Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(навчально-науковий інститут)

факультет інформаційних технологій
(факультет)

Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи ступеня бакалавра

(бакалавра, магістра)

Здобувача вищої освіти Шеченкова Володимира Андрійовича
(ПІБ)

академічної групи 126-21-1
(шифр)

спеціальності 126 «Інформаційні системи та технології»
(код і назва спеціальності)

спеціалізації за освітньо-професійною (освітньо-науковою) програмою
(за наявності)
«Інформаційні системи та технології»
(офіційна назва)

на тему Веборієнтована інформаційна система контрольованого вирощування агрокультур з використанням RFID технології
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	доц. Соколова Н. О.			
розділів:				

Рецензент	доц. Клименко А. В.			

Нормоконтролер	проф. Коротенко В.О.			

Дніпро
2025

ЗАТВЕРДЖЕНО:
завідувач кафедри
інформаційних технологій та комп'ютерної інженерії
(повна назва)

_____ В.В. Гнатушенко
(підпис) (ініціали та прізвище)

«_____» _____20___року

ЗАВДАННЯ

на кваліфікаційну роботу
ступеня бакалавра
(бакалавра, магістра)

здобувача вищої освіти Шеченков В.А. **академічної групи** 126-21-1
(прізвище та ініціали) (шифр)

спеціальності 126 «Інформаційні системи та технології»

спеціалізації за освітньою-професійною програмою _____
(за наявності)

на тему Веборієнтована інформаційна система контрольованого вирощування агрокультур з використанням RFID технології

затверджену наказом ректора НТУ «Дніпровська політехніка» від _____ № _____

Розділ	Зміст	Термін виконання
Розділ 1. Аналіз інформаційних систем типу інтернету речей та їх складових	Огляд існуючих рішень для контрольованого вирощування агрокультур, аналіз цих рішень та їх окремих компонентів, оптимізація відповідно до вимог області рішення задачі.	21.04.2025 – 07.05.2025
Розділ 2. Архітектура інформаційної системи	Опис взаємодії розглянутих компонентів інформаційної системи, огляд налаштувань та особливостей встановлення цих компонентів для досягнення виконання попередньо оголошених вимог.	08.05.2025 – 25.05.2025
Розділ 3. Демонстрація роботи розробленої інформаційної системи	Опис взаємодії розглянутих компонентів інформаційної системи, огляд налаштувань та особливостей встановлення цих компонентів для досягнення виконання попередньо оголошених вимог.	26.05.2025 – 31.05.2025

Завдання видано _____ _____
(підпис керівника) (ініціали та прізвище)

Дата видачі _____

Дата подання до екзаменаційної комісії _____

Прийнято до виконання _____ Шеченков В.А.
(підпис здобувача вищої освіти) (ініціали та прізвище)

РЕФЕРАТ

Пояснювальна записка: 69с., 25 рис., 2 табл., 2 додатки, 15 джерел.

ІНФОРМАЦІЙНА СИСТЕМА, IOT, АРХІТЕКТУРА, АГРОКУЛЬТУРА, ВЕБДОДАТОК, КОНТРОЛЬОВАНЕ ВИРОЩУВАННЯ, REST API, RFID, ESP32, БАЗА ДАНИХ

Об’єкт кваліфікаційної роботи: веборієнтована моніторингова інформаційна система типу IoT.

Предмет кваліфікаційної роботи: компоненти архітектури створюваної інформаційної системи підходящі для умов зрощування агрокультур типу ферм або теплиць.

Мета роботи: Створення веборієнтованої інформаційної системи для моніторингу навколишнього середовища в якому ростуть агрокультури, створення можливості порівняння отриманих даних з рекомендованими показниками, таким чином підвищуючи ефективність росту агрокультур.

У вступі описано перспективу поєднання інформаційних технологій та сільського господарства та ситуацію на ринку моніторингових систем для фермерства.

У першому розділі розглянуто порівняння та вибір найбільш підходящих компонентів для створення веборієнтованої моніторингової інформаційної системи в умовах вирощування агрокультур.

У другому розділі описано взаємодію попередньо обраних компонентів інформаційної системи для досягнення поставленої мети а також особливості налаштування цих компонентів для коректного сполучення між компонентами.

У третьому розділі продемонстровано різні сценарії використання інформаційної системи та результати виконання цих сценаріїв.

ABSTRACT

Explanatory note: 69 pages, 25 figures, 2 tables, 2 applications, 15 sources.

