COMSATS University Islamabad,
Vehari campus

SOFTWARE DESIGN DESCRIPTION

(SDD DOCUMENT)

for

Weather disaster and early warning application

Version 2.0

By

Faizan Sharif	**CIIT/FA21-BSE-007/VHR**
Salman Sharif	**CIIT/FA21-BSE-056/VHR**

Supervisor

Kalim Sattar

Bachelor of Software Engineering (2021-2025)

Table of Contents

Revision History

Name	Date	Reason for changes	Version
Models Updates	16 Dec 2024	Enhancing Models	2.0

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_______________

Introduction

The **Weather Disaster and Early Warning Application** is designed to provide **real-time alerts and weather forecasts** for **cyclones, tornadoes, heavy rain, and floods**. It helps users in high-risk areas stay informed and take necessary precautions.

Developed using **Flutter (frontend) and Python (backend)**, the app integrates with **CMA and OpenWeatherMap** to deliver **accurate weather data**. It offers **personalized notifications, interactive maps, and AI-based predictions** to enhance public safety and disaster preparedness.

The system ensures **fast alerts (within 10 seconds)** and updates weather data **every 5 minutes**, helping users respond quickly to severe weather conditions.

Design methodology and software process model

The **Weather Disaster and Early Warning Application** follows a **modular and layered architecture** to ensure **scalability, maintainability, and real-time data processing**. The app is structured into:

1. **User Interface Layer (Flutter)** – Provides an interactive and user-friendly experience.
2. **Business Logic Layer (Python & AI)** – Handles data processing, weather predictions, and alerts.
3. **Data Management Layer (Database & APIs)** – Stores user preferences, retrieves weather data from **CMA & OpenWeatherMap**, and manages notifications.

The **design principles** focus on **real-time data integration, cloud-based scalability, and customizable alerts** to ensure an efficient and user-centric experience.

Software Process Model

The application development follows the **Iterative Model**, ensuring continuous improvement based on user feedback and testing. The key phases include:

1. **Requirement Analysis** – Gathering user needs and defining key functionalities.
2. **System Design** – Creating UI/UX wireframes, database structures, and API integrations.
3. **Development (Incremental Approach)**
    - **Frontend (Flutter)** – Mobile UI design, user authentication, and map integration.

    - **Backend (Python, APIs)** – Weather data fetching, AI-based predictions, and notifications.
4. **Testing & Integration** – Unit testing, API validation, and performance checks.
5. **Deployment & Maintenance** – Launching on app stores, monitoring performance, and updating features based on feedback.

System overview

The Weather Disaster and Early Warning Application provides real-time alerts for cyclones, tornadoes, heavy rains, and floods using Flutter (frontend), Python (backend), and AI-based predictions. It integrates with CMA and OpenWeatherMap APIs for accurate forecasts.

Key Features:

- Real-time weather tracking & alerts based on user location.
- AI-driven predictions for storm intensity and disaster impact.
- Push notifications (within 10 seconds) for severe weather events.
- Customizable alerts & interactive weather maps for easy tracking.
- Cloud-based storage for user preferences and historical alerts.

How It Works:

1. Fetches real-time weather data based on GPS or user input.
2. Analyzes risks & generates alerts using AI models.
3. Sends instant push notifications with safety recommendations.
4. Displays interactive maps & detailed forecasts for user action.

Functionality

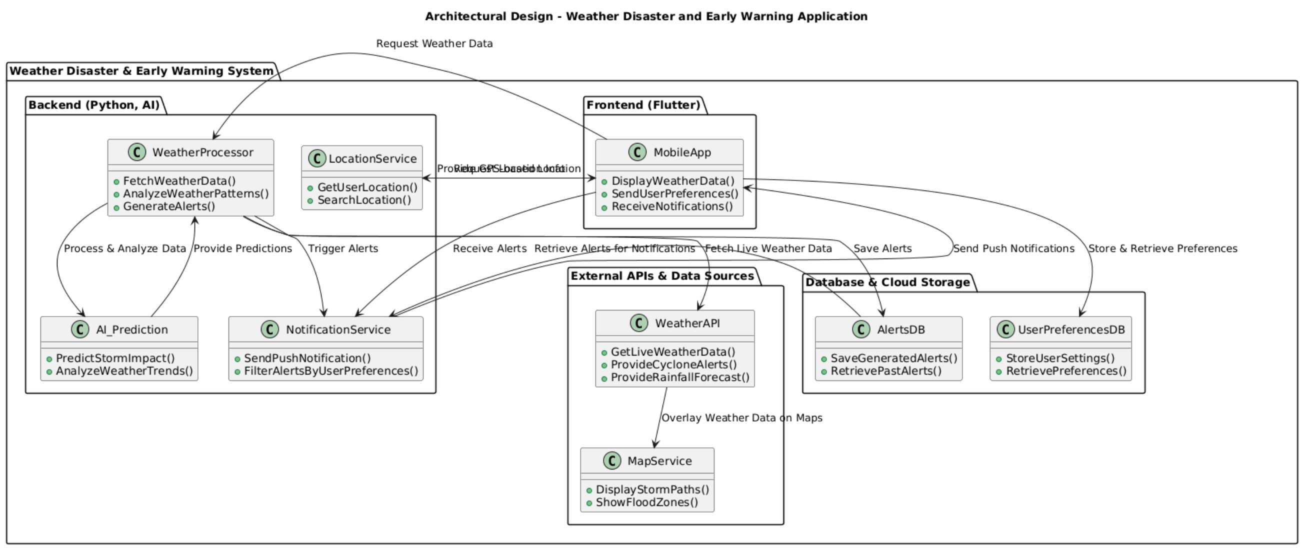
The **Weather Disaster and Early Warning Application** provides real-time weather alerts and tracking for **cyclones, tornadoes, heavy rains, and floods**. It ensures **timely notifications** and **interactive data visualization** for users in high-risk areas.

