**Phase 5**

**Project Title: Flood Monitoring and Early Warning System**

**Project Objectives**

* **Objective:** Develop a real-time flood monitoring and early warning system.
* **Aim:** Enhance public safety and improve emergency response coordination during flood events.

**IoT Sensor Deployment**

Sensor Selection

* Selected IoT sensors capable of monitoring water levels.
* Chose sensors suitable for outdoor use and data transmission.

Sensor Placement

* Strategically deployed sensors near water bodies, rivers, and flood-prone areas.
* Ensured optimal sensor placement for accurate data collection.

Data Transmission

* Configured sensors to transmit data to a central server using communication protocols (e.g., Wi-Fi, LoRa, cellular networks).
* Established reliable data transmission mechanisms.

**Central Data Processing**

Data Processing Platform

* Set up a central data processing platform (server) to receive and process sensor data.

Data Ingestion

* Implemented an API or data ingestion mechanism to receive data from IoT sensors.

Data Storage

* Utilized a secure and scalable database system for storing received data (e.g., MySQL, MongoDB).

Data Processing with Python

* Developed Python scripts for data processing and storage.
* Ensured data quality checks and data structuring.

**Flood Prediction and Early Warning Logic**

Machine Learning

* Employed machine learning algorithms using Python libraries (e.g., scikit-learn, TensorFlow) to analyze historical data and predict floods.

Warning Thresholds

* Defined specific thresholds and rules for issuing flood warnings based on analysis.

Warning Issuance

* Developed Python logic to issue flood warnings and generate alerts.
* Implemented notifications to relevant authorities and the public.

**Early Warning Platform**

Web Development

* Created a web-based early warning platform using web development technologies (HTML, CSS, JavaScript).
* Utilized Python frameworks (e.g., Flask, Django) for server-side development.

Real-Time Data Display

* Implemented JavaScript to fetch and display real-time water level data from the central server.
* Used charting libraries (e.g., Chart.js) for data visualization.

Flood Warnings

* Periodically checked for flood warnings and displayed notifications on the platform.

**Integration**

API Integration

* Enabled API calls between the early warning platform and the central server for real-time data updates and flood warning information.

Websockets

* Implemented WebSockets or server-sent events for real-time communication between the platform and the central server.

Notifications

* Integrated notification services (e.g., email, SMS, push notifications) to alert users when flood warnings are issued.

**Testing and Deployment**

Testing

* Thoroughly tested the integration and functionality of the system.
* Simulated various scenarios, including normal conditions and flood situations using mock data.

Deployment

* Deployed the entire system, including IoT sensors, central data processing, and the early warning platform, on the chosen infrastructure (e.g., cloud hosting, on-premises servers).

Maintenance and Monitoring

* Set up monitoring and alerting systems to detect and address issues proactively.
* Established a maintenance schedule to ensure security and reliability.

**Enhancing Public Safety**

The real-time flood monitoring and early warning system enhances public safety by:

* Providing timely flood warnings to the public and relevant authorities.
* Improving emergency response coordination during flood events.
* Enabling informed decision-making for disaster management.

**Submission**

**GitHub Repository:** [Include your GitHub repository link here]

**Replication Instructions:**

To replicate this project, follow these steps:

1. Deploy IoT sensors at strategic locations near water bodies.
2. Set up a central server for data reception and processing. Implement APIs for data ingestion.
3. Develop Python scripts for data processing, analysis, and flood warning logic.
4. Create a web-based early warning platform using web development technologies and Python.
5. Integrate the platform with the central server for real-time data updates and notifications.
6. Test the system with mock data to ensure functionality.
7. Deploy the entire system to your chosen infrastructure (cloud or on-premises).
8. Maintain and monitor the system for continuous operation.

