МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна система дистанційного контролю і керування поливом рослин»
здобувача групи ІН - 93 Аль-Шававреx Ахмад Хакем

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Ахмад Аль-Шававреx
_____
(підпис)

Керівник, доцент, кандидат
технічних наук Петров Сергій _____
(підпис)

Суми – 2023

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-93 Аль-Шававрех Ахмад Хакем

1. Тема роботи: «Інформаційна система дистанційного контролю і керування поливом рослин» затверджую наказом по СумДУ від « »
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми водного управління та постановка завдань дослідження.
2) Аналіз проблеми водного управління та постановка завдань дослідження. 3) Розробка інтелектуальної системи для ефективного управління водними ресурсами. 4) Аналіз результатів дослідження.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз вимог*		
2	*Проектування системи*		
3	*Проектування системи*		
4	*Тестування та налагодження*		
5	*Завершення та написання звіту*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 67 стр., 1 рис., 1 додаток, 10 використаних джерел.

Обґрунтування актуальності теми роботи – Зростаючі проблеми водних ресурсів та потреба в сталому водокористуванні роблять тему роботи про Водне управління з Arduino актуальною.

Об’єкт дослідження — розробка системи управління водними ресурсами з використанням Arduino для ефективного використання водних ресурсів у житлових умовах.

Мета роботи — розробка та впровадження надійної системи управління водними ресурсами з використанням Arduino, що забезпечує автоматичне наповнення резервуарів водою та зрошення рослин для сприяння сталому використанню води в будинках.

Методи дослідження — аналіз сенсорних даних, програмування на Arduino та виконання експериментів для валідації ефективності водного управління.

Результати — успішну реалізацію водного управління з використанням Arduino, точне заповнення водних баків і автоматичне поливання рослин, що сприяє ефективному використанню водних ресурсів та підтримує сталий розвиток в домашньому середовищі.

ЗМІСТ

ВСТУП

Актуальність

В сучасному світі проблема ефективного управління водними ресурсами є дедалі більш актуальною. Зростаюча свідомість про необхідність збереження води та раціонального її використання вимагає розробки нових технологій та систем управління, що забезпечують оптимальне використання водних ресурсів в різних сферах, включаючи побутове використання та полив рослин.

Об’єкт дослідження

Об’єктом дослідження є процес управління водними ресурсами для заповнення водних баків в будинках та поливу рослин. Основна увага зосереджена на розробці системи, що забезпечує автоматизацію цих процесів та ефективне використання води.

Предмет дослідження

Предметом дослідження є розробка та реалізація інтелектуальної системи для управління водними ресурсами, що використовує платформу Arduino. Система передбачає контроль рівнів води у баках та автоматичний полив рослин, що дозволяє ефективно використовувати доступну воду та забезпечувати оптимальні умови поливу.

Гіпотеза

Гіпотеза даної роботи полягає в тому, що розроблена інтелектуальна система для управління водними ресурсами на основі Arduino здатна забезпечити ефективне використання води, заповнення водних баків у будинках та полив рослин з урахуванням їх потреб.

Новизна

Новизною даної роботи є поєднання технологій автоматизації заповнення водних баків та поливу рослин з використанням платформи Arduino. Розроблена система пропонує інноваційний підхід до управління водними ресурсами, що може забезпечити збалансоване та ефективне використання води.

Структура

Робота складається з наступних розділів:

- Розділ 1: Аналіз проблеми та постановка завдань дослідження.
- Розділ 2: Огляд технологій, що використовуються для управління водними ресурсами.
- Розділ 3: Розробка та реалізація інтелектуальної системи управління водними ресурсами на основі платформи Arduino.
- Розділ 4: Аналіз отриманих результатів та оцінка ефективності розробленої системи.
- Розділ 5: Висновки та рекомендації щодо подальших досліджень у цій області.

Кожен розділ міститиме відповідні теоретичні та практичні аспекти, необхідні для досягнення поставлених цілей та вирішення завдань дослідження.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Основні поняття, терміни та відомості

1. Управління водними ресурсами: Це процес планування, контролю та ефективного використання водних ресурсів для задоволення потреб людей, промисловості та сільськогосподарської діяльності.
2. Arduino: Це відкрите апаратне та програмне середовище для створення електронних пристроїв. Arduino платформа використовується для розробки інтелектуальної системи управління водними ресурсами в даній роботі.
3. Водні баки: Це контейнери або резервуари, призначені для зберігання води. Вони використовуються в домашньому господарстві для забезпечення водопостачання та поливу рослин.
4. Полив рослин: Це процес надання води рослинам для їхнього зростання та розвитку. Полив може проводитись різними методами, включаючи автоматичний полив, який розробляється в даній роботі.
5. Ефективне використання води: Це оптимальне використання водних ресурсів з метою забезпечення необхідних потреб, при цьому мінімізуючи втрати та зберігаючи воду для майбутнього використання.
6. HTML (HyperText Markup Language): Це мова розмітки, використовувана для створення структури та вигляду веб-сторінок.
7. CSS (Cascading Style Sheets): Це мова стилів, яка використовується для оформлення веб-сторінок та керування їх зовнішнім виглядом.
8. JavaScript: Це мова програмування, що використовується для створення динамічного та інтерактивного контенту на веб-сторінках.
9. Веб-браузер: Це програмне забезпечення, призначене для відображення веб-сторінок та навігації в Інтернеті.
10. Backend: Це частина веб-додатку, яка займається обробкою даних та взаємодіє з базою даних. У даній роботі використовується мова

програмування C для розробки backend-функціоналу.

Ці поняття та терміни є важливими для розуміння та реалізації проекту з управління водними ресурсами, включаючи розробку веб-інтерфейсу з використанням HTML, CSS та JavaScript, інтеграцію з веб-браузером та розробку backend-логіки на мові програмування C для Arduino.

1.2 Аналіз аналогічних проєктів

Web-сервер на основі Arduino з використанням мікрохвильових сенсорів для вимірювання відстані та здатністю функціонувати як хостер для фронтенду та бекенду, а також підключенням до API веб-сервісів.

1. Arduino та мікрохвильові сенсори: Використання Arduino та мікрохвильових сенсорів дозволяє точно виміряти відстань та здійснювати контроль водних ресурсів з високою точністю та надійністю.
2. Функції веб-сервера: Arduino може бути налаштований як веб-сервер, що дозволяє розробляти та розгортати фронтенд та бекенд додатки безпосередньо на самому пристрої. Це дозволяє забезпечити зручний доступ та управління системою через веб-інтерфейс.
3. Підключення до API веб-сервісів: Arduino може встановлювати з'єднання з різними API веб-сервісів, що відкриває можливості для обміну даними та інтеграції з іншими системами.

Використання Arduino як веб-сервера з мікрохвильовими сенсорами та здатністю підключення до API веб-сервісів дозволяє створити розширений та гнучкий систему управління водними ресурсами з можливістю дистанційного контролю та оптимізації.

1.3 Постановка задачі

Метою роботи є розробка системи управління водними ресурсами на базі Arduino з використанням мікрохвильових сенсорів та функцій веб-сервера.