INFORMATION SYSTEM, IOT, ARCHITECTURE, AGRICULTURE, WEB APPLICATION, CONTROLLED CULTIVATION, REST API, RFID, ESP32, DATABASE

Object of qualification work: web-oriented monitoring information system of the IoT type.

Subject of qualification work: components of the architecture of the created information system suitable for growing crops such as farms or greenhouses.

Purpose of the work: Creation of a web-oriented information system for monitoring the environment in which crops grow, creating the ability to compare the data obtained with the recommended values, thus increasing the efficiency of crop growth.

The introduction describes the perspective of combining information technology and agriculture and the situation on the market of monitoring systems for farming.

The first section discusses the comparison and selection of the most suitable components for creating a web-based monitoring information system for growing crops.

The second section describes the interaction of the pre-selected components of the information system to achieve the goal, as well as the peculiarities of setting up these components for correct communication between the components.

The third section demonstrates various scenarios of using the information system and the results of these scenarios.

ЗМІСТ

ВСТУП

Агрокультурне ремесло безумовно відіграє невимовно важливу роль у нашому повсякденному житті, знаходячись чи не найвищому місці по критичності як для виробників та власників агрокультурного бізнесу, так і для споживачів виробництва цих бізнесів. Деякі регіони та навіть країни будують свою економіку та капітал виключно на рослинництві, що робить цю галузь невід'ємною частиною безпосереднього існування цього регіону чи країни.

Інформаційні технології з кожним роком все більше проникають у наше повсякдення, і фермерство тому не виключення. Світло в загонах може вмикатися автоматично з настанням темряви, а ворота відкриватися за допомогою додатку у смартфоні.

Поєднання IT та сільського господарювання містить не абиякий позитивний вплив майже на всіх етапах створення та споживання агрокультурного виробництва. Отримання та обробка поточних даних про статус різних фермерських активів можуть вплинути на подальше існування виробництва загалом, що запобігає виникненню критичних ситуацій.

Моніторинг та аналіз станів навколишнього середовища аграрних культур є одною з найголовніших умов отримання успішного врожаю. Ручну перевірку оточення в якому ростуть культури можна цілком замінити на автоматизовану перевірку за допомогою елементів розумних речей (Internet of Things), а подальше збереження та обробка результатів може бути виконана за допомогою програмного коду та відображено у застосунку або вебсайті.

Поточний ринок налічує немалу кількість готових рішень для відслідковування та аналітики показників оточення для рослин, але ці рішення не добре оптимізовані для обробки масштабів підприємства. Інші ж рішення, які є пристосованими до великих об'ємів збору та обробки інформації є доволі дорогими та менш гнучкими у плані розширення та додавання нових позицій до існуючої системи.

РОЗДІЛ 1
АНАЛІЗ ІНФОРМАЦІЙНИХ СИСТЕМ ТИПУ ІНТЕРНЕТУ РЕЧЕЙ ТА ЇХ СКЛАДОВИХ

Інтернет речей (IoT) представляє собою різноманітні фізичні об'єкти, які оснащені датчиками і програмами, що дають змогу їм взаємодіяти між собою з невеликим або без втручання людини, збираючи та передаючи дані через мережу. IoT охоплює безліч «розумних», комп'ютеризованих пристроїв, які сьогодні можна зустріти дуже часто в повсякденному житті, і які здатні підключатися до Інтернету чи взаємодіяти через бездротові мережі; «розумні» елементи IoT включають телефони, побутову техніку, термостати, освітлювальні системи, системи для поливу, камери відеоспостереження, автомобілі, навіть тварин та цілі міста.

Принцип взаємодії елементів IoT системи між собою починається з датчиків або сенсорів, які є джерелами даних про стан фізичного середовища у місцях знаходження цих елементів, та з яких ці дані будуть потім передані проміжними пристроями (мікроконтролерами) до серверів на яких безпосередньо буде виконані отримання, обробка, аналіз та збереження отриманих даних з сенсорів.

Початок Інтернету речей датуються кінцем 1960-х років. Саме тоді група видатних дослідників почала вивчати способи з'єднання комп'ютерів і систем. Яскравим прикладом цієї роботи була ARPANET, мережа, створена Агентством передових дослідницьких проектів (ARPA) Міністерства оборони США; ця мережа була попередницею сучасного Інтернету [1].

