COMSATS University Islamabad (CUI)

Software Requirement Specification

for

Weather Disaster and Early Warning Application

Version 1.0

By

Faizan Sharif CIIT/FA21-BSE-007/VHR

Salman Sharif CIIT/FA21-BSE-056/VHR

Supervisor

Kalim Sattar

Bachelor of Science in Computer Science

(2021-2025)

Revision History

Name	Date	Reason for Changes	Version

Application Evaluation History

Comments (by committee) *include the ones given at scope time both in doc and presentation	**Action Taken**

Supervised By

Kalim Sattars

Signature

Abstract

The Weather Disaster and Early Warning Application enables users to get current weather information for cyclones, tornadoes, heavy rains, and floods amongst others. It targets the risky regions, providing reliable information and prediction for the purpose of improving the disaster response. Developed with Flutter and Python, the app provides personalized alerts, storm chasing, and flood notices using factual information from CMA(China Meteorological Administration) Datasets and OpenWeatherMap. The purpose is to assist the users in initiating appropriate actions and minimizing the hazards resulting from such climate changes.

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 Project Scope

The Weather Disaster and Early Warning Application is designed to address the provision of alerts and forecasts of hazardous weather events, namely cyclones, tornadoes, heavy rain, and floods. The application will be useful to users in high-risk zones, enabling them to receive up-to-date information.

Key components of the project scope include:

- Cyclone tracking in real-time, with high and accurate forecasts of storm surges or identification of areas prone to red zone effects.
- Precipitation alerts, including floods and heavy rain, with real-time precipitation information.

- Selective, place-specific notifications for various types of weather conditions.
- An easy-to-navigate UI for mobile.

Some existing integrations with other sources include CMA and OpenWeatherMap.

1.2 Project Significance

The role of the Weather Disaster and Early Warning Application is crucial in supplementing safety and disaster preparations in areas prone to climate disasters such as cyclones, excessive rains, and floods. Its significance lies in:

1. **Improved Early Warning System:** With real-time, location-based alerts, the application gives users ample time to prevent the occurrence of disasters, thereby reducing deaths and losses.
2. **Filling a Market Gap:** Most current weather applications lack specificity and accuracy in providing information about extreme weather conditions. This application fills that gap by offering specialized alerts and forecasts for life-threatening scenarios.
3. **Risk Reduction:** With timely, accurate, geolocation-based tracking and early alerts, users can make informed decisions, such as leaving the affected area or preparing for floods, thus reducing the impact of natural disasters.

4. **Customizable Alerts:** The location-based notification system increases relevance for users, ensuring they receive time-sensitive information to take necessary action when hazardous weather is present.

1.3 Target Audience

The Weather Disaster and Early Warning Application is designed for:

1. **Residents in High-Risk Areas:** Individuals residing in cyclone-prone, tornado, torrential rain, and flood zones.
2. **Emergency Responders:** Individuals dependent on timely alarms or sirens, including local authorities, disaster management, and first responders.
3. **Travelers and Tourists:** People traveling to regions where there are high risks, allowing them to seek safety when the weather is unfavorable. .

1.4 Project Timeline

The development of the Weather Disaster and Early Warning Application is expected to follow these extended phases:

1. **Phase 1: Requirements Collection and Research (Weeks 1–6)**
    - Finalizing detailed functional and non-functional requirements.
    - Identifying stakeholders, data providers (e.g., CMA), and potential integration partners.
    - Conducting feasibility studies and risk assessments.
2. **Phase 2: Design and Planning (Weeks 7–14)**
    - Creating high-fidelity UI/UX designs for mobile and web platforms.
    - Finalizing system architecture and database schemas.
    - Defining API contracts and data processing workflows.
3. **Phase 3: Core Development (Weeks 15–38)**
    - **Frontend Development (Flutter):** Building responsive UI components for multi-platform deployment.

- **Backend Development (Python):** Creating services for weather data ingestion, alert logic, and user management.
- Iterative builds and internal testing every 4 weeks.

4. **Phase 4: Integration and System Testing (Weeks 39–46)**
   - Integrating real-time weather APIs and location services.
   - Performing unit, integration, and system testing.
   - Involving end-users in early UAT sessions.
5. **Phase 5: Pilot Launch and Evaluation (Weeks 47–52)**
   - Deploying the system to a limited audience in selected regions.
   - Monitoring performance, usability, and feedback collection.
   - Making necessary revisions based on stakeholder input.
6. **Phase 6: Full Deployment and Stabilization (Weeks 53–56)**
   - Publishing the application across all target platforms (iOS, Android, Web).

- Providing user onboarding support and tutorials.

7. **Phase 7: Maintenance, Monitoring, and Feature Expansion (Weeks 57–62)**
   - Routine bug fixes, performance optimizations, and security patches.
   - Planning for version 2.0 feature enhancements based on feedback.
   - Establishing a roadmap for long-term support and updates.

Total Project Duration: 62 weeks.