Key Functionalities:

1. **Real-Time Weather Tracking**
    - Fetches live weather data from CMA & OpenWeatherMap APIs.
    - Tracks cyclones, tornadoes, rainfall intensity, and flood risks.
2. **Customizable Alerts & Notifications**
    - Sends instant push notifications when severe weather is detected.
    - Users can set alert preferences based on location and event severity.
3. **Interactive Weather Maps**
    - Displays real-time storm paths, rainfall distribution, and flood zones.
    - Allows users to zoom in on affected areas for better visibility.
4. **Location-Based Services**
    - Supports GPS tracking for automatic location-based alerts.
    - Allows manual location search & multiple location management.
5. **AI-Based Predictions**
    - Uses machine learning models to forecast storm intensity & disaster impact.
    - Provides safety recommendations based on predicted weather conditions.
6. **User Feedback & Report System**
    - Users can report incorrect alerts or provide feedback for improvement.
    - The system learns from user input to enhance accuracy over time.
7. **Cloud-Based Storage & Scalability**
    - Stores user preferences, past alerts, and weather history for analysis.
    - Supports high availability & scalability for a large user base.

.

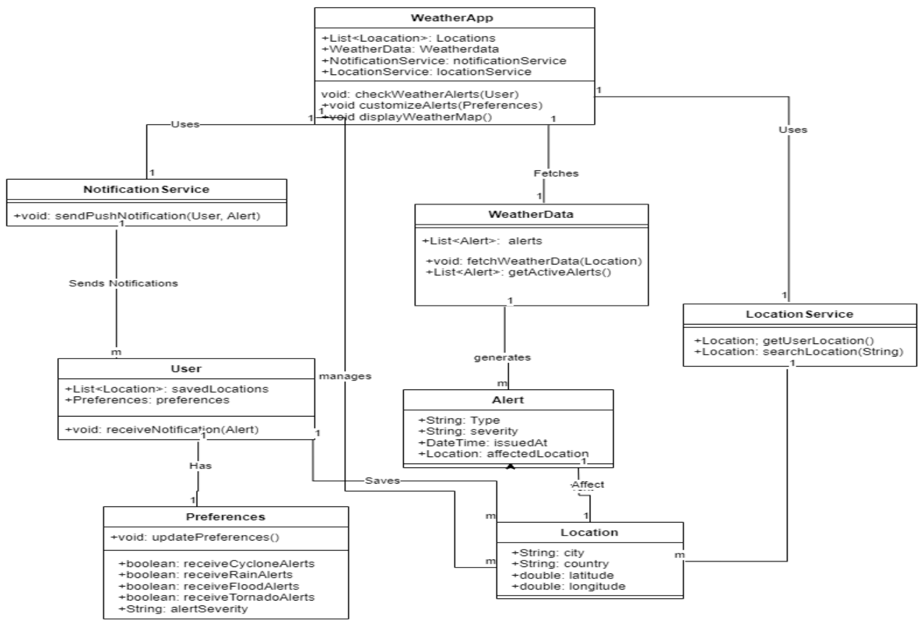
Architectural design



Design models

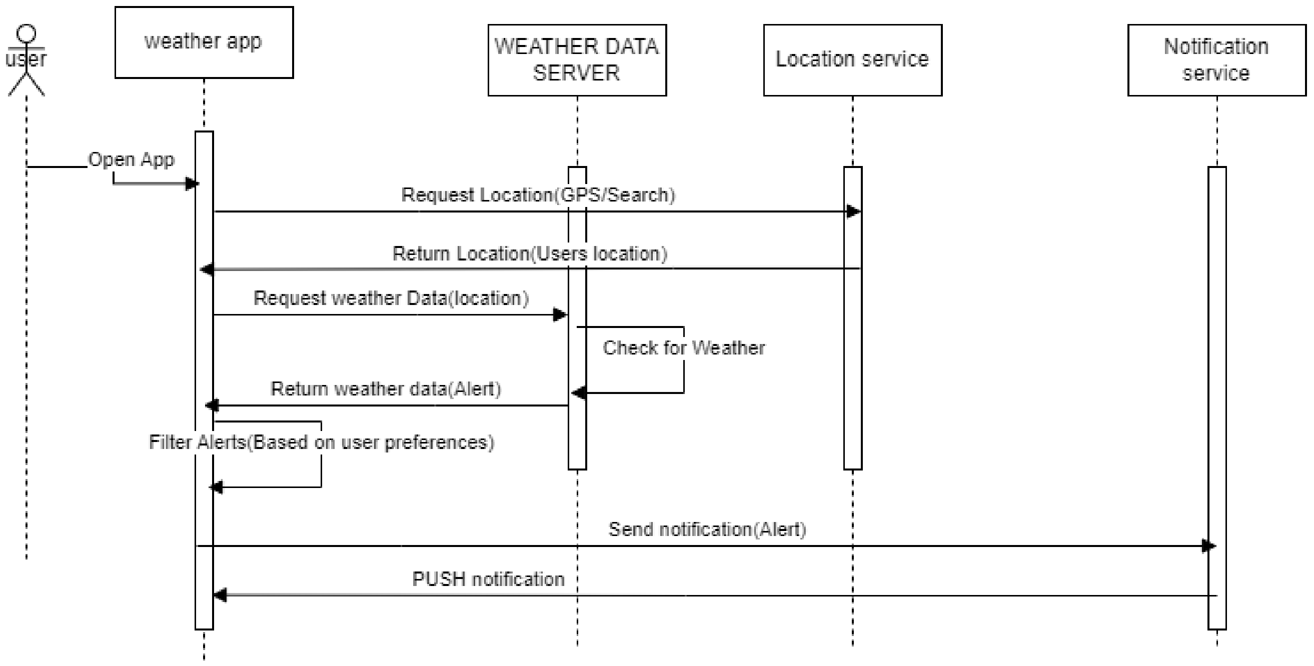
Class Diagram

The most important aspect of the presented diagram is the class diagram. It defines the concept and structure of the Weather Disaster and Early Warning Application, including classes, attributes, methods, and inter-class relationships. It illustrates the roles of system components and how weather data is processed, analyzed, and used to generate real-time alerts and notifications for users in high-risk areas.