Для досягнення поставленої мети необхідно вирішити наступні задачі:

1. Розробити алгоритм вимірювання відстані з використанням мікрохвильових сенсорів.
2. Побудувати веб-сервер на основі Arduino для забезпечення доступу та управління системою через веб-інтерфейс.
3. Реалізувати функцію збереження та аналізу даних про водні ресурси.
4. Забезпечити можливість підключення до API веб-сервісів для обміну даними та інтеграції з іншими системами.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Математична (або інформаційна) модель

Для вимірювання відстані за допомогою мікрохвильових сенсорів SR-04 використовується математична модель, яка базується на властивостях ультразвукових хвиль та розрахунках затримки сигналу.

Методологія розрахунку відстані полягає у наступних кроках:

1. Відправлення ультразвукового сигналу: Мікроконтролер Arduino відправляє короткий ультразвуковий сигнал за допомогою мікрохвильового сенсора SR-04.
2. Отримання ехо-сигналу: Сенсор сприймає ехо-сигнал, який відбивається від перешкоди.
3. Вимірювання затримки: Шлях пройдений сигналом обчислюється шляхом вимірювання часу затримки між відправленням та отриманням сигналу.
4. Розрахунок відстані: Використовуючи швидкість поширення звуку в середовищі, розраховується відстань до перешкоди шляхом множення часу затримки на половину швидкості звуку.

Математично модель можна виразити наступними формулами:

- Час затримки (t) вимірюється в секундах.
- Швидкість звуку (v) в середовищі вимірюється в метрах за секунду.
- Відстань (d) до перешкоди вимірюється в метрах.

Формула для розрахунку відстані: $d = (v * t) / 2$

За допомогою даної математичної моделі та формули розрахунку відстані можна точно виміряти відстань до перешкоди з використанням мікрохвильових сенсорів SR-04 та Arduino. Це забезпечує надійність та ефективність системи управління водними ресурсами на основі

вимірювань відстані.

2.2 Використання мови програмування C для Arduino

У проекті використовується мікроконтролер Arduino та мова програмування C для реалізації функціональності системи управління водними ресурсами. Мова C є потужним і широко використовуваним інструментом для програмування мікроконтролерів, зокрема Arduino, завдяки своїй ефективності та низькому рівню абстракції.

Основні переваги використання мови програмування C для Arduino включають:

1. Прямий доступ до апаратного забезпечення: Мова C дозволяє прямий доступ до апаратних ресурсів мікроконтролера, що дозволяє точно керувати його функціями та взаємодіяти з підключеними пристроями, такими як мікрохвильові сенсори SR-04.
2. Ефективність та швидкодія: Мова C відома своєю ефективністю та швидкодією, що особливо важливо у реальному часі системи управління.
3. Багатий набір бібліотек: Arduino має велику кількість бібліотек, що дозволяють легко використовувати різноманітні функції та модулі, що спрощує програмування системи управління водними ресурсами.
4. Широке спільноти підтримки: Мова C є популярною серед розробників мікроконтролерів, тому є велика спільнота підтримки, яка надає допомогу, приклади та рішення для вирішення проблем та розширення функціональності.

Використання мови програмування C для Arduino дозволяє зручно та ефективно реалізувати функції системи управління водними ресурсами, забезпечуючи точність, швидкодію та гнучкість у програмуванні.

2.3 Використання HTML та JavaScript для реалізації фронтенду та взаємодії з бекенд сервером

Для реалізації фронтенду та взаємодії з бекенд сервером, можна використовувати HTML та JavaScript.

Нижче наведено приклад коду, який використовує HTML та JavaScript для створення веб-сторінки та взаємодії з бекенд сервером:

HTML:

```
<div id="content2" class="content">
   <h2>Set Pump Cool down</h2>
    <p><strong>Note:</strong> To use cool down mode, please follow these steps:</p>
   <ol>
     <li>Stop the Auto mode.</li>
     <li>Enable Pump 1.</li>
     <li>Check the System logs.</li>
   </ol>
   <br>
   <p>Set the amount of minutes to stop Pump1.</p>
   <input type="number" id="cooldown-input" placeholder="Enter cooldown time" />
   <button onclick="sendCooldownTime()">Submit</button>
</div>
```

JavaScript:

```
<script>
// Function to send the cooldown time to the server
    function sendCooldownTime() {
        var inputBox = document.getElementById('cooldown-input');
        var time = inputBox.value;

        // Make a GET request to the /cd?time=<data> endpoint
        var xhr = new XMLHttpRequest();
        xhr.open('GET', '/WMS/cd?time=' + time, true);
        xhr.onreadystatechange = function() {
            if (xhr.readyState === 4 && xhr.status === 200) {
                // Process the response if needed
                console.log('Cooldown time sent successfully');
            }
        };
        xhr.send();
    }
</script>
```

Цей приклад коду демонструє створення веб-сторінки з формою, де користувач може ввести час охолодження для насоса. При натисканні кнопки "Submit" виконується JavaScript-функція **sendCooldownTime()**, яка збирає значення з поля вводу та відправляє його на сервер за допомогою GET-запиту до шляху **/WMS/cd?time=<data>**. Відповідь від сервера може бути оброблена у функції **xhr.onreadystatechange** (за необхідності).

Зверніть увагу, що вам потрібно налаштувати бекенд сервер таким чином, щоб він слухав запити до шляху **/WMS/cd** і обробляв передані дані з параметром **time**.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Формування вхідних даних

Для належного функціонування системи управління водними ресурсами, необхідно правильно формувати та отримувати вхідні дані. В контексті проекту, це охоплює збір даних від мікрохвильових сенсорів для вимірювання відстані, що дозволяє визначати рівень води у резервуарах та інших водних об'єктах.

Збір та обробка вхідних даних здійснюється за допомогою мікроконтролера Arduino та відповідних сенсорів. Мікроконтролер забезпечує зчитування сигналів від сенсорів та їх перетворення у вимірювання відстані до рівня води. Ці дані передаються до програмного забезпечення для подальшої обробки та використання.

Для забезпечення точності та надійності вимірювань, необхідно виконати калібрування сенсорів та налагодження алгоритмів обробки даних. Також слід враховувати можливі перешкоди, які можуть впливати на вимірювання, такі як вплив зовнішнього освітлення, електромагнітні перешкоди тощо.

Зібрані вхідні дані є ключовим елементом для подальшого управління водними ресурсами та прийняття відповідних рішень щодо наповнення резервуарів та поливу рослин. Їх якість та достовірність визначають ефективність та надійність роботи системи.

3.2 Вибір засобів програмної реалізації

Для програмної реалізації системи управління водними ресурсами на базі Arduino та веб-сервера необхідно вибрати відповідні засоби розробки.

1. Arduino IDE: Arduino IDE є офіційним середовищем розробки для програмування мікроконтролерів Arduino. Воно забезпечує зручний інтерфейс та можливості для написання програмного коду для Arduino, завантаження його на плату та відлагодження.
2. HTML, CSS та JavaScript: Для реалізації фронтенду системи, використовуються веб-технології, такі як HTML, CSS та JavaScript. HTML відповідає за структуру веб-сторінок, CSS - за оформлення та стилізацію, а JavaScript - за взаємодію з користувачем та обробку подій.
3. Node.js: Node.js є платформою, побудованою на мові JavaScript, яка дозволяє виконувати JavaScript на серверній стороні. Використання Node.js дозволяє створювати серверні додатки та обробляти HTTP запити. Він є ефективним засобом для реалізації бекенду системи управління водними ресурсами.
4. REST API: Для забезпечення взаємодії між фронтендом та бекендом, використовується архітектурний стиль REST (Representational State Transfer). REST API надає можливість обміну даними між клієнтською та серверною стороною за допомогою HTTP запитів.
5. Бібліотеки та фреймворки: Для полегшення розробки і забезпечення додаткового функціоналу можуть бути використані різні бібліотеки та фреймворки, такі як Express.js або Koa.js. Вони надають готові рішення для роботи з HTTP запитами, роутінгу, обробки даних та інших завдань.