У 1997 році британський технолог Кевін Ештон, співзасновник Центру автоматичної ідентифікації в Масачусетському технологічному інституті, почав досліджувати технологічну основу - радіочастотну ідентифікацію (RFID), яка дозволила б фізичним пристроям з'єднуватися за допомогою

мікрочіпів і бездротових сигналів, і саме у своїй промові в 1999 році Ештон ввів в обіг фразу «Інтернет речей» [1].

Використання мікроконтролерів у поєднанні з сенсорами стану навколишнього середовища може дуже легко оптимізувати збір та передачу даних. Оскільки програмування мікроконтролерів відбувається безпосередньо розробником, у процес зчитування та передачі можна закласти будь-яку логіку та послідовність. Обробка даних в цьому випадку може бути виконана як локально, в тому числі використовуючи одноплатні комп'ютери, так і за допомогою хмарних обчислень, що також дає додаткову гнучкість та заощадливість за наявності великої бази для аналізу. Технологія RFID (Radio Frequency IDentification) у вигляді міток може бути найлегшим та найшвидшим методом отримання доступу до зібраних та/або оброблених даних, також додає можливість повторного використання цих міток шляхом перевизначення отримання різних типів даних.

1.1 Огляд вибору IoT сенсорів

Сенсори виступатимуть головним джерелом даних про навколишнє середовище та фізичні властивості ґрунту. Базуючись на показниках отриманих з сенсорів будуть прийняті подальші рішення щодо покращення умов вирощування рослин.

Показники ґрунту в якому ростуть культури майже так само, якщо не більше, впливають на ефективність вирощування рослин. Одними з найголовніших показників є температура ґрунту та вологість ґрунту в якому ростуть культури. Оптимальні умови вологості та температури ґрунту сприяють кращому засвоєнню мікроелементів, таких як мідь, цинк і бор, такими культурами, як пшениця, горох і ріпак, що призводить до збільшення врожайності біомаси [2].

Задля вимірювання вологості ґрунту можна перелічити безліч пропозицій серед різноманіття сенсорів, але основними типами сенсорів

можна виокремити ємнісні та резистивні сенсори. Резистивні сенсори трохи дешевші, але це єдина позитивна відмінність від ємнісних датчиків вологості ґрунту. Ємнісні ж більш точні у вимірах, не схильні до корозії, оскільки не потребують прямого електричного контакту із ґрунтом, тому в додаток до попередніх переваг ще й дуже довговічні [3].

Навколишні умови за яких ростуть культури також відіграють монументальну роль в успішному зрощенні. Серед найголовніших можна виділити температуру повітря, навколишня вологість та освітленість. Ці показники безпосередньо впливають на внутрішні процеси росту агрокультур, такі як фотосинтез, проростання, випаровування води та інші.

Існують універсальні сенсори які одночасно вимірюють температуру, вологість та атмосферний тиск, доволі точні у вимірах та недорогі. Сенсор BME280 зможе надавати потрібні дані задля контролю навколишнього середовища в якому ростуть рослини.

Оскільки універсальні сенсори які б включали абсолютно усі потрібні сенсори дуже дорогі або робляться на замовлення, що може викликати затримку в часі та нераціональне використання коштів, а мікроконтролери містять достатню кількість пінів, для вимірювання освітленості можна використати окремий сенсор. VEML7700 містить широкий діапазон виміру та має гарну точність вимірювання.

1.2 Аналіз вибору мікроконтролерів

Мікроконтролери це фактично дуже малі у розмірах, споживанні енергії та продуктивності комп’ютери. Їх можна дуже легко запрограмувати на виконання будь-якого програмного коду. Існує багато різновидів мікроконтролерів за призначенням, потужністю, розмірами та вартістю, але саме для IoT можна виділити декілька найпопулярніших варіантів.

В області рішення поставленої задачі, мікроконтролер повинен збирати інформацію з сенсорів, за потреби форматувати її у вигляд готовий для

перегляду та / або обробки, бути компактним у розмірах, енергоефективним, в міру потужним та недорогим.

Серед найпопулярніших мікроконтролерів для IoT можна виділити наступні плати розробки: Arduino UNO, Arduino NANO, Raspberry Pi 3A+, ESP32-C3 SuperMini та ESP8266 D1 mini. Саме ці плати розробки будуть порівнюватись між собою задля обрання найбільш оптимального варіанту для поставленого завдання.