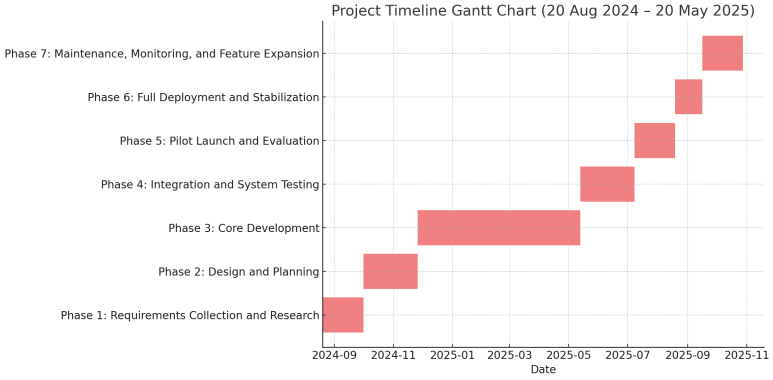

FIGURE 1.1: Gant Chart

Chapter 2

Software Requirement Specification

2.1 Software Requirements Technique

This document presents the *Software Requirement Specification (SRS)* of the Weather Disaster and Early Warning Application to deliberate on techniques and requirement to produce an effective system. The requirement collection techniques include stakeholder interviews, target audience surveys, workshop where client and SAP developers, analysts work hand in hand and use case development to ensure that the users' requirements meet with the system requirements. This category includes features which state rules as to what the product must do for it to be useful; these include real-time weather, customizable alerts, weather information, search by location, and feedback. This means that alerts on cyclone, tornado, and floods have to be available at real-time, user

notification settings, interactive weather map, and multiple location functionality. Non-functional requirements highlight time and again, response time less than 10 seconds for alerts, update on every 5 minutes, availability and fault tolerance of 99.9 %, user friendly and disabled friendly interface, horizontal scaling and cloud compatibility. Thus, the integration of these features and requirements is expected to allow the application to offer timely, accurate as well as easy-to-use disaster forecasting and alerts for its users.

2.2 Functional Requirements

The Weather Disaster and Early Warning Application is an application aimed at providing weather forecasts and disaster early warnings for weather disasters such as cyclones, tornadoes, heavy rain, floods, and the like. The following are the key functional requirements for the system:

Real-Time Weather tracking

- **Cyclone Tracking and Alerts:** The system must track cyclones in real time, providing location-based alerts, including:
  - Predicted path of the cyclone for the next 24 hours.
  - Current intensity and forecasted intensity changes.
  - Storm surge alerts for areas at risk.

- **Heavy Rain and Flood Alerts:** Users must receive alerts for:
  - Real-time rain intensity and forecasts.
  - Potential flooding risks due to heavy rainfall.
- **Tornado Alerts:** The system must provide real-time tornado warnings and expected paths based on location.

Customizable Notifications

- **Alert Customization:** Users can choose the types of alerts they want to receive (e.g., cyclone, heavy rain, floods) and set the severity threshold for notifications (e.g., high-risk events only).
- **Push Notifications:** The system must provide timely push notifications to mobile devices of users, ensuring that they are immediately informed of potential threats.

Weather Data Display

- **Weather Maps:** The system must display interactive weather maps that show:
  - Cyclone tracking and intensity.
  - Rainfall distribution and flood-prone areas.
  - Tornado movement and affected areas.

- **Forecast Summary:** Provide users with a summary of weather conditions for the next 24-48 hours for their specified location.

Search and Location Selection

- **Location Search:** Users must be able to search and select their location manually or enable automatic detection via GPS.
- **Multiple Locations:** Users can save and manage multiple locations (e.g., home, workplace) for which they want to receive weather alerts.

User Feedback

- **Feedback Mechanism:** Users must be able to provide feedback or report issues within the app (e.g., false alerts, app performance).
- **Ratings and Reviews:** The app must allow users to rate the accuracy of alerts and overall experience.