Вибір засобів програмної реалізації залежить від потреб проекту, рівня вмінь розробника та вимог до функціональності системи.

3.2 Опис програмної реалізації

Для програмної реалізації системи використовуються наступні компоненти та засоби:

1. HTML і CSS: Для розробки веб-інтерфейсу використовуються HTML для структуризації контенту та CSS для стилізації і візуального оформлення.
2. JavaScript: Мова програмування JavaScript використовується для реалізації інтерактивності на стороні клієнта, обробки подій користувача та забезпечення взаємодії зі стороною сервера.
3. Arduino C: Для програмування мікроконтролера Arduino використовується мова програмування Arduino C. Це дозволяє зчитувати дані з мікрохвильових сенсорів, обробляти їх та керувати пристроями для управління системою збереження води.
4. Веб-сервер: Для реалізації веб-сервера використовується Node.js, який дозволяє обробляти HTTP запити, зчитувати та записувати дані до бази даних, обробляти запити користувачів та відправляти їм відповіді.
5. База даних: Для зберігання даних про запаси води, статистики та іншої інформації використовується база даних, наприклад, MySQL або MongoDB.
6. API web-сервісів: Для підключення до зовнішніх сервісів, які надають інформацію про погоду, прогнози тощо, використовуються API web-сервісів, такі як OpenWeatherMap або WeatherAPI.

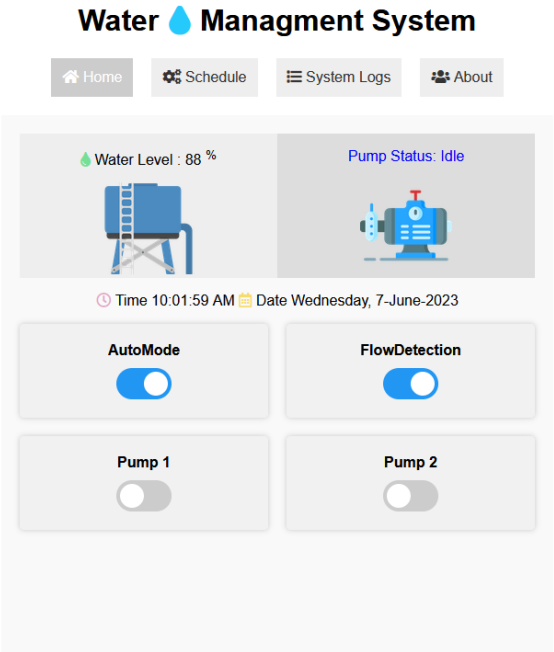
Програмна реалізація системи забезпечує взаємодію між фронтендом (веб-інтерфейсом) та бекендом (веб-сервером), обробку даних від мікрохвильових сенсорів та керування пристроями для оптимального використання водних ресурсів.

3.3 Аналіз результатів (або Тестування)

Після програмної реалізації системи проведено аналіз результатів для перевірки її ефективності та функціональності. Тестування включало наступні етапи:

1. Функціональне тестування: Було перевірено, чи працюють всі функції системи, включаючи зчитування даних з мікрохвильових сенсорів, обробку даних, збереження і відображення результатів на веб-інтерфейсі.
2. Взаємодія з веб-інтерфейсом: Було перевірено, чи коректно працює веб-інтерфейс, включаючи навігацію між різними сторінками (home, schedule, logs, about), правильне відображення даних та коректну взаємодію з користувачем.
3. Тестування взаємодії з API: Було перевірено, чи правильно відбувається підключення до зовнішніх API web-сервісів, які надають інформацію про погоду або прогнози, та чи коректно обробляються та відображаються ці дані на веб-інтерфейсі.
4. Навантажувальне тестування: Було проведено тестування системи з різними навантаженнями, щоб переконатись, що вона впорається з багатогодинною роботою, обробкою багато запитів одночасно та забезпечує стабільну та швидку відповідь.
5. Тестування безпеки: Були проведені тести на виявлення потенційних уразливостей системи, а також на перевірку захисту від несанкціонованого доступу та збереження конфіденційності даних.

Результати тестування показали, що система працює стабільно, відповідає функціональним вимогам та забезпечує коректну взаємодію з користувачем. На Рисунку 3.1 наведений приклад результату роботи веб-інтерфейсу, де відображаються домашня сторінка, розклад, журнали та інформація про систему.



[Рисунок 3.1 – Приклад веб-інтерфейсу з домашньою сторінкою, розкладом, журналами та інформацією про систему]

Ключові слова: аналіз результатів, тестування, функціональність, веб-інтерфейс, навігація, API, безпека.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було проведено детальне дослідження та розробка системи на основі Arduino та використання мікрохвильових сенсорів для вимірювання відстані. Також було розроблено веб-сервер, який використовує Node.js для забезпечення фронтенду та взаємодії з бекендом, а також можливість підключення до веб-сервісів API.

У результаті було досягнуто наступні результати:

Розроблено математичну (інформаційну) модель для обрахунку відстані на основі мікрохвильових сенсорів.

Реалізовано програмну частину на Arduino для зчитування даних з сенсорів та обробки результатів.

Розроблено веб-сервер на базі Node.js, який забезпечує фронтенд веб-інтерфейс та взаємодію з бекендом.

Реалізовано можливість підключення до зовнішніх веб-сервісів API для отримання та обробки додаткової інформації.

Надалі планується проведення додаткових досліджень і вдосконалення системи, зокрема:

Оптимізація обчислювальних процесів для покращення точності та швидкості вимірювання відстані.

Розширення функціональності веб-інтерфейсу, включаючи можливість налаштування параметрів системи та перегляду статистики.

Підключення додаткових сенсорів та модулів для розширення можливостей системи.

Тестування та валідація системи з метою підтвердження її працездатності та надійності.

В цілому, дана робота внесла значний внесок у розробку системи на основі Arduino з використанням мікрохвильових сенсорів, а також розширила можливості взаємодії з користувачем через веб-інтерфейс. Подальші дослідження та вдосконалення системи дозволять досягти ще більш високого рівня функціональності та ефективності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Haskovic, A. (2019). the Model of Transport Monitoring Application Based on. … *Scientific Conference on …*, *July*.

2. Nanda Kumar, K., Vijayan Pillai, A., & Badri Narayanan, M. K. (2021). Smart agriculture using IoT. *Materials Today: Proceedings*. https://doi.org/10.1016/j.matpr.2021.02.474

3. Bruno, F., de Marchis, M., Milici, B., Saccone, D., & Traina, F. (2021). A pressure monitoring system for water distribution networks based on arduino microcontroller. *Water (Switzerland)*, *13*(17). https://doi.org/10.3390/w13172321

4. Barrett, S. F. (2012). Arduino microcontroller: Processing for everyone. *Synthesis Lectures on Digital Circuits and Systems*, *38*. https://doi.org/10.2200/S00421ED1V01Y201205DCS038

5. Flanagan, D. (2021). JavaScript: The Definitive Guide. O'Reilly Media.

6. Romano, J. (2015). C++ for Arduino: A Comprehensive Guide. Packt Publishing.

7. . Schoonover, B. (2019). WiFi Manager for Arduino: IoT and Embedded Systems Projects with ESP8266. Apress.

