ENVIRONMENTAL MONITORING(APP DEVELOPMENT)

**To create a mobile app that displays real-time temperature and humidity data received from a Raspberry Pi using Python, you would typically use a mobile app development framework like Flutter, as mentioned. However, it's essential to clarify that Flutter primarily uses Dart as its programming language, not Python. If you want to use Python for mobile app development, you might consider frameworks like Kivy, BeeWare, or Pyqtdeploy.**

Designing an app for real-time environmental monitoring using Flutter involves creating a model to gather and display data from various environmental sensors. Here's a high-level overview of the process:

1.Define App Requirements:

Determine the specific environmental parameters you want to monitor, such as temperature, humidity, air quality, pollution levels, or any other relevant data.

Select Data Sources:

Identify the source of environmental data. This could involve IoT sensors, external APIs, weather stations, or other data providers.

2.Create a Flutter Project:

Start a new Flutter project using the Flutter framework, using the Flutter CLI or your preferred IDE.

3.Design the User Interface (UI):Create the UI for your app. This may include widgets for displaying environmental data, charts for historical data, and settings for user preferences.

4.Connect to Data Sources:

Integrate with your selected data sources. This may involve making HTTP requests, connecting to IoT devices, or using third-party APIs.

5.Real-Time Data Updates:

Implement real-time data updates. You can use libraries like WebSocket or StreamBuilder to handle real-time updates and refresh the UI as new data becomes available.

6.Display Data:

Display environmental data on the app's user interface. You can use Flutter widgets like Text, Charts, or custom widgets as needed to present the data in a user-friendly manner.

7.User Settings:

Implement settings where users can configure their preferences, such as units (Celsius/Fahrenheit), alerts, and notification settings.

8.Historical Data Storage:

If you want to display historical data, consider using local storage (e.g., SQLite) or cloud-based databases to store and retrieve this data.

9.Notifications and Alerts:

Set up notifications and alerts based on user preferences and predefined thresholds for environmental parameters.

10.Mapping and Visualization:Implement maps and visualizations to display the geographical distribution of environmental data, especially if you are monitoring data from multiple locations.

11.Testing:

Thoroughly test your app to ensure that it functions correctly and is responsive to real-time data updates.

12.Optimization and Performance:

Optimize your app for performance, especially when dealing with real-time data. Minimize unnecessary updates and network requests.

13.Deployment:

Prepare your app for deployment to app stores (Google Play Store, Apple App Store). Follow platform-specific guidelines for app submission.

14.Documentation:

Document your app's features, functions, and how to use it for both users and developers.

15.User Support:

Be ready to provide user support and address issues that may arise after deployment.

RECEIVING DATA FROM RASPBERRY PI:

import requests

class EnvironmentalMonitor:

def \_init\_(self, base\_url):

self.base\_url = base\_url

def get\_temperature(self):

try:

response = requests.get(f"{self.base\_url}/temperature")

if response.status\_code == 200:

return response.json()

else:

return None

except requests.exceptions.RequestException as e:

print(f"Error: {e}")

return None

def get\_humidity(self):

try:

response = requests.get(f"{self.base\_url}/humidity")

if response.status\_code == 200:

return response.json()

else:

return None

except requests.exceptions.RequestException as e:

print(f"Error: {e}")

return None

CREATING A MODEL TO REPRESENT REAL TIME TEMPERATURE AND HUMIDITY:

class EnvironmentalData:

def \_init\_(self, temperature, humidity):

self.temperature = temperature

self.humidity = humidity

def \_str\_(self):

return f"Temperature: {self.temperature}°C, Humidity: {self.humidity}%"

# Usage:

# Create an instance of the EnvironmentalData class with real-time data

real\_time\_data = EnvironmentalData(25.5, 60.0)

# Access the temperature and humidity properties

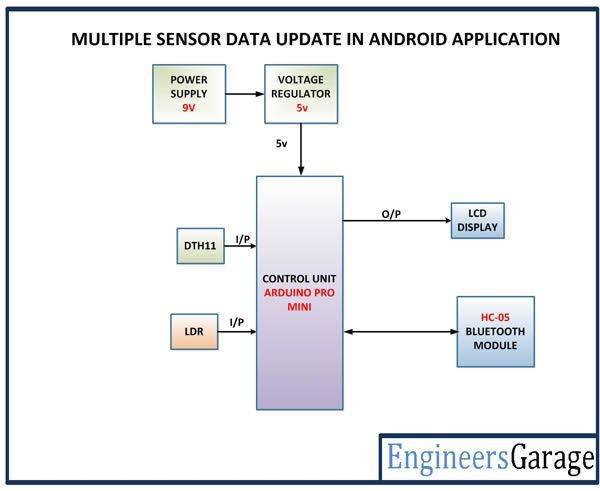
print(f"Current Temperature: {real\_time\_data.temperature}°C")

print(f"Current Humidity: {real\_time\_data.humidity}%")

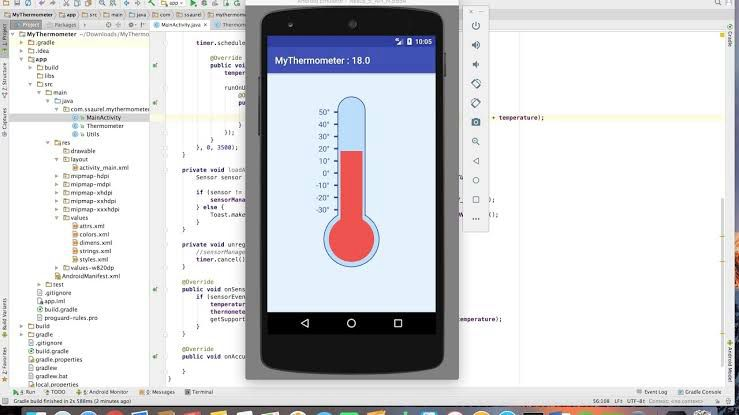
# Display the data using the \_str\_ method

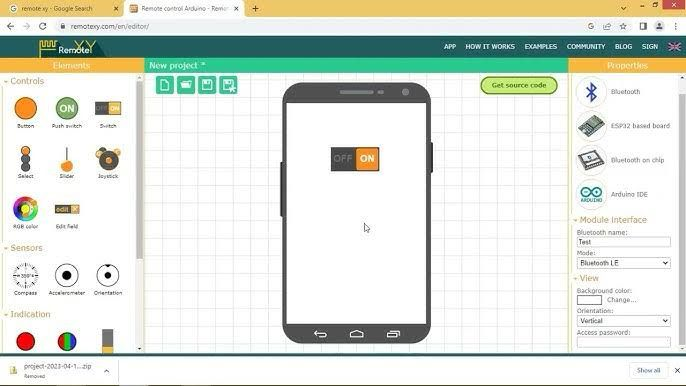
print(real\_time\_data)

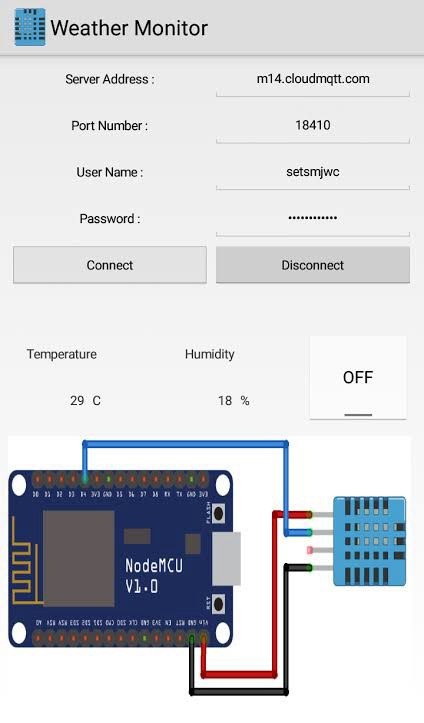
MODEL FOR ENVIRONMENTAL MONITORING:



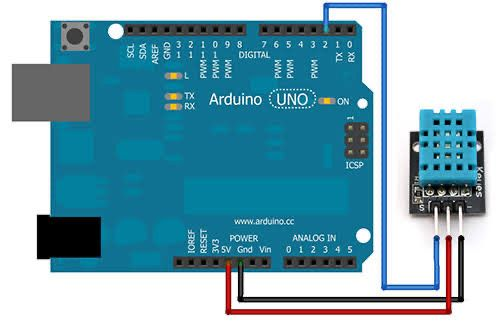
DEVELOPING THE APP:

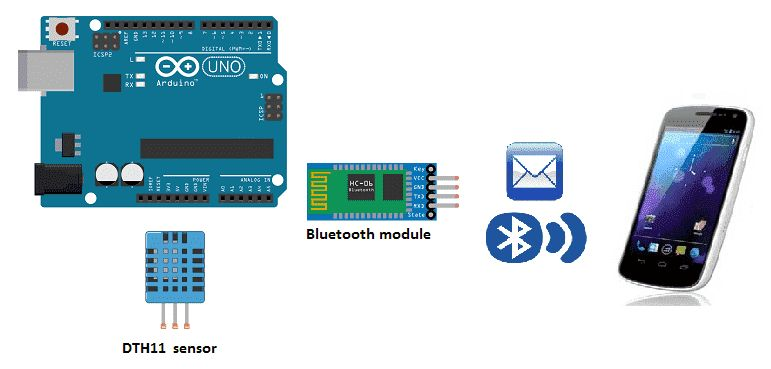






CONNECTIONS:





HUMIDITY AND TEMPERATURE MONITORING OVER THINGSPEAK:

