1) Unit Tests

a) API tests

Sr.No	Test Scenario	Pre-Condition	Steps to Reproduce	Expected Behavior
1	Test GET /disasters endpoint	API server running	1. Send a GET request to /disasters	Returns status code 200 and a valid JSON response
2	Test POST /alerts with missing data	API expects JSON payload	1. Send a POST request to /alerts with missing fields in the payload	Returns status code 422 with a validation error message
3	Test response structure of /disasters	Endpoint returns JSON array	1. Send GET request to /disasters2. Parse the JSON response	Each item in the response includes type, location, source, timestamp keys
4	Test rate limiting (if implemented)	Rate limit config enabled	1. Send many requests rapidly to any endpoint (e.g., /disasters)	Returns status code 429 after exceeding the allowed limit
5	Test invalid route handling	N/A	1. Send a GET request to an undefined endpoint like /unknown	Returns status code 404 and a “Not Found” error message

b) Processing Module (Python) tests

Sr.No	Test Scenarios	Pre-Conditions	Steps to Reproduce	Expected Behavior
1	Normalize and clean raw text input	Raw input with noise	Call text cleaner function	Returns lowercase, punctuation-free text
2	Extract location entities from input text	Text with named locations	Run entity recognition module	Identifies and returns location names
3	Categorize event type from context	Keywords in input data	Call classification function	Returns relevant category label
4	Detect presence of numerical measurements	Text contains numbers like "5.6 magnitude"	Run numeric extractor	Returns list of identified numerical values

c) Dashboard tests

Sr.No	Test Scenario	Pre-Condition	Steps to Reproduce	Expected Behavior
1	Test rendering with no data	Streamlit app runs	1. Start app with an empty dataset 2. Observe main UI section	A friendly message like “No data available” is shown; app does not crash
2	Test rendering with mocked data	Mocked input injected	1. Mock or load sample disaster data 2. Run the app	UI displays disaster cards/lists with correct content
3	Test component visibility toggle	Toggle button exists	1. Run app 2. Click visibility toggle for a section (e.g., filters/sidebar)	The selected UI component appears or disappears appropriately
4	Test real-time refresh behavior	Refresh logic implemented	1. Inject new data 2. Trigger st.experimental_rerun()	Dashboard reloads and displays updated data
5	Test filtering by disaster type	Filtering dropdown exists	1. Select a specific disaster type from dropdown (e.g., “Flood”)	Only disasters of the selected type are displayed in the UI
6	Filter outdated events from Dashboard	Dataset with timestamps	Run time filter function	Returns only recent events

2) Integration Tests

Sr.No	Test Scenarios	Pre-Conditions	Steps to Reproduce	Expected Behavior
1	Integration of NewsAPI fetching with categorization module	NewsAPI API key configured, valid query parameters	1. Trigger news fetch for disaster-related news 2. Pass retrieved news articles to categorization module	News articles fetched and processed, relevant disaster categories assigned correctly
2	Combined data processing from diff api's etc	Correct output from all the api's	1. Simultaneously fetch data from diff api's 2. Merge and process the combined data stream	Combined data from diff api's is aggregated, duplicates removed, and categorization applied correctly
3	Processed data feeding into Streamlit front-end	Streamlit app running, processed data available	1. Start Streamlit app 2. Push processed and categorized data into Streamlit for display	Streamlit interface updates dynamically to show latest disaster info from diff api's
4	Error handling when one or more Api's fail	API rate limits or connectivity issues occur	1. Simulate API failure or rate limit from api's. Trigger data fetch or streaming	System handles errors gracefully, retries or shows appropriate error message without crashing