "InvalidDataError: Missing or invalid temperature field."
    - No data is displayed.

3.2.4. Test with an unreachable API and ensure cached data is used seamlessly.

- **Precondition:** Ensure the API endpoint is unavailable (e.g., incorrect URL or server offline) and cache is preloaded.
- **Steps**
    - Use an invalid API URL (https://invalid-api.com/data).
    - Verify that the system displays cached data and logs an error.
- **Expected Output:** Cached data is displayed, and an error is logged "APIError: Endpoint unreachable at https://invalid-api.com/data."
- **Cached data example:**

```
{
    "temperature": 70,
```

"humidity": 38

}

3.2.5. Simulate high traffic and verify response time and success rate.

- **Precondition:** The API endpoint is live and capable of handling stress testing.
- **Steps:**
  - Simulate 100 concurrent requests to the API endpoint.
  - Measure the average response time and success rate.

- **Expected Output:**
  - Average response time ≤ 2 seconds.
  - Success rate ≥ 95%.

3.2.6. Test with an empty cache and unreachable API.

- **Precondition:** Clear the cache and simulate an unavailable API.
- **Steps:**
  - Clear the cache to ensure no fallback data is available.
  - Simulate an unreachable API endpoint.
- **Expected Output:**
  - Error message: "Data unavailable: API and cache both failed."
  - Error log is created, and no data is displayed.

3.3. Test Output

3.3.1. Climate data is retrieved successfully and matches the expected API output.

- **Expected Output:**

  - Data retrieved matches the response from the API.
  - Example API Response:

```
{
    "temperature": 72,
    "humidity": 40,
    "wind_speed": 10
}
```

  - No errors logged

3.3.2. Cached data is displayed when API requests timeout.

- **Expected Output:**
    - Cached data is retrieved and displayed seamlessly.
    - Timeout error is logged:

      TimeoutError: API request failed after 2 seconds. [Timestamp]
    - Example Cached Data:

```
{
    "temperature": 68,
    "humidity": 35
}
```

3.3.3. Invalid API responses are logged, and processing stop

- System logs an error for invalid or malformed API responses.
- No data is processed or displayed.
- Example Log Entry:

  InvalidDataError: Missing or invalid temperature field. [Timestamp]

3.3.4. Errors are logged for unreachable APIs, and cached data is displayed.

- **Expected Output:**
    - Cached data is retrieved and displayed to the user
    - Error log entry created

      APIError: Endpoint unreachable at https://invalid-api.com/data. [Timestamp]
- Example Cached Data:

```
{
    "temperature": 70,
    "humidity": 38
}
```

3.3.5. High traffic load is handled with acceptable response times and success rates.

- **Expected Output:**
    - Average response time ≤ 2 seconds.
    - Success rate ≥ 95%.
    - Example Metrics:

      Total Requests: 100

      Successful Requests: 97

      Average Response Time: 1.8 seconds

3.3.6. Error is displayed when the cache is empty and API is unavailable.

- **Expected Output:**
    - User is notified: "Data unavailable: API and cache both failed."
    - Error log entry created:

      FallbackError: No cached data available, and API unreachable. [Timestamp]

3.4. Test Controls

- **Run each test multiple times under varying conditions (e.g., different network speeds, API configurations, server loads).**
- **Maintain a consistent testing environment:**
    - Use identical mock data, preloaded cache, and hardware/software configurations.
- **Verify error logs:**
    - All errors must include correct timestamps and error codes (e.g., TimeoutError).
- **Monitor Performance:**
    - Ensure API response times ≤ 2 seconds and success rates ≥ 95%.
- **Use controlled input datasets:**
    - Fixed inputs (e.g., valid/malformed API data, timeouts) for reproducible results.
- **Validate cache integrity:**
    - Cached data must remain unaltered and match preloaded values.
- **Record and analyze test results:**

- Document successes and failures for each test run, logging any anomalies.

3.5. Software or Structure Attribute Tested

- The Data Fetching Module is Tested for the following attributes:
  - **Data Retrieval Functions:** Accurate data fetching for external APIs.
  - **Error-Handling Mechanisms:** Logging errors and managing invalid or delayed API responses.
  - **Caching Functionality:** Proper use of cached data for fallback and consistency.
  - **Performance Metrics:** Ensuring response time and reliability under high traffic.

3.6. Detailed Testing of API Data Retrieval, Error Handling, and Caching

Software Project: ClimateView
Software Module Name: Data Fetching Module
Test Name: Detailed Testing of API Data Retrieval, Error Handling, and Caching
Test Number: DF-002
Date: 11/20/24

Sequence	Source	Script Event	Evaluation Criteria	Comments
1	NASA API	Send a valid API request.	Data matches the API response format and values.	Example response: { "temperature": 72, "humidity": 40 }.
2	Mock API Server	Simulate a timeout scenario (>2s delay).	Cached data is displayed. Error log: "TimeoutError: Request failed.".	Cache must be preloaded with valid data before this test.
3	Mock API Server	Provide invalid API response (malformed data).	System logs an error and halts processing.	Example response: { "temperature": null, "humidity": "N/A" }.
4	Mock API Server	Use an unreachable API endpoint.	Cached data is displayed, and error is logged: "APIError: Endpoint down."	Simulate network disconnection or invalid URL.
5	Mock API Server	Simulate high traffic (100 concurrent requests).	Average response time ≤ 2 seconds. Success rate ≥ 95%.	Log metrics for performance under stress.

6	Mock API Server	Clear cache and simulate unreachable API.	Error message displayed: "Data unavailable: API and cache failed.".	Simulate an empty cache scenario to test fallback mechanisms.

Table No 3.6.1 Data Fetching Module Detailed Testing of API Data Retrieval, Error Handling, and Caching

3.6.1. Sequence Of Testing:

- **Sequence 1: Valid API Request**
  - **Source:** NASA API (or mock server replicating it).
  - **Event:** Send a valid API request to fetch climate data.
  - **Evaluation Criteria:** Data received matches the API response format and values
  - **Expected output:**

```
{
    "temperature": 72,
    "humidity": 40,
    "wind_speed": 10
}
```

  - **Comments**: Test ensures basic functionality and data accuracy.

- **Sequence 2: Timeout Scenario**
  - **Source:** Mock API Server.
  - **Event:** Simulate a delayed server response (>2 seconds).
  - **Evaluation Criteria:** System retrieves cached data and logs a timeout error.
  - **Expected Output:**

Cached data:

```
{
    “temperature”: 68,
    “humidity”: 35
}
```

Error log: “TimeoutError: Request failed after 2 seconds.”

  - **Comments:** Cache must be preloaded with valid data for this test.

- **Sequence 3: Invalid API Response**
  - **Source:** Mock API Server.
  - **Event:** Simulate malformed API response data.
  - **Evaluation Criteria:** System logs the error and halts further processing.
  - **Expected Output:**

        - Error log: "InvalidDataError: Missing temperature field."
        - No data is displayed or saved.
    - **Comments:** Ensures robust error-handling mechanisms.

- **Sequence 4: Unreachable API**
    - **Source:** Mock API Server.
    - **Event:** Simulate an unreachable API endpoint (e.g., invalid URL or offline server).
    - **Evaluation Criteria:** Cached data is displayed. Error is logged: "APIError: Endpoint unreachable."
    - **Expected Output:**
        - Cached data:

```
{
"temperature": 70,
"humidity": 38
}
```

        - Error log: “APIError: Endpoint unreachable at https://invalid-api.com."
    - **Comments:** Ensures fallback to cached data when the API is unavailable?

- **Sequence 5: High Traffic Load**.
    - **Source:** Mock API Server.
    - **Event:** Simulate 100 concurrent API requests.
    - **Evaluation Criteria:** System handles the load with response time ≤ 2 seconds and success rate $\geq 95\%$.
    - **Expected Output:**
        - **Example metrics:**

Total Requests: 100
Successful Requests: 97
Average Response Time: 1.8 seconds

    - **Comments:** Verifies the module’s performance under stress

- **Sequence 6: Empty Cache and Unreachable API**
    - **Source:** Mock API Server.
    - **Event:** Clear the cache and simulate an unreachable API endpoint.
    - **Evaluation Criteria:** System displays an error message: "Data unavailable: API and cache failed."

- **Expected Output:**
  - Error log**:** "FallbackError: No cached data available, and API unreachable."
- **Comments:** Tests edge case handling when both API and cache fail.

3.7. Data Fetching Pseudocode

```
FUNCTION fetchData(apiUrl):
  TRY:
    SEND GET request to apiUrl with 2-second timeout
    IF response is successful:
      PARSE response as JSON
      VALIDATE response data (CALL validateData(response))
      SAVE valid response to CACHE
      RETURN response
    ELSE:
      LOG error "InvalidDataError: Malformed data received"
      RETURN NULL
  EXCEPT TimeoutError:
    LOG error "TimeoutError: API request timed out"
    RETURN cached data if available, ELSE NULL
  EXCEPT RequestError:
    LOG error "APIError: Failed to connect to API"
    RETURN cached data if available, ELSE NULL

FUNCTION validateData(data):
  IF "temperature" EXISTS in data AND data["temperature"] IS NOT NULL:
    RETURN TRUE
  ELSE:
    THROW ValidationError("Missing or invalid temperature field")

FUNCTION saveToCache(data):
  STORE data in local cache

FUNCTION getCachedData():
  RETRIEVE and RETURN data from local cache if available, ELSE NULL

MAIN:
```

```
DEFINE apiUrl as "https://example-api.com/data"
CALL fetchData(apiUrl) AND store result in fetchedData
IF fetchedData IS NOT NULL:
    DISPLAY "Fetched Data: ", fetchedData
ELSE:
    DISPLAY "No valid data available"
```

4. AI Prediction Module Test Plan

4.1 Validation of AI Prediction Module

Software Project: ClimateView
Software Module Name: AI Prediction Module
Test Name: AI Climate Prediction Module Functional and Performance Validation
Test Number: DF-003
Date: 11/21/20024

4..1.1 Test Objectives:

To ensure that the AI Prediction Module delivers accurate, reliable, and efficient climate projections based on historical data, adhering to the specified benchmarks for accuracy, response time, and seamless integration with the Visualization Engine under various operational and edge-case scenarios.

Sequence	Source	Script	Event	Evaluation Criteria	Comments
1	Historical Data	Provide validated historical climate data for benchmarking.	Test the accuracy of predictions using historical trends.	Prediction accuracy exceeds 90%.	Validates reliability of AI models against historical baselines.
2	Mock Request Generator	Simulate multiple requests to the module in real-time.	Test response time for climate projections.	Projections generated within 5 seconds per request under standard	Ensures real-time usability of predictions.

				conditions.	
3	Visualization Engine	Integrate prediction output with the visualization engine.	Validate the compatibility of generated data format.	Output is presented in a structured format compatible with the engine.	Tests seamless integration of prediction and visualization modules.
4	Simulated Edge Cases	Provide incomplete or noisy input data.	Test modules' robustness to handle data quality issues.	System logs errors but avoids crashes; outputs flagged for anomalies	Ensures system reliability under non-ideal conditions.
5	Concurrent Requests	Simulate high traffic (100 concurrent requests).	Test performance under load.	Average response time <= 5 seconds with success rate >= 95%.	Verifies system scalability.

Table No. 4.1.1.1 AI prediction Module Objectives

4.1.2 Test Input:

Prepare the following input types:

1. **Valid Historical Data**: Ensure the input contains validated historical climate data.
2. **Incomplete Data**: Simulate missing or incomplete fields in the input dataset.
3. **Noisy Data**: Provide input with inconsistent formats or outliers.
4. **Synthetic Future Data**: Input hypothetical climate scenarios (e.g., projected CO2 levels).
5. **High Traffic Load**: Simulate concurrent input requests to evaluate performance.
6. **Empty Dataset**: Test how the module behaves when no input data is provided.

Input Type	**Description**	**Example**
Valid Historical Data	Complete, validated data derived from reliable climate databases.	{ "temperature": 72, "humidity": 40, "CO2": 400 }
Incomplete Data	Input dataset with missing critical fields or partially recorded data.	{ "temperature": null, "humidity": 35 }

Noisy Data	Input dataset containing outliers, mixed units, or inconsistent formats.	{ "temperature": "75F", "humidity": "forty", "CO2": 9999 }
Synthetic Future Data	Hypothetical projections based on predictive models	{ "temperature": 85, "humidity": 30, "CO2": 450 }
High Traffic Load	Concurrent requests with varying regions and parameters to stress-test the module.	100 parallel requests querying for different regions and timeframes.
Empty Dataset	An empty or null dataset provided as input.	{}

Table No. 4.1.2.1 AI Prediction Module Test Input

4.2 Test procedures:

4.2.1 Valid Historical Data

- **Precondition**: Ensure the dataset contains validated historical climate data.
- **Steps**:
    1. Load the validated dataset into the AI Prediction Module.
    2. Execute the prediction process using the input data.
    3. Compare the predicted outputs with historical trends.

- **Expected Output:** Predictions closely align with the historical data, achieving accuracy>90%.

- **Example Input:**

```
{
        "temperature": 72,
        "humidity": 40,
         "CO2": 400
}
```

4.2.2 Incomplete Data

- **Precondition**: Prepare input with missing or null fields.
- **Steps**:
    1. Load the incomplete dataset into the AI Prediction Module.
    2. Observe system behavior and logs for error handling.
    3. Verify if flagged outputs are returned for incomplete data.

- **Expected Output:** Errors are logged for missing fields, and predictions are flagged for anomalies.

- **Example Input:**

```
{
        "temperature": null,
        "humidity": 35
}
```

4.2.3 Noisy Data

- **Precondition**: Prepare input with inconsistent formats or outliers.
- **Steps**:
    1. Provide the noisy dataset to the AI Prediction Module.
    2. Execute the prediction process.
    3. Observe if the system normalizes the data or logs errors for invalid entries.

- **Expected Output:** Invalid entries are flagged or corrected, and predictions remain consistent.
- **Example Input:**

```
{
        "temperature": "75F",
        "humidity": "forty",
        "CO2": 9999
}
```

4.2.4 Synthetic Future Data

- **Precondition**: Provide hypothetical future climate projections
- **Steps**:
    1. Load the synthetic dataset into the AI Prediction Module.
    2. Run predictions for the provided future scenarios.
    3. Assess the outputs for plausibility and structured formatting.

- **Expected Output:** Predictions align with the expected trends of the synthetic input scenarios.

- **Example Input:**

```
{
```

```
    "temperature": 85,
    "humidity": 30,
    "CO2": 450
}
```

4.2.5 High Traffic Load

- **Precondition**: Simulate multiple concurrent requests using a traffic generator.
- **Steps**:
    1. Send 100 concurrent requests to the AI Prediction Module.
    2. Record response times and success rates.
    3. Verify if performance metrics meet benchmarks (response time ≤ 3 seconds, success ≥ 95%).

- **Expected Output:** All requests are processed within specified performance thresholds.
- **Example Input:** Multiple parallel requests with varied parameters:

```
[
  {
    "region": "North America",
    "year": 2030
  },
  {
    "region": "Europe",
    "year": 2040
  }
]
```

4.2.6 Empty Dataset

- **Precondition**: Provide an empty or null dataset.
- **Steps**:
    1. Input the empty dataset into the AI Prediction Module.
    2. Observe how the module handles missing data.
    3. Check for proper error handling and logging.

- **Expected Output:** Errors are logged, and the system returns a descriptive message indicating missing input.
- **Example Input:**

```
{

}
```

4.3 Test Output

4.3.1 Valid Historical Data

- **Expected Output:**
    1. Predicted data closely matches historical trends with >90% accuracy.
    2. No errors logged during the prediction process.
    3. Example Prediction:

```
{
        "predicted_temperature": 73,
        "predicted_humidity": 39,
         "predicted_CO2": 405
}
```

4.3.2 Incomplete Data

- **Expected Output:**
    1. Error is logged indicating missing or null fields in the input data.
    2. Prediction stops, and no output is generated.
    3. Example Log Entry:

       MissingFieldError: Temperature field is null. [Timestamp]

4.3.3 Noisy Data

- **Expected Output:**
    1. Invalid or inconsistent input data is normalized or flagged.
    2. Predictions are generated only for valid entries.
    3. Example Prediction:

```
{
  "predicted_temperature": 75,
  "predicted_humidity": 40
}
```

    4. Example Log Entry:

       DataNormalizationWarning: Wind speed format corrected. [Timestamp]

4.3.4 Synthetic Future Data

- **Expected Output:**
    1. Predictions are aligned with the input future scenarios and trends.
    2. No errors are logged during processing.

3. Example Prediction:

```
{
  "predicted_temperature": 85,
  "predicted_humidity": 30,
  "predicted_CO2": 450
}
```

4.3.5 High Traffic Load

- **Expected Output:**
  1. All requests are processed successfully within the defined time (≤5 seconds).
  2. System maintains a >95% success rate.
  3. Example Log Entry

     LoadTestResult: 100 requests processed successfully. Average response time: 2.8 seconds. [Timestamp]

4.3.6 Empty Dataset

- **Expected Output:**
  1. Error is logged indicating missing input data.
  2. No predictions are generated.
  3. Example Log Entry:

     EmptyDataError: Input dataset is empty. [Timestamp]

5. Visualization Engine Test plan

5.1. Test Objective

To ensure that the Visualization Engine accurately renders data-driven visualizations, handles large datasets efficiently, and responds to user interactions.

Objective ID	Test Objective	Priority	Completion Criteria
OBJ-VE-001	Verify that visualizations (charts, maps) are rendered accurately.	High	Visualizations reflect data without distortion or misinterpretation.

OBJ-VE-002	Validate performance with large datasets and complex visualizations.	High	Visualizations load within acceptable timeframes.
OBJ-VE-003	Ensure correct responses to user interactions (zooming, filtering).	Medium	Interaction features function seamlessly.
OBJ-VE-004	Confirm error handling for invalid or missing data.	Medium	Errors result in informative messages or placeholders.

Table No 5.1.1 Visualization Engine Test Objective.

5.2. Test Procedures and Expected Outputs

5.2.1. Visualization Rendering Accuracy Test

Objective: Ensure visualizations match the provided data.
Procedure: Test rendering with various datasets and visualization types.
Expected Output: Accurate representation of data points and correct labels.

5.2.2. Performance and Large Dataset Test

Objective: Assess performance with large data inputs.
Procedure: Provide large datasets and measure rendering times.
Expected Output: Visualizations load quickly, with no noticeable lags.

5.2.3. User Interaction Test

Objective: Verify interactivity of visualizations.
Procedure: Test zooming, panning, and filtering on charts and maps.
Expected Output: Interactions are smooth, with updated data displays.

5.2.4. Error Handling and Placeholder Test

Objective: Validate error handling for incomplete/invalid data.
Procedure: Simulate missing or incorrect datasets.
Expected Output: Error messages or placeholders are displayed, and the engine remains functional.

5.3. High-Level Code for Error Handling.

5.3.1. Visualization Engine - Error Handling

function RenderVisualization(data):

Check if the format of data is correct and all required elements are present.

If the format is valid:

Proceed to render the visualization.

Otherwise:

Log an error with the message: "Invalid data format received. Unable to render visualization."

Display a placeholder message: "Visualization unavailable due to data issues."

function DisplayPlaceholder(message):

Replace the visualization area with the given message to inform the user about the issue.

6. Content Management System Test Plan

6.1. Objective:

to verify essential features such as content upload, retrieval, role-based permissions, error handling, and performance in a range of situations in order to guarantee the Content Management System's (CMS) dependable operation.

6.2. Test Objectives:

Objective ID	Test Objective	Priority	Completion Criteria
OBJ-001	Validate the upload and retrieval of supported file types.	High	Files upload successfully, are stored, and accessible without errors.

OBJ-002	Ensure unsupported file types are rejected with clear messages.	High	Upload fails with an error: “File type not supported.”
OBJ-003	Verify proper handling of storage capacity limitations.	High	Upload fails with a message: “Storage limit reached.”
OBJ-004	Enforce role-based permissions for content actions.	Medium	Actions are restricted based on user roles.
OBJ-005	Test performance for concurrent user actions.	Medium	Response time ≤ 5 seconds with ≥ 98% success rate under high traffic.
OBJ-006	Validate error handling for corrupted file uploads.	High	Errors are logged, and corrupted files are not processed.

Table Number 6.2.1 test objectives

6.3. Input Scenarios:

Input Type	Description	Example
Valid Content	Well-formatted, supported files that meet CMS requirements.	File: “document.pdf”, Type: PDF, Size: 2MB
Unsupported Content	Files with formats not supported by the CMS.	File: “script.exe”, Type: Executable File
Storage Full	Simulated scenario where storage capacity is exceeded or nearly full.	Upload fails when server reports storage utilization of 100%.

Role-Restricted Action	Attempt to perform unauthorized actions (e.g., delete content as a viewer).	User Role: Viewer, Action Attempted: Delete file “example.jpg”
High Traffic Load	Simulate multiple users uploading files simultaneously to test scalability.	200 simultaneous upload requests for files of size 5MB each.
Corrupted File	Content files that are damaged, missing metadata, or contain incomplete data.	File: “image_corrupt.jpg”, Type: JPEG, Issue: Missing headers
Empty Content	Uploads where files are empty or contain no data.	File: “empty_file.txt”, Size: 0 bytes

Table Number 6.3.1 Input Scenarios

6.4. Test Procedures:

6.4.1. Validate Upload of Supported Files

- **Precondition**: CMS is operational; user has valid upload permissions.
- **Steps**:
    1. Upload a valid file (e.g., PDF, JPEG).
    2. Verify that the file is stored and accessible.
- **Expected Output**: File uploads successfully, with a confirmation message.

6.4.2. Unsupported File Types

- **Precondition**: CMS accepts only specific formats (e.g., PDF, JPEG).
- **Steps**:

1. Attempt to upload an unsupported file (e.g., .exe).
2. Verify the error message.

- **Expected Output**: Upload fails with an error: “File type not supported.”

6.4.3. Storage Full

- **Precondition**: Simulate storage nearing maximum capacity.
- **Steps**:
  1. Attempt to upload a file when storage is full.
  2. Verify the error message.
- **Expected Output**: Upload fails with an error: “Storage limit reached.”

6.4.4. Role-Based Permissions

- **Precondition**: Users have predefined roles (e.g., admin, editor, viewer).
- **Steps**:
  1. Attempt content actions for each role (upload, delete).
- **Expected Output**: Actions are restricted based on roles (e.g., viewers cannot delete).

6.4.5. High Traffic Simulation

- **Precondition**: CMS is deployed in a test environment.
- **Steps**:

1.Simulate 100+ users uploading files simultaneously.

- **Expected Output**: Response time ≤ 5 seconds with ≥ 98% success rate.

6.4.6. Corrupted File Handling

- **Precondition**: Prepare files with missing metadata or invalid format.
- **Steps**:

  1. Upload a corrupted file.

- **Expected Output**: Upload fails; an error log is created: "InvalidDataError: Corrupted file detected."

6.4.7. Empty File Upload

- **Precondition**: CMS allows file uploads.
- **Steps**:

  1. Attempt to upload an empty file (size = 0 bytes).

- **Expected Output**: Upload fails with an error: "File is empty."

6.5. Expected Outputs:

Scenario	Expected Output
Valid Content	Files are uploaded, stored, and accessible.
Unsupported Content	Upload fails with an error: "File type not supported."

Storage Full	Upload fails with an error: “Storage limit reached.”
Role-Restricted Action	Unauthorized actions fail with an error: “Access Denied.”
High Traffic Load	System handles concurrent uploads with response time ≤ 5 seconds and ≥ 98% success rate.
Corrupted File	Upload fails, and an error is logged: “InvalidDataError: Corrupted file detected.”
Empty Content	Upload fails with an error: “File is empty.”

Table Number 6.5.1 Expected Outputs

6.6. Performance Metrics:

- Average response time ≤ 5 seconds for uploads.
- Success rate ≥ 98% for valid file uploads under high traffic.
- System uptime ≥ 99.9% during testing.