WeatherApp
+List<Loacation>: Locations
+WeatherData: Weatherdata
+NotificationService: notificationService
+LocationService: locationService
void: checkWeatherAlerts(User)
+void customizeAlerts(Preferences)
+void displayWeatherMap()
Uses
Fetches
Uses
Notification Service
+void: sendPushNotification(User, Alert)
Sends Notifications
WeatherData
+List<Alert>: alerts
+void: fetchWeatherData(Location)
+List<Alert>: getActiveAlerts()
Location Service
+Location; getUserLocation()
+Location: searchLocation(String)
generates
User
+List<Location>: savedLocations
+Preferences: preferences
+void: receiveNotification(Alert)
manages
Alert
+String: Type
+String: severity
+DateTime: issuedAt
+Location: affectedLocation
Has
Saves
Affect
Preferences
+void: updatePreferences()
+boolean: receiveCycloneAlerts
+boolean: receiveRainAlerts
+boolean: receiveFloodAlerts
+boolean: receiveTornadoAlerts
+String: alertSeverity
Location
+String: city
+String: country
+double: latitude
+double: longitude
1
m

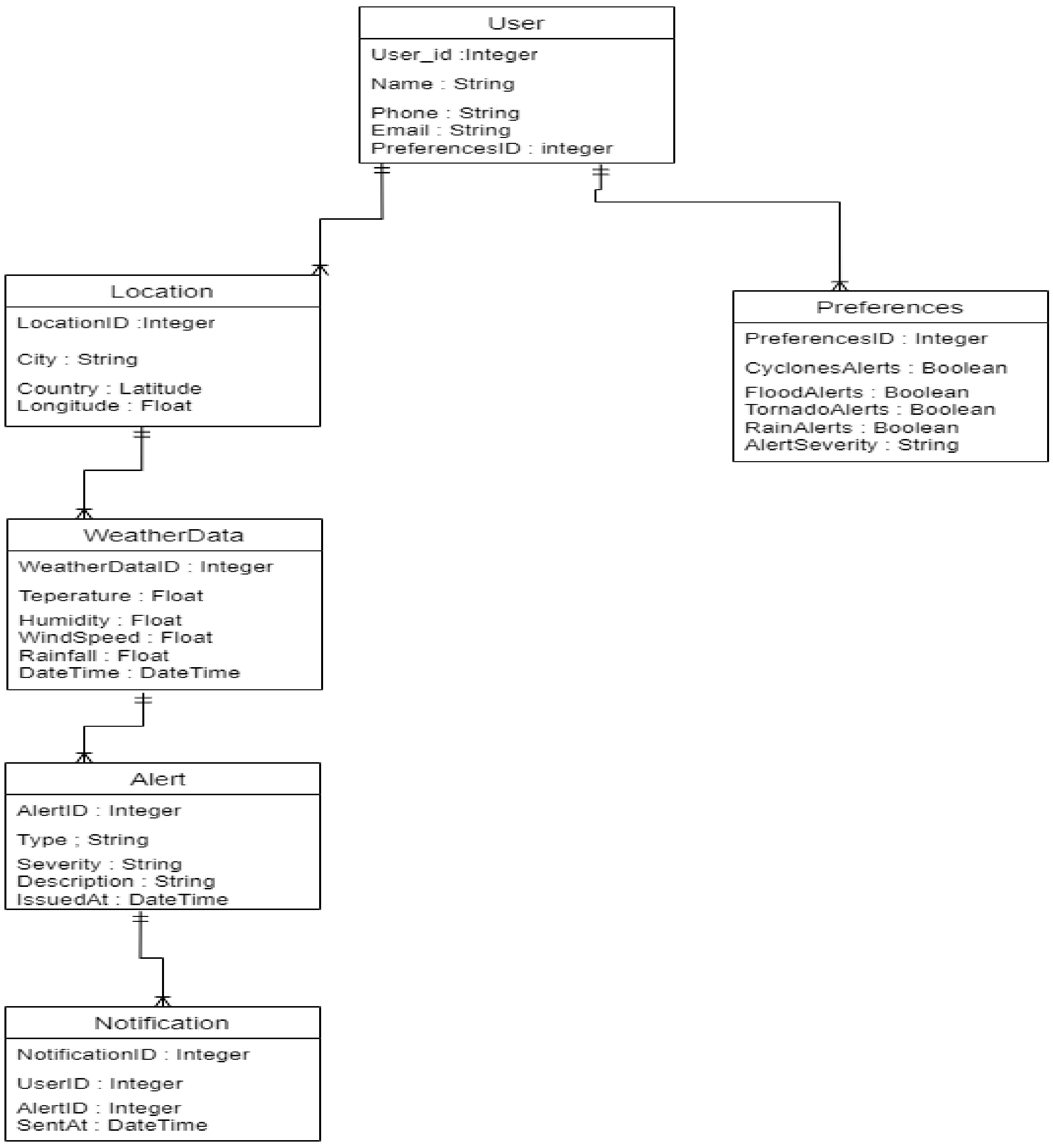
Sequence Diagram

The main activity of the system, ranging from **weather data retrieval to real-time alert generation and user notification**, is well illustrated by the **Sequence Diagram**


user
weather app
WEATHER DATA SERVER
Location service
Notification service
Open App
Request Location(GPS/Search)
Return Location(Users location)
Request weather Data(location)
Check for Weather
Return weather data(Alert)
Filter Alerts(Based on user preferences)
Send notification(Alert)
PUSH notification

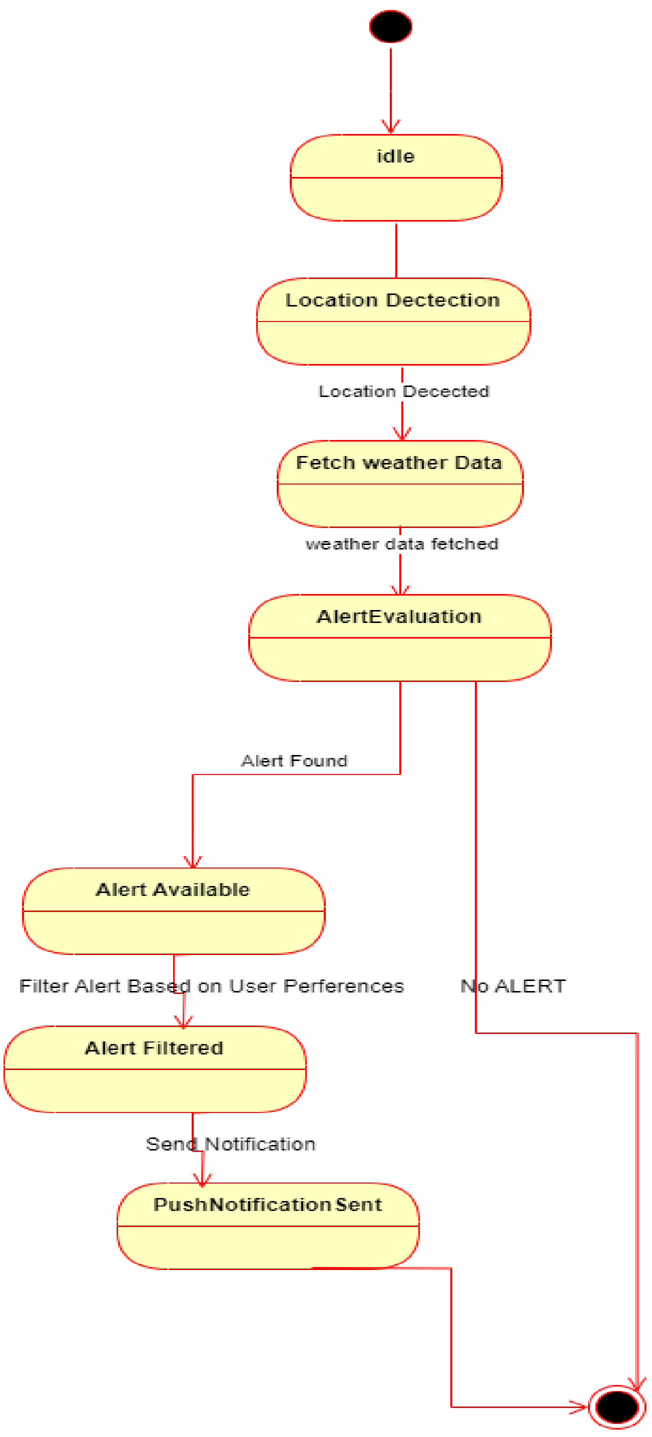
Erd Diagram :

The Entity-Relationship model was established to specifically focus on making relationships understandable and explaining the types of data stored in the entities. The data flow of weather information can be depicted in the Weather Data Flow Chart, which provides an understanding of how weather data is retrieved, processed, and stored in the system. The major and related entities have been defined in the system through an Entity-Relationship Diagram (ERD).



State Transition Diagram

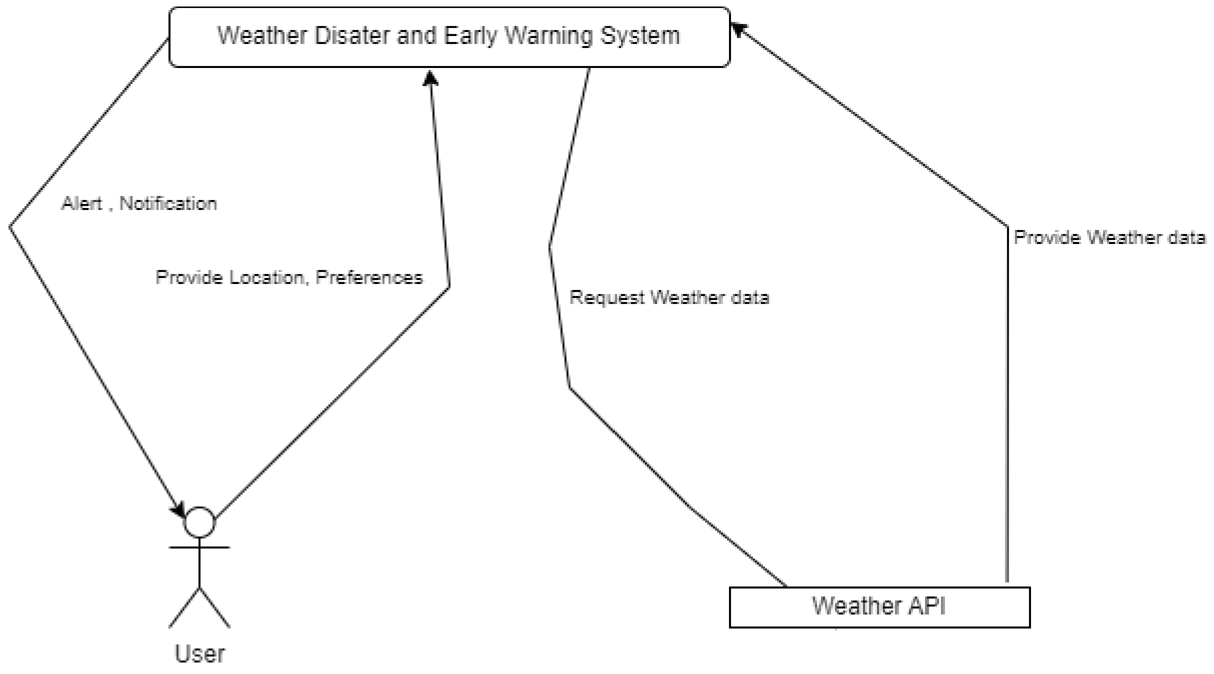
The State Dynamic Viewpoint captures the activities and communication of the system elements and applies to weather data analysis, providing the process flow initiated at the Weather Data Retrieval tier and completed at the User Notification tier.