2.2.1 Use Cases

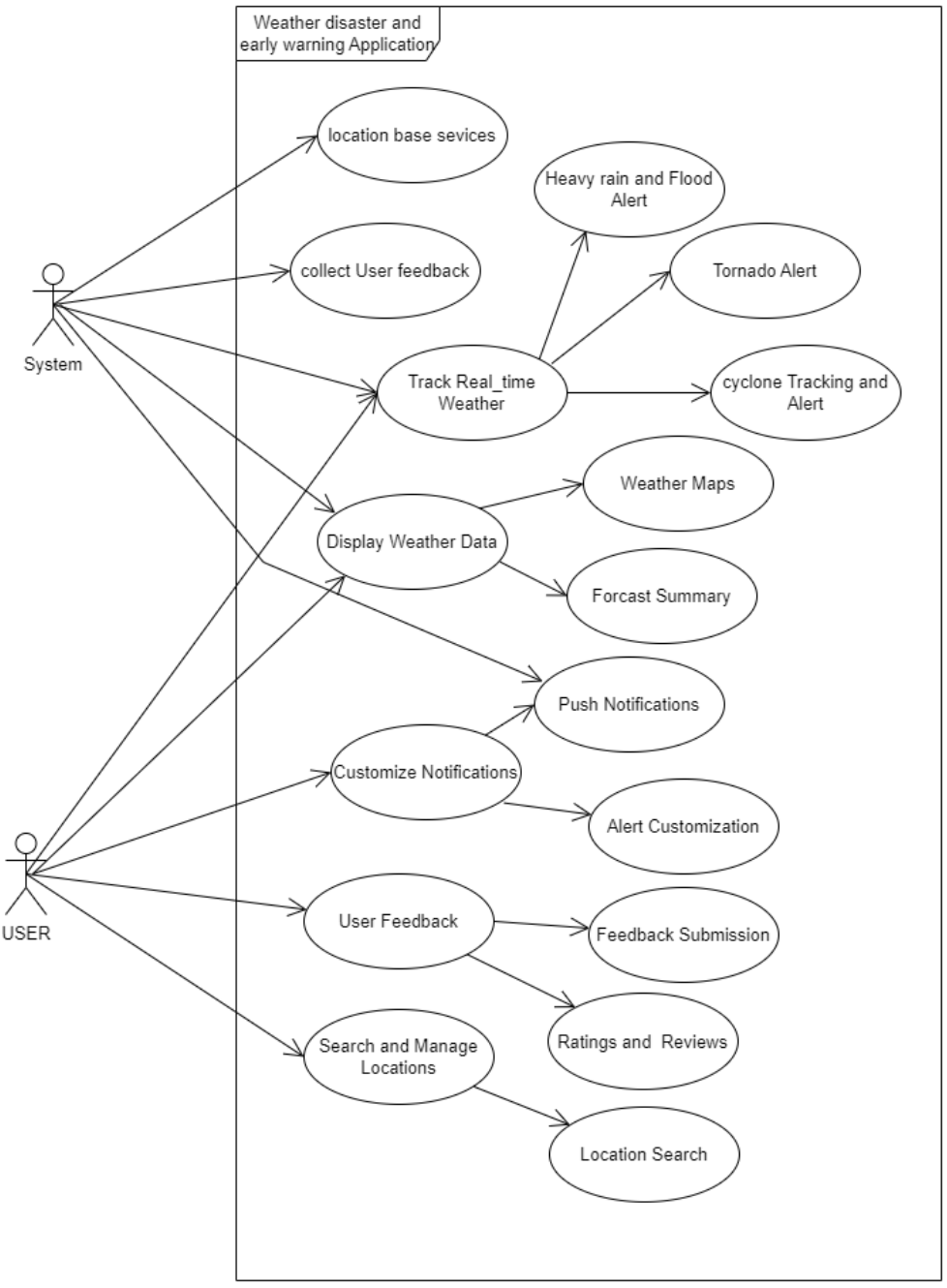

FIGURE 2.1: Usecase

2.2.2 Action Tables for Use Cases

Actor	**Event**	**Trigger**	**Use Case**	**System Response**
User	View Real-time Weather	User launches app	Track Real-time Weather	Retrieves and displays the latest weather data to the user
User	Receive Weather Alerts	Weather alerts generated based on user location	Push Notifications	Sends heavy rain, flood, tornado, and cyclone alerts based on real-time tracking
User	Customize Notifications	User chooses notification preferences	Customize Notifications	Allows user to set preferences for alerts (type, frequency, etc.)
User	Submit Feedback	User provides feedback within app	Feedback Submission	Collects and stores user feedback for future analysis
User	Search Location	User searches for a specific location	Location Search	Retrieves weather data for specified location
User	Manage Favorite Locations	User adds or removes favorite locations	Search and Manage Locations	Allows user to store, update, or delete preferred locations for quick access
User	View Weather Forecast	User accesses forecast data	Forecast Summary	Provides summarized forecast information (e.g., hourly, daily)
User	Rate and Review	User rates app or leaves review	Ratings and Reviews	Collects and stores user reviews and ratings
System	Track Real-time Weather	Continuous real-time updates	Track Real-time Weather	Continuously monitors and updates weather data, tracking potential hazards
System	Send Weather Alerts	Hazardous weather detected	Push Notifications	Sends alerts for heavy rain, flood, tornado, and cyclone events
System	Display Weather Data	Real-time or requested data update	Display Weather Data	Updates UI with current weather conditions, maps, and relevant alerts
System	Collect Feedback	Feedback submission by user	Collect User Feedback	Stores feedback to enhance user experience and system accuracy

Table 2.1: Event Table

2.3 Non Functional Requirement

The system has to guarantee high availability, and the alerts of abnormal situations should appear in 10 seconds, while the data is updated in five minutes. Other non-functional requirements include availability that should be 99.9 % and should support fault tolerance because of server or network problems usability should have clean and easy interface for visually impaired with features like large fonts, high contrast themes and compatibility with screen readers. Lastly, the system should be Scalable; the architecture should be capable of horizontal scaling and must integrate with the cloud for improving the system's functionality based on user traffic.

1. **Performance:**
    - **Response Time:**They have to get delivered to the users within 10 seconds of identifying a severe weather event.
    - **Data Refresh Rate:**Weather information should reflect the current information and therefore should be updated every five minutes.
2. **Reliability:**
    - **Availability:**Make sure that the system is always up and running, virtually 99.9% of the time, including some severe weather conditions.