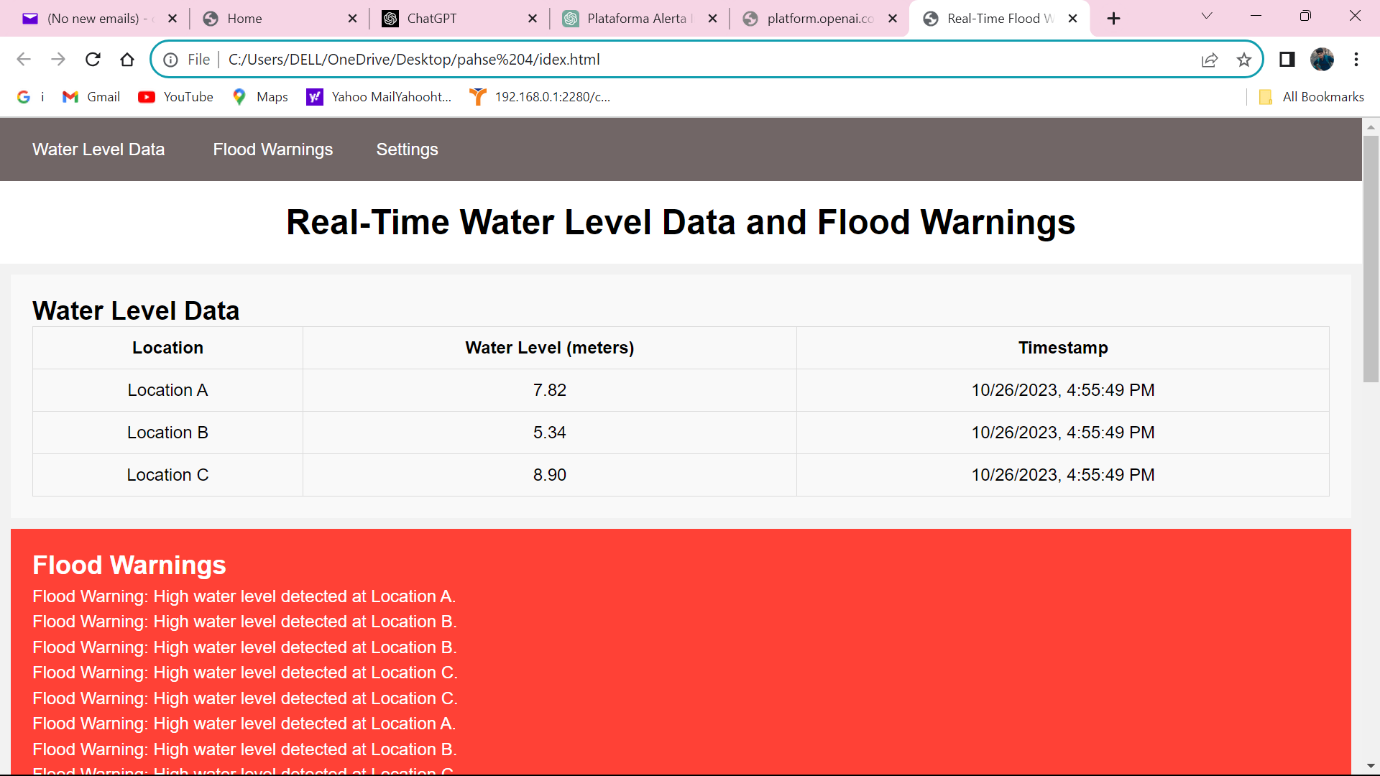
**HTML Code (index.html)**

1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title>Real-Time Flood Warning System</title>
5. <link rel="stylesheet" type="text/css" href="styles.css">
6. </head>
7. <body>
8. <nav class="nav-2">
9. <a href="#data-display" class="nav-1">Water Level Data</a>&nbsp;
10. <a href="#warning-section"class="nav-1">Flood Warnings</a>
11. <a href="#settings" class="nav-1">Settings</a>
13. </nav>
14. <header>
15. <h1>Real-Time Water Level Data and Flood Warnings</h1>
16. </header>
17. <section id="data-display">
18. <h2>Water Level Data</h2>
19. <table>
20. <thead>
21. <tr>
22. <th>Location</th>
23. <th>Water Level (meters)</th>
24. <th>Timestamp</th>
25. </tr>
26. </thead>
27. <tbody id="data-table">
28. <!-- Real-time data will be displayed here -->
29. </tbody>
30. </table>
31. </section>
32. <section id="warning-section">
33. <h2>Flood Warnings</h2>
34. <ul id="warnings-list">
35. <!-- Real-time flood warnings will be displayed here -->
36. </ul>
37. </section>
38. <section id="settings">
39. <h2>System Settings</h2>
40. <form id="settings-form">
41. <label for="server-url">Server URL:</label>
42. <input type="text" id="server-url" name="server-url" placeholder="Enter the server URL" required>            <label for="sensor-threshold">Sensor Threshold (meters):</label>
43. <input type="number" id="sensor-threshold" name="sensor-threshold" step="0.01" placeholder="Enter the sensor threshold" required>
44. <label for="notification-interval">Notification Interval (seconds):</label>
45. <input type="number" id="notification-interval" name="notification-interval" placeholder="Enter the notification interval" required>
47. <button type="submit">Save Settings</button>
48. </form>
49. </section>
50. <div id="water-level-chart">
51. <!-- JavaScript will populate and render the chart here -->
52. </div>
53. <section id="notifications">
54. <h2>Notifications</h2>
55. <ul>
56. <li>System is operational.</li>
57. <!-- Display real-time flood warnings here -->
58. </ul>
59. </section>
60. <footer>
61. <div style="align-content: center;">
62. <p>&copy; 2023 YourCompany. All rights reserved.</p>
63. <p><a href="privacy-policy.html">Privacy Policy</a> | <a href="terms-of-service.html">Terms of Service</a></p></div>