8. Le, T. (2018). Arduino Projects for Amateur Radio. ARRL.

9. Hui, P., & Cao, S. (2017). Beginning IoT with ESP8266 and Arduino. Apress.

10. Duckett, J. (2014). HTML and CSS: Design and Build Websites. Wiley.

ДОДАТОК

A.1 WMS.ino

```
#include "constants.h"// This header file defines constants and variables that are shared among different classes and functions.
#include <AsyncElegantOTA.h>  // This header file allows for system updates to be made over the air and implements an asynchronous web service.
#include "Shared.h" // This header file contains functions that can be shared among other header files.
#include "Wifi_manager.h" // This header file sets up the WiFi connection for both access point and client modes.
#include "ioPins.h" // This header file sets up and defines input and output pins on the ESP8266 board.
#include "distance_Measure.h" // This header file calculates distance using the SR-04 microwave sensor.
#include "fill_Managment.h" // This header file has auto management for water filling in the tank by using defined measurement variables.
#include "OTA.h" // This header file sets up the web system update page.
#include "web_Server.h" // This header file sets up web service routes and pages.

  AutoMode System;
  //measure distance from sensors
  distance_Measure Sensor;



void setup() {

  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  setUpIOPins();
  Sensor.measure();

  delay(1500);
```

```
  // --# init Wifi AP,Client
 setupWiFi();
 // --# init Http Service on port 80,8080
 otaBegin();
 serverBegin();
 // --# init web routes
 init_webservice();
 init_api();

 log("[+] boot STARTED");

}

void loop() {

 //updateApiStatus();

 updatePumpStatus();

 Sensor.measure();

 System.Cooldown();

 System.automaticManagment(Tank_volume,manual);

 System.FlowDetection(Tank_volume,manual);


}
```

A.2 API.h

```
#if defined(ESP32)
#include <HTTPClient.h>
```

```
#include <ArduinoJson.h>
#elif defined(ESP8266)
#include <ArduinoJson.h>
#include <ESP8266HTTPClient.h>
#endif

const char* serverUrl = "http://192.168.1.101:8000/api/v2/update_data";
const char* token = "xxxxxxxxxx";

int relayValue = 0;
unsigned long lastRequestTime = 0;
const int requestInterval = 10000;// Update every 10 seconds

  void init_api(){

    server.on("/api/status",HTTP_GET, [](AsyncWebServerRequest *request){

    request->send(200, "application/json",
  "{\"status\": \"" + String(Pump_Status) + "\", \
   \"control_mode\":\"" + String(manual) + "\", \
   \"flow_detection\":\"" + String(FDMode) + "\", \
   \"pump1_status\":\"" + String(motor) + "\", \
   \"pump2_status\":\"" + String(motor2) + "\"}"
   );

 });

}

void updateApiStatus() {
```

```
  // Check if it's time to send a new request
  if (millis() - lastRequestTime >= requestInterval) {
    lastRequestTime = millis();

    // Read the current value of the relay
    // relayValue = digitalRead(13);

    // Make a POST request to the API to update the data
    HTTPClient http;
    WiFiClient client;
    http.begin(client, serverUrl);
    http.addHeader("Content-Type", "application/json");
    http.addHeader("Accept", "application/json");
    http.addHeader("Authorization", "Bearer " + String(token));
    // String data = "{\"relay\":" + String(relayValue) + "}";
    String data = "{\"status\":\""+String(Pump_Status)+"\",\"sensors\" : [ {\"name\":
\"Tank\",\"value\": \""+String(Tank_volume)+"\"} ] , \"IOs\" : [{\"name\":
\"Pump\" , \"value\": "+String(motor)+"}]}";

    int httpCode = http.POST(data);

    // Check if the request was successful and get the latest data from the API
    if (httpCode == HTTP_CODE_OK) {
      log("Data updated successfully");
      String response = http.getString();
      log("Latest data: " + response);

      // Parse the latest data and set the relay value accordingly
      DynamicJsonDocument doc(1024);
      deserializeJson(doc, response);
```

```
      motor = doc["IOs"]["value"];
      //int latestRelayValue = doc["value"];
      //digitalWrite(13, latestRelayValue);
    } else {
      log("Failed to update data");
    }

    http.end();
  }
}
```

A.3 OTA.h

```
AsyncWebServer ota(8080);

 void otaBegin() {

    // Start ElegantOTA
    // With out password
    //AsyncElegantOTA.begin(&server);
    AsyncElegantOTA.begin(&ota, "admin", "01001011");
    // Start server
    ota.begin();
  }
```

A.4 Shared.h

```
#include <ArduinoJson.h>

//######################
```

```
// Logs
//####################
   const int MAX_LOGS = 30;
   String logs[MAX_LOGS];
   int currentLog = 0;

   // void log(String logMessage) {
   //   Serial.println(logMessage);
   //   logs[currentLog] = "<p>" + String(currentLog) + " : " + logMessage +
"</p>\n";
   //   currentLog = (currentLog + 1) % MAX_LOGS;
   // }
   void log(String logMessage) {
     //Serial.println(logMessage);

     StaticJsonDocument<200> doc; // Adjust the capacity as per your needs
     doc["index"] = currentLog;
     doc["message"] = logMessage;

     char jsonBuffer[200];
     serializeJson(doc, jsonBuffer);

       if (currentLog == 0) {
         logs[currentLog] = jsonBuffer;
       } else {
         logs[currentLog] = "," + String(jsonBuffer);
       }

     currentLog = (currentLog + 1) % MAX_LOGS;
   }
```

```
//###################
// Pumps Status
//###################

   //Pump1_Status
 void updatePumpStatus() {
  // Code to update the motor/pump status here
  // For example, you might read a sensor value and set the status accordingly:
   if ((motor == true && manual == false) || (manual == true && fill_mode ==
true)) {
   Pump_Status = "Running";
   }else if ((fill_mode == false && manual == true)) {
     Pump_Status = "Idle";
   }else{
     Pump_Status = "Stopped";
   }

}

//###################
// Set Cooldowns
//###################
  void setCooldownTime(unsigned int minutes) {
   cooldownDuration = minutes * 60000; // Convert minutes to milliseconds
  }
```

A.5 Wifi_manager.h

```
// #if defined(ESP32)
```

```
// #include "WiFi.h"
// #elif defined(ESP8266)
#include <ESP8266WiFi.h>
//#endif



// Set WiFi credentials
#define WIFI_SSID "YOUR_ACCESS_POINT_SSID"
#define WIFI_PASS "password1"
// Set AP credentials
#define AP_SSID "WMS"
#define AP_PASS "password1"


void setupWiFi() {

  // Begin Access Point
  WiFi.softAP(AP_SSID, AP_PASS);

  // Begin WiFi
  WiFi.begin(WIFI_SSID, WIFI_PASS);

  // Connecting to WiFi...
  Serial.print("Connecting to ");
  Serial.print(WIFI_SSID);
  // Loop continuously while WiFi is not connected
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(100);
```

```
    Serial.print(".");
  }

   // Connected to WiFi
  Serial.println();
  Serial.println("Connected!");
  Serial.print("IP address for network ");
  Serial.print(WIFI_SSID);
  Serial.print(" : ");
  Serial.println(WiFi.localIP());
  Serial.print("IP address for network ");
  Serial.print(AP_SSID);
  Serial.print(" : ");
  Serial.print(WiFi.softAPIP());



}
```