- **Fault Tolerance:**Reassign work to other servers or networks while dealing with server or network problems with ease.

3. **Usability:**

- **Ease of Use:**The application should involve the use of an interface/ graphical user interface that must be basic/ easy to operate/ talent friendly.
- **Accessibility:**Some options to turn into options for the disabled are large text, high-contrast themes, and screen-reader support.

4. **Scalability:**

- **Horizontal Scalability:**Include facility for scaling horizontally in order to handle many users properly.

Chapter 3

Design

3.1 Data Flow Diagrams (DFDs)

3.1.1 Zero Level DFDs

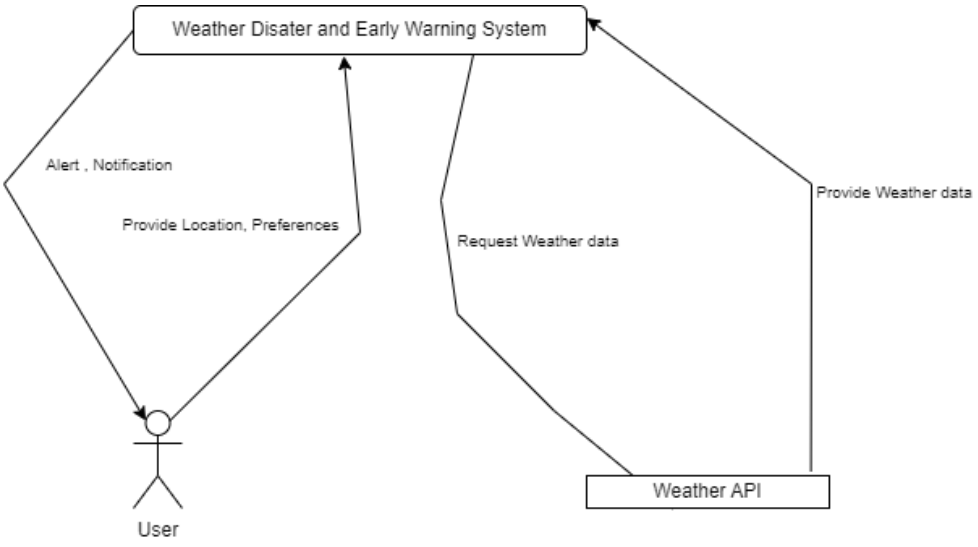

FIGURE 3.1: DFD Zero level

The following figure shows a **Zero Level Data Flow Diagram (DFD)** of a Weather Disaster and Early Warning System (WDEWS). Here's its description:

The system provides users with timely weather disaster alerts based on their location and preferences.

Actors

- **User**: The main user of the system who supplies their current location and desired preferences. For example, the system issues alerts for weather disasters considered emergencies.
- **Weather API**: An external data source that supplies the system with real-time weather data upon request.

Processes

- **Provide Location and Preferences**: The user enters their location and preferences into the system.
- **Request Weather Data**: The system sends a request to the external Weather API to fetch weather data for the user's location.
- **Provide Weather Data**: The Weather API responds with the requested weather data.
- **Alert and Notification**: Based on the received weather data, the system analyzes the information and generates the necessary alerts and notifications for the user.

Data Flow

- The user provides their location and/or preferences to the system.

- The system requests weather data from the Weather API and receives the response.
- The system processes the weather data and sends alerts and notifications back to the user.

3.1.2 First Level DFDs

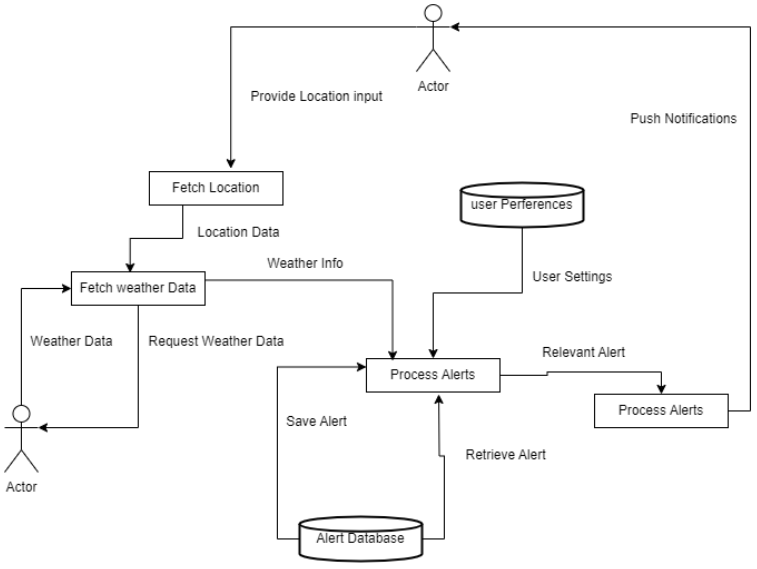

FIGURE 3.2: DFD One level

System Name This is the Weather Disaster and Early Warning System. This system's goal is to notify a user regarding various forms of weather-related disasters as preferred by the user and depending on current weather conditions.