64. </footer>
65. <script>
66. // Simulate fetching and updating real-time data
67. function fetchData() {
68. // Simulate data from IoT sensors (you should replace this with real data)
69. const locations = ["Location A", "Location B", "Location C"];
70. const waterLevels = [Math.random() \* 10, Math.random() \* 10, Math.random() \* 10];
71. // Get the data table element
72. const dataTable = document.getElementById("data-table");
73. // Clear previous data
74. dataTable.innerHTML = '';
75. // Loop through the data and populate the table
76. for (let i = 0; i < locations.length; i++) {
77. const row = document.createElement("tr");
78. row.innerHTML = `
79. <td>${locations[i]}</td>
80. <td>${waterLevels[i].toFixed(2)}</td>
81. <td>${new Date().toLocaleString()}</td>
82. `;
83. dataTable.appendChild(row);
84. // Check for flood warnings (you can set your own thresholds)
85. if (waterLevels[i] > 7.0) {
86. const warningsList = document.getElementById("warnings-list");
87. const warning = document.createElement("li");
88. warning.textContent = `Flood Warning: High water level detected at ${locations[i]}.`;
89. warningsList.appendChild(warning);
90. }
91. }
92. }
93. document.getElementById("settings-form").addEventListener("submit", function (event) {
94. event.preventDefault(); // Prevent the default form submission
95. // Get values from the input fields
96. const serverUrl = document.getElementById("server-url").value;
97. const sensorThreshold = parseFloat(document.getElementById("sensor-threshold").value);
98. const notificationInterval = parseInt(document.getElementById("notification-interval").value);
99. // Use the values to update system settings (you can send them to the server or store them locally)
100. // For example, display the updated settings in the console
101. console.log("Server URL:", serverUrl);
102. console.log("Sensor Threshold:", sensorThreshold);
103. console.log("Notification Interval:", notificationInterval);
104. // You can also send these settings to your server for further processing
105. // Optionally, provide feedback to the user that settings were saved
106. alert("Settings saved successfully!");
107. }); // Update data every 5 seconds (adjust the interval as needed)
108. setInterval(fetchData, 5000);
109. </script>
110. </body>
111. </html>

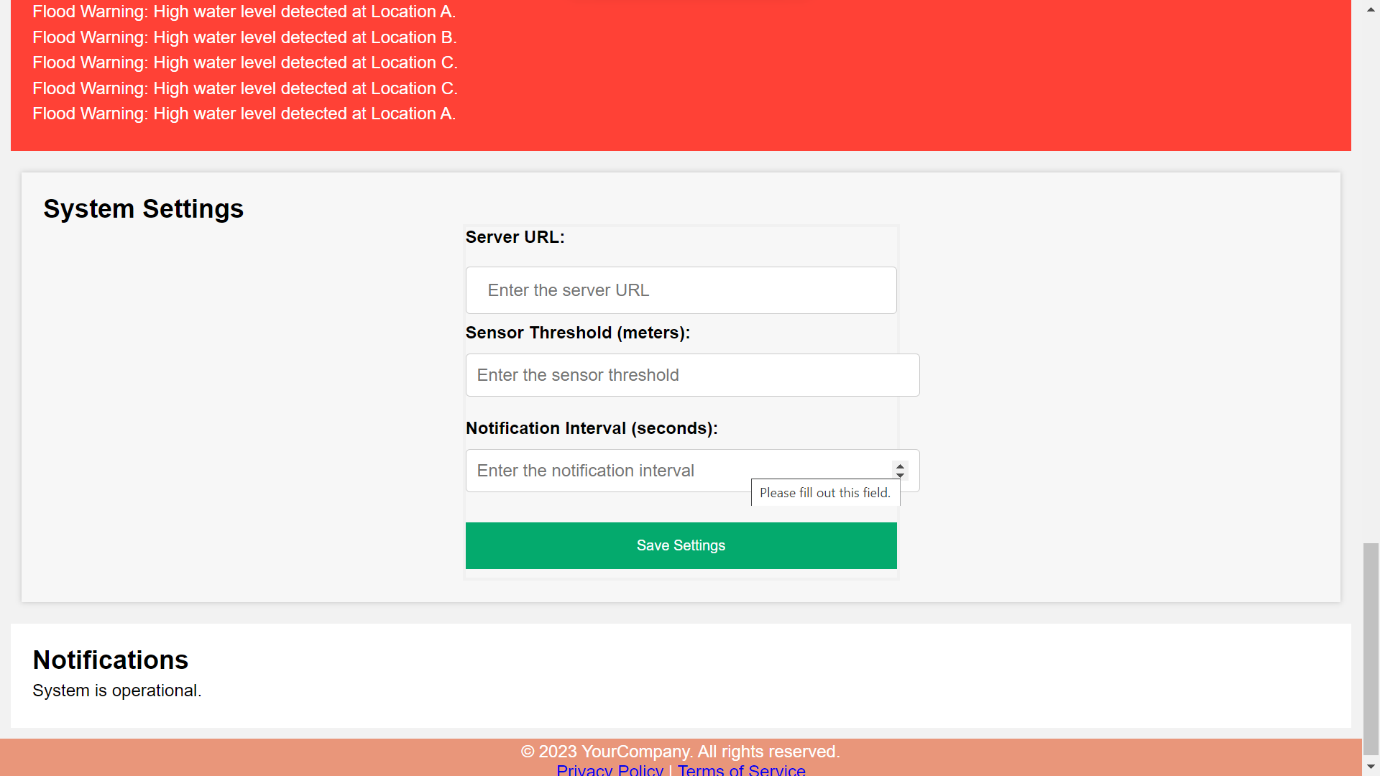
**CSS Code : (style.css)**

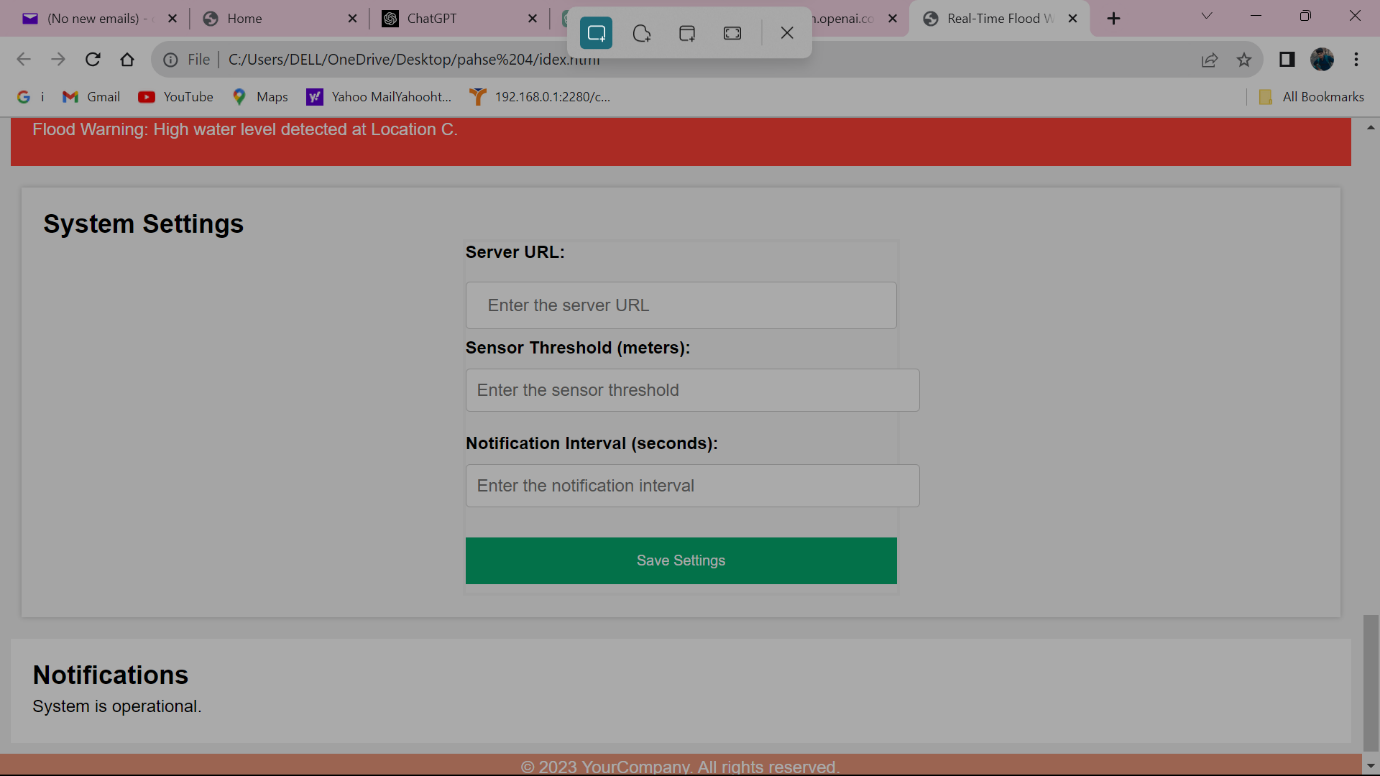
1. /\* CSS styles for your Real-Time Flood Warning System \*/
2. /\* Reset some default browser styles \*/
3. body, h1, h2, ul, li, p {
4. margin: 0;
5. padding: 0;
6. }
7. /\* Define general styles for your page \*/
8. body {
9. font-family: Arial, sans-serif;
10. background-color: #f2f2f2;
11. }
12. header {
13. background-color: white;
14. color:  black;
15. padding: 20px;
16. text-align: center;
17. }
18. h1 {
19. margin: 0;
20. }
21. section {
22. background-color: #fff;
23. padding: 20px;
24. margin: 10px;
25. }
26. table {
27. width: 100%;
28. border-collapse: collapse;
29. }
30. table, th, td {
31. border: 1px solid #ddd;
32. }
33. th, td {
34. padding: 10px;
35. text-align: center;
36. }
37. ul {
38. list-style-type: none;
39. padding: 0;
40. }
41. li {
42. margin: 5px 0;
43. }
44. #data-display {
45. background-color: #f9f9f9;
46. }
47. #warning-section {
48. background-color: #FF4136;
49. color: #fff;
50. }
51. #data-table th {
52. background-color: #333;
53. color: #fff;
54. }
55. .nav-2 {
56. background-color: #716666;
57. display: flex;
58. justify-content: left;
59. padding: 10px;
60. }
61. .nav-1 {
62. text-decoration: none;
63. color: #fff;
64. padding: 10px 20px;
65. transition: background-color 0.3s;
66. }
67. /\* Change link color on hover \*/
68. .nav-1:hover {
69. background-color: #555; /\* Background color on hover \*/
70. }
71. /\* Style the system settings section \*/
72. #settings {
73. background-color: #f7f7f7;
74. padding: 20px;
75. margin: 20px;
76. box-shadow: 0 0 5px rgba(0, 0, 0, 0.2);
77. }
78. /\* Style form elements \*/
79. #settings-form {
80. max-width: 400px;
81. margin: 0 auto;
82. }
83. label {
84. display: block;
85. margin-bottom: 10px;
86. font-weight: bold;
87. }
88. input[type="text"],
89. input[type="number"] {
90. width: 100%;
91. padding: 10px;
92. margin-bottom: 20px;
93. border: 1px solid #ccc;
94. border-radius: 4px;
95. font-size: 16px;
96. }
97. input[type="submit"] {
98. background-color: #0074D9;
99. color: #fff;
100. border: none;
101. padding: 10px 20px;
102. cursor: pointer;
103. }
104. input[type="submit"]:hover {
105. background-color: #0056b3;
106. }
107. body {font-family: Arial, Helvetica, sans-serif;}