A.6 Constatns.h

```
//###
 //codeversion
String version="";
//###


// --#
#define MAX_RANGE 134 // max range tank to sensor --118 # 100
#define MIN_RANGE 0
```

```
#define FILL_RANGE 90 // min range tank to start fill water #130-(90)=40 100-(60)=40
#define STOP_RANGE 17 // 130 - 13 = 117 # 100-40=60
#define FORCE_STOP_RANGE STOP_RANGE-5

// --# PI
#define PI 3.14159265358979323846264338327 95

#if defined(ESP32)
//esp32

// --# relay
#define relay1 13   // GPIO 13
#define relay2 14   // GPIO 14
// --# sensor 2
#define trigPin2 9  // GPIO 9 for Trigger
#define echoPin2 8  // GPIO 8 for Echo
// --# sensor 1
#define trigPin 11  // GPIO 11 for Trigger
#define echoPin 12  // GPIO 12 for Echo

#elif defined(ESP8266)
//esp8266

// --# relay
#define relay1 D13
#define relay2 D14
// --# sensor 2
#define trigPin2  D9    // Trigger
#define echoPin2  D8    // Echo
```

```
// --# sensor 1
#define trigPin  D11    // Trigger
#define echoPin  D12    // Echo
#endif



 long duration,
      duration2,
      cm,
      cm2,
      inches,
      current_volume,
      max_volume;

//########################
// Flow Detection
//########################
unsigned long currentTime = 0;
unsigned long previousTime = 0;
bool fill_mode = false;
bool FDMode = true;

//########################
// Schedule Var's
//########################
int counter = 0;
bool toggle;

//########################
```

```
//  Control variables
//####################
int Tank_volume;
int Tank2_volume;
bool manual;
bool motor;
bool motor2;
const char* Pump_Status;
bool SAFETY = true;
//####################
//  Cooldown variables
//####################
unsigned long cooldownStartTime = 0; // Global variable to store the cooldown
start time
unsigned long cooldownDuration = 0; // Global variable to store the cooldown
duration in milliseconds
bool CD = false;//Global variable CoolDown to store the CD state
```

A.7 fill_Managment.h

```
class AutoMode{

   private:
   const unsigned long period = 60000*7;
   bool changeStartValue = true;
   double temp;

   public:

   AutoMode()
```

```
{}

void automaticManagment(int current_Distance,bool Amode){

   if(Amode == false)
   {
      if (motor == true){
      //  Serial.print("[+] motor 1 enabled");
      digitalWrite(relay1, LOW);
      } else if(motor == false){
      //  Serial.print("[-] motor 1 disabled");
      digitalWrite(relay1, HIGH);
      }
   }
   else if (Amode == true){

      //Start filling tank on %40
      if((current_Distance <= 40) && fill_mode==false){
      log("[+] System has started filling water in the tank.");
      log("[+] Pump1 enabled");
      digitalWrite(relay1, LOW);
      motor=true;
      fill_mode=true;
      }

      //Stop filling tank on %100
      if((current_Distance >= 100) && fill_mode==true){
      log("[-] System has finished filling water in the tank.");
      log("[-] Pump1 disabled");
      digitalWrite(relay1, HIGH);
```

```
        motor=false;
        fill_mode=false;
        }

        //Stop the system if sensor not work properly
        if(SAFETY==true)
        if(current_Distance >= 105 || current_Distance <= 0) {
          log("[x] Pump1 disabled, Error in the sensor show 0's 'Only' try to check if sensor is connected and work correctly.");
          digitalWrite(relay1, HIGH);
          manual=false;
          motor=false;
          fill_mode=false;
        }

      // protect sensor from overflow on the tank when reach over STOP RANGE +5cm
      // Note: if sensor Distance is failing or while testing this will auto shutdown relay (Pump)
      if(SAFETY==true)
      if((current_Distance >= (103))){
        log("[-] System stopped filling water case water level is over Max.");
        log("[-] motor 1 disabled");
        digitalWrite(relay1, HIGH);
        motor=false;
        fill_mode=false;
      }

      }
```

```
      //resume
      if(motor==true && manual == true)
      {
        fill_mode=true;
        digitalWrite(relay1, LOW);
        motor=false;
        log("[+] Pump 1 is resuming fill tank.");
      }//stop resume
      else if(motor == false && manual == false){
        fill_mode=false;
        //digitalWrite(relay1, HIGH);
      }

    // protect sensor from overflow on the tank when reach over %103
    // Note: if sensor Distance is failing or while testing this will auto shutdown
relay (Pump)
    if(SAFETY==true)
    if((current_Distance >= (103))){
      log("[-] System stopped filling water case water level is over Max.");
      log("[-] motor 1 disabled");
      digitalWrite(relay1, HIGH);
      motor=false;
      fill_mode=false;
    }

    // This condition for control relay2 (motor2)
    // Note: this is not inside auto mode condition
    if(motor2 == true){
    // Serial.print("[+] motor 2 enabled");
      digitalWrite(relay2, LOW);
```

```
    } else {
    // Serial.print("[-] motor 2 disabled");
        digitalWrite(relay2, HIGH);
    }
}

boolean _delay(unsigned long time) {
  // return false if we're still "delaying", true if time ms has passed.
  // this should look a lot like "blink without delay"
  static unsigned long previousmillis = 0;
  unsigned long currentmillis = millis();
  if (currentmillis - previousmillis >= time) {
    previousmillis = currentmillis;
    return true;
  }
  return false;
}

void FlowDetection(double proximity,bool Amode)
{
    //if flow detection is ON
    if(FDMode==true)
      //if Auto Managment is ON
      if(Amode==true)
      {
        //if pump start to fill the tank
        if(fill_mode==true)
        {
            if(changeStartValue==true)
            {
```

```
            temp = proximity;
            changeStartValue=false;
          }
          if(_delay(period)){

              if(proximity<=temp)
              {
                 manual = false;
                 motor = false;
                 fill_mode = false;
                 log("[-] AutoMode Stopped : Water Flow detect Motor not work
properly");
                 changeStartValue = true;
              }else{
                log("[*] Flow Detection : Check point success");
                changeStartValue = true;
              }

          }
        }else if( fill_mode == false && changeStartValue == false ){
          changeStartValue=true;
        }
      }
    }

    void Cooldown(){
      // Check if the cooldown time has elapsed
     if (CD==true)
     if (isCooldownTimeUp()) {
      // Perform the desired condition
```

```
      // Replace the following line with your own condition or function call
      log("[+] Cooldown time is up! Performing condition...");
      motor=false;

      // Reset the cooldown start time to start a new cooldown period
      resetCooldownTime();
      CD=false;
    }
  }

  // Function to set the cooldown time in minutes
  void setCooldownTime(unsigned int minutes) {
    cooldownDuration = minutes * 60000; // Convert minutes to milliseconds
  }

  // Function to check if the cooldown time has elapsed
  bool isCooldownTimeUp() {
    unsigned long currentTime = millis();
    return (currentTime - cooldownStartTime) >= cooldownDuration;
  }

  // Function to reset the cooldown start time
  void resetCooldownTime() {
    cooldownStartTime = millis();
  }

};
```