За порівнянням параметрів компактності з рисунку 1.1, найбільш компактними можна виокремити плати ESP32-C3 SuperMini, ESP8266 D1 mini та Arduino NANO з середньою площею у 6.05 см що можна порівняти з площею поштової марки. Незважаючи на свою компактність, ця плата не поступається доступністю пінів підключення та налічує 16 пінів різних видів. Компактність цієї плати дозволяє зручніше розташувати її біля сенсора або змонтувати її прямо на сенсор, що дозволяє більш зручно розташувати компоненти зчитування біля рослин.

Рисунок 1.1 – Діаграма порівняння розмірів плат розробки

Порівняння споживання енергії за рисунком 1.2 (сторінка 8) вказує, що більшість плат споживає до 0.5 Ватт енергії окрім Raspberry Pi 3A+. Оскільки плати будуть працювати автономно, тобто від акумуляторів, електроспоживання є дуже важливим параметром при виборі мікроконтролера, особливо в умовах великих агропідприємств, де підтримка автономності багатьох систем моніторингу одночасно може бути проблематичним. Плата Raspberry Pi 3A+ підтримує живлення виключно від блоку живлення, що кардинально зменшує компактність та можливість компактного живлення.

Рисунок 1.2 – Діаграма порівняння енергоспоживання плат розробки

Наступним не менш важливим параметром буде ціна, оскільки чим більше агрокультурне підприємство тим більше потрібно закупити мікроконтролерів. За рисунком 1.3 (сторінка 9) найдешевшими платами виявились ESP8266 D1 mini та ESP32-C3 SuperMini. Закупівля цих плат дозволить розмістити якомога більше систем моніторингу за меншу вартість.

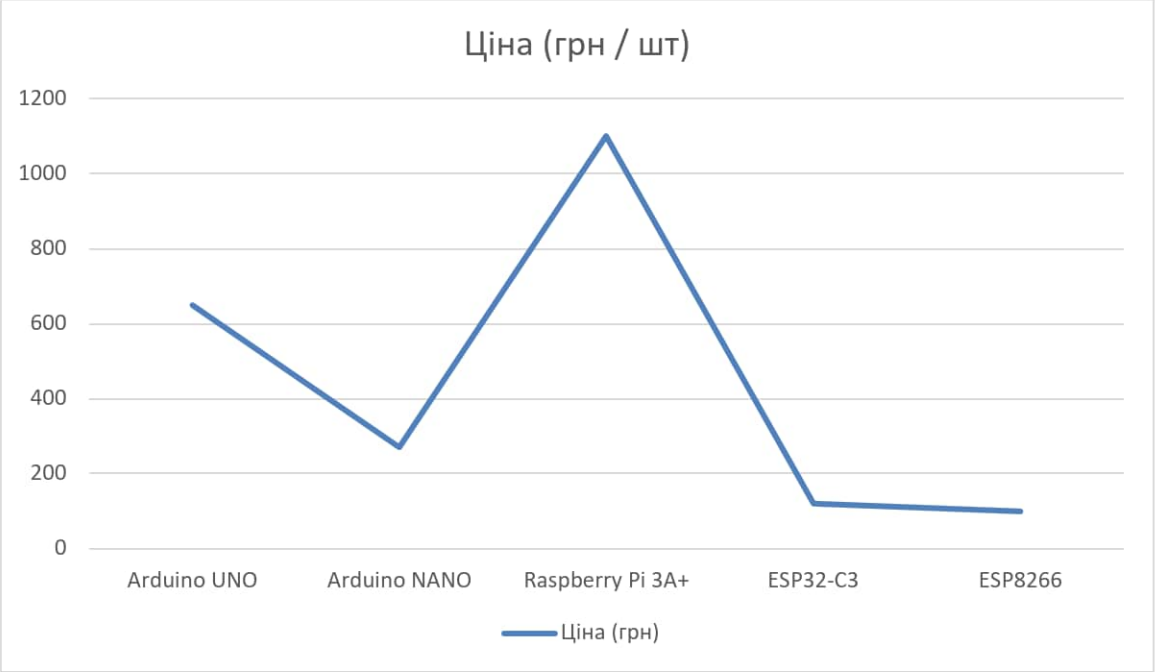

Рисунок 1.3 – Діаграма порівняння роздрібної ціни плат розробки

Продуктивність плат можна визначити за частотою процесора, кількістю ядер та кількість оперативної пам'яті. Таблиця 1 показала, що найпотужнішою платою виявилась Raspberry Pi 3A+, випереджаючи усіх з дуже великим відривом. Але для IoT потужність цієї плати буде надлишковою, тому незважаючи на показники, буде задіяна дуже мала частина використання ресурсів цієї плати.