108. form {border: 3px solid #f1f1f1;}
109. input[type=text], input[type=password] {
110. width: 100%;
111. padding: 12px 20px;
112. margin: 8px 0;
113. display: inline-block;
114. border: 1px solid #ccc;
115. box-sizing: border-box;
116. }
117. button {
118. background-color: #04AA6D;
119. color: white;
120. padding: 14px 20px;
121. margin: 8px 0;
122. border: none;
123. cursor: pointer;
124. width: 100%;
125. }
126. button:hover {
127. opacity: 0.8;
128. }
129. .cancelbtn {
130. width: auto;
131. padding: 10px 18px;
132. background-color: #f44336;
133. }
134. .imgcontainer {
135. text-align: center;
136. margin: 24px 0 12px 0;
137. }
138. img.avatar {
139. width: 40%;
140. border-radius: 50%;
141. }
142. .container {
143. padding: 16px;
144. }
145. span.psw {
146. float: right;
147. padding-top: 16px;
148. }
149. /\* Change styles for span and cancel button on extra small screens \*/
150. @media screen and (max-width: 300px) {
151. span.psw {
152. display: block;
153. float: none;
154. }
155. .cancelbtn {
156. width: 100%;
157. }
158. }
159. footer {
160. text-align: center;
161. padding: 3px;
162. background-color: DarkSalmon;
163. color: white;
164. }

**Result :**

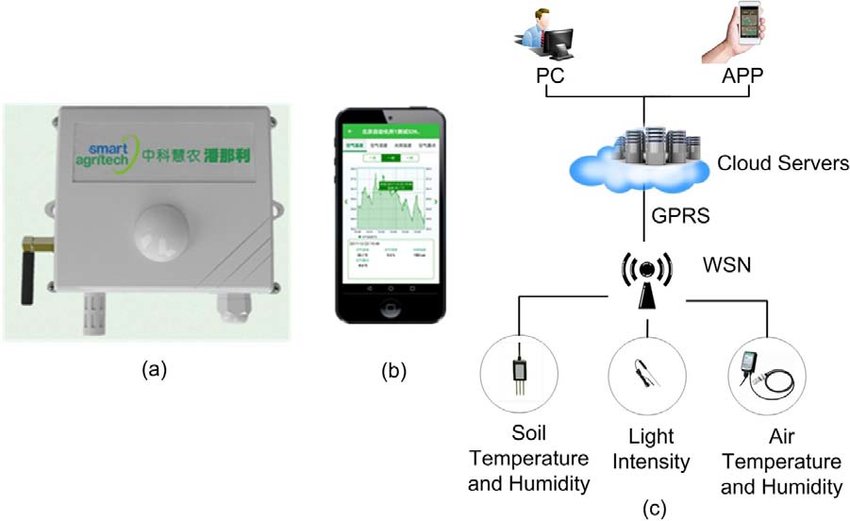


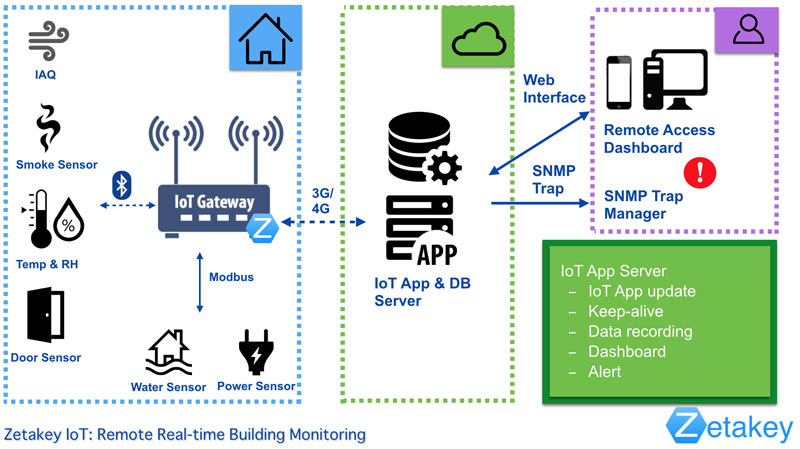


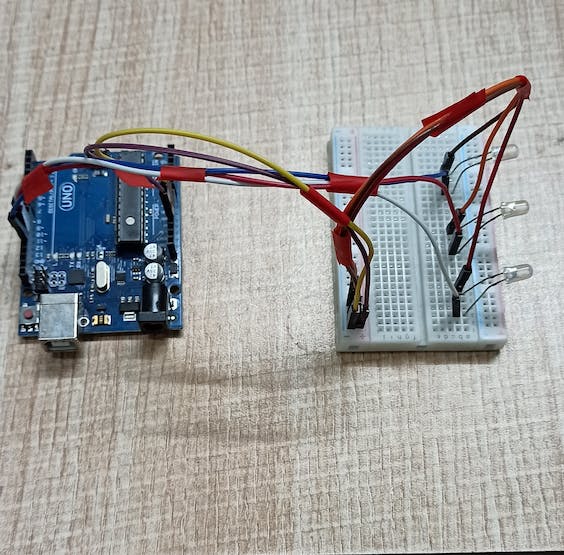
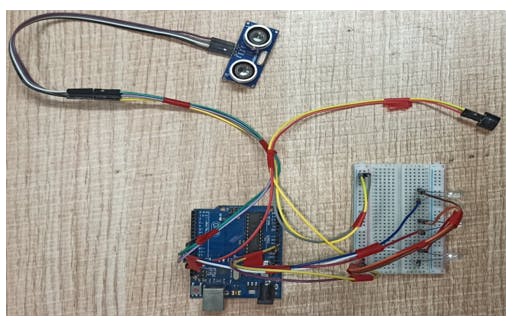