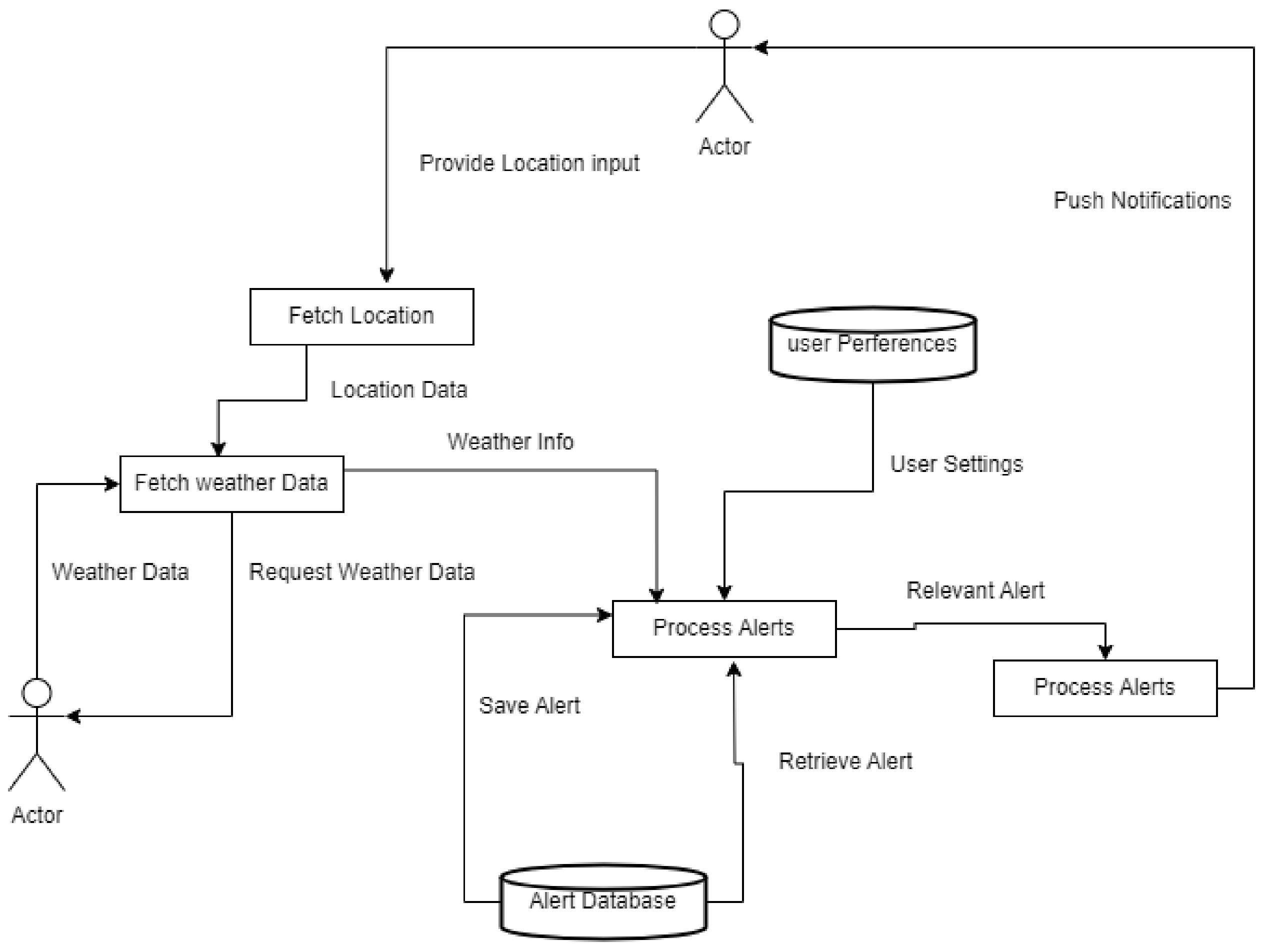
Data Flow Diagram

The Data Flow Diagram (DFD) provides an excellent system overview of how weather data enters, is processed, analyzed, and stored, and how alerts are generated and delivered to users. This simplifies the presentation of how data flows through the system, undergoing transformations before producing the desired output, such as real-time alerts, notifications, and interactive **weather maps**.

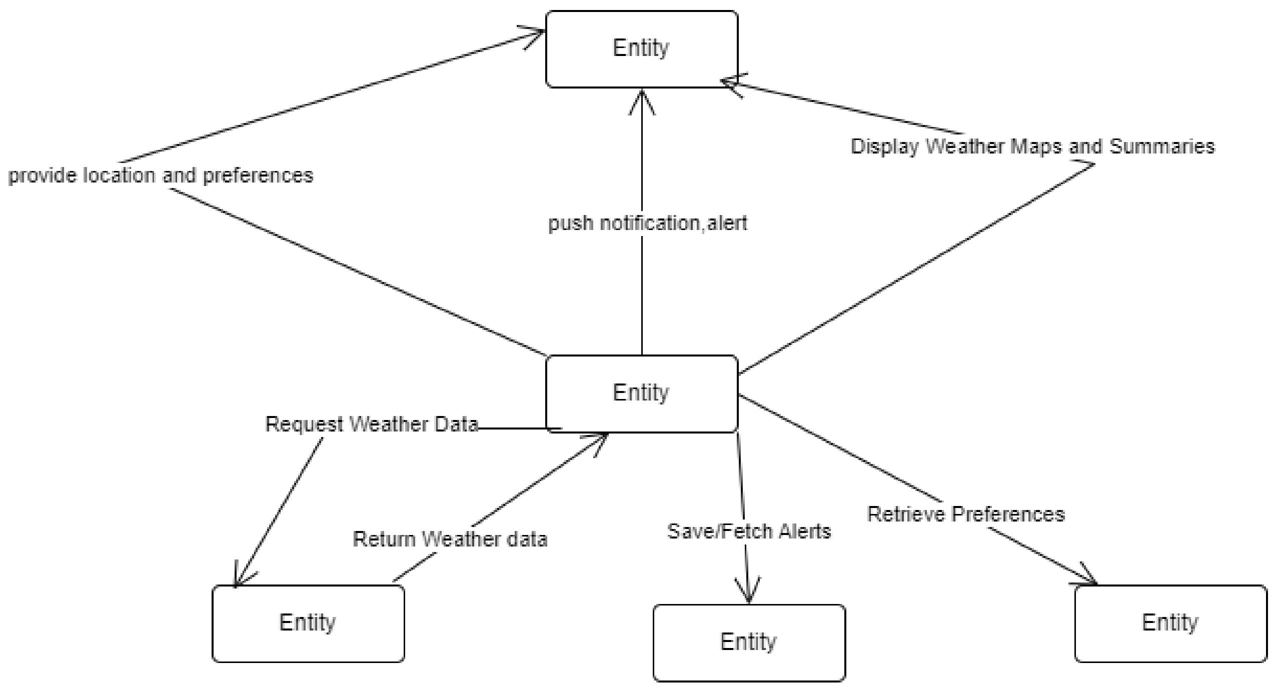
Zero level:



One level:

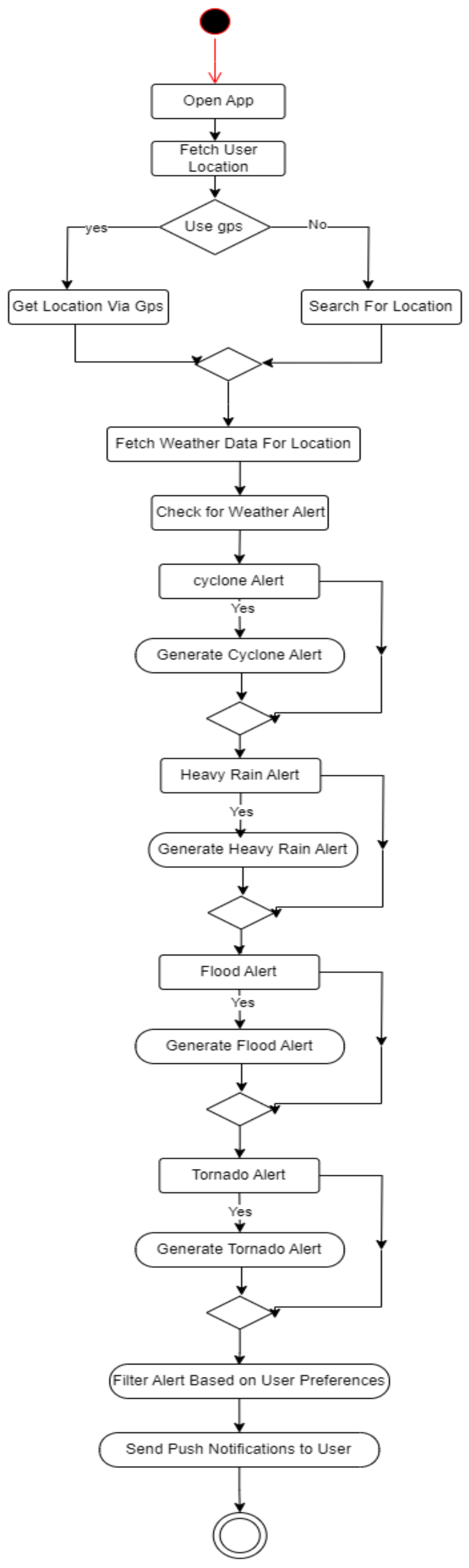


Two level:



Activity Diagram:

The Composite Viewpoint brings together multiple perspectives and compares them with the goal of creating a Weather Disaster and Early Warning System that meets the needs of users by providing accurate alerts and forecasts while ensuring data reliability and security.


Open App
Fetch User Location
Use gps
yes
No
Get Location Via Gps
Search For Location
Fetch Weather Data For Location
Check for Weather Alert
cyclone Alert
Yes
Generate Cyclone Alert
Heavy Rain Alert
Yes
Generate Heavy Rain Alert
Flood Alert
Yes
Generate Flood Alert
Tornado Alert
Yes
Generate Tornado Alert
Filter Alert Based on User Preferences
Send Push Notifications to User

Algorithm & Implementation

The Weather Disaster and Early Warning Application processes real-time weather data, detects severe weather conditions, and sends alerts to users. The implementation follows a modular approach, integrating Flutter (frontend), Python (backend), AI-based predictions, and external APIs (CMA, OpenWeatherMap).

---

1. Algorithm for Weather Data Processing & Alerts

Step 1: User Location Detection

1. Get user location (via GPS or manual input).
2. Send location data to the backend for weather updates.

Step 2: Fetch & Analyze Weather Data

1. Request live weather data from APIs (CMA, OpenWeatherMap).
2. Extract parameters (cyclone speed, rainfall intensity, wind speed, temperature).
3. Compare values with predefined thresholds:
    - If cyclone wind speed > threshold, trigger Cyclone Alert.
    - If rainfall intensity > flood risk level, trigger Flood Alert.
    - If tornado detected, trigger Tornado Alert.

Step 3: AI-Based Predictions

1. Process historical weather data to predict storm trajectory & intensity.
2. Generate risk assessment reports for affected regions.