Entities

1. **Actor (User)**:
   - Provides the system with their location data.
   - Receives appropriate push notifications concerning the weather disasters.

Processes

1. **Fetch Location**: Uses the input of a location provided by the user and converts it into location data.
2. **Fetch Weather Data**:
   - Requests weather data from the location data obtained from other weather data sources such as application programming interfaces (APIs).
   - Receives weather information as a response.
3. **Process Alerts**:
   - Accepts parameters such as weather information and user preferences as inputs.
   - Converts the data into appropriate alerts by analyzing and processing it.
   - Retains alerts in the **Alert Database** for later use.
   - Retrieves alerts from the database whenever required.
4. **Push Notifications**: Serves notifications regarding weather events noted by the user and current weather conditions.

Data Stores

1. **User Preferences**:
   - Contains user-related data in the application, such as the type of alerts or number of notifications to be provided.

- Provides this information to the **Process Alerts** system to make appropriate notifications to each user based on this data.

2. **Alert Database**:
   - Stores processed alerts for use or even for future reference.
   - Enables the system to call back saved alerts for reprocessing if necessary or for resending the alerts.

Data Flow

1. **Provide Location Input**: The user provides the system with their current location.
2. **Location Data**: Transmitted from the **Fetch Location** process and issued to the **Fetch Weather Data** process.
3. **Request Weather Data**: The system requests weather information using geographical coordinates.
4. **Weather Info**: Forwarded to the **Process Alerts** process for the required weather data.
5. **User Settings**: Observable parameters that fit the user's preferences, including alert categories or thresholds, are transferred to the **Process Alerts** process.

6. **Relevant Alert**: Specific alert messages are initiated by the system based on analyzed weather conditions and the user's preferences.

7. **Save Alert**: Alerts are stored in the **Alert Database** for later use.

8. **Retrieve Alert**: The system fetches certain alerts from the database when required.

9. **Push Notifications**: The system sends meaningful alerts back to the user.

3.1.3 Second Level DFDs

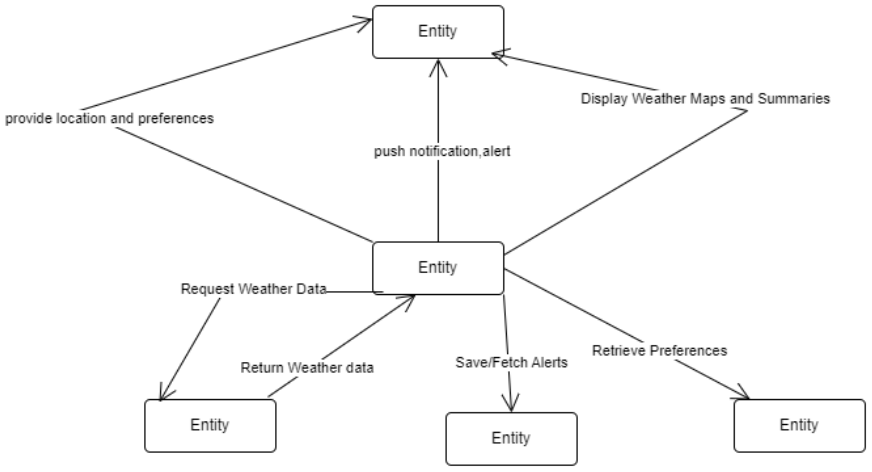

FIGURE 3.3: DFD Two level

The given diagram is a two-level Data Flow Diagram (DFD) for a Weather Disaster and Early Warning System. Here's a detailed description of the components and their interactions in the DFD:

Entities and Components

1. **User (External Entity)**:

- The input from the user includes their geographic location and the details of the alerts including the type and level of alert they want.
- The output from the system is in the form of push notifications, alerts, and weather maps/summaries to the user.

2. **Central System (Process)**:
   - Functions as a traffic control point for the application and interfaces with other entities in the processing of the data.
   - Deals with the user input on the interface, interacts with outside data sources, processes weather data, and disseminates notifications.
3. **Weather Data Source (External Entity)**:
   - Provides current weather details to the system like cyclone tracks, tornado traces, rain rates, and flood probabilities.
   - This external source includes the actual weather data since the system sends requests for the data and receives the responses.
4. **Alert Database (Data Store)**:
   - Contains records created by the system regarding particular events and locations.

- Permits the system to store or retrieve alerts for the users whenever needed.

5. **Preferences Database (Data Store)**:

- Holds all users' options such as changing of the alerts, location bookmarks, and severity levels.
- The system uses these preferences to alert users based on their preferences, as displayed in the diagram.

Data Flows

1. **User to System**:

- **Input**: Users enter location information and select delivery of alerts.
- **Output**: The system provides push notifications, alarms, and weather maps or brief interactively; they are returned to the user.

2. **System to Weather Data Source**:

- **Request Weather Data**: The system transmits a request to obtain current weather information such as cyclone tracks, rain rates, and tornadoes.
- **Return Weather Data**: The external data source replies in the form of the provided weather data.