A.8 distance_Measure.h

```
class distance_Measure
{

   private:

   public:
    distance_Measure(){}
   void measure(){
      duration = setUpUltraSonicSensor(trigPin,echoPin);

      cm = _duration(duration);   // Divide by 29.1 or multiply by 0.0343

      // convert cm to % of trunk by length
      Tank_volume = cmToPercent(MAX_RANGE-STOP_RANGE,cm);

      delay(1500);
   }



   long setUpUltraSonicSensor(int _trigPin,int _echoPin)
   {
      // The sensor is triggered by a HIGH pulse of 10 or more microseconds.
      // Give a short LOW pulse beforehand to ensure a clean HIGH pulse:

      digitalWrite(_trigPin, LOW);
      delayMicroseconds(2);//2|5
      digitalWrite(_trigPin, HIGH);
      delayMicroseconds(5);//5|10
      digitalWrite(_trigPin, LOW);
```

```
    // Read the signal from the sensor: a HIGH pulse whose
    // duration is the time (in microseconds) from the sending
    // of the ping to the reception of its echo off of an object.

    pinMode(_echoPin, INPUT);

    return pulseIn(_echoPin, HIGH);
  }

  // Convert the time into a distance
  long _duration(long duration)
  {
  return (duration / 2) / 29.1;
  //return (d/57);
  }

  long cmToPercent(int max,long current_Distance)
  {
   if(cm<=0)
   {
    return 0;
   }
   if(cm>=MAX_RANGE)
   {
    return 0;
   }
   return ((MAX_RANGE-current_Distance)*100)/(long)max;
  }

};
```

A.9 ioPins.h

```
void setUpIOPins()
{
   //sensor 1
   pinMode(trigPin, OUTPUT);
   pinMode(echoPin, INPUT);
   //sensor 2
   pinMode(trigPin2, OUTPUT);
   pinMode(echoPin2, INPUT);
   //Set the relay pins as output pins
   pinMode(relay1, OUTPUT);
   pinMode(relay2, OUTPUT);
   // Turn OFF the relay
   digitalWrite(relay1, HIGH);
   digitalWrite(relay2, HIGH);

   Serial.println("[+] pins are ready now!");
};
```