Step 4: Alert Generation & Notification

1. Check user preferences for alert types & severity.
2. Generate customized notifications (push alerts, in-app warnings).
3. Send real-time notifications (within 10 seconds) to users.

---

2. Implementation Details

Frontend (Flutter)

- User Interface: Displays real-time weather data & alerts.
- Google Maps Integration: Shows affected areas & storm paths.
- User Preferences Module: Allows users to customize alerts.

Backend (Python & AI)

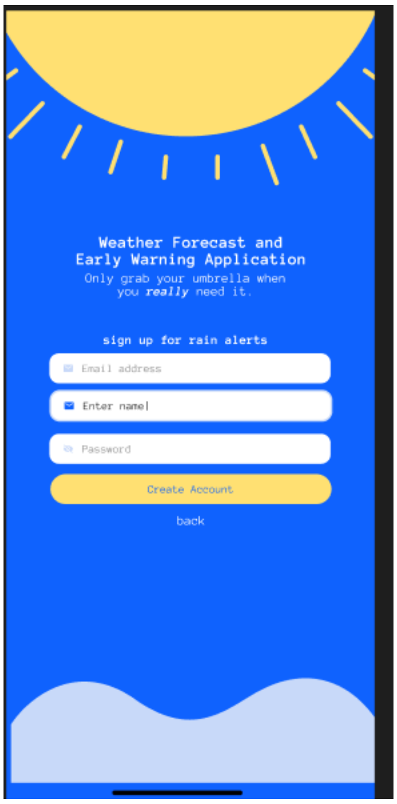
- Weather Data Processing: Fetches and analyzes data from APIs.
- AI Prediction Model: Uses historical patterns for forecast accuracy.

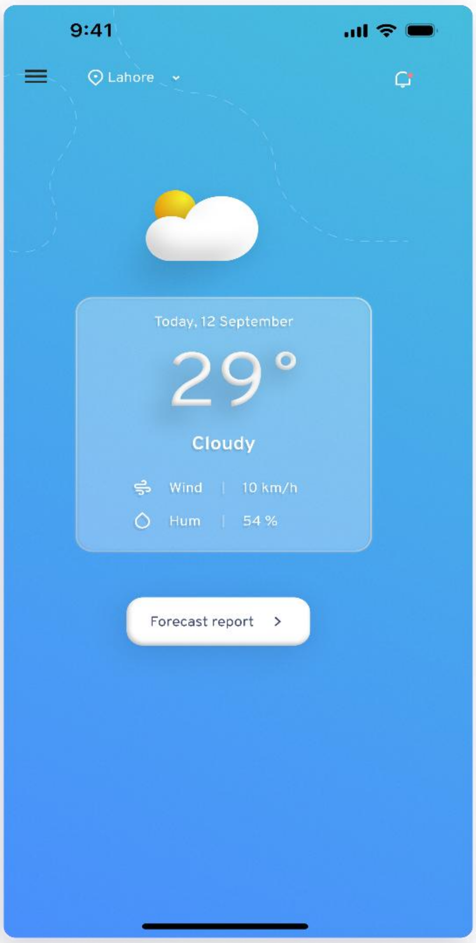
- Alert Generation System: Sends notifications based on risk levels.

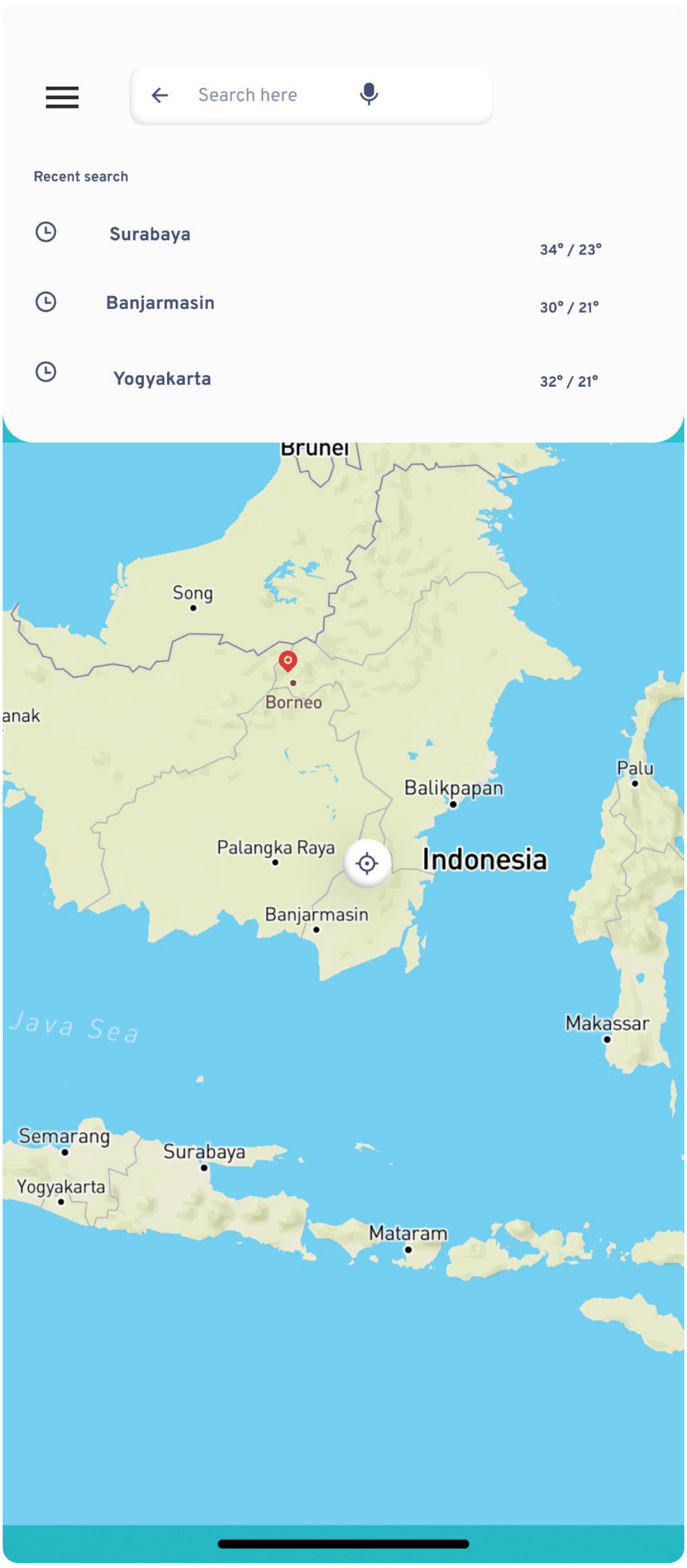
Database & Cloud Storage

- Stores user preferences, past alerts, and weather history.
- Ensures fast query retrieval for real-time alerts.

Human interface design





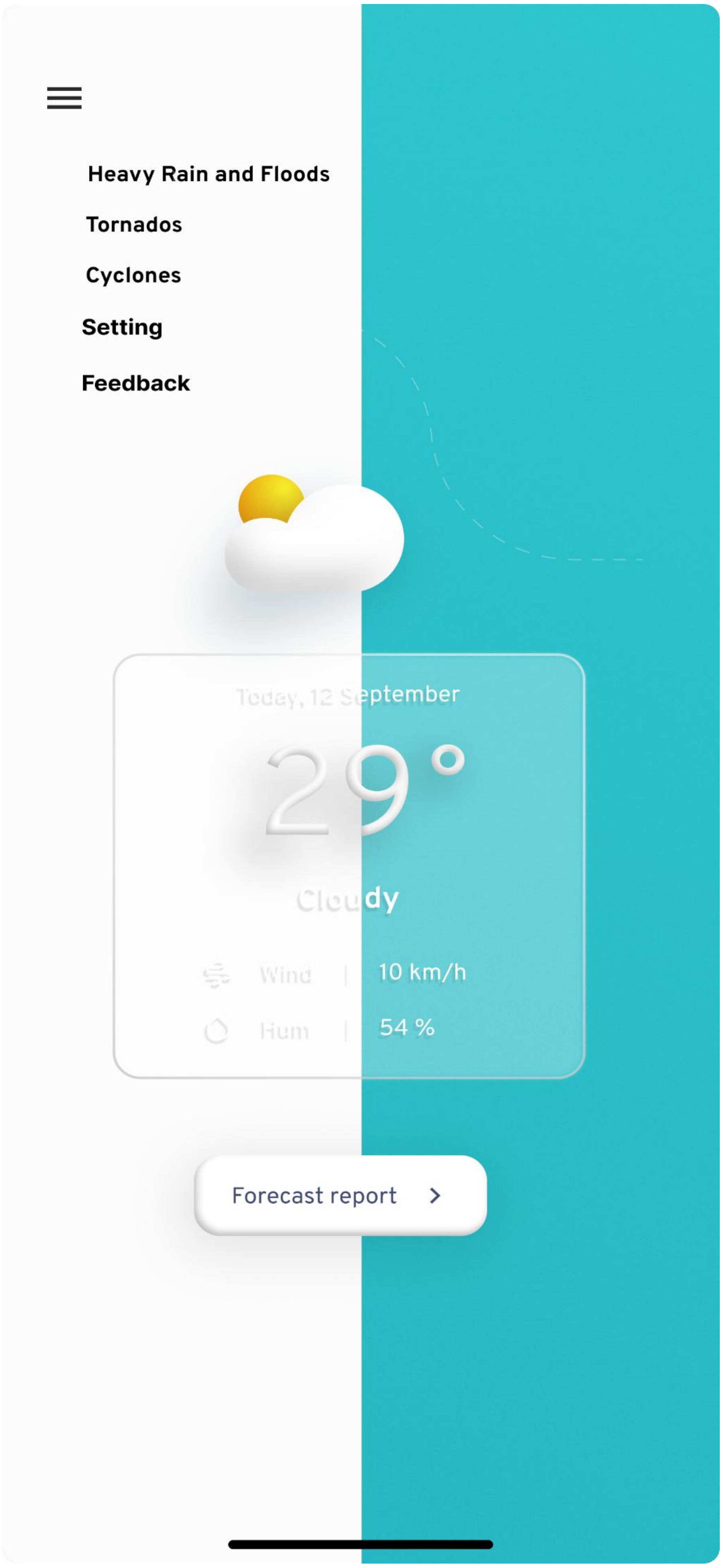

Search here

Recent search

Surabaya 34° / 23°

Banjarmasin 30° / 21°

Yogyakarta 32° / 21°


Heavy Rain and Floods
Tornados
Cyclones
Setting
Feedback
Today, 12 September
29°
Cloudy
Wind | 10 km/h
Hum | 54 %
Forecast report

Bibliography

Flutter Documentation – Google Developers. *Flutter Framework for Cross-Platform App Development.* https://flutter.dev/docs

Python for Backend Development – Python Software Foundation. *Python 3 Official Documentation.* https://docs.python.org/3/

OpenWeatherMap API – OpenWeather Ltd. *Weather Data API for Real-Time Forecasting.* https://openweathermap.org/api

China Meteorological Administration (CMA) – CMA Official Website. *Meteorological Data and Weather Alerts.* http://www.cma.gov.cn/

AI for Weather Forecasting – Deep Learning in Meteorology. *AI-Based Climate Prediction and Disaster Response.* https://arxiv.org/

Google Maps API – Google Cloud. *Geolocation and Interactive Map Integration.* https://developers.google.com/maps

Software Engineering Principles – Ian Sommerville. *Software Engineering (10th Edition), Pearson Education.*

UML and Software Architecture – Booch, G., Rumbaugh, J., Jacobson, I. *The Unified Modeling Language User Guide, Addison-Wesley.*