umar rashid

weather_application

FYP_7th_semester
kalim_sattar
COMSATS Institute of Information Technology

Document Details

Submission ID
trn:oid:::1:3128548292

Submission Date
Jan 13, 2025, 2:44 PM GMT+5

Download Date
Jan 13, 2025, 2:48 PM GMT+5

File Name
Weather_diaster_and_early_warning_application_1.pdf

File Size
7.8 MB

73 Pages

7,691 Words

43,953 Characters

0% detected as AI

The percentage indicates the combined amount of likely AI-generated text as well as likely AI-generated text that was also likely AI-paraphrased.

Caution: Review required.

It is essential to understand the limitations of AI detection before making decisions about a student's work. We encourage you to learn more about Turnitin's AI detection capabilities before using the tool.

Detection Groups

1 AI-generated only 0%
Likely AI-generated text from a large-language model.

2 AI-generated text that was AI-paraphrased 0%
Likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

Frequently Asked Questions

How should I interpret Turnitin's AI writing percentage and false positives?
The percentage shown in the AI writing report is the amount of qualifying text within the submission that Turnitin's AI writing detection model determines was either likely AI-generated text from a large-language model or likely AI-generated text that was likely revised using an AI-paraphrase tool or word spinner.

False positives (incorrectly flagging human-written text as AI-generated) are a possibility in AI models.

AI detection scores under 20%, which we do not surface in new reports, have a higher likelihood of false positives. To reduce the likelihood of misinterpretation, no score or highlights are attributed and are indicated with an asterisk in the report (*%).

The AI writing percentage should not be the sole basis to determine whether misconduct has occurred. The reviewer/instructor should use the percentage as a means to start a formative conversation with their student and/or use it to examine the submitted assignment in accordance with their school's policies.

What does 'qualifying text' mean?
Our model only processes qualifying text in the form of long-form writing. Long-form writing means individual sentences contained in paragraphs that make up a longer piece of written work, such as an essay, a dissertation, or an article, etc. Qualifying text that has been determined to be likely AI-generated will be highlighted in cyan in the submission, and likely AI-generated and then likely AI-paraphrased will be highlighted purple.

Non-qualifying text, such as bullet points, annotated bibliographies, etc., will not be processed and can create disparity between the submission highlights and the percentage shown.

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)

CHAPTER 0

Revision History

Name	Date	Reason for Changes	Version

CHAPTER 0

Application Evaluation History

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

Supervised By

Kalim Sattar

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 personalised alerts, storm chasing, and floods notice using factual information from CMA and OpenWeatherMap. The purpose is to assist the users in initiating appropriate action and minimize the hazards resulting from such climate changes.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

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. **Public Safety Enhancement:** It aids individuals, communities, and emergency response teams in preparing before, during, and after severe weather, thus enhancing overall public safety and disaster preparedness.
5. **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.
4. **Government and Municipal Agencies:** Institutions that require weather predictions for safety precautions, especially agencies involved in disaster management.
5. **Businesses in Affected Regions:** Enterprises that plan for search and rescue operations or have facilities in vulnerable areas, requiring continuity despite severe weather conditions.

1.4 Project Timeline

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

1. **Phase 1: Requirements Collection (Weeks 1 & 2)**
   - Finalizing functional and non-functional specifications.
   - Defining stakeholders and data suppliers (e.g., CMA).
2. **Phase 2: Design and Planning (3 weeks)**
   - Developing system architecture and designing the user interface (UI/UX) for both mobile and web applications.
   - Planning API integration and data processing.

3. **Phase 3: Development (8 weeks)**
    - **Frontend Development (Flutter):** Developing interfaces using UI controls across platforms.
    - **Backend Development (Python):** Processing weather data and issuing alerts.
4. **Phase 4: Integration & Testing (4 weeks)**
    - Incorporating APIs to fetch real-time weather data.
    - Conducting unit tests, integration tests, and user acceptance testing (UAT).
5. **Phase 5: Deployment and Launch (2 weeks)**
    - Hosting the application on mobile operating systems (iOS/Android) and web platforms.
    - Monitoring the initial use and addressing post-launch issues.
6. **Phase 6: Maintenance & Updates (Ongoing)**
    - Frequent updates including bug fixes, performance improvements, and feature enhancements.

Total Project Duration: 19 weeks.

Chapter 2

Description

The Weather Disaster and Early Warning Application for Android is a complete application packed with alerts and real time information to inform the users of cyclones, tornadoes, heavy rainfall and floods. It features current and proximal notification that will enable users to act appropriately and decrease probabilities of perishing in cases of natural disasters. The application is created with Flutter, which guarantees a fast and stable run of the app on every Android device.

2.1 Key Features:

1. **Cyclone Tracking and Alerts:**

   - The app offers cyclone positioning; the location of the storm, cyclone trajectory, and the strength of

the cyclone in the course of the subsequent one day period.

- It will provide users real time alerts of storm surge risks and Red zone when they are in the path of a dangerous cyclone.

2. **Heavy Rain and Flood Warnings:**

- The app predicts an occurrence of a heavy rainfall event and alerts the users of flooding in their locations.
- Correct rainfall data assists the users to mitigate or avoid water related disasters like floods from flash floods or river bursts.

3. **Customizable Notifications:**

- It allows its users to personalize the kind of alert that is depending on the event severity and the users location. Thus, only those messages that require the attention of a particular user will be transmitted.
- People can turn alerts on for weather events such as cyclones or floods, according to the usage of every individual.

4. **Real-Time Notifications:**

- The system delivers an alert to the user through an instant push notification the moment a likelihood

of adverse weather condition is realized close to the user.

- Notifications are meant to be brief but informative and give the notification recipient sufficient information on the event in question and sensible safety instructions.

5. **User-Friendly Interface:**
   - Various conveniences include an easy to use and easy to navigate Android application where users can view weather information at a glance as well as see warnings and follow storms via maps.
   - Users can choose a specific state or allow GPS to set up notifications for notifying them in real time for the regions they are currently in.

How it Works:

- **Data Integration:** The maps also get real weather data from authenticated sources like CMA & Open Weather Map to offer accurate data.
- **Interactive Maps:** The app provides updated real time tracking on storm, precipitation and areas affected by floods, and even weather map pictures so that the users can easily have the picture of the weather in their area.

- **Custom Alerts:** Users can select specific regions or use their current GPS location to receive customized alerts about weather events that affect them directly.
- **Safety Recommendations:** Furthermore, the app provides specific safety tips and measures that the user has to take when threatened by serious weather conditions.

2.2 Technical Details:

- **Built with Flutter:** Flutter was used to the development of the app so it feels fast and as if it's native especially to the android devices in addition to this the app is easy to update and maintain since all the platforms share the same code.
- **Backend Processing:** This is a Python application backend, used for data collection and weather predictions, making sure that the forecast is on time using Artificial Intelligence.
- **Google Play Store Distribution:** This app will be stored at the Google Play Store which makes it easy for Android users to access, update and manage the application.

