AIR QUALITY MONITORING SYSTEM

Building an air quality monitoring system is a complex project that typically involves collecting data Raspberry Pi from various sensors and sources. To get started, we'll create a simplified example by using a synthetic dataset. In practice, it would obtain real-time data from air quality sensors and sources, but this example will demonstrate the loading and preprocessing steps.

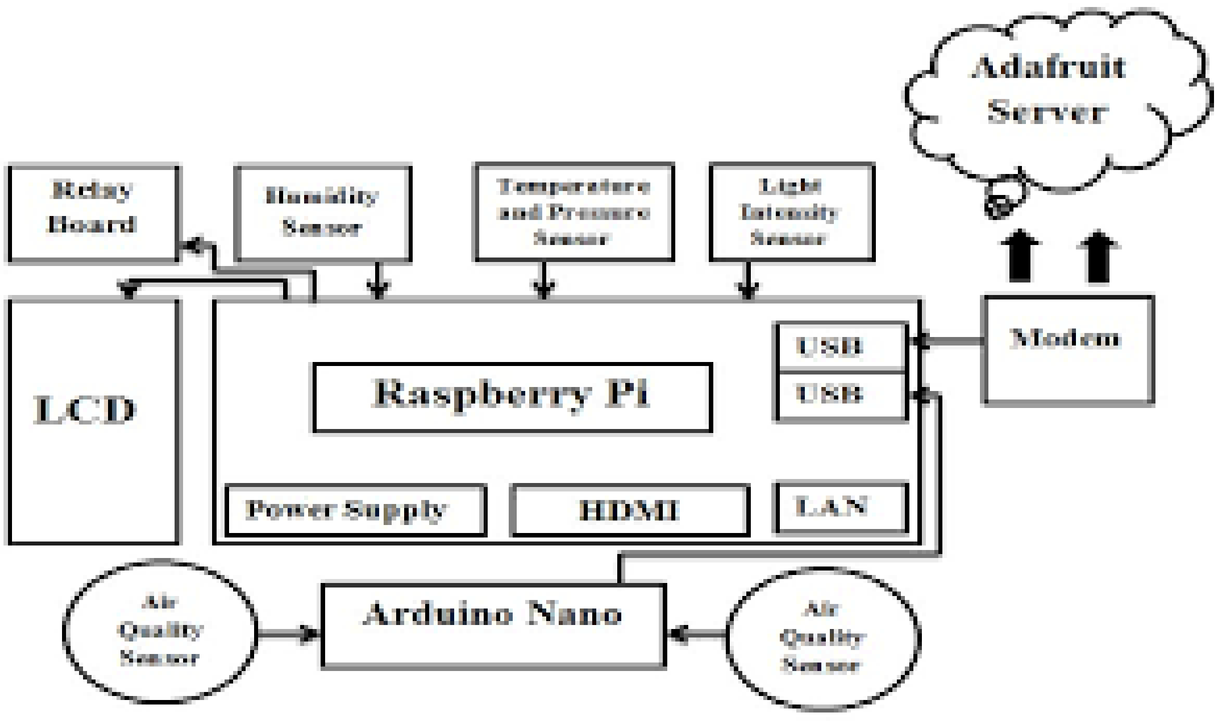
In a real air quality monitoring system, you would continuously collect data from sensors, preprocess it, and potentially store it in a database for further analysis and visualization. The preprocessing steps will depend on the specific needs of your project and the quality of data collected from sensors.

Remember that this example uses synthetic data for simplicity, but in practice, you would need to integrate with actual sensors and data sources, which may require additional data collection and preprocessing steps.

Components specifications

Hardware Specifications	**Software Specifications**
• Raspberry Pi 3 • Air Quality Sensor • Mic Sensor • Wifi Module • LCD Display • Resistors • Capacitors • Transistors • Cables and Connectors • Diodes • PCB and Breadboards • LED • Transformer/Adapter • Push Buttons • Switch • IC • IC Sockets	• Linux • Programming Language: Python • Thingspeak

BLOCK DIAGRAM



Generate a Synthetic Dataset:

First, generate a synthetic dataset to simulate air quality data. For simplicity, we'll create a Pandas DataFrame with random values. In a real project, you would connect to actual sensors or data sources.

```
import pandas as pd

import numpy as np

# Create a synthetic dataset

data = pd.DataFrame({

    'Timestamp': pd.date_range(start='2023-01-01', periods=100, freq='H'),

    'CO2 (ppm)': np.random.randint(350, 700, 100),

    'PM2.5 (µg/m³)': np.random.randint(5, 50, 100),

    'Temperature (°C)': np.random.uniform(10, 30, 100),

    'Humidity (%)': np.random.uniform(30, 70, 100)

})
```

Handling Imbalanced Data (If Applicable):

If your dataset has imbalanced classes, consider techniques like oversampling, undersampling, or using different evaluation metrics to handle the class imbalance.

These are general preprocessing steps, and the specifi steps you need to perform depend on the nature of your dataset and the algorithms you plan to use. Always tailor your preprocessing steps to the requirements of your specific project.

Exploratory Data Analysis (EDA):

. Explore the synthetic dataset to understand its structure.

```
# Display the first few rows of the dataset

print(data.head())

# Check data types and missing values

print(data.info())

# Summary statistics

print(data.describe())
```

Data Preprocessing:

In a real-world scenario, air quality data might require more extensive preprocessing, including handling missing data, smoothing sensor noise, and more. For this synthetic dataset, we'll keep it simple.

```
# No missing values in this synthetic dataset

# Smooth sensor noise (optional)

# Convert the 'Timestamp' column to the datetime format

data['Timestamp'] = pd.to_datetime(data['Timestamp'])

# Set the 'Timestamp' column as the DataFrame's index (useful for time-series data)

data.set_index('Timestamp', inplace=True)
```

You can save this synthetic dataset for further analysis or visualization.

```
data.to_csv('air_quality_data.csv')
```