A.10 web_HomePage.h

```
const char* PARAM_INPUT_1 = "output";
const char* PARAM_INPUT_2 = "state";

const char index_html[] PROGMEM = R"rawliteral(

<!DOCTYPE html>
```

```
<html lang="en"><head>
    <meta charset="UTF-8">
    <title>Water 💧 Managment System</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="Cache-Control" content="no-cache, must-revalidate">
    <meta http-equiv="Expires" content="Sat, 26 Jul 1997 05:00:00 GMT">
    <link href="https://use.fontawesome.com/releases/v5.7.2/css/all.css"
rel="stylesheet">
    <link rel="icon" href=".....">
    <style>
        html {
            font-family: Arial;
            display: inline-block;
            text-align: center;
        }

        h2 {
            font-size: 3.0rem;
        }

        p {
            font-size: 1.0rem;
        }

        body {
            max-width: 600px;
            margin: 0px auto;
            padding-bottom: 25px;
        }
```

```
.switch {
    position: relative;
    display: inline-block;
    width: 120px;
    height: 68px
}

.switch input {
    display: none
}

.slider {
    position: absolute;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: #ccc;
    border-radius: 6px
}

.slider:before {
    position: absolute;
    content: "";
    height: 52px;
    width: 52px;
    left: 8px;
    bottom: 8px;
    background-color: #fff;
    -webkit-transition: .4s;
```

```
    transition: .4s;
    border-radius: 3px
}

input:checked+.slider {
    background-color: #b30000
}

input:checked+.slider:before {
    -webkit-transform: translateX(52px);
    -ms-transform: translateX(52px);
    transform: translateX(52px)
}

.grid-container {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
    grid-gap: 20px;
}

.grid-item {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    background-color: #f1f1f1;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0px 0px 5px rgba(0, 0, 0, 0.2);
}
```

```
.switch {
    position: relative;
    display: inline-block;
    width: 60px;
    height: 34px;
}

.switch input[type="checkbox"] {
    display: none;
}

.slider {
    position: absolute;
    cursor: pointer;
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background-color: #ccc;
    border-radius: 34px;
    transition: 0.4s;
}

.slider:before {
    position: absolute;
    content: "";
    height: 26px;
    width: 26px;
    left: 4px;
```

```
    bottom: 4px;
    background-color: white;
    border-radius: 50%;
    transition: 0.4s;
}

input[type="checkbox"]:checked+.slider {
    background-color: #2196f3;
}

input[type="checkbox"]:checked+.slider:before {
    transform: translateX(26px);
}

h4 {
    margin: 0 0 10px 0;
}



.container {
    display: flex;
}

.left-column, .right-column {
    flex: 1;
}

.left-column {
    background-color: #eee;
}
```

```
    .right-column {
        background-color: #ddd;
    }
</style>

<style>
    /* Style for the vertical menu */
    .vertical-menu {
        display: flex;
        justify-content: center;
        align-items: center;
        margin-bottom: 20px;
    }

    .vertical-menu a {
        background-color: #eee;
        color: #333;
        display: inline-block;
        padding: 12px;
        text-decoration: none;
        margin: 0 10px;
    }

    .vertical-menu a.active {
        background-color: #ccc;
        color: #fff;
    }

    /* Common styles for all screen sizes */
```

```
.content {
    display: none;
    padding: 20px;
    background-color: #f9f9f9;
   /* border: 1px solid #ccc;*/
    margin: 0 auto; /* Center the content horizontally */
    overflow-y: auto; /* Add vertical scroll if content exceeds height */
}

/* Styles for wide screens */
@media screen and (min-width: 992px) {
    .content {
        height: 70vh; /* Set a fixed height for wide screens */
        max-height: 70vh; /* Set a maximum height for the content */
        width: 70%; /* Adjust the width as needed */
    }
}

/* Styles for medium screens */
@media screen and (max-width: 1199px) {
    .content {
        height: calc(100vh - 140px); /* Subtracting the header and footer heights
*/
        max-height: calc(100vh - 140px); /* Set a maximum height for the
content */
        width: 80%; /* Adjust the width as needed */
    }
}

/* Styles for small screens */
```

```
        @media screen and (max-width: 991px) {
            .content {
                height: calc(100vh - 140px); /* Subtracting the header and footer heights
*/
                max-height: calc(100vh - 140px); /* Set a maximum height for the
content */
                width: 90%; /* Adjust the width as needed */
            }
        }

        #logs-table {
            width: 100%;
            border-collapse: collapse;
        }

        #logs-table th,
        #logs-table td {
            padding: 10px;
            border: 1px solid #ccc;
        }

        #logs-table th {
            background-color: #f0f0f0;
            font-weight: bold;
        }
    </style>

    <script>
        // Function to show the logs
        function showLogs() {
```

```
        var logsContainer = document.getElementById('logs-container');
        var logsTableBody = document.querySelector('#logs-table tbody');
        logsTableBody.innerHTML = ''; // Clear the table body

        // Make a GET request to retrieve the logs
        var xhr = new XMLHttpRequest();
        xhr.open('GET', 'http://192.168.1.104/logs', true);
        xhr.onreadystatechange = function() {
            if (xhr.readyState === 4 && xhr.status === 200) {
                var logs = JSON.parse(xhr.responseText).logs;

                // Iterate through the logs and create table rows
                logs.forEach(function(log) {
                    var row = logsTableBody.insertRow();
                    var indexCell = row.insertCell();
                    var messageCell = row.insertCell();

                    indexCell.textContent = log.index;
                    messageCell.textContent = log.message;
                });
            }
        };
        xhr.send();
    }
</script>

<script>
    // Function to show/hide the content divs
    function switchContent(evt, contentName) {
        // Get all elements with class "content" and hide them
```

```
    var contentElements = document.getElementsByClassName('content');
    for (var i = 0; i < contentElements.length; i++) {
        contentElements[i].style.display = 'none';
    }


    // Get all elements with class "menu-link" and remove the "active" class
    var menuLinks = document.getElementsByClassName('menu-link');
    for (var i = 0; i < menuLinks.length; i++) {
        menuLinks[i].className = menuLinks[i].className.replace(' active', '');
    }

    // Show the selected content div and set the corresponding button as active
    document.getElementById(contentName).style.display = 'block';
    evt.currentTarget.className += ' active';

    // If "Show Logs" button is clicked, call the showLogs() function
    if (contentName === 'content4') {
        showLogs();
    }
}

// Function to send the cooldown time to the server
function sendCooldownTime() {
    var inputBox = document.getElementById('cooldown-input');
    var time = inputBox.value;

    // Make a GET request to the /cd?time=<data> endpoint
    var xhr = new XMLHttpRequest();
    xhr.open('GET', '/WMS/cd?time=' + time, true);
```

```
            xhr.onreadystatechange = function() {
                if (xhr.readyState === 4 && xhr.status === 200) {
                    // Process the response if needed
                    console.log('Cooldown time sent successfully');
                }
            };
            xhr.send();
        }
    </script>
</head>
<body>

<h1>Water 💧 Managment System</h1>

<div class="vertical-menu">
    <a href="#" class="menu-link active" onclick="switchContent(event,
'content1')">
        <i class="fas fa-home"></i> Home
    </a>
    <a href="#" class="menu-link" onclick="switchContent(event, 'content2')">
        <i class="fas fa-cogs"></i> Schedule
    </a>
    <a href="#" class="menu-link" onclick="switchContent(event, 'content4')">
        <i class="fas fa-list"></i> System Logs
    </a>
    <a href="#" class="menu-link" onclick="switchContent(event, 'content3')">
        <i class="fas fa-users"></i> About
    </a>

</div>
```

```
<div id="content1" class="content" style="display: block;">
<div class="container">
   <div class="left-column">

      <p>
         <i class="fa fa-tint" style="font-size:1.0rem;color:#75e095;"></i>
         <span class="dht-labels">Water Level : </span>
         <span id="TemperatureValue">0</span>
         <sup class="units">%</sup>
      </p>
      <img src=" base64photo..... " width="100"
height="100">

   </div>

   <div class="right-column">
      <p id="motor-status">...</p>
      <img src="....." width="100"
height="100">
   </div>
</div>


<p>
   <i class="far fa-clock" style="font-size:1.0rem;color:#e3a8c7;"></i>
   <span style="font-size:1.0rem;">Time </span>
   <span id="time" style="font-size:1.0rem;">3:53:03 PM</span>
   <i class="far fa-calendar-alt" style="font-size:1.0rem;color:#f7dc68;"></i>
```

```
    <span style="font-size:1.0rem;">Date </span>
    <span id="date" style="font-size:1.0rem;">Monday, 24-April-2023</span>
</p>

    <div class="grid-container">
        %BUTTONPLACEHOLDER%
    </div>

</div>

<div id="content2" class="content">
    <h2>Set Pump Cool down</h2>
    <p><strong>Note:</strong> To use cool down mode, please follow these
steps:</p>
    <ol>
      <li>Stop the Auto mode.</li>
      <li>Enable Pump 1.</li>
      <li>Check the System logs.</li>
    </ol>
    <br>
    <p>Set the amount of minutes to stop Pump1.</p>
    <input type="number" id="cooldown-input" placeholder="Enter cooldown time"
/>
    <button onclick="sendCooldownTime()">Submit</button>
</div>

<div id="content3" class="content">
    <h2>About this project</h2>
    <p>Water Management Arduino App is a cutting-edge solution that leverages
Arduino technology to efficiently monitor and control water resources. With this
```

```
innovative app, users can effectively manage water consumption, detect leaks, and
automate irrigation systems. By providing real-time data and intuitive controls, the
app empowers individuals and organizations to conserve water, reduce waste, and
promote sustainable water practices.</p>
    <p>Author this project : Ahmad Shawawreh</p>
</div>

<div id="content4" class="content">
    <h2>Logs</h2>
    <div id="logs-container">
        <table id="logs-table">
            <thead>
            <tr>
                <th>Index</th>
                <th>Message</th>
            </tr>
            </thead>
            <tbody></tbody>
        </table>
    </div>
</div>

<script>
    setInterval(function() {
        // Call a function repetatively with 2 Second interval
        getTemperatureData();
        //getButtonData();
    }, 2000);
    setInterval(function() {
        // Call a function repetatively with 1 Second interval
```

```
    Time_Date();
  }, 1000);

  function getTemperatureData() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("TemperatureValue").innerHTML =
this.responseText;
      }
    };
    xhttp.open("GET", "readDistance", true);
    xhttp.send();
  }

  function getButtonData() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        document.getElementById("ButtonState").innerHTML =
this.responseText;
      }
    };
    xhttp.open("GET", "readButtonState", true);
    xhttp.send();
  }

  function Time_Date() {
    var t = new Date();
    document.getElementById("time").innerHTML = t.toLocaleTimeString();
```

```
        var d = new Date();
        const dayNames = ["Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"];
        const monthNames = ["January", "February", "March", "April", "May",
"June", "July", "August", "September", "October", "November", "December"];
        document.getElementById("date").innerHTML = dayNames[d.getDay()] + ",
" + d.getDate() + "-" + monthNames[d.getMonth()] + "-" + d.getFullYear();
    }
</script>

<script>

    // define the URL of the server API
    const apiUrl = '/api/status';



    var buttons = [
        { id: "111", property: "automode" },
        { id: "222", property: "flowdetectionmode" },
        { id: "14", property: "pump1status" },
        { id: "4", property: "pump2status" }
    ];

    // define a function to fetch the status and update the <p> element
    function updateStatus() {
        fetch(apiUrl)
            .then(response => response.json())
            .then(data => {
                const pump1Button = document.getElementById('14');
                const pump2Button = document.getElementById('4');
```

```
const automodeButton = document.getElementById('111');
const flowdetectmode = document.getElementById('222');

if (data.pump1_status === '1') {
   pump1Button.checked = true;
} else {
   pump1Button.checked = false;
}

if (data.pump2_status === '1') {
   pump2Button.checked = true;
} else {
   pump2Button.checked = false;
}

if (data.control_mode === '1') {
   automodeButton.checked = true;
} else {
   automodeButton.checked = false;
}

if (data.flow_detection === '1') {
   flowdetectmode.checked = true;
} else {
   flowdetectmode.checked = false;
}

const statusElement = document.getElementById('motor-status');
statusElement.textContent = `Pump Status: ${data.status}`;
if (data.status === 'Running') {
```

```
                statusElement.style.color = 'green';
            } else if (data.status === 'Idle') {
                statusElement.style.color = 'blue';
            } else if (data.status === 'Stopped') {
                statusElement.style.color = 'red';
            }
        })
        .catch(error => console.error(error));
    }

    // update the status immediately
    updateStatus();

    // update the status every 3 seconds
    setInterval(updateStatus, 3000);


</script>


<script>
    function toggleCheckbox(element) {
        var xhr = new XMLHttpRequest();
        if (element.checked) {
            xhr.open("GET", "/data_update?output=" + element.id + "&state=1", true);
        } else {
            xhr.open("GET", "/data_update?output=" + element.id + "&state=0", true);
        }
        xhr.send();
    }
```

```
</script>
</body></html>

)rawliteral";

String outputState(int output){

 if(output == 111)
 {
  if(manual) return "checked"; else return "";
 }

 if(output == 222)
 {
  if(FDMode) return "checked"; else return "";
 }

if(output == 13)//D13
 {
  if(motor) return "checked"; else return "";
 }

  if(output == 14)//D14
 {
  if(motor2) return "checked"; else return "";
 }

 return "";

}
```

```
String processor(const String& var){
  //Serial.printvar);
  if(var == "BUTTONPLACEHOLDER"){
    String buttons = "";
  buttons += "<div class=\"grid-item\"><h4>AutoMode</h4><label
class=\"switch\"><input type=\"checkbox\" onchange=\"toggleCheckbox(this)\"
id=\"111\" " + outputState(111) + "><span
class=\"slider\"></span></label></div>";
  buttons += "<div class=\"grid-item\"><h4>FlowDetection</h4><label
class=\"switch\"><input type=\"checkbox\" onchange=\"toggleCheckbox(this)\"
id=\"222\" " + outputState(222) + "><span
class=\"slider\"></span></label></div>";
  buttons += "<div class=\"grid-item\"><h4>Pump 1</h4><label
class=\"switch\"><input type=\"checkbox\" onchange=\"toggleCheckbox(this)\"
id=\"14\" " + outputState(13) + "><span class=\"slider\"></span></label></div>";
  buttons += "<div class=\"grid-item\"><h4>Pump 2</h4><label
class=\"switch\"><input type=\"checkbox\" onchange=\"toggleCheckbox(this)\"
id=\"4\" " + outputState(14) + "><span class=\"slider\"></span></label></div>";

    return buttons;
  }
  return String();
}
```