3.2 Interaction Diagram

3.2.1 Sequence Diagrams

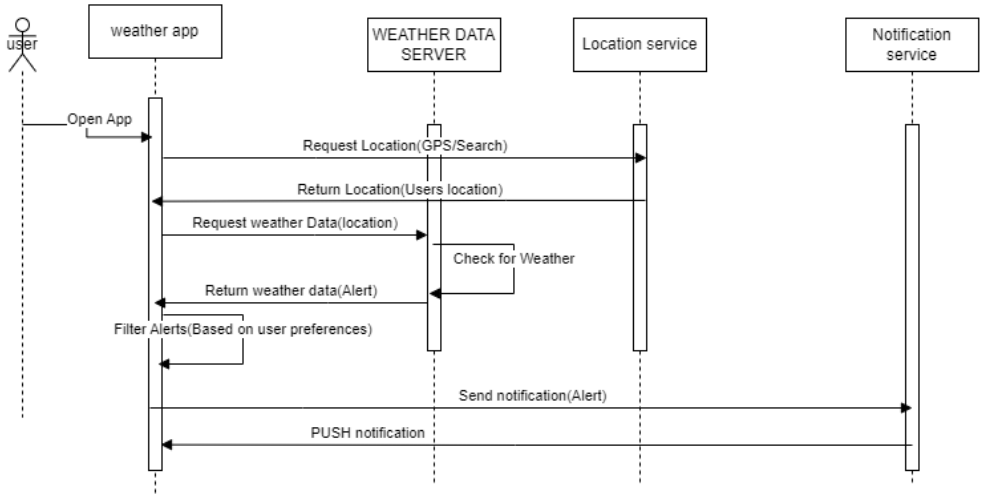

FIGURE 3.4: Sequence Diagram

Actors and Components

- **User**: The user who is communicating with the system through the user interface.
- **Weather App**: The tool that the user interacts with to operate the main application.
- **Weather Data Server**: The service responsible for pulling weather information and sending alerts.
- **Location Service**: A service to identify either the user's current or intended geographic location.
- **Notification Service**: A service to display push notifications to the user.

Sequence of Actions

(a) **Open App** The user opens the weather application, which initiates the process.

(b) **Request Location**

- The **Weather App** asks the user to enter their location using GPS or a search engine.
- This request is forwarded to the **Location Service**.

(c) **Return Location** The **Location Service** sends the current or searched location of the user to the **Weather App**.

(d) **Request Weather Data** The **Weather App** shares the location information with the **Weather Data Server** and requests weather details for that location.

(e) **Check for Weather Alerts** The **Weather Data Server** analyzes the weather conditions for the provided location and decides whether any kind of alert is required (e.g., cyclone, flood, tornado).

(f) **Return Weather Data (Alert)** If there are any alerts, the **Weather Data Server** reports them back to the **Weather App**.

(g) **Filter Alerts** The **Weather App** filters the received alerts based on the user's preferences, including severity levels and types of alerts.

(h) **Send Notification** The filtered alerts are forwarded to the **Notification Service** for delivery to the user.

(i) **Push Notification** The Notification Service provides push notifications about the relevant weather alerts to the user.

3.2.2 ERD

This figure represents an ERD of the system which focuses on the process of monitoring the weather data, alerts, and notifications to users. Below is a description of each entity and their relationships:

Entities and Attributes:

1. User

- **Attributes:**
  - **UserID (Integer):** Unique identifier for a user.
  - **Name (String):** Name of the user.
  - **Phone (String):** Contact phone number.
  - **Email (String):** Email address.

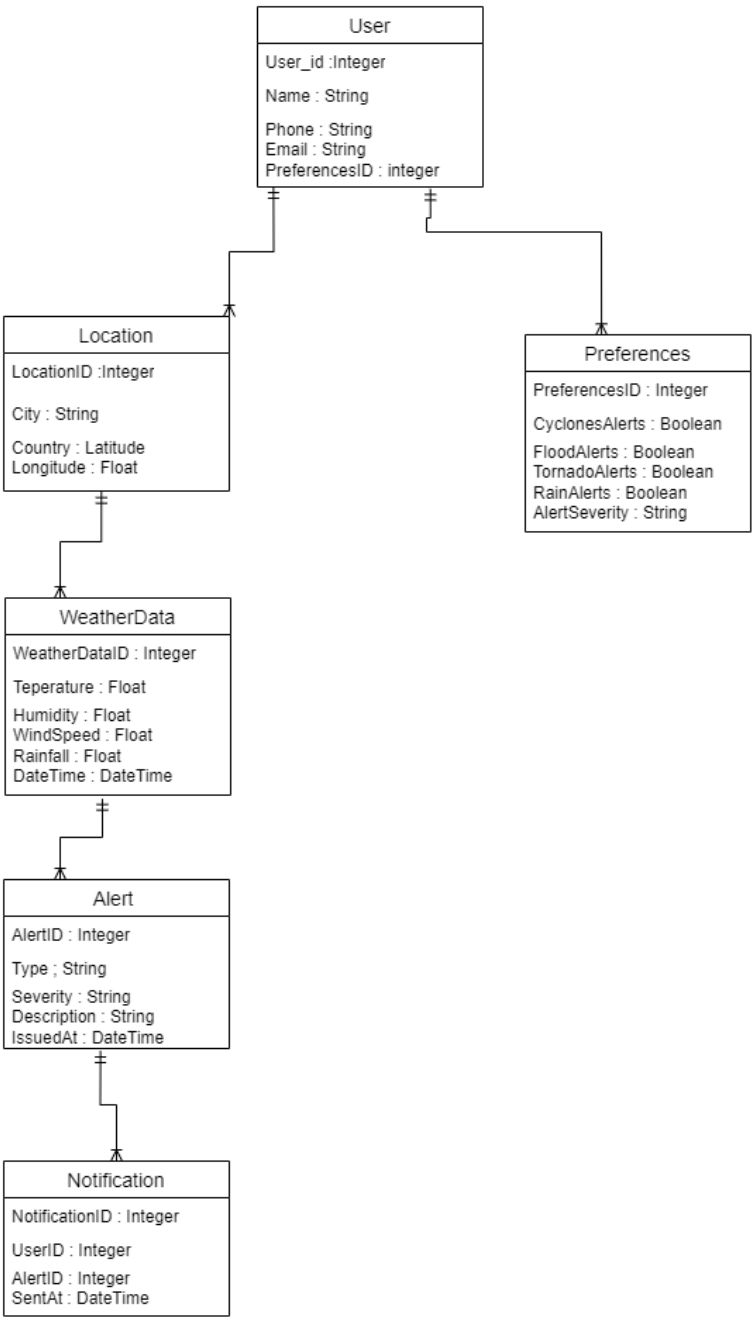

FIGURE 3.5: ERD

- **PreferencesID (Integer):** Foreign key linking to the user's preferences.

- **Description:** The user entity represents a system account, allowing users to set alert preferences.

2. **Preferences**

- **Attributes:**
  - **PreferencesID (Integer):** Unique identifier for preferences.
  - **CyclonesAlerts (Boolean):** Preference for cyclone alerts.
  - **FloodAlerts (Boolean):** Preference for flood alerts.
  - **TornadoAlerts (Boolean):** Preference for tornado alerts.
  - **RainAlerts (Boolean):** Preference for rain alerts.
  - **AlertSeverity (String):** Minimum severity level for sending alerts.
- **Description:** Stores user-specific preferences for receiving weather alerts.

3. **Location**

- **Attributes:**
  - **LocationID (Integer):** Unique identifier for a location.
  - **City (String):** Name of the city.

  - **Country (String):** Name of the country.
  - **Latitude (Float):** Latitude coordinate.
  - **Longitude (Float):** Longitude coordinate.
- **Description:** Represents geographic locations monitored for weather data.

4. **WeatherData**

- **Attributes:**
  - **WeatherDataID (Integer):** Unique identifier for weather data.
  - **Temperature (Float):** Temperature at the location.
  - **Humidity (Float):** Humidity percentage.
  - **WindSpeed (Float):** Wind speed at the location.
  - **Rainfall (Float):** Rainfall measurement.
  - **DateTime (DateTime):** Timestamp of the weather data entry.
- **Description:** Captures real-time weather information for a specific location.

5. **Alert**

- **Attributes:**
  - **AlertID (Integer):** Unique identifier for an alert.
  - **Type (String):** Type of the alert (e.g., cyclone, flood).
  - **Severity (String):** Severity level of the alert.
  - **Description (String):** Description of the alert.
  - **IssuedAt (DateTime):** Timestamp when the alert was issued.
- **Description:** Represents weather-related alerts generated based on weather data.

6. Notification

- **Attributes:**
  - **NotificationID (Integer):** Unique identifier for a notification.
  - **UserID (Integer):** Foreign key linking to the notification recipient.
  - **AlertID (Integer):** Foreign key linking to the related alert.
  - **SentAt (DateTime):** Timestamp when the notification was sent.

- **Description:** Stores notifications sent to users based on weather alerts.

Relationships:

- **User Preferences:** A one-to-one relationship where each user has a single set of preferences.
- **User Notification:** A one-to-many relationship where a user can receive multiple notifications.
- **Location Weather Data:** A one-to-many relationship where a location can have multiple weather data entries over time.
- **Alert Notification:** A one-to-many relationship where a single alert can trigger multiple notifications for different users.
- **Alert Weather Data:** Alerts are generated based on specific weather data conditions.

Chapter 4

Testing

4.1 Test Cases

TC ID	**Test Case Description**	**Input**	**Expected Result**	**Status**
TC01	User registration with valid data	Name, Email, Password	Account created successfully	Pass
TC02	User login with correct credentials	Email, Password	Login successful, redirect to home page	Pass
TC03	User login with incorrect password	Email, Wrong Password	Display error message	Pass

TC ID	**Test Case Description**	**Input**	**Expected Result**	**Status**
TC04	Fetch weather data using valid GPS location	GPS enabled	Display current weather & alerts	Pass
TC05	Fetch weather data using manual location input	City name	Display weather & alert info for that city	Pass
TC06	Display real-time alert for cyclone	Simulated cyclone condition	Push notification sent with details	Pass
TC07	Customize alert preferences	Cyclone: ON, Flood: OFF	Only cyclone alerts received	Pass
TC08	Save user location preferences	Multiple saved locations	Alerts match all saved locations	Pass
TC09	UI displays weather map and alert zones correctly	App loaded, map requested	Map with overlay displayed	Pass

TC ID	Test Case Description	Input	Expected Result	Status
TC10	App handles slow or no internet connection	Turn off network	Show network error / cached data if available	Pass
TC11	Push notification received in real-time	Trigger backend alert	Alert received within 10 seconds	Pass
TC12	Feedback form submission	Valid feedback text	Confirmation message shown, data saved	Pass
TC13	Accessibility features (high contrast, large text)	Toggle accessibility mode	UI adjusts accordingly	Pass
TC14	System updates weather data every 5 minutes	Wait for interval	New data shown without manual refresh	Pass
TC15	Alert severity filtering works	Low-severity alert generated	Not shown if filter set to high severity only	Pass

Table 4.1: Test Cases

4.2 Test Reports

4.2.1 Objective of Testing

The objective of testing is to ensure that the application meets all functional and non-functional requirements by verifying the reliability, accuracy, usability, and performance of its key features including real-time alerts, GPS tracking, and user preferences.

4.2.2 Testing Types Performed

Test Type	**Description**
Functional Testing	Verified that each feature behaves according to the system requirements.
Manual Testing	Conducted real-world testing on physical Android devices.
Unit Testing (Backend)	Tested alert generation logic and data integration in Python.
Usability Testing	Evaluated the interface and user experience for intuitiveness.
Performance Testing	Measured response times of notifications and maps.
Accessibility Testing	Verified features like large text, high contrast, and screen reader compatibility.

4.2.3 Test Summary

Total Test Cases	Passed	Failed	Blocked	Skipped
15	15	0	0	0

4.2.4 Environment Details

- **Devices Tested:** Android 11 (Pixel 4a), Android 13 (Samsung Galaxy A32)
- **Tools Used:** Firebase Console (Auth, Messaging, Firestore), Postman (API Testing), Android Emulator, VS Code
- **Test Data Sources:** Simulated data using Python, Live data from CMA and OpenWeatherMap APIs

4.2.5 Key Results

Feature	Test Outcome	Notes
User Registration/Login	Pass	Auth via Firebase, handles valid and invalid input
Real-Time Alert Generation	Pass	Simulated alerts triggered delivery under 10 seconds
GPS and Manual Location Input	Pass	Both input modes returned accurate results

Alert Preferences	Pass	Custom alert types and severity filters applied correctly
Push Notifications	Pass	Firebase Cloud Messaging worked across devices
Weather Map Display	Pass	Overlays for storm zones worked correctly
Accessibility Options	Pass	High contrast and large text verified

4.2.6 Issues Identified & Resolved

Issue ID	**Description**	**Status**	**Resolution**
001	Map marker lag on older devices	Resolved	Reduced map update frequency
002	GPS error not handled gracefully	Resolved	Added manual fallback prompt
003	UI clipping on smaller screens	Resolved	Used responsive layout widgets

Chapter 5

Conclusion

Since the number of weather disasters like cyclones, floods, tornadoes and heavy rainfall is increasing, the Weather Disaster and Early Warning Application was made to offer fast, reliable and up-to-date alerts. Its designers wanted users, mainly in high-risk regions, to have access to time-sensitive information and warnings to help them handle danger from severe weather.

A mobile application was created with Flutter on the frontend and Python (Flask) on the backend for this project. The **CMA** and the OpenWeatherMap APIs were used to collect current weather information from these data sources. Within the system, anyone who uses Firebase can manage user authentication, handle data storage in Firestore and use notifications via FCM.

Included are the following prominent features:

- Style settings can enable instant alerts and tracking of current weather.
- Impacts also trigger localized alerts based on GPS or manually entered locations.
- You can modify alerts to suit the level and type of security events.
- Storm and flood layers can be viewed on interactive weather maps.
- A navigation system that is effortless to use and accessible to various types of users.

The system is designed so that alerts reach you quickly (in 10 seconds or fewer) after danger is spotted and weather data is updated every 5 minutes. Specialized diagrams such as ERD, Activity, Sequence, State Transition and DFDs were developed to reflect the system's design and operations.

Tests were performed that involved manual, functional, unit and usability subtypes. All 15 tests were executed and none failed which means the application is reliable and stable. The first users of the application discovered that its clear design and usefulness are especially appreciated by individuals who live in regions that frequently experience disasters.

The project applies software skills and helps improve safety as well as disaster control. This makes it possible to add iOS support, multilingual features and boost the AI-based predictions in the future.

In short, the system helps users by combining updated mobile apps, working with real data and taking an user-friendly approach to reduce damage from natural disasters.

Bibliography