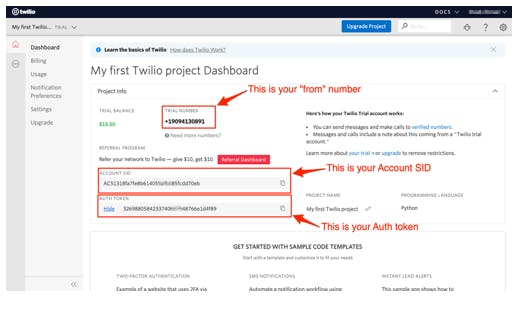
**Environmental sensors :**

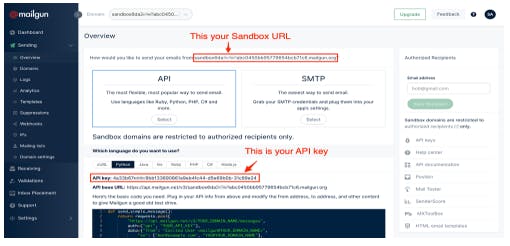




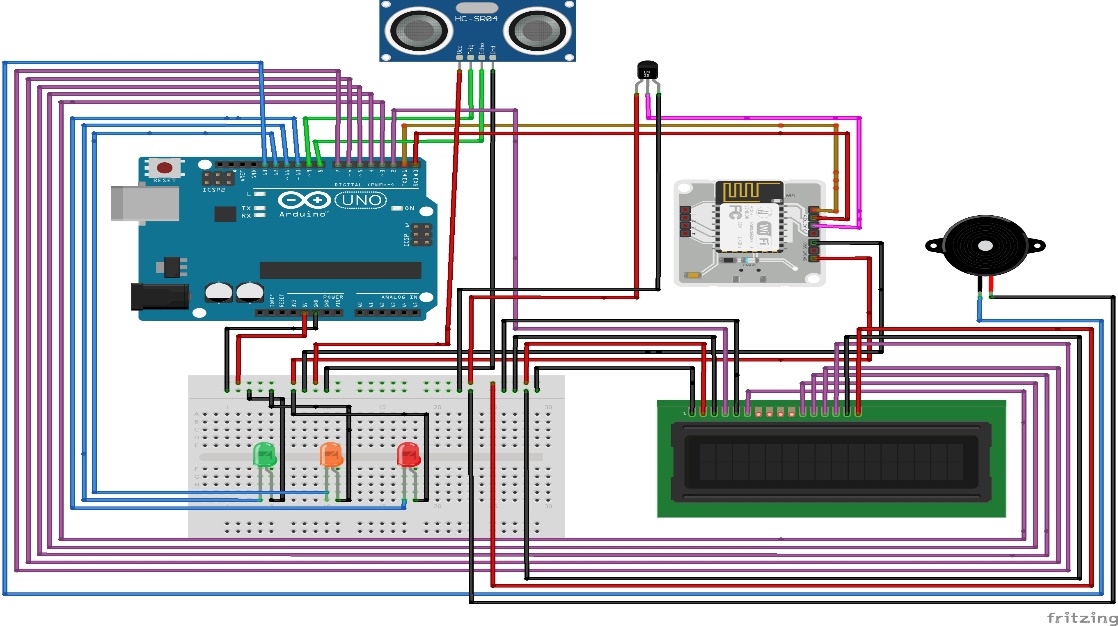


**App Development**





**Schematics Diagram of the Project**



**Code for conf.py**

#twillo details for sending alert sms

SID = 'You can find SID in your Twilio Dashboard'

AUTH\_TOKEN = 'You can find on your Twilio Dashboard'

FROM\_NUMBER = 'This is the no. generated by Twilio. You can find this on your Twilio Dashboard'

TO\_NUMBER = 'This is your number. Make sure you are adding +91 in beginning'

#bolt iot details

API\_KEY = 'XXXXXXXXX' #This is your Bolt cloud API

Key.

DEVICE\_ID = 'BOLTXXXXXXXXX' #This is the ID of your Bolt device.

#mailgun details for sending alert E-mails

MAILGUN\_API\_KEY = 'This is the private API key which you can find on your Mailgun Dashboard'

SANDBOX\_URL= 'You can find this on your Mailgun Dashboard'

SENDER\_EMAIL = 'test@ + SANDBOX\_URL' # No need to modify this. The sandbox URL is of the format test@YOUR\_SANDBOX\_URL

RECIPIENT\_EMAIL = 'Enter your Email ID Here'

