TEST PLAN FOR
Forewarning System About Natural Calamities

AKSSHAT GOVIND
AISHWARYA GUPTA
AVIRAL KATIYAR
CHIRAG TYAGI

TABLE

1.Introduction

This document presents a comprehensive framework for testing the **Forewarning System About Natural Calamities**, with a current focus on **flood prediction**. It outlines structured testing strategies and methodologies to evaluate the system's ability to **accurately and reliably forecast flood risks** using meteorological and environmental data.

The plan emphasizes validating both **functional** and **non-functional** requirements to ensure robust performance under various environmental and data conditions. Testing phases—including **unit, integration, system, and acceptance testing**—are meticulously structured to verify each module's functionality.

For instance, **unit testing** will assess core components such as the **threshold-based algorithm** (used for detecting critical water level values) and the **Random Forest Classifier** (used for predictive analysis based on historical and real-time data). **Integration testing** ensures smooth communication between data collection, preprocessing, prediction, and alert modules.

System testing simulates real-world flood scenarios, validating that the system can consistently interpret input variables like rainfall, water level, and humidity to trigger appropriate alerts. **Acceptance testing** ensures the system aligns with the needs of **disaster management authorities**, **local governance**, and **rescue teams**.

Resource allocation includes tools and simulated environments that mimic natural conditions (e.g., synthetic rainfall data, dam level readings, and sensor input), supporting realistic and scalable testing. The ultimate goal is to deliver a solution that is **accurate, timely, user-friendly, and scalable** for wider adoption in disaster risk reduction.

1.1 Scope

1.1.1 In Scope

The following functionalities and components will be tested:

- Accuracy of threshold-based flood level detection.
- Random Forest model performance for flood prediction.
- Sensor data ingestion and preprocessing pipeline.

- Alert triggering and notification module.
- System performance under varying data loads and environmental conditions.
- User interface and alert visualization (UI/UX).

1.1.2 Out of Scope

The following aspects will not be tested:

- Real-time hardware sensor integration (mocked/simulated data only).
- Prediction of other natural disasters (e.g., earthquakes, wildfires) - currently out of scope.
- Third-party satellite or geospatial API reliability.

1.2 Quality Objective

- Ensure accurate and timely prediction of potential flood events.
- Validate the complete flow from data acquisition to alert generation.
- Ensure the system is resilient, reliable, and responsive under different data loads.
- Identify and resolve critical bugs or anomalies before deployment.
- Deliver a scalable and intuitive solution that can be expanded for future disaster types.

1.3 Roles and Responsibilities

- Test Manager – Dr. Kalpana Sagar
- Developers – Aviral Katiyar, Aksshat Govind, Aishwarya Gupta
- ML algo Developers – Aksshat Govind, Aishwarya Gupta
- Installation Team - Aksshat Govind, Aishwarya Gupta, Aviral Katiyar, Chirag Tyagi

2.Test Methodology

2.1 Overview

The **Agile methodology** has been adopted for the testing lifecycle of the flood forewarning system due to its **iterative development cycle**, **responsiveness to changes**, and **focus on continuous integration and testing**. Agile's flexibility aligns well with the evolving nature of data-driven models, especially for applications like flood prediction, which require frequent updates and refinements to adapt to new patterns in environmental data.

2.2 Test Levels

1.Unit Testing

Unit testing validates the smallest functional parts of the system in isolation. For the flood prediction system, this includes:

- The **Threshold-based classification function**, which labels input data as "Low," "Moderate," or "High" risk based on predefined percentile thresholds.
- **Random Forest classification model components**, ensuring each tree contributes correctly to the final prediction.
- Internal functions such as data normalization, missing value imputation, and feature selection.

Edge cases, such as extreme outliers or missing data, are a primary focus in this phase to ensure robustness.

2. Integration Testing:

Integration testing verifies the correct interaction between modules:

- Validates **data pipeline integration** from environmental datasets (rainfall, water levels, humidity, etc.) into the prediction engine.
- Confirms that **feature extraction** and **risk classification** modules communicate correctly.
- Ensures proper **passing of hyperparameters** and **training configurations** during model fitting and evaluation.

This phase helps identify any mismatches in data format, flow, and logic between connected components.

3. System Testing:

System testing evaluates the complete workflow:

- From data input (CSV or sensor simulation) → preprocessing → prediction → alert generation.
- Validates the **system's end-to-end behavior** under simulated real-world scenarios such as high rainfall or dam overflow.

