Project Title: Environmental Monitoring

Phase 4: Development Part 2

To continue building an environmental monitoring platform that displays real-time temperature and humidity data from IoT devices, let's expand on the previous code example. We'll simulate the server's behaviour for data emission. For this demonstration, I'll create a basic simulation using HTML,CSS and JavaScript with random data for temperature and humidity.

HTML (index.html):

```
<!DOCTYPE html>
<html>
<head>
    <title>Arduino Data Display</title>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.3.2/socket.io.js"></script>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <h1>Temperature and Humidity Data</h1>
    <div class="data-display">
        <p>Temperature (°C): <span id="tempValue">--</span></p>
        <p>Humidity (%): <span id="humidityValue">--</span></p>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

CSS (styles.css):

```
body {
    font-family: Arial, sans-serif;
    text-align: center;
    margin: 20px;
```

```
}

h1 {
    color: #333;
}

.data-display {
    text-align: left;
    margin: 20px;
}

p {
    font-size: 18px;
    margin-bottom: 8px;
}
```

JavaScript (script.js):

```
document.addEventListener('DOMContentLoaded', () => {
    // Create a WebSocket connection
    const socket = io('http://localhost:3000'); // Replace with your server URL

    // Listen for 'data' event emitted by the server
    socket.on('data', (data) => {
        const { temperature, humidity } = data; // Data received from Arduino

        // Display the temperature and humidity data on the web page
        document.getElementById('tempValue').textContent = temperature;
        document.getElementById('humidityValue').textContent = humidity;
    });
});
```

Server-side (Node.js for WebSocket Server):

```
const app = require('express')();
const http = require('http').createServer(app);
const io = require('socket.io')(http);

io.on('connection', (socket) => {
  console.log('A user connected');

  socket.on('disconnect', () => {
    console.log('User disconnected');
  });

  socket.on('data', (data) => {
    io.emit('data', data); // Broadcast received data to all connected clients
  });
});

http.listen(3000, () => {
  console.log('Server listening on port 3000');
});
```

Code Overview:

This HTML, CSS, and JavaScript code creates a basic web page that displays temperature and humidity data received from a WebSocket server, presumably connected to an Arduino or similar device.Here's a breakdown of what each part of the code does:

HTML (index.html):

- **Document Type Declaration (`<!DOCTYPE html>`):** Specifies the document type and version.

- HTML Structure: Basic structure of an HTML document with the `<html>`, `<head>`, and `<body>` tags.

- **Title (`<title>`) and External Scripts/Styles:** Sets the title of the web page and links external scripts and a stylesheet.

- **Content:** Contains an `<h1>` heading for title and a `<div>` container for displaying temperature and humidity data. The data is shown within `<p>` tags and specific values are updated using `<span>` tags with unique IDs (`tempValue` and `humidityValue`).

CSS (styles.css):

- **Styling:** Defines basic styles for the HTML elements to create a visually appealing layout.

JavaScript (script.js):

- **Event Listener (`DOMContentLoaded`):** Waits for the HTML document to load before executing JavaScript.

- **WebSocket Connection:** Initiates a connection to a WebSocket server (assuming it's running on `http://localhost:3000`).

- **Event Listener for Data Reception:** Listens for the 'data' event emitted by the server.

- **Updating Data Display:** When data is received, it updates the temperature and humidity values on the web page by selecting the respective elements using their IDs and setting their text content.

Server-side (Node.js for WebSocket Server):

- The JavaScript file now initiates a simulated data stream for temperature and humidity every 3 seconds. It updates the displayed values and emits this data to the WebSocket server.
- The server-side code creates a basic WebSocket server that listens for connections and data from IoT devices and broadcasts it to all connected clients.

Conclusion:

This example provides a foundation for an environmental monitoring platform. In a real-world scenario, the server-side logic would be replaced with code to receive actual data from IoT devices. The simulated data and emission to the server in this code can be replaced with actual data received from sensors attached to IoT devices.