Таблиця 1. Характеристика ефективності плат розробки

	Частота процесору (МГц)	Кількість ядер процесору (шт)	Об'єм оперативної пам'яті (Кб)
Arduino UNO	16	1	2
Arduino NANO	16	1	2
Raspberry Pi 3A+	1400	4	524288

Продовження таблиці №1

ESP32-C3	160	1	400
ESP8266	80	1	80

Тип живлення було обрано спираючись на реалії оточення, в якому будуть рости агрокультури та характер споживання енергії мікроконтролерами. Оскільки проведення постійного джерела живлення до усіх рослин може бути доволі складним викликом, використання портативних акумуляторів є дуже гнучким та стійким рішенням для живлення мікроконтролерів, в тому числі через ефективне енергоспоживання самими платами розробки.

Підбиваючи підсумки можна дійти до наступних заключень: плати Arduino майже ніде не показали лідерство серед порівняння, маючи лише доволі високу компактність для плати NANO; Raspberry Pi 3A+ виявилась більш підходящою для IoT систем в приміщенні, де немає прив’язки до розмірів та типу живлення, але є потреба в обробці багатьох сенсорів одночасно, що потребуватиме великої потужності. Останніми залишаються плати ESP, які є чи не ідеальними майже по всім параметрам: компактні, дешеві, достатньо потужні; але в цьому випадку плата ESP32-C3 SuperMini буде найбільш оптимальним варіантом за співвідношенням ціни до потужності та потужності до енергоспоживання.

1.3 Середовище програмування мікроконтролерів

Для того щоб мікроконтролери збирали, обробляли та відправляли інформацію на сервер де ця інформація буде в подальшому оброблена та збережена, плати розробки треба запрограмувати на виконання логіки, яка буде відтворювати вище перелічені дії.

Для обраної плати розробки ESP32-C3 SuperMini існує доволі багато Інтегрованих середовищ розробки (IDE). Ці середовища підтримують

широкий функціонал який допоможе під час створення програмного коду для мікроконтролеру. Функціонал може включати форматування коду, автоматичну табуляцію, підтримку безлічі бібліотек для спрощення впровадження додаткового функціоналу в код та інші.

Задля початку роботи з кодом для мікроконтролеру, потрібно встановити саме IDE. Серед рекомендованих середовищ для створення програмного коду для ESP32-C3 пропонується використовувати Visual Studio Code або Eclipse. Ці середовища розробки є одними з найрозповсюдженіших серед розробників не тільки у сфері IoT, але й у всіх сферах інформаційних технологій. Єдине що може бути досить незручним під час створення коду для мікроконтролерів це налаштування IDE. Перед початком безпосереднього програмування потрібно виконати дуже великий ряд дій, такий як встановлення фреймворку для роботи з мікроконтролером, налаштування окремого проєкту, наявність додаткових компонентів та інші.

Arduino IDE це середовище розробки, призначене саме для програмування мікроконтролерів. Тільки маючи посилання на файл конфігурацій можна ледве не в одне натискання мишею встановити усі сумісні драйвера, бібліотеки та фреймворки потрібні для роботи з платою розробки. Саме Arduino IDE буде найзручнішим у створенні програмного коду для IoT логіки та мікроконтролерів.

1.4 Server-side hardware

В умовах зрощування агрокультур іноді може існувати проблема з наявністю вільного простору для інструментів догляду за культурами, не кажучи про комп’ютерне або серверне обладнання.

Попередньо під час розглядання плат розробки було виокремлено плату, яка має усі якості мікроконтролеру, але не підходить в якості мікроконтролеру для зчитування даних з сенсорів. Мова йдеться про плату Raspberry Pi, і якщо подивитися на цю плату розробки не зі сторони

мікроконтролеру, а як на рішення компактного серверу, це може бути чудовим варіантом для фермерських умов.

Цей одноплатний комп'ютер підтримує не тільки програмування в якості мікроконтролера, але й може незалежно існувати як повноцінний комп'ютер під управлінням операційної системи Linux – найпопулярнішої ОС для серверів. Більші параметри продуктивності, компактні розміри, модульність та економне споживання енергії можуть вказувати на те, що цей варіант для використання в якості серверної архітектури буде одним з найбільш підходящих для області рішення цього завдання.

