Phase 4: Development part 2

Air quality monitor

Requirements:

- Web development technology. (for example, HTML, CSS, and JavaScript)
- IoT devices to collect data about air quality.
- A server that receives and stores data.
- A database is used to store data.

Design:

The platform will have two major components:

- A front-end web application that displays real-time air quality information.
- A backend server that receives and stores data from IoT devices.

HTML, CSS, and JavaScript will be used to create the front-end web application. It will leverage a WebSocket connection to the back-end server to obtain real-time updates on air quality data. The back-end server will be written in your preferred programming language, such as Python, Java, or Node.js. It will be in charge of receiving and storing data from IoT devices, as well as giving real-time data updates to the front-end web application.

Implementation

Configure the back-end server:

Installing the required prerequisites, such as a programming language runtime and a database server, is required. You'll also need to write code for the back-end server, which will be in charge of receiving and storing data from IoT devices.

Set up the front-end web application

This includes building the web application's HTML, CSS, and JavaScript files. You will also need to write code to connect to the back-end server and receive real-time air quality data updates.

Configure the IoT devices

Connecting the IoT devices to the back-end server and configuring them to deliver air quality data at regular intervals are required.

Deploy the platform

This entails putting the back-end server and front-end web application into production.

Web Application

HTML, CSS, and JavaScript are used to create the web application. It receives real-time air quality data from the server over the WebSocket API.

A JavaScript library is also used by the web application to display the air quality data on a map.

The following is a high-level summary of web application architecture:

[WebSocket connection] -> [Web app] -> [Map]

The WebSocket connection is used by the web application to receive air quality data from the server. Using the JavaScript library, the web application then plots the air quality data on a map.

Front-end code:

```
<!DOCTYPE html>
<html>
<head>
  <title>Real-Time Air Quality Monitoring Platform</title>
  <script src="https://cdn.jsdelivr.net/npm/socket.io/client/dist/socket.io.js"></script>
  <script src="script.js"></script>
</head>
<body>
  <h1>Real-Time Air Quality Monitoring Platform</h1>

  <div id="air_quality"></div>

  <script>
    // Create a WebSocket connection to the back-end server
    var ws = new WebSocket("ws://localhost:5000/api/air_quality/ws");

    // Receive updates on the air quality data from the back-end server
    ws.onmessage = function(event) {
      var data = JSON.parse(event.data);
```

```
// Update the air quality display
document.getElementById("air_quality").innerHTML
```

Server:

The server is written in Python and built with the Flask framework. It creates a REST API using the Flask-RESTful extension that accepts air quality data from IoT devices and sends it to the web application.

The server also stores the air quality data and other information in a database, such as the location of the air quality sensors and the time the data was collected.

A high-level overview of the server architecture follows:

[Internet of Things devices] -> [Server] -> [Web application]

The REST API is used by IoT devices to communicate air quality data to the server. The data is stored in the database by the server and then broadcasted to the web application via a WebSocket connection.

The web application receives and shows real-time air quality data from the server over a WebSocket connection.

Back-end code:

```
# Back-end server

from flask import Flask, render_template, request
import json

app = Flask(__name__)

# Database to store the air quality data
db = {}

# WebSocket connection to the front-end web application
ws = None

# Receive air quality data from IoT devices
@app.route("/api/air_quality", methods=["POST"])
def receive_air_quality_data():
    data = json.loads(request.data)
```

```
# Store the data in the database
    db[data["sensor_id"]] = data

    # Send the data to the front-end web application
    ws.send(json.dumps(data))

# Send real-time updates on the air quality data to the front-end web application
def send_air_quality_updates():
    while True:
        # Get the latest air quality data from the database
        data = db

        # Send the data to the front-end web application
        ws.send(json.dumps(data))

if __name__ == "__main__":
    # Start the WebSocket server
    ws = app.socket("/api/air_quality/ws")

    # Start the Flask server
    app.run(debug=True)
```