Chapter 3

Software Requirement Specification

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.

3.1 Requirement Collection Techniques

Thus, it is possible to use different requirement collection techniques to obtain precise and inclusive data on the requirements for the **Weather Disaster and Early Warning Application**. All the techniques can actually assist in the confirmation of the alignment of the system to the user needs, the technical requirements, and the business requirements. Here are the most effective techniques:

Interviews

- **Stakeholder Interviews:**Face-to-face/moderated focus groups with key stakeholders including the disaster management agencies Met office and the target users.
- **Key Questions:** Which kind of severe weather do you prefer the app to concentrate on? That triggers these

questions: What are the most essential facets of alerting and forecasting for the users?

The following are surveys and questionnaires:

- **Target Audience Surveys:**Survey inhabitants of the extreme-risk zones, regarding their past experiences in extreme weather conditions and their measures on the preparation to such events.
- **Benefits:** Gather significant amount of data about target audience about what features they want to see in application, their preferred alert style, and interfaces.

Workshops and Focus Groups

- **Collaborative Sessions:** This includes inviting emergency responders, weather experts, and community members to conduct focus group sessions where they group generate ideas and features necessary during natural disasters and identify the typical obstacles the encountered during such calamities.
- **Focus Groups:** Involve selected target user to discuss about the experience of the users in the app's notifications as well the right time of alert and information required.

Use Case Development

- **Scenario-Based Discussions:** Create detailed use cases to capture how users will interact with the system in different scenarios, such as during a cyclone, flood, or tornado.
- **Benefits:** Identifies functional requirements by outlining step-by-step actions users will take during critical weather events.

There are common exercise including brainstorming sessions that are often adopted in the organization.

- **Collaborative Idea Generation:**Group multi-discipline teams (developers, product managers, climatologists) to define new features, notification, and the behavior of the system.
- **Goal:** Collect ideas that would not otherwise be obtained through specified conventional requirement gathering process.

3.2 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.

3.2.1 Use Cases

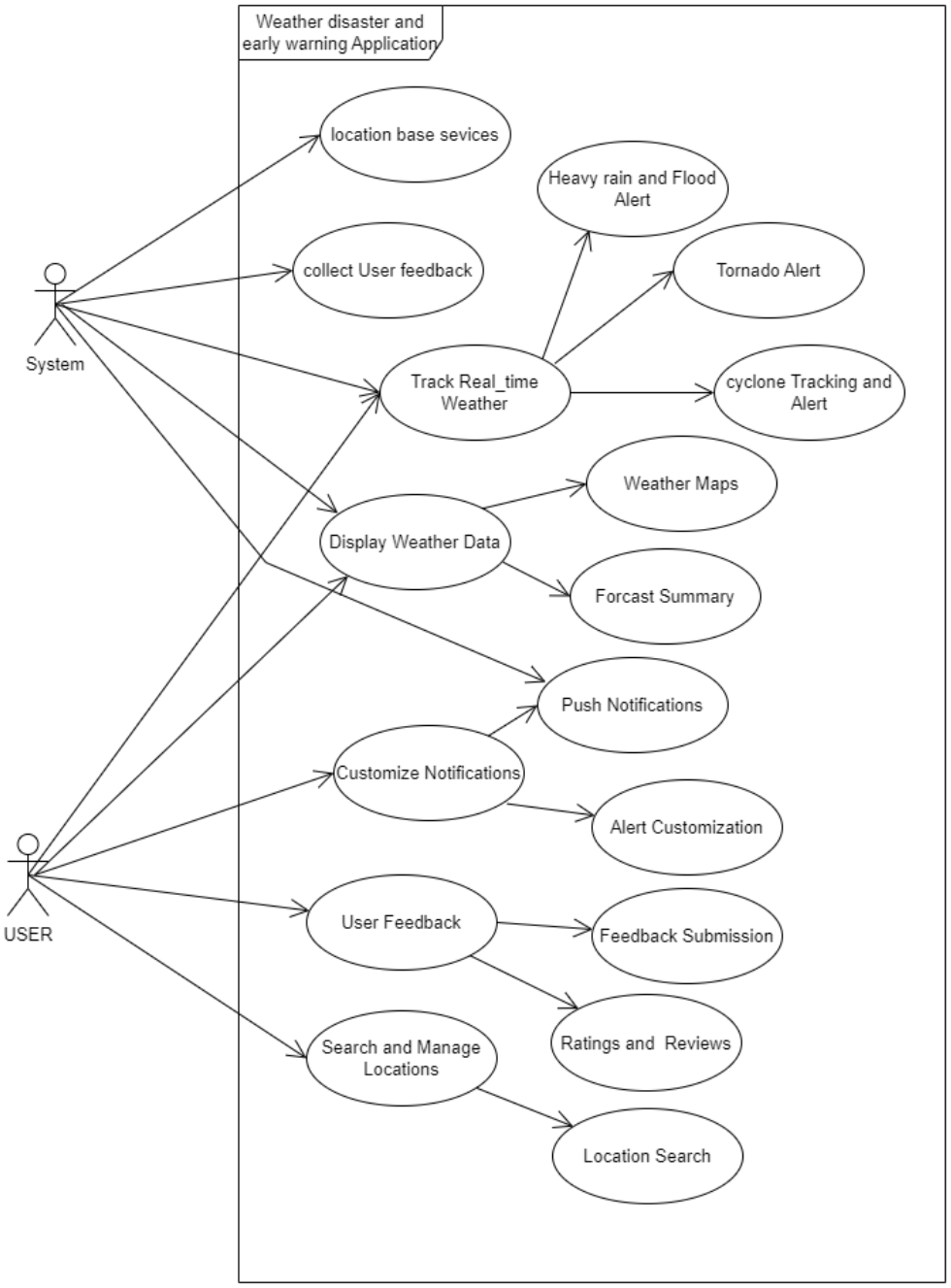

FIGURE 3.1: Usecase of Weather Disaster and Early Warning Application

3.2.2 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 3.1: Event Table for Weather Disaster and Early Warning Application

3.3 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.
- **Cloud Integration:**Make use of cloud services in IT infrastructure, so that the resource can be easily obtained as the users change frequently.

Chapter 4

External Interface Requirements

The weather app also has an interface that was created with the purpose of enhancing usability and menu navigation. They are able to view temperature, clouds, wind, humidity, and general summery of today's weather on the main screen. Other options include a search option for searching weather data in various cities, and the history of the latest searches is available for use. It also provides alerts for any extreme weather condition such as thunderstorm, flood, tornado, and cyclones. Customers can register using their name and email and password as well as log in to numerous features like the rain alert. This brings out the times when one has to carry an umbrella hence better prepared in case of weather fluctuations.

Start page

FIGURE 4.1: Start Page of Weather Disaster and Early Warning Application

Login and Sign up Page

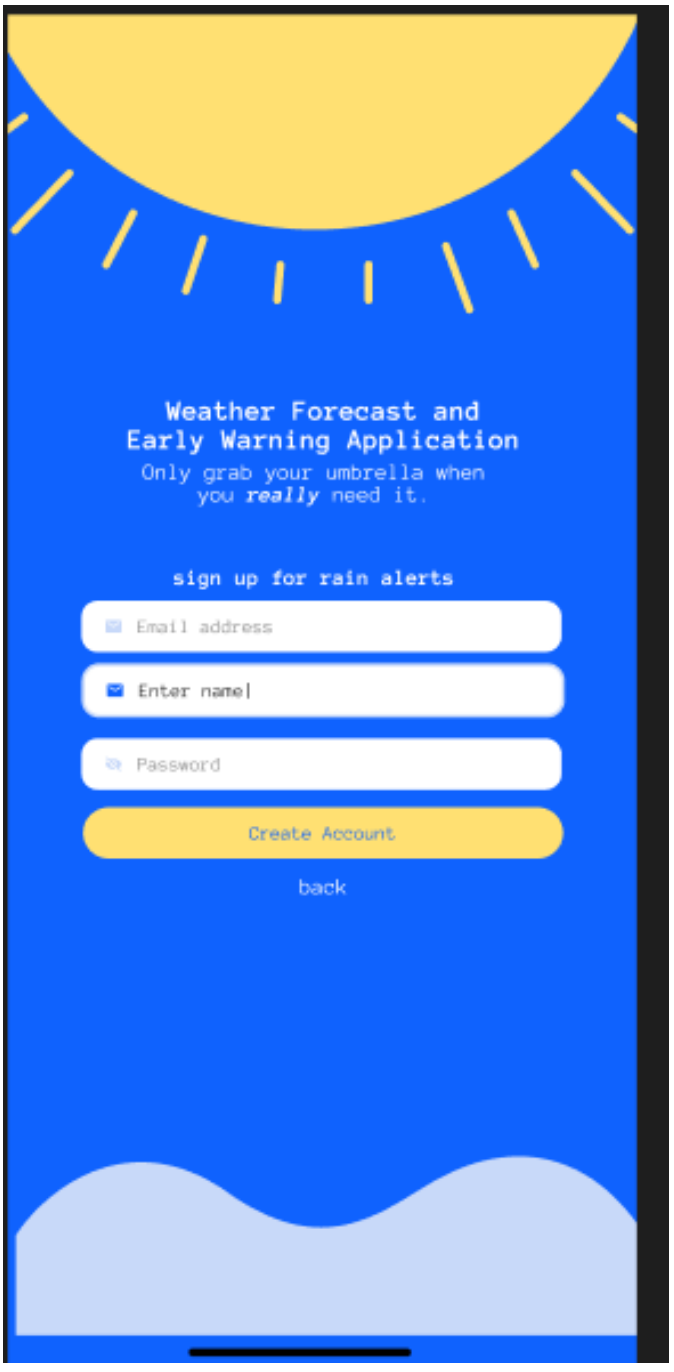

FIGURE 4.2: Login and Sign up page of Weather Disaster and Early Warning Application

Home page

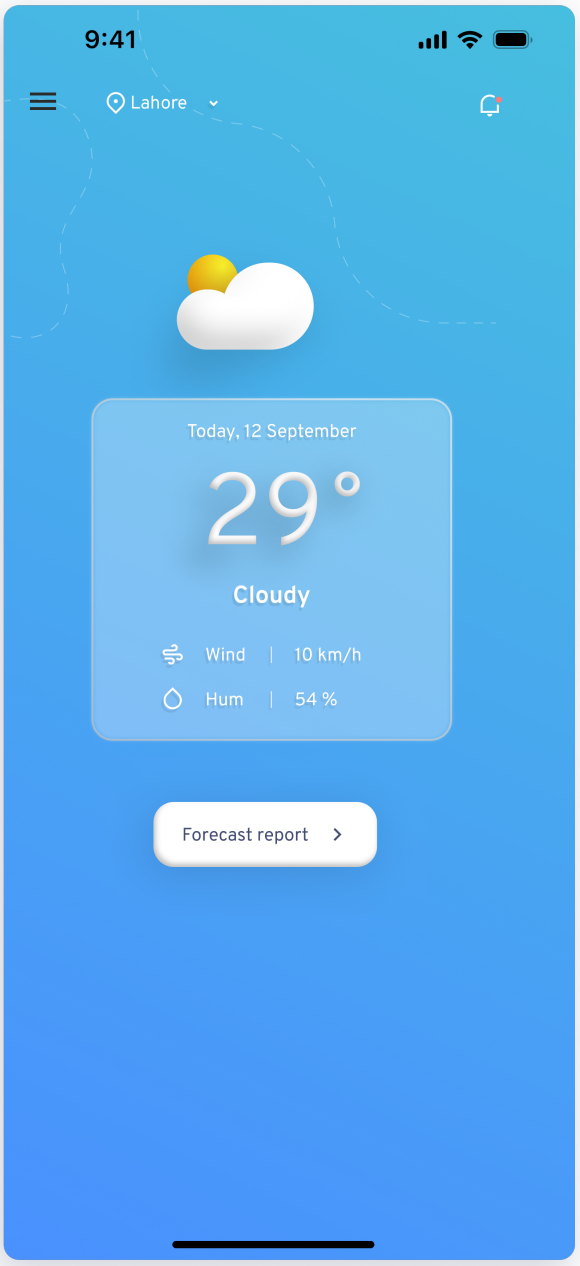

FIGURE 4.3: Home Page of Weather Disaster and Early Warning Application

Search page

FIGURE 4.4: Search Page of Weather Disaster and Early Warning Application

App Bar Screen

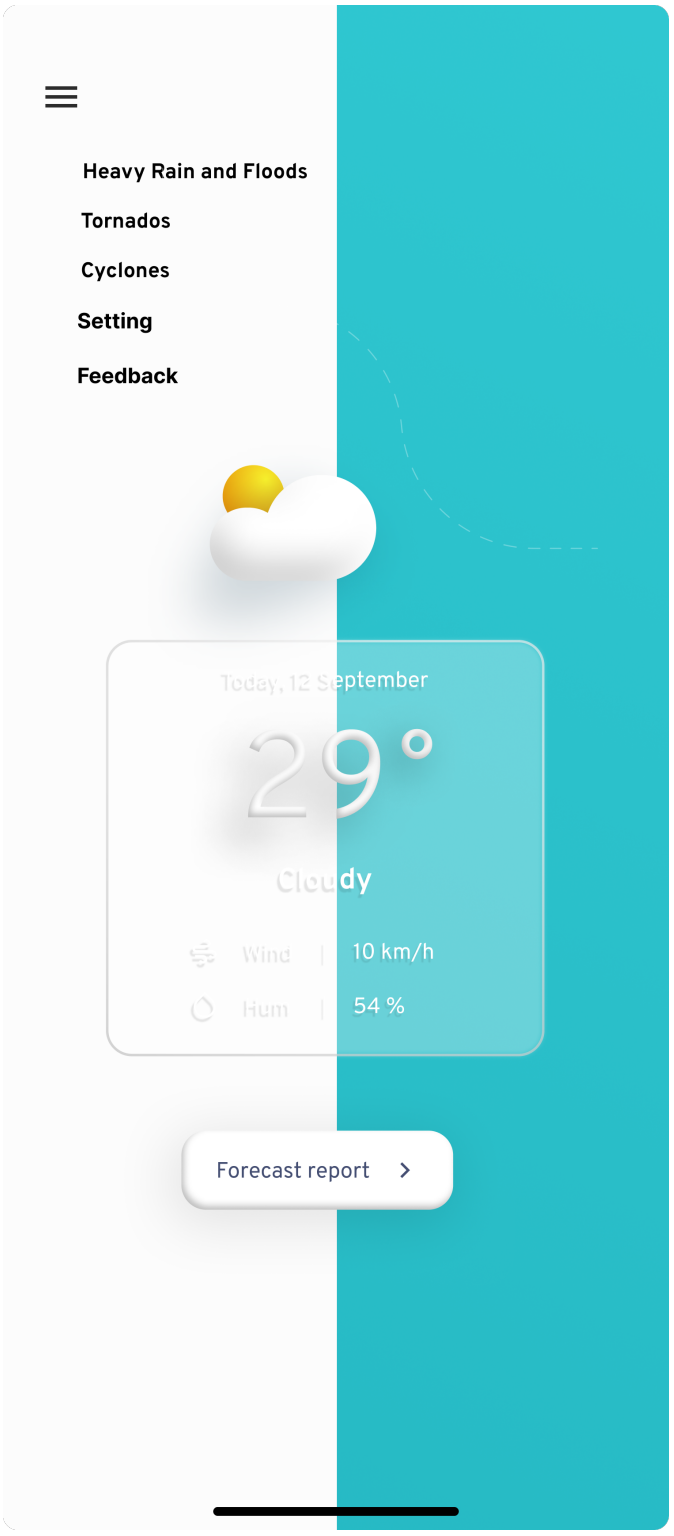

FIGURE 4.5: App Bar Screen of Weather Disaster and Early Warning Application

Chapter 5

Design and Implementation

The **Weather Disaster and Early Warning System (WDEWS)** is web-based application designed to be easily scalable to add new modules as needed and utilizes geographic location and user preferences to deliver localized weather disturbances warning messages in real time. The actions form a sequence starting with the user launching the app, entering their location either through GPS or manually, a function facilitated by the *Location Service*. The app sends a request to the *Weather Data Server* which, in turn, considers conditions for some alerts like cyclones or floods. There are user preferences on which alerts to receive and notifications are provided in the form of push notification through what is referred to as the *Notification Service*. The system works within states at states, pre-sleep, location

detection, weather data, alert evaluation, and notification. Alerts are further filtered in real-time and then kept in an *Alert Database* for reuse and resending, in order to ensure only important notifications are provided.

5.1 ERD Diagram

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.
  - **PreferencesID (Integer):** Foreign key linking to the user's preferences.
- **Description:** The user entity represents a system account, allowing users to set alert preferences.

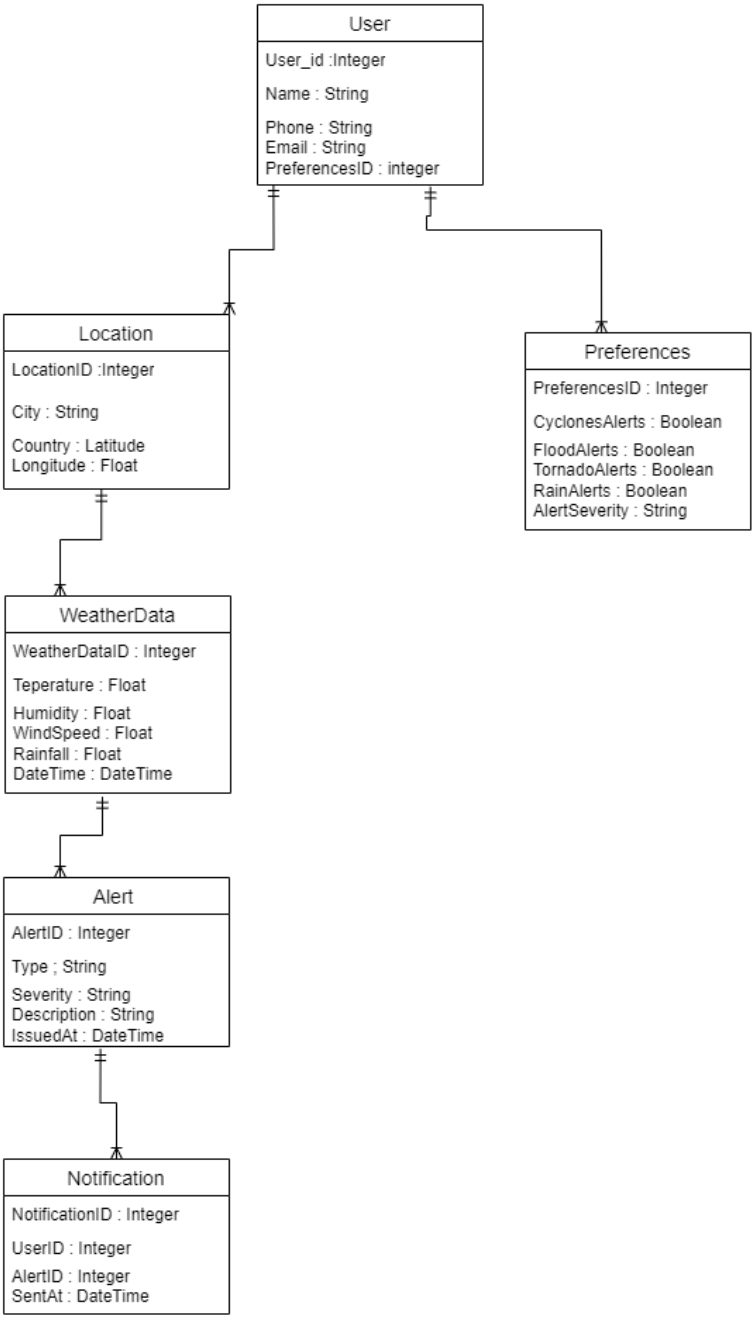

FIGURE 5.1: ERD of Weather Disaster and Early Warning Application

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.

5.2 Activity Diagram

1. **Open App** Upon launch of the application, the process of monitoring is initiated by the user.
2. **Fetch User Location** If the app is able to localise the user, it will make the identification of the user's location its priority.
3. **Use GPS?**
   - If **Yes**, the application then uses the GPS to get the current location of the user.
   - If **No**, the user independently types in their desired location.

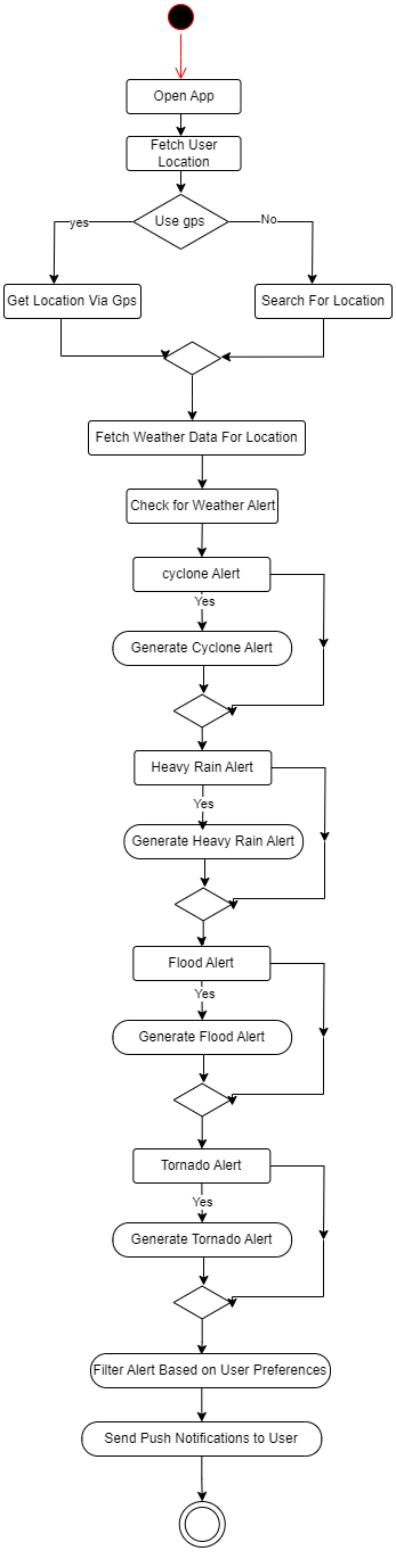

FIGURE 5.2: Activity Diagram of Weather Disaster and Early Warning Application

4. **Get Weather of Specified Place** In case the place where the user wants to go is defined with the help of

GPS or by typing it, the application retrieves the climate data of that place.

5. **Check for Weather Alerts** The application analyzes the weather data to determine the type of disaster or alert in the given area.
6. **Cyclone Alert**
   - If cyclone conditions are present, a **Cyclone Alert** is created.
   - The process then continues to check for other alerts.
7. **Heavy Rain Alert**
   - If heavy rainfall conditions are observed, a **Heavy Rain Alert** is created.
   - The process continues to check for the next alert type.
8. **Flood Alert**
   - If high flood conditions are identified, a **Flood Alert** is created.
   - The process continues to check for other possible alerts.
9. **Tornado Alert**
   - If tornado conditions are observed, a **Tornado Alert** is created.

- The process moves to the next phase.

10. **Work with Alerts Depending on the User Profile** The generated alerts are filtered based on the user's preferences. Only the preferred alerts are selected for notification.

11. **Broadcast Push Notifications to the User** The application sends push notifications for the selected alerts to the user.

12. **End Process** The process ends, and the user receives the alerts based on their preferences and geographic location.

Key Notes

- **Decision Points:** Decision points such as "Use GPS?", "Cyclone Alert?", etc., are included to ensure the appropriate actions are taken based on the situation.
- **Customization:** User preferences allow for customization, ensuring that notifications are sent only for the types of alerts the user wants.
- **Alert Types:** The supported weather alerts include cyclone alerts, heavy rain alerts, flood alerts, and tornado alerts.

This workflow provides a clean and user-oriented method for disseminating weather alerts in real time.

5.3 Class Diagram

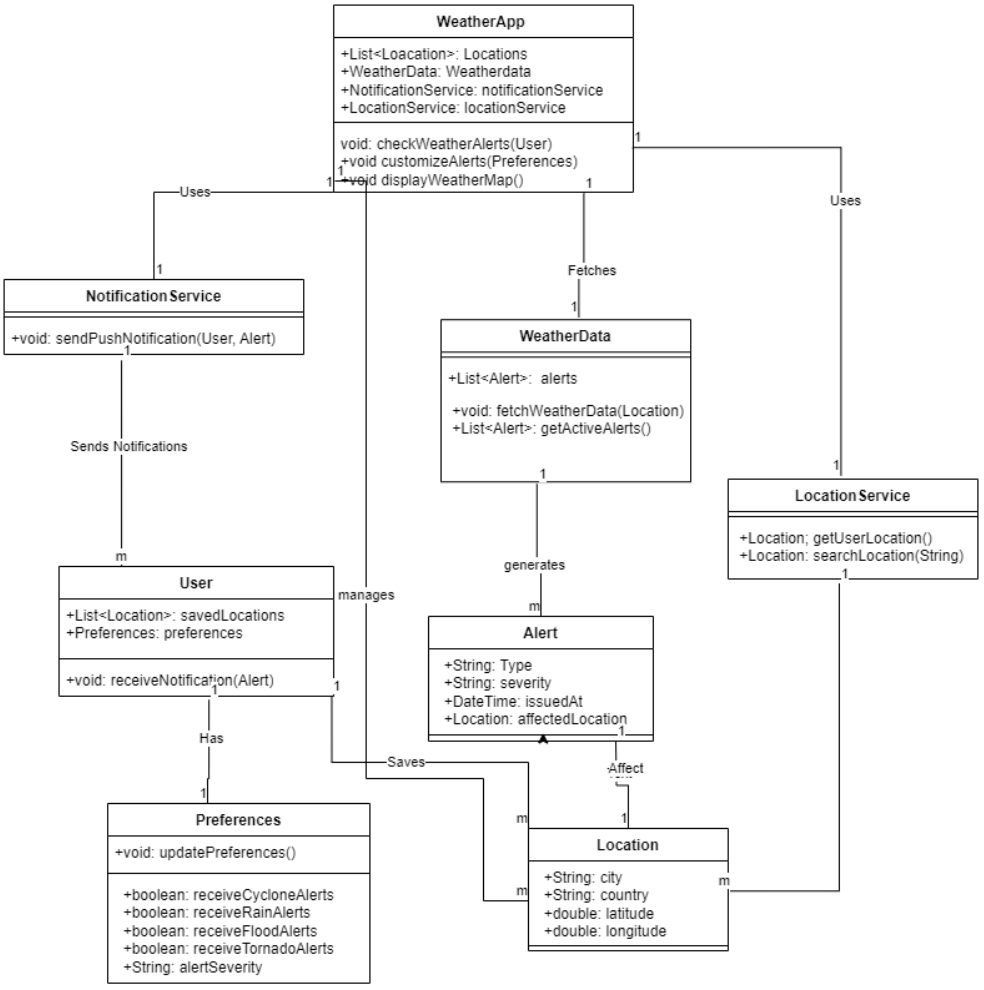

FIGURE 5.3: Class diagram of Weather Disaster and Early Warning Application

The following diagram shows A class diagram for the weather disaster and early warning system, including the system, classes, attributes, methods, and relationships. Here's a detailed explanation:

1. WeatherApp

- **Attributes:**
  - `+List<Location>: Locations` – A list of locations where the application is tracking data.

  - `+WeatherData: Weatherdata` – Associated table with weather data for the application.
  - `+NotificationService: notificationService` – Responsible for sending notifications to the alert service.
  - `+LocationService: locationService` – This service handles all location-related functionalities.

- **Methods:**
  - `+void checkWeatherAlerts(User user)` – Checks for weather alerts for the user based on their location and preferences.
  - `+void customizeAlerts(Preferences)` – Customizes alerts based on the respective preferences of a user.
  - `+void displayWeatherMap()` – Displays a map showing climate conditions to the user.
- **Relationships:** Utilizes the services of `NotificationService`, `WeatherData`, and `LocationService` to deliver functionalities.

2. Notification Service

- **Methods:**

  - `+void sendPushNotification(User, Alert)` – Notifies a user of an alert that was previously generated.

- **Relationships:** Uses `Alert` data to deliver notifications to users and integrates with `WeatherApp` to provide alerts.

3. WeatherData

- **Attributes:**
  - `+List<Alert>: alerts` – The current list of active alerts generated based on weather reports.
- **Methods:**
  - `+void fetchWeatherData(Location)` – Retrieves weather data for a specific location.
  - `+List<Alert>: getActiveAlerts()` – Returns a list of currently active alerts.
- **Relationships:** Fetches weather details for a specific `Location` and creates `Alert` objects based on weather conditions.

4. LocationService

- **Methods:**
  - `+Location getUserLocation()` – Retrieves the current location of the user.

  - `+Location searchLocation(String)` – Searches for a location based on user input.

- **Relationships:** Supports the `WeatherApp` in identifying user locations and other specific places.

5. **User**

- **Attributes:**
  - `+List<Location>: savedLocations` – A list of locations saved by the user.
  - `+Preferences: preferences` – User preferences for receiving alerts.
- **Methods:**
  - `+void receiveNotification(Alert)` – Processes notifications received by the user.
- **Relationships:** Manages preferences through the `Preferences` class and interacts with `NotificationService` to receive notifications.

6. **Preferences**

- **Attributes:**
  - `+boolean receiveCycloneAlerts` – User preference for receiving cyclone alerts.
  - `+boolean receiveRainAlerts` – User preference for receiving rain alerts.

  - `+boolean receiveFloodAlerts` – User preference for receiving flood alerts.
  - `+boolean receiveTornadoAlerts` – User preference for receiving tornado alerts.
  - `+String alertSeverity` – Minimum severity level for triggering alerts.

- **Methods:**
  - `+void updatePreferences()` – Allows the user to change their preferences.

- **Relationships:** Manages user-specific alert settings in conjunction with the `User` class.

7. **Alert**

- **Attributes:**
  - `+String Type` – The type of alert, e.g., cyclone or flood.
  - `+String severity` – The severity level of the alert.
  - `+DateTime issuedAt` – The timestamp when the alert was issued.
  - `+Location affectedLocation` – The location affected by the alert.

- **Relationships:** Alerts are generated by `WeatherData` and sent to users through `NotificationService`.

8. Location

- **Attributes:**
  - `+String city` – Name of the city.
  - `+String country` – Name of the country.
  - `+double latitude` – Latitude coordinate.
  - `+double longitude` – Longitude coordinate.
- **Relationships:** Locations are linked to weather information, alerts, and user preferences.

System Summary This class diagram demonstrates how the various system components in the weather disaster and early warning application integrate with each other. It highlights how:

- `WeatherApp` orchestrates the overall workflow using `WeatherData`, `NotificationService`, and `LocationService`.
- `NotificationService` delivers user-specific alerts based on preferences stored in the `Preferences` class.
- `WeatherData` retrieves and processes weather information for specific `Location` instances and generates `Alert` objects.
- Users interact with the system to save locations, update preferences, and subscribe to notifications.

This modular system ensures scalability and provides users with real-time weather alerts.

5.4 Sequence Diagram

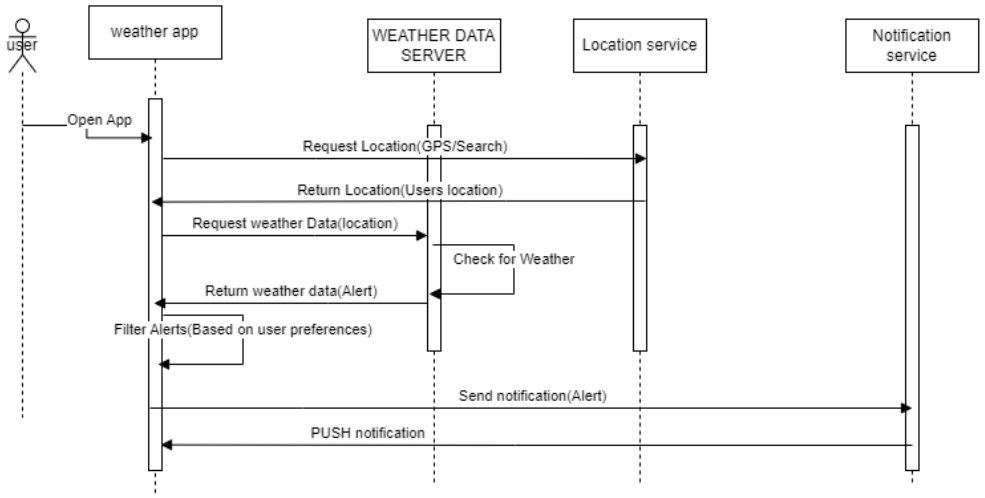

FIGURE 5.4: Sequence Diagram of Weather Disaster and Early Warning Application

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

1. **Open App** The user opens the weather application, which initiates the process.
2. **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**.
3. **Return Location** The **Location Service** sends the current or searched location of the user to the **Weather App**.
4. **Request Weather Data** The **Weather App** shares the location information with the **Weather Data Server** and requests weather details for that location.
5. **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).
6. **Return Weather Data (Alert)** If there are any alerts, the **Weather Data Server** reports them back to the **Weather App**.
7. **Filter Alerts** The **Weather App** filters the received alerts based on the user's preferences, including severity levels and types of alerts.

8. **Send Notification** The filtered alerts are forwarded to the **Notification Service** for delivery to the user.
9. **Push Notification** The **Notification Service** provides push notifications about the relevant weather alerts to the user.

Key Notes

- **Real-Time Updates**: Ensures that weather data and alerts are pulled and processed in real-time.
- **User Preferences**: Alerts delivered to the user are refined according to their interests and preferences.
- **Component Interaction**: This diagram illustrates the relationships between the app itself, the data server, and external services such as location and notification services.
- **Scalability**: The system structure is designed to effectively support multiple users and multiple locations.

This sequence ensures smooth and user-oriented delivery of weather alerts, as planned.

5.5 State Transition Diagram

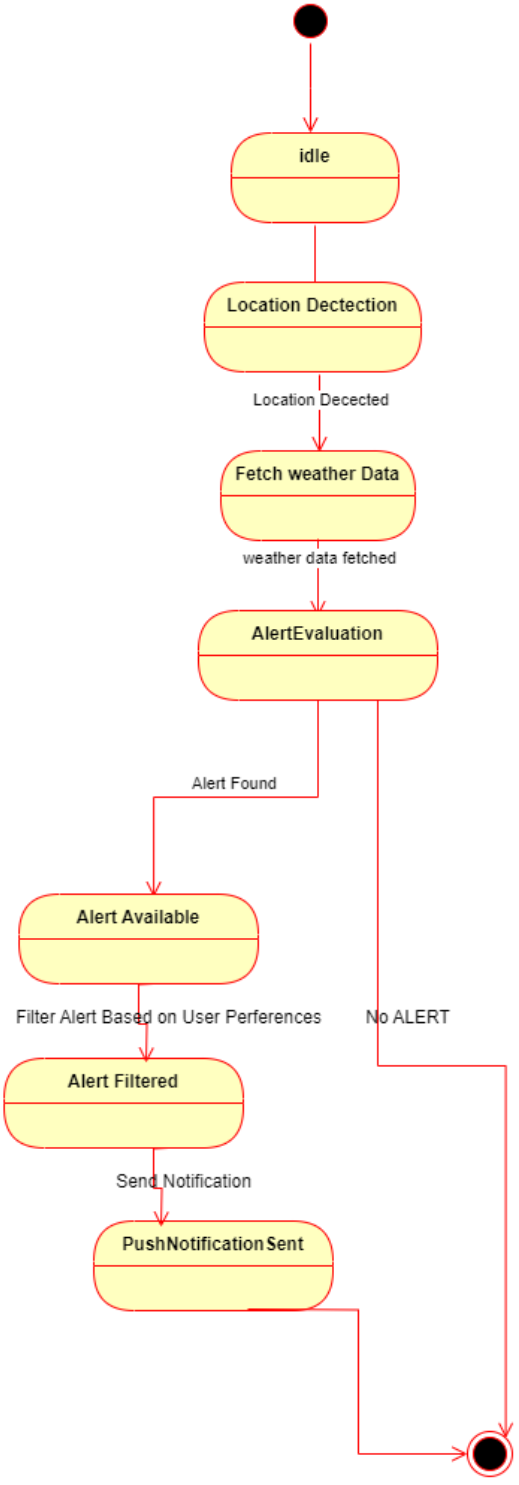

FIGURE 5.5: State Transition Diagram of Weather Disaster and Early Warning Application

States and Transitions

The process of modeling is divided into states and transitions, as described below:

1. **Idle** The system is initiated in the idle state, waiting to execute its core functionality once an action is initiated.
2. **Location Detection**
   - **Transition:** GPS or manual entry is used, and the system identifies the user's position.
   - **Output:** The location is successfully detected.
3. **Fetch Weather Data**
   - **Transition:** The program obtains current weather information for the identified geographical location from the weather data server.
   - **Output:** The weather data is successfully fetched.
4. **Alert Evaluation**
   - **Transition:** The fetched weather data is analyzed by the system to determine if any weather alerts have been issued.
   - **Output:**
     - If an alert is detected, the system moves to the **Alert Available** state.

  - If no alerts are found, the system returns to the **Idle** state.

5. **Alert Available**
   - **Transition:** An alert is identified and filtered based on the user's preferences, such as alert type and severity level.
   - **Output:** The alert is filtered as per the user's requirements.
6. **Alert Filtered**
   - **Transition:** The filtered alert is prepared to be delivered to the user.
   - **Output:** The system moves to the notification state.
7. **Push Notification Sent**
   - **Transition:** The system forwards the filtered alert to the user via a push notification.
   - **Output:** The notification is dispatched, and the system transitions back to the **Idle** state.

Key Notes

- **Dynamic Evaluation:** The system can self-analyze weather information and send signals only when necessary conditions are detected.

- **User Preferences:** Alerts are refined according to user preferences to ensure that only the most relevant notifications are produced.
- **Loop back:** If no alerts are discovered, the system returns to the idle state, awaiting the next action.
- **End State:** The process is complete once a push notification has been issued, or if no alerts are present.

This state diagram guarantees the efficiency of the procedure for searching, evaluating, and notifying users about weather conditions in real-time.

5.6 Zero Level Data Flow Diagram (DFD)

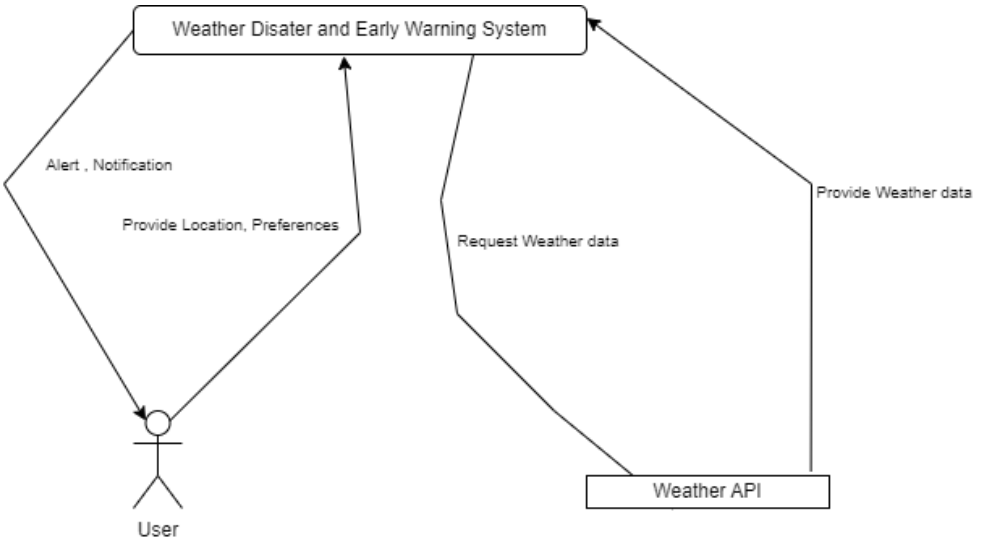

FIGURE 5.6: DFD Zero level Diagram of Weather Disaster and Early Warning Application

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

System Name Weather Disaster and Early Warning System

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.

Overview This Zero Level DFD provides an overview of the Weather Disaster and Early Warning System (WDEWS), highlighting the major participants, data flows, and processes while excluding specific implementation details.

5.7 One Level Data Flow Diagram (DFD)

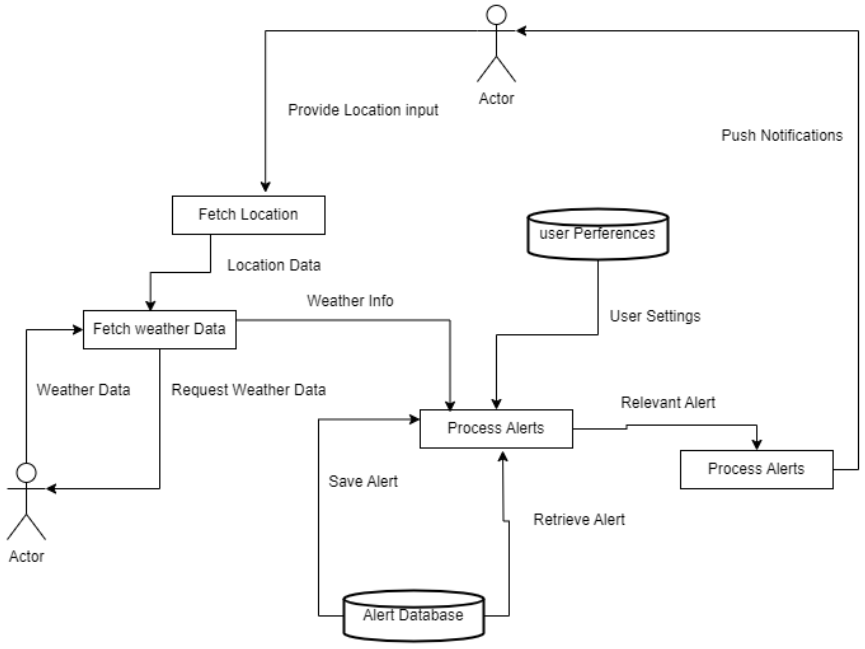

FIGURE 5.7: DFD One level Diagram of Weather Disaster and Early Warning Application

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 re-processing 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.

5.8 Two Level Data Flow Diagram (DFD)

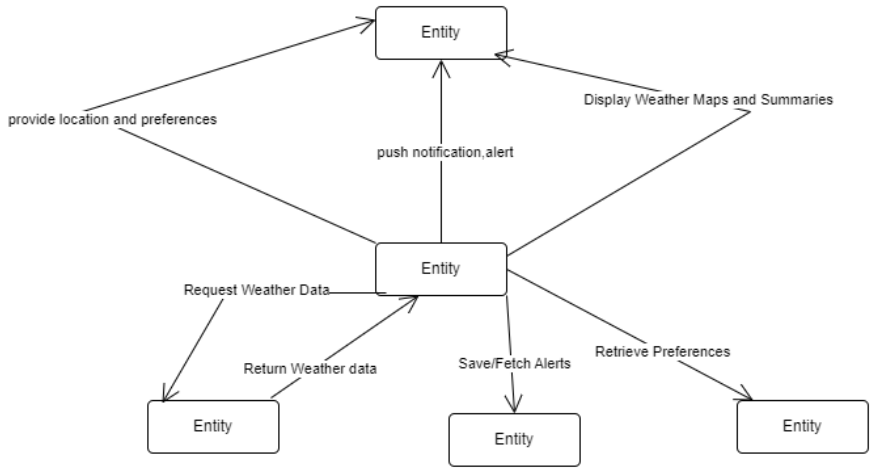

FIGURE 5.8: DFD Two level Diagram of Weather Disaster and Early Warning Application

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. **System to Alert Database**:
    - **Save Alerts**: This system saves generated alerts concerning specific weather events and risks existing in a specific area.
    - **Fetch Alerts**: The system also pulls out saved alerts for further use or to present them to the user.
4. **System to Preferences Database**:

- **Retrieve Preferences**: User preferences are used by the system in order to produce alerts and notifications according to the users' needs.

Purpose of the DFD This two-level DFD shows the details of how the Weather Disaster and Early Warning System works. It illustrates:

- How data accumulates from users and how it is processed in the system.
- How the system integrates with external data sources to obtain live weather data.
- The storage of alerts and user preferences.
- The delivery of notifications at appropriate times alongside user preferences and real-time weather conditions.

This structured approach is useful for explaining what a given application is capable of and how its components are related.

5.9 Algorithms and Pseudo-code for Weather Disaster and Early Warning Application

The system utilizes inputs from end users, weather APIs, and databases to generate real-time alerts in the workflow. Below is the structure of the major processes:

Algorithm of the Main Activities

a) User Registration and Preference Selection Goal: Users must be allowed to register or sign up, establish their settings, and store them in the preferences table.

Steps:

1. Retrieve user registration information, namely, name, email address, and geographical location.
2. Gather the choices for the type of alerts to be issued, whether major or minor, and how the alerts are to be disseminated.
3. Validate the data and choices.
4. Update the preferences database by writing data to the file.

b) Weather Data Retrieval Goal: Retrieve raw current weather information from an external weather data service.

Steps:

1. Determine the user's location through preferences.
2. Construct an API request with parameters (location, type of event, event severity level).
3. Pass the request to the API for the weather data source.
4. Analyze the response and process the subsequent subtasks.

5. Retrieve cyclone tracks, tornado occurrences, precipitation rates, or flood information.

c) Alert Generation Goal: Create and store alerts using weather data.

Steps:

1. Check the weather values against specific thresholds for severe events.
2. If thresholds are exceeded, write an alert message specifying the location, type of event, and severity.
3. Store this alert in the alert database.

d) Notification Delivery Goal: Present notifications according to user preferences.

Steps:

1. Retrieve alerts saved in the alert database.
2. Cross-check alerts with user-based preferences, such as event type and severity level.
3. Select the type of notification (push, email, or in-app alert).
4. Send the notification to the user.

Algorithms for Major Processes

a) User Registration and Preference Configuration

```
def register_user(name, email, location, preferences):
    if validate_input(name,email,location,preferences):
        save_to_database('preferences', {
            'name': name,
            'email': email,
            'location': location,
            'preferences': preferences
        })
        return "Registration successful"
    return "Invalid input"
```

b) Weather Data Retrieval

```
import requests

def fetch_weather_data(location, event_type):
    api_url = "https://weather-api.example.com/data"
    params = {
        'location': location,
        'event_type': event_type
    }
    response = requests.get(api_url, params=params)
    if response.status_code == 200:
        return response.json()
    else:
        return "Error fetching data"
```

c) Alert Generation

```
def generate_alert(weather_data, location):
    thresholds = {
        'cyclone': 75,
        'flood': 100,
        'tornado': 50
    }
    for event, value in weather_data.items():
        if value > thresholds[event]:
        alert = {
                'location': location,
                'event': event,
                'severity': "High",
    'details': f"{event} detected with value {value}"
            }
            save_to_database('alerts', alert)
            return alert
    return None
```

Categorization of Resources

1. **Requests Library:** Used to call weather data from external APIs. *Example:* `requests.get()` for API requests.
2. **Firebase (Optional):** Used to send messages to users via the application. *Example:* `firebase_admin.messaging` for sending alerts.

3. **Database Library:** For managing databases like SQLite or MySQL. *Example:* Handling the insert, update, and fetch of preferences and alerts.

4. **JSON Library:** For analyzing and structuring API responses. *Example:* `json.loads()` for decoding API responses.

Bibliography