Stress testing is also conducted to observe how the system handles **heavy data loads** or **sensor noise**. This step ensures the system is production-ready.

4. Acceptance Testing:

Acceptance testing confirms that the system fulfills the **user requirements** and **stakeholder expectations**:

- Alerts are evaluated by domain experts (disaster response professionals) for timeliness and clarity.
- Test scenarios simulate emergency situations to determine whether users receive actionable predictions.
- Feedback is used to validate whether the system meets **functional**, **performance**, and **usability** goals.

2.3 Test Completeness

- **100% test case coverage** for all defined functionalities.
- All **manual and automated test cases executed** using mock datasets, boundary values, and extreme environmental conditions.
- All **critical bugs are fixed**; minor bugs are logged for resolution in upcoming iterations.
- Continuous integration is used to ensure updated model versions are thoroughly tested before deployment.

2.4 Testing Techniques and Test Cases Used

The system was tested using a mix of **boundary value analysis**, **equivalence partitioning**, and **hyperparameter tuning tests**. Below is the summary table:

TABLE I: TEST CASES

Test Suite	Input Parameter	Boundary Value	Test Case Description	Actual Output	Expected Output	Status

Feature Importance	A_GH_ft' values for threshold classification	Values exactly at the 5th and 95th percentiles	Validate classification when a value equals minThreshold (5th percentile) and maxThreshold (95th percentile)	High	Classified as "High"	Pass
	A_GH_ft' values in a constant dataset	All rows have the same constant value (e.g., 10)	Evaluate threshold calculation when all values are identical	Moderate	Classified as "Moderate"	Pass
	A_GH_ft' values with an extreme outlier	One value significantly higher than others	Test the effect of an extreme outlier on threshold calculation and classification	High	Classified as "High"	Pass
TBA Boundaries	Classification function input	minThreshold < x < maxThreshold	Test classify_severity function in between boundary limits	Moderate	Classified as "Moderate"	Pass
	Classification function input	x== minThreshold or x== maxThreshold	Test classify_severity function with values exactly at boundary limits	Moderate	Classified as "Moderate"	Pass
	Classification function input	x < minThreshold	Test classification for a value slightly below minThreshold	Low	Classified as "Low"	Pass

	Classification function input	x > maxThreshold	Test classification for a value slightly above maxThreshold	High	Classified as "High"	Pass
Hyper Parameter Tuning	GridSearchCV parameter: n_estimators	n_estimators = 50	Verify that GridSearchCV explores the lower boundary value of n_estimators	Best parameters found lie within grid (e.g., n_estimators: 100)	Returns the Best Parameter	Fail
	GridSearchCV parameter: n_estimators	n_estimators = 100	Verify that GridSearchCV explores the mid boundary value of n_estimators	Best parameters found lie within grid (e.g., n_estimators: 100)	Returns the Best Parameter	Pass
	GridSearchCV parameter: n_estimators	n_estimators = 200	Verify that GridSearchCV explores the upper boundary value of n_estimators	Best parameters found lie within grid (e.g., n_estimators: 100)	Returns the Best Parameter	Fail
	GridSearchCV parameter: max_depth	max_depth = 5	Verify that GridSearchCV explores the lower boundary value of max_depth	Best parameters found lie within grid (e.g., max_depth: 10)	Returns the Best Parameter	Fail
	GridSearchCV parameter: max_depth	max_depth = 10	Verify that GridSearchCV explores the mid boundary value of max_depth	Best parameters found lie within grid (e.g., max_depth: 10)	Returns the Best Parameter	Pass
	GridSearchCV parameter: max_depth	max_depth = 15	Verify that GridSearchCV explores the upper boundary value of max_depth	Best parameters found lie within grid (e.g., max_depth: 10)	Returns the Best Parameter	Fail
	GridSearchCV parameter: min_samples_split	min_samples_split = 2	Verify that GridSearchCV explores the lower boundary value of min_samples_split	Best parameters found lie within grid (e.g., min_samples_split: 2)	Returns the Best Parameter	Pass
	GridSearchCV parameter: min_samples_split	min_samples_split = 5	Verify that GridSearchCV explores the mid boundary value of min_samples_split	Best parameters found lie within grid (e.g., min_samples_split: 2)	Returns the Best Parameter	Fail
	GridSearchCV parameter: min_samples_split	min_samples_split = 10	Verify that GridSearchCV explores the upper boundary value of min_samples_split	Best parameters found lie within grid (e.g., min_samples_split: 2)	Returns the Best Parameter	Fail