A.11 web_Route.h

```
String getLogPage();
void handleCooldownRequest(AsyncWebServerRequest *request);
```

```
void init_webservice() {

  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
  request->send_P(200, "text/html", index_html, processor);
    });

  server.on("/version",HTTP_GET, [](AsyncWebServerRequest *request){

  request->send(200, "text/plain", String(version));

  });

 server.on("/readDistance",HTTP_GET, [](AsyncWebServerRequest *request){

  request->send(200, "text/plain", String(Tank_volume));

  });  //--> Routine to handle the call procedure handleDHT11Temperature

  server.on("/getDistance",HTTP_GET, [](AsyncWebServerRequest *request){


  request->send( 200, "application/json","{\"status\":
\"OK\",\"distance\":\""+String(Tank_volume)+"\"}" );

  });  //--> Routine to handle the call procedure



  server.on("/WMS/SFT",HTTP_GET,
```

```
  [](AsyncWebServerRequest* request){
    if(!SAFETY)
    {
    SAFETY = true;
    request->send(200, "text/plain", "SAFETY mode is On");
    }else{
    SAFETY = false;
    request->send(200, "text/plain", "SAFETY mode is Off");
    }
  });

  server.on("/logs", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send(200, "application/json", getLogPage());
   });

  server.on("/WMS/cd", HTTP_GET, [](AsyncWebServerRequest *request){
   handleCooldownRequest(request);
  });

 // Send a GET request to
<ESP_IP>/update?output=<inputMessage1>&state=<inputMessage2>
 server.on("/data_update", HTTP_GET, [] (AsyncWebServerRequest *request) {
  String inputMessage1;
  String inputMessage2;
  // GET input1 value on
<ESP_IP>/update?output=<inputMessage1>&state=<inputMessage2>
  if (request->hasParam(PARAM_INPUT_1) && request-
>hasParam(PARAM_INPUT_2)) {
    inputMessage1 = request->getParam(PARAM_INPUT_1)->value();
    inputMessage2 = request->getParam(PARAM_INPUT_2)->value();
```

```
if(inputMessage1.toInt() == 111)
{
  if(inputMessage2.toInt() == 1)
  {
    manual = true;

  }
  else{
    manual = false;
  }
}

if(inputMessage1.toInt() == 222)
{
  if(inputMessage2.toInt() == 1)
  {
    FDMode = true;

  }
  else{
    FDMode = false;
  }
}

if(inputMessage1.toInt() == D13)//D13
{
  if(inputMessage2.toInt() == 1)
  {
    motor = true;
```

```
      }
      else{
        motor = false;
      }
    }

    if(inputMessage1.toInt() == D14)//D14
    {
      if(inputMessage2.toInt() == 1)
      {
        motor2 = true;

      }
      else{
        motor2 = false;
      }
    }
  }
  else {
    inputMessage1 = "No message sent";
    inputMessage2 = "No message sent";
  }
  request->send(200, "text/plain", "OK");
});

}
```

A.12 Web_Server.h

```
#if defined(ESP32)
#define D14 14
#define D13 13
#define D12 12
#elif defined(ESP8266)
#include <sys/pgmspace.h>
#endif

#include <Arduino.h>
#include "web_HomePage.h"
AsyncWebServer server(80);
#include "API.h"
#include "web_Route.h"

void init_api();
void init_webservice();

void serverBegin() {

   // Start ElegantOTA
   // With out password
   // AsyncElegantOTA.begin(&server);
   // With password
   // AsyncElegantOTA.begin(&server, "admin", "01001011");
   // Start server
   server.begin();
}
```

```
String getLogPage() {
     String logPage = "";
     for (int i = 1; i < MAX_LOGS; i++) {
       logPage = logPage + (logs[(currentLog + i) % MAX_LOGS]);
     }
     return ("{ \"logs\":[\n" + logPage + "\n] }");
}

void handleCooldownRequest(AsyncWebServerRequest *request) {
  if (request->hasArg("time")) {
    // Get the value of the "time" parameter from the URL query string
    String timeParam = request->arg("time");
    unsigned int cooldownTime = timeParam.toInt();

    // Set the cooldown time
    setCooldownTime(cooldownTime);
    CD = true;

    // Send a response
    request->send(200, "text/plain", "Cooldown time set: " + String(cooldownTime)
+ " minutes");
  } else {
    // Invalid or missing "time" parameter
    request->send(400, "text/plain", "Invalid request");
  }
}
```