1.5 Backend частина серверу

Backend - це логічний простір з функціоналом та операціями з програмного забезпечення або інформаційної системи. Одним з його реалізацій в цій веборієнтованій інформаційній системі буде отримання, аналіз, форматування та зберігання даних отриманих сенсорів. [5]

Загалом Backend можна назвати набір сервісів сполучених між собою іншими сервісами. Головна особливість цієї програмної серверної частини що всі дії відбуваються у фоні, без фактичного відображення дій. Дані спочатку входять до Backend'у, там обробляються, а на виході ми отримуємо готові оброблені дані, набори даних або інші дії з даними, такі як збереження в базі даних. Виконані дії залежать від набору сервісів які містить Backend та яким чином запрограмовані ті чи інші сервіси.

1.5.1 Вибір бази даних. Для зберігання даних які будуть зібрані з сенсорів та потім будуть використані в подальшому буде використовуватись база даних. Оскільки наразі існує багато продуктів в цій сфері, потрібно ретельно обрати коректний варіант для поточної задачі.

Оскільки дані будуть надходити з десятків, якщо не сотень сенсорів, швидкодія та ефективність бази даних буде стояти в голові вимог для обрання фінального продукту. Окрім цього, бажано щоб база даних була у

відкритому доступі (open source) щоб скоротити витрати на ліцензії. Також наявність постійної підтримки є вимогою, щоб можливі вразливості безпеки були виправлені та база даних мала оновлення в цілому.

Найбільш підходящими базами даних за перерахованими вимогами можна виділити MySQL, PostgreSQL та SQLite. Ці продукти є дуже популярними серед проєктів та можуть бути підходящими для цього завдання.

Легкість MySQL допомагає скоротити час, що витрачається на встановлення та налаштування, дозволяючи розробникам швидко створювати прототипи рішень. Однак багатофункціональність PostgreSQL може краще відповідати вимогам певної програми або варіанту використання, зменшуючи залежність від створення кастомних рішень для реалізації подібної функціональності. В умовах великої агрокультурної компанії PostgreSQL буде більш підходящим завдяки наявності потужної індексації та паралельних запитів. Хоч MySQL буде краще для читання даних, PostgreSQL теж має непогану ефективність в цій області. [7]

SQLite краще підійде для легких, портативних додатків, а PostgreSQL для надійних, багатофункціональних систем з високими вимогами до масштабованості. За результатами поверхневого аналізу цих двох продуктів які відображені у таблиці 2, PostgreSQL є переможцем майже в усіх категоріях. Єдиною категорією де PostgreSQL програв це ліцензування, де SQLite є суспільним надбанням, тобто може використовуватися без будь-яких обмежень, пов'язаних з ліцензуванням. З іншого боку, PostgreSQL працює під PostgreSQL License, дозвільною ліцензією з відкритим вихідним кодом.

Таблиця 2. Результати порівняння PostgreSQL та SQLite

Категорія	**Переможець**
Типи даних	PostgreSQL
Побудова	PostgreSQL

Продовження таблиці №2

ACID-відповідність	PostgreSQL
Ліцензія	SQLite
Адміністрація	PostgreSQL
Ефективність	PostgreSQL
Сумісність з SQL	PostgreSQL
Функціонал	PostgreSQL
Масштабованість	PostgreSQL
Абсолютний переможець: PostgreSQL	

1.5.2 Application Programming Interface (API). Для відображення зібраних даних із сенсорів на майбутньому вебдодатку, потрібно зробити запит до бази даних на отримання даних. Це можна зробити за допомогою виклику SQL запитів напряму з Frontend частини серверу або використовуючи API.

У цьому випадку кращим варіантом буде створення власного API, оскільки виконання SQL запитів напряму має дуже великі ризики безпеки, такі як SQL ін’єкції та розкриття структури бази даних, а також дуже обмежений функціонал, який не дозволить виконати обробку та аналіз отриманих даних з бази даних.

Найкращою мовою програмування саме для API буде TypeScript у поєднані з модулем Express та додатковими бібліотеками які забезпечать точність отриманих та надісланих даних, додаткові перевірки, функціонал для аналізу та обробки даних та безпеку.

