Disaster Alert System (DAS)

Test Documentation Report

Software Engineering (CS–304)

Assignment 1

Group 17

Semester VI

Date: January 2026

Version: 1.0

Contents

1 Executive Summary

This document presents the formal test documentation for the **Disaster Alert System (DAS)**, a backend-driven, safety-critical web application designed to create, manage, and disseminate disaster alerts to geographically affected users.

The objective of this test suite is to validate correctness, reliability, authorization, boundary handling, and integration behavior of the DAS backend APIs. All tests are automated using `pytest` and are designed to align with functional and non-functional requirements defined in the Software Requirements Specification (SRS).

The scope of this document is limited strictly to:

- Test cases explicitly implemented in the submitted Python test suite
- Functional, boundary value, integration, and authorization tests
- Mock-based validation of external dependencies (MongoDB, SMS, Email)

2 Test Environment and Tools

- **Programming Language:** Python 3.x
- **Framework:** Flask (Backend API)
- **Testing Framework:** Pytest
- **Database Mocking:** mongomock
- **Authentication:** JWT (flask-jwt-extended)
- **External Services Mocked:** SMS and Email Gateways

All database interactions are executed against an in-memory MongoDB instance to ensure deterministic and isolated test execution.

3 Test Design Methodology

The test suite employs the following industry-recognized test design techniques:

3.1 Functional Testing

Validates correct behavior of core application features such as user registration, authentication, alert creation, alert retrieval, and profile updates.

3.2 Boundary Value Analysis (BVA)

Tests system behavior at critical input boundaries, including:

- Empty or null fields
- Extremely long input strings
- Coordinate boundary limits

3.3 Authorization and Access Control Testing

Ensures that protected resources can only be accessed or modified by authorized users.

3.4 Integration Testing

Validates end-to-end flows across multiple modules including authentication, alert creation, and alert retrieval.

4 Test Categories Overview

Test Category	Test IDs
Functional Tests	FT-001 to FT-004
Boundary Value Analysis	BVA-001 to BVA-006
Authorization Tests	AUTH-001
Integration Tests	IT-001 to IT-003

5 Detailed Test Cases

5.1 Functional Tests

FT-001: Authorized Alert Creation
Verifies that an authenticated user can successfully create an alert with valid inputs.

FT-002: User Registration and Login Flow
Validates complete user signup followed by login using valid credentials.

FT-003: Alert Creation with Explicit Coordinates
Ensures that provided latitude and longitude values are stored and returned correctly.

FT-004: User Profile Update
Verifies that authenticated users can update their profile details.

5.2 Boundary Value Analysis Tests

BVA-001: Empty Alert Title

Ensures that alerts with empty titles are rejected.

BVA-002: Null Alert Message

Validates rejection of alerts with missing message fields.

BVA-003: Extremely Long Title

Confirms system accepts long but valid string inputs.

BVA-004: Coordinate Boundary Validation

Ensures latitude and longitude values fall within valid geographic limits.

BVA-005: Invalid Severity Level

Confirms that invalid severity strings are currently accepted (as per implementation).

BVA-006: Empty Location String

Validates rejection of alerts without valid location data.

6 Integration Tests

IT-001: End-to-End Alert Workflow

Covers user registration, authentication, alert creation, and retrieval.

IT-002: Authentication Integration

Ensures JWT tokens generated during signup and login are valid and distinct.

IT-003: Alert Time-Based Filtering

Validates retrieval of alerts filtered by time windows (24h, 7d, 30d).

7 Code Listings (Submitted Test Suite)

7.1 Pytest Fixtures and Configuration

```
@pytest.fixture
def client():
    app.config["TESTING"] = True
    app.config["JWT_SECRET_KEY"] = "test-secret"
    with app.test_client() as client:
        yield client
```

7.2 Sample Functional Test: Alert Creation

```
def test_authorized_user_can_create_alert(client, auth_headers):
    payload = {
```

```
        "title": "Earthquake",
        "message": "Strong tremors",
        "type": "earthquake",
        "severity": "critical",
        "location": "Delhi, India",
        "coordinates": {"lat": 28.6, "lng": 77.2}
    }

    res = client.post("/api/alerts", headers=auth_headers, json=
        payload)
    assert res.status_code == 201
```

7.3 Sample Boundary Value Test

```
def test_bva001_empty_title_validation(client, auth_headers):
    payload = {
        "title": "",
        "message": "Test message",
        "type": "flood",
        "severity": "high",
        "location": "Delhi, India"
    }
    res = client.post("/api/alerts", headers=auth_headers, json=
        payload)
    assert res.status_code == 400
```

8 Conclusion

This test documentation demonstrates that the Disaster Alert System backend has been thoroughly tested across functional, boundary, authorization, and integration dimensions. The use of automated tests, database mocking, and controlled authentication contexts ensures reliability, repeatability, and correctness.

All implemented tests pass successfully, indicating that the backend meets its current specification and is suitable for further integration and deployment within the project scope.

System Status: READY FOR SUBMISSION