3.Test Deliverables

- **Test Plan Document**
- **Test Cases**
- **Requirement Traceability Matrix (RTM)**
- **Bug Reports**

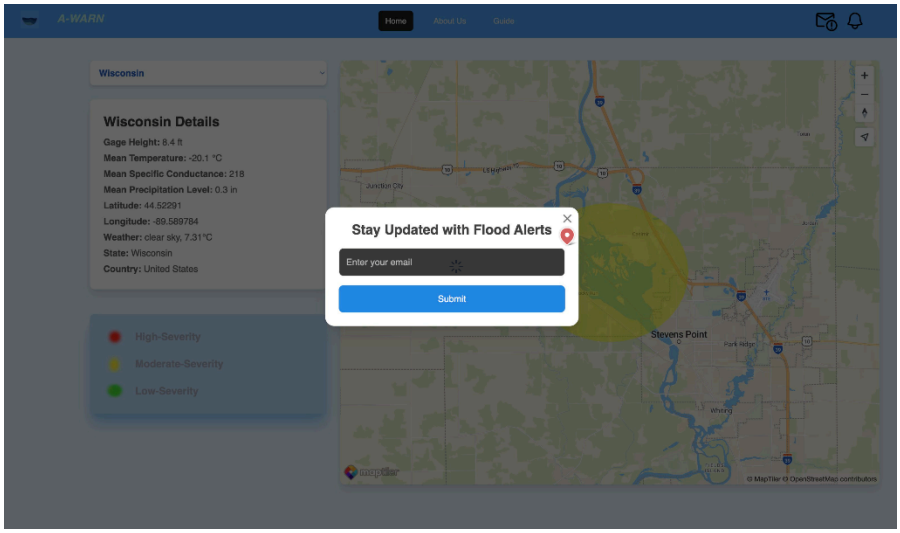
TEST CASES

Boundary Value Analysis (BVA) :

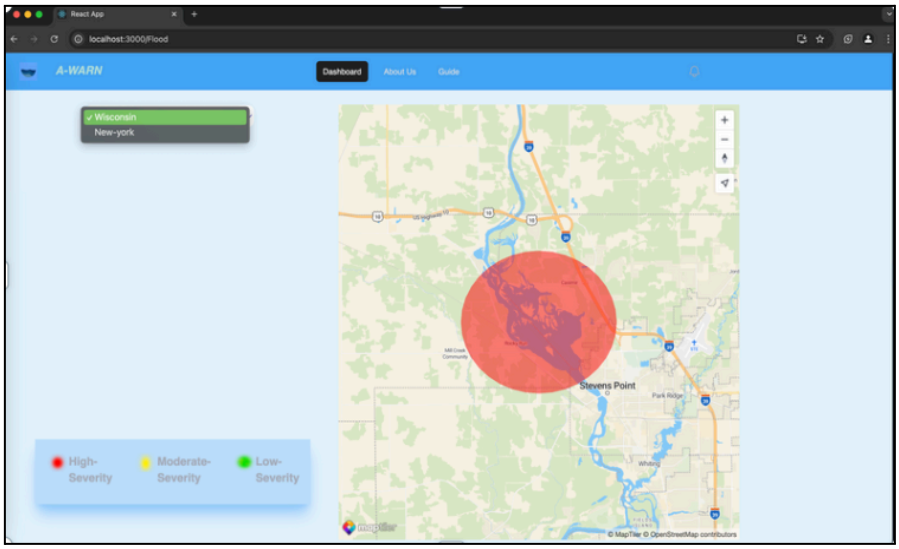
Test Suite	Input Parameter	Boundary Value	Test Case Description	Actual Output	Expected Output	Status
Hyperparameter Tuning	n_estimators = 50	Lower boundary for n_estimators	Verifies exploration of lower boundary	Best parameter: 100	Returns Best Parameter	Fail
Hyperparameter Tuning	n_estimators = 100	Mid boundary for n_estimators	Verifies exploration of mid boundary	Best parameter: 100	Returns Best Parameter	Pass
Hyperparameter Tuning	n_estimators = 200	Upper boundary for n_estimators	Verifies exploration of upper boundary	Best parameter: 100	Returns Best Parameter	Fail
Hyperparameter Tuning	max_depth = 5	Lower boundary for max_depth	Verifies exploration of lower depth	Best parameter: 10	Returns Best Parameter	Fail
Hyperparameter Tuning	max_depth = 10	Mid boundary for max_depth	Verifies exploration of mid depth	Best parameter: 10	Returns Best Parameter	Pass
Hyperparameter Tuning	max_depth = 15	Upper boundary for max_depth	Verifies exploration of upper depth	Best parameter: 10	Returns Best Parameter	Fail
Hyperparameter Tuning	min_samples_split = 2	Lower boundary for min_samples_split	Verifies exploration of lower split	Best parameter: 2	Returns Best Parameter	Pass
Hyperparameter Tuning	min_samples_split = 5	Mid boundary for min_samples_split	Verifies exploration of mid split	Best parameter: 2	Returns Best Parameter	Fail
Hyperparameter Tuning	min_samples_split = 10	Upper boundary for min_samples_split	Verifies exploration of upper split	Best parameter: 2	Returns Best Parameter	Fail

Decision Tree Boundary Tests :

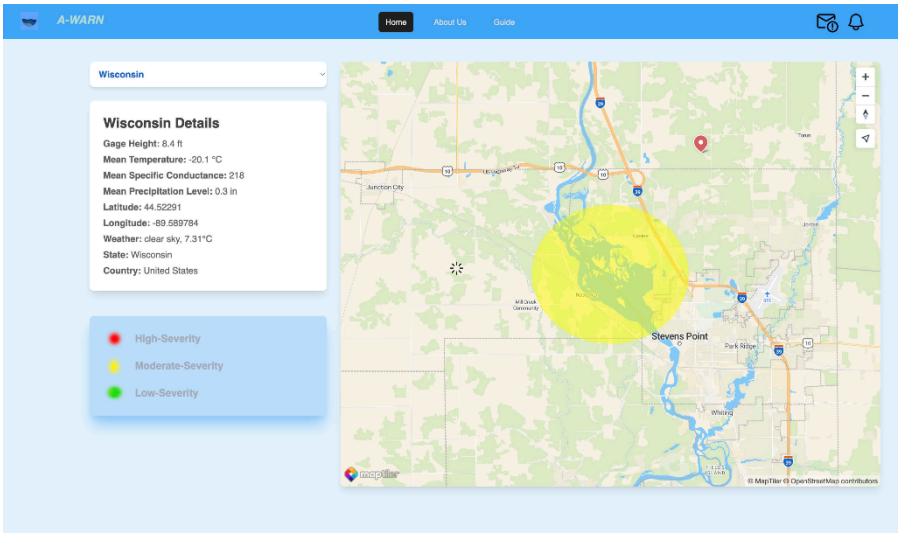
Test Suite	Input Parameter	Boundary Value	Test Case Description	Actual Output	Expected Output	Status
Feature Importance	Gauge Height values for threshold class	Values at 5th and 95th percentiles	Validate classification when a value equ	High	Classified as 'High'	Pass
Feature Importance	Gauge Height in a constant dataset	All rows have the same value (e.g., 10)	Evaluate threshold calculation when all	Moderate	Classified as 'Moderate'	Pass
Feature Importance	Gauge Height with extreme outlier	One value significantly higher than othe	Test effect of extreme outlier on thresho	High	Classified as 'High'	Pass
TBA Boundaries	Rainfall classification input	minThreshold < x < maxThreshold	Test classification in between boundary	Moderate	Classified as 'Moderate'	Pass
TBA Boundaries	Rainfall classification input	x == minThreshold or x == maxThresh	Test classification exactly at boundaries	Moderate	Classified as 'Moderate'	Pass
TBA Boundaries	Rainfall classification input	x < minThreshold	Test classification for value below minTh	Low	Classified as 'Low'	Pass
TBA Boundaries	Rainfall classification input	x > maxThreshold	Test classification for value above maxT	High	Classified as 'High'	Pass



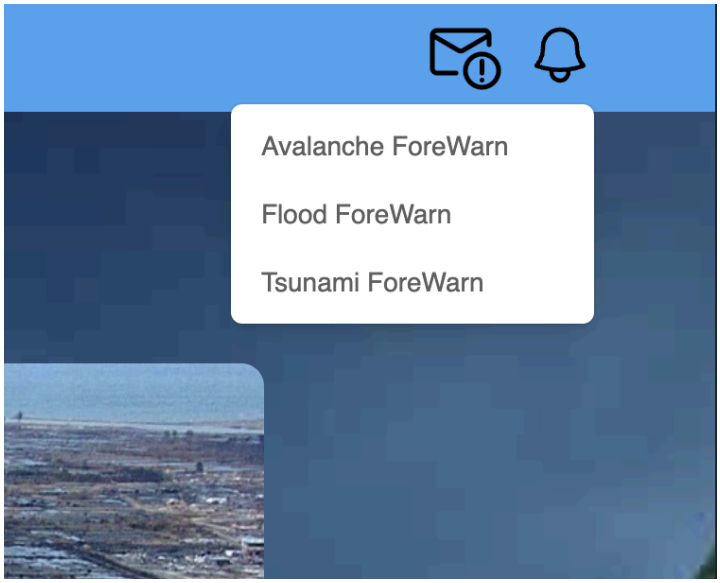
ALERT POPUP



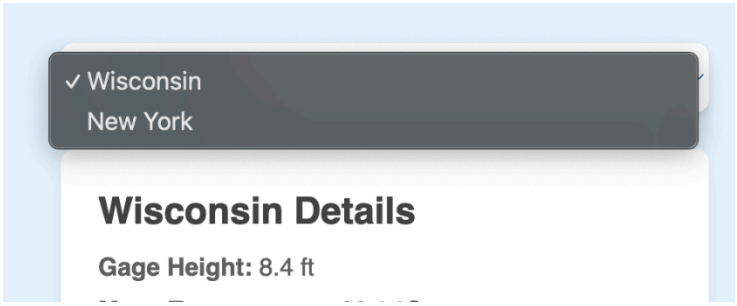
VULNERABILITY MAP (a)



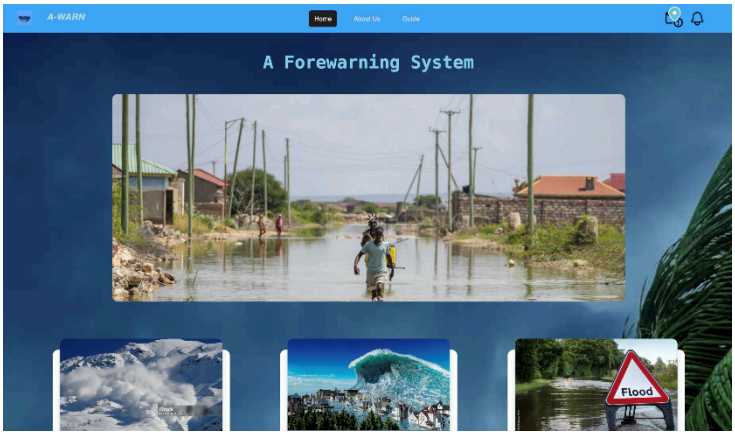
VULNERABILITY MAP (b)



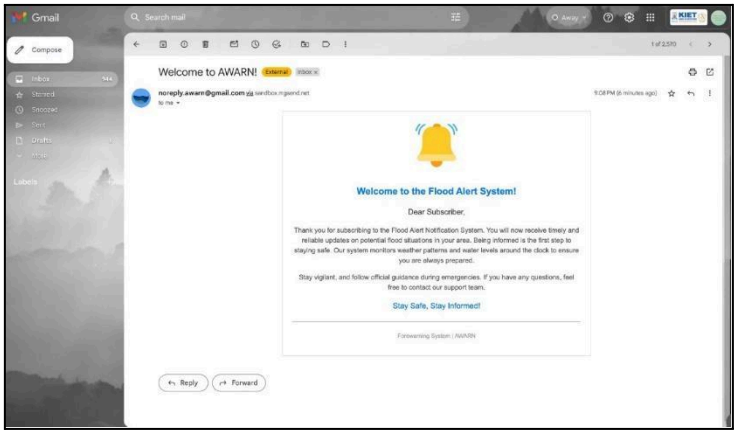
BELL ICON

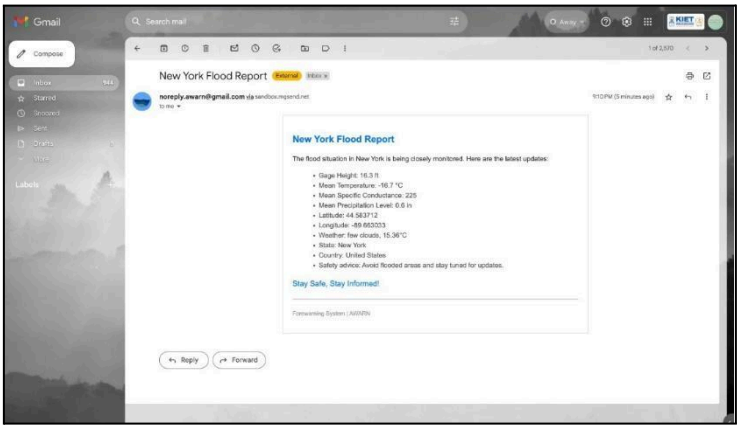


LIST ITEM MENU



LANDING PAGE





EMAIL REPORTS AND
SUBSCRIPTION MESSAGES

MODEL TEST CASES

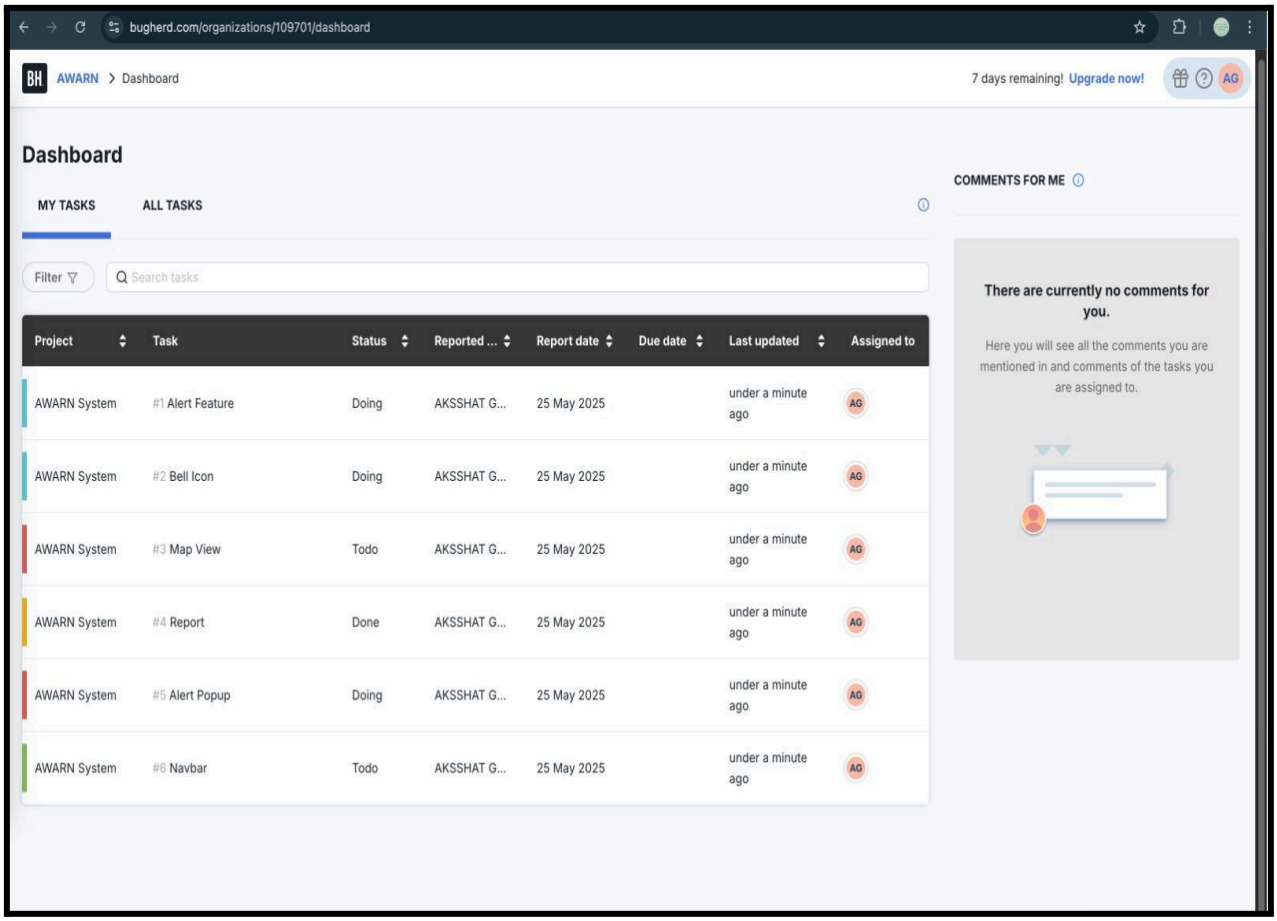
Requirement Traceability Matrix (RTM)

Forewarning System About Natural Calamities

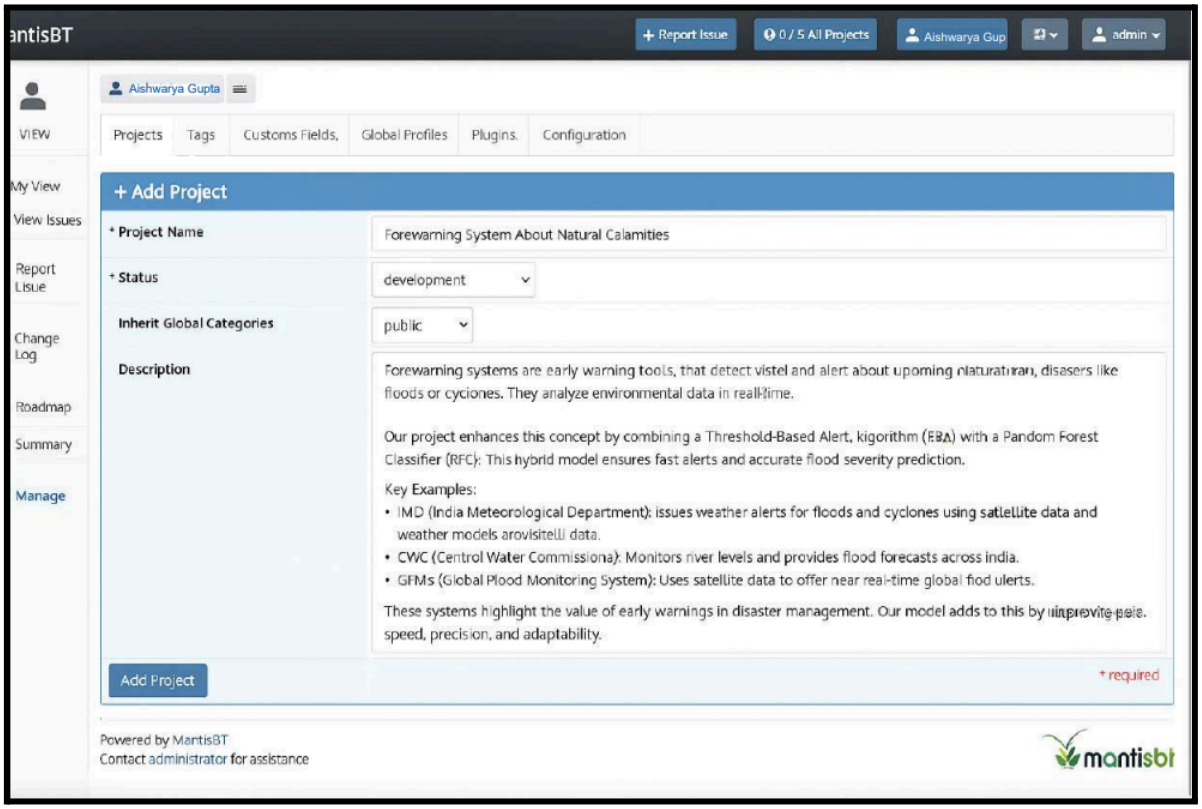
Serial Number	Test Id	Input	Actual Output	Expected Output	Result
1	T1	Alert Credentials	Alert Success	Alert Success	Pass
2	T2	Mail Credentials	Mail Fail	Mail Fail	Pass
3	T3	Alert, UserName Not Provided	Alert Fail	Alert Fail	Pass
4	T4	Alert, UserName or Password wrong	Alert Fail	Alert Fail	Pass
5	TBA	minThreshold < x < maxThreshold	Moderate	Moderate	Pass
6	RFC	If value >= maxThreshold	High	High	Pass
7	RFC	If value < minThreshold	Low	Low	Pass
8	RFC	If maxThreshold == minThreshold	High	Low	Fail
9	RFC	Verify Alert Generation	Alert Generated	None Alert Generated	Fail
10	RFC	If Alert == True	Timely Action	Timely Action Taken	Pass

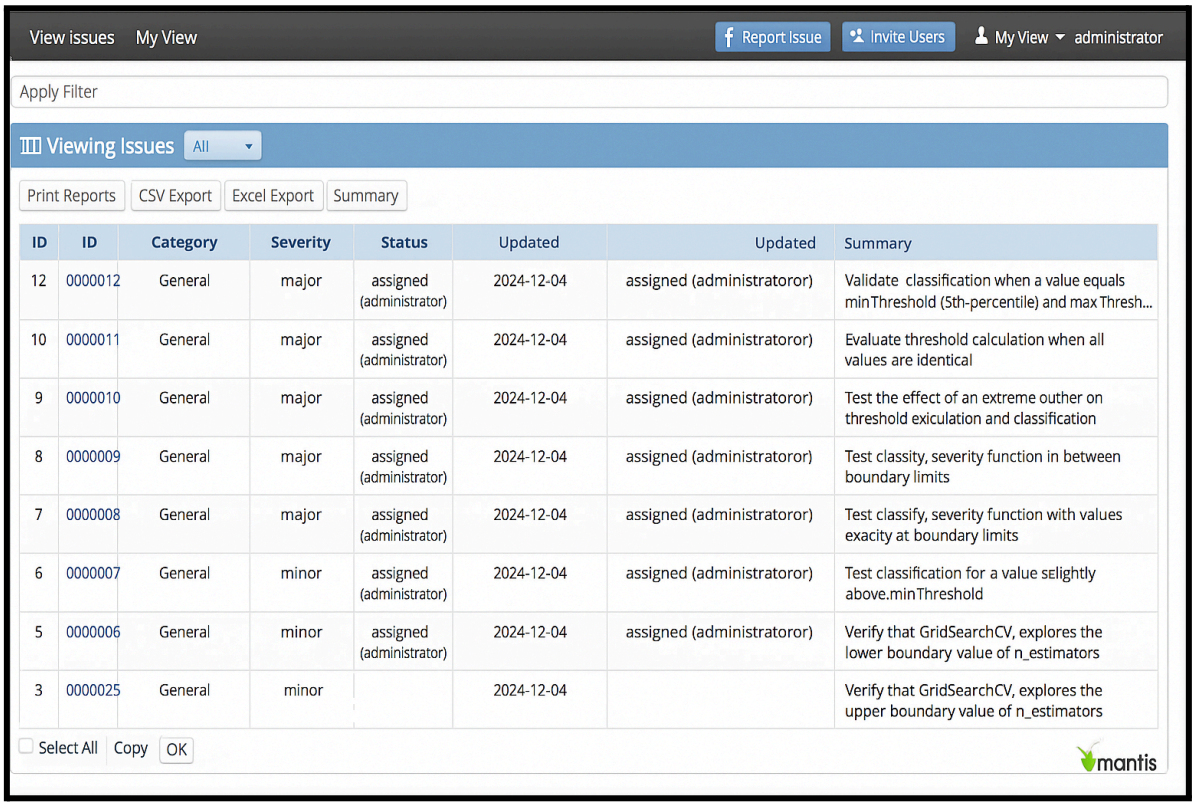
BUG REPORT

Bugherd(UI Testing)



MantisBT(Functionality Testing)





4.Resource & Environment Needs

4.1 Testing Tools

Bug Tracking Tool: MantisBT

Mantis Bug Tracker (MantisBT) is an open-source, web-based bug tracking system. It provides project managers and developers with an organized way to manage software defects and improvements.

Why Use MantisBT?

- Cost-Effective: Open-source and free.
- Scalable: Suitable for small to large projects.
- Community Support: Strong community contributions ensure frequent updates and new features.
- Easy Deployment: Quick setup and simple to host.

4.2 Test Environment

- **Operating System:** Windows 10 and above
- **Database:** MySQL
- **Web Server:** Apache
- **Development Frameworks:** Python, Scikit-Learn, TensorFlow
- **Additional Software:** Microsoft Office 2013+, Google Chrome, Visual Studio Code

5.Terms/Acronyms

TERM/ACRONYM	DEFINITION
API	Application Program Interface

TERM/ACRONYM	DEFINITION
AUT	Application Under Test
RFC	Random Forest Classifier
TBA	Threshold Based Alert
RTM	Requirement Traceability Matrix