Статична типізація та розширений інструментарій TypeScript у поєднанні з гнучкістю та простотою Express.js роблять їх виграшною комбінацією для розробки API. Ефективно використовуючи ці технології та дотримуючись найкращих практик, буде створено надійне, продуктивне та масштабоване API, що відповідає вимогам сучасних додатків. Також, завдяки ранньому виявленню помилок, спрощенню співпраці та інтеграції з

сучасними інструментами, цей дует забезпечує підтримку та масштабування Backend-сервісів. [8]

1.5.3 Обґрунтування вибору вебсерверу. Apache2 - це безкоштовний крос-платформний вебсервер з відкритим вихідним кодом. Користувацький агент, зазвичай веббраузер, робить запит на вебсторінку або інший ресурс за допомогою HTTP протоколу, а вебсервер відповідає вмістом цього ресурсу або повідомленням про помилку.

Вебсервер Apache - це надійний, гнучкий і безпечний вебсервер, який вже понад два десятиліття є найкращим вибором для веброзробників і адміністраторів. Хоча він має деякі обмеження, він є чудовим вибором для більшості вебсайтів завдяки своїй стабільності, функціям безпеки та широкій підтримці спільноти. [10]

У Apache2 є конкурент у вигляді NGINX, який також є вебсервером, але більше підходить для сайтів, які мають особливо високу відвідуваність та більше підходить для використання в якості точки входу (ingress) для мікросервісів. В свою ж чергу Apache2 є дуже гнучким традиційним вебсервером, що є підходящим варіантом для локального сервера, який також має підтримку .htaccess, який спрощує налаштування доступу за кожним каталогом.

1.6 Frontend частина серверу

Головною частиною Frontend’у буде сторінка яка відображатиме дані про фізичні властивості ґрунту та навколишнього середовища рослини біля якої було відскановано NFC RFID мітку (рис. 1). Окрім даних про ґрунт та середовище також можна побачити фотографію рослини та її назву.

Рисунок 1.4 – Приклад вебсторінки

Вебсторінка використовує стандартний набір інструментів для формування шаблону сторінки, такі як HTML 5 та CSS. Вебсервер міститиме лише один файл HTML, один файл CSS та файли які містять шрифти.

Дані про властивості ґрунту та середовища рослини будуть підставлятися в універсальний шаблон сторінки. Самі дані будуть братися з двох API: дані про середовище з локального API та рекомендовані значення з OpenPlantBook API. Для отримання цих даних та підставлення їх у шаблон сторінки існуватиме окремий каталог який міститиме JavaScript скрипти, які й будуть виконувати запити та підставляти отримані дані в потрібні поля. Окрім цього, для ідентифікації кожної рослини в URL сторінки буде використовуватись параметр plant_id за яким і будуть робитися запити до API. Наприклад, на рис. 1 відображено дані про рослину Anthurium andraeanum, відповідно адреса цієї сторінки буде наступна: http://agrocult.local/webpages/template_plant.html?plant_id=anthurium%20andraeanum0, де найменування рослини буде anthurium%20andraeanum і у випадку якщо рослин з однаковим типом декілька, після найменування йде порядковий номер рослини. Далі найменування та ідентифікатор

видобувається з параметра з посилання та за ними відбувається запит на отримання даних з обох API, після чого отримані дані розміщуються в поля шаблону сторінки.

1.7 NFC RFID мітки

RFID розшифровується як радіочастотна ідентифікація, технологія, яка дозволяє зчитувати та зберігати інформацію бездротовим способом. Це робиться за допомогою мікрочіпа (мітки) та зчитувача (сканера). Мітка містить дані, які зчитувач може зчитувати дистанційно за допомогою радіохвиль. [11]

У випадку нашої веб орієнтованої системи, мітки міститимуть посилання на веб сайт з де буде відображено інформацію про середовище в якому росте рослина. В кожної рослини буде унікальна яка буде вести на сторінку про середовище виключно рослини біля якої була розміщена мітка.

Заздалегідь сформоване посилання на сторінку рослини буде записано на RFID мітку за допомогою смартфону, який підтримує технологію NFC (Near-Field Communication, дослівно «зв'язок на невеликих відстанях»), на якому повинно бути встановлено додаток NFC Tools, за допомогою якого і буде виконуватись запис заздалегідь підготовлених посилань на мітки.

Для зчитування посилань та переходу за ними у браузер потрібен тільки смартфон з підтримкою технології NFC який матиме попередньо встановлений будь-який варіант браузерного клієнту. Для зчитування мітки потрібно просто піднести телефон до RFID мітки, після цього посилання яке містить ця мітка буде автоматично відкрито у браузері на смартфонах під управлінням Android або буде запропоновано відкрити посилання у браузері у вигляді спливаючого повідомлення на телефонах під управлінням iOS.

1.8 Зберігання та контроль версій файлів та коду

Безумовно, якщо проєкт створюється як продукт який планується використовувати довгий період часу і вірогідніше у багатьох клієнтів одночасно, з часом буде виявлено помилки або недоліки роботи програмної частини інформаційної системи.

Задля легкого впровадження принципу CICD (continuous integration and continuous development, безперервна інтеграція та безперервна розробка) існують сервіси керування версіями файлів та спільної розробки типу Git. Git часто використовується для контролю вихідного коду програмістами, які спільно розробляють програмне забезпечення або просто в якості хостингу файлів які містять код програм. [15]

Серед найпопулярніших платформ контролю версій розробки існують два найбільших рішення - GitHub та GitLab. Незважаючи на схожі назви це дві різних платформи які за принципом доволі схожі, але надають унікальні інструменти в порівнянні один до одного.

Історично GitLab мав наголос на приватності та ексклюзивності, та був більше націлений на великі корпорації, які потребують приватність та гарантовану безпеку коду. Також GitLab має дуже велику все-в-одному DevOps (Developer Operations) платформу, яка включає усі функції потрібні великій компанії з багатьма розробниками. Окрім цього існує можливість виділеного хмарного хостингу Git сервісу від GitLab, який забезпечує контроль над даними, операціями та середовищем розробки. Але в порівнянні з GitHub, ця платформа має набагато меншу спільноту, тому в разі виникнення питань або складнощів роботи з сервісом буде складніше знайти інструкції або статті написані користувачами платформи для інших користувачів.

GitHub – платформа більш націлена на Open Source розробку (з вільним кодом), але також надає можливість створення приватних репозиторіїв, як на GitLab, в тому числі безкоштовно для некорпоративних користувачів. Загалом GitHub має більш зрозумілий та інтуїтивний інтерфейс, що дозволить почати роботу з цією платформою навіть недосвідченому

користувачу. Також перевагою є сервіс GitHub CLI що додає легкість в налаштуванні Git пайплайну. Але на відміну від GitLab, GitHub має менше вбудованих DevOps інструментів, що може створити потребу у впровадженні зовнішніх сервісів та інструментів.

Якщо розглядати ці дві платформи в межах сфери цього проєкту, GitHub буде найкращим варіантом. Він містить багато інструментів та сервісів які можна отримати безкоштовно з деякими обмеженнями, яких буде достатньо для виконання поставленої задачі. Також наявність великої спільноти дозволить швидше та комфортніше опанувати саму платформу. Загалом, використання GitHub є більш перспективним варіантом в порівнянні з GitLab, не зважаючи на наявність функцій які забезпечують приватність та більший вбудований DevOps функціонал. Наявність зручного та швидкого налаштування, схожого функціоналу з GitLab та більшої спільноти робить GitHub більш привабливим варіантом для використання саме для цього проєкту.

1.9 Середовище розробки коду

Оскільки ця робота вимагатиме розробку окремих рішень у вигляді програмного коду, використання IDE (Integrated Development Environment, інтегроване середовище розробки) покращить досвід розробки. Для успішного виконання поставленої задачі потрібно обрати середовище розробки яке найбільш підходило би для створення вебдодатків, оскільки створювана інформаційна система є веборієнтовною.

Більшість сучасних IDE мають приблизно однаковий набір функціоналу, наприклад:

- Перевірка лексичної коректності коду;
- Контроль помилок та пропозиції щодо їх вирішення;
- Надання послуг від зовнішніх постачальників та інші.

За доступністю, наявністю потрібних інструментів та відповідністю до мети, серед найпопулярніших рішень можна виокремити два середовища розробки – JetBrains WebStorm та Visual Studio Code.

Visual Studio Code є одним з найпопулярніших, якщо не найпопулярнішим IDE. Найголовнішою перевагою над іншими середовищами розробки є відкритість та легкість цієї програми. Це IDE можливо встановити майже на будь-який девайс під управлінням багатьох різних архітектур. Разом з популярністю також завжди йде наявність великої спільноти, що ініціює створення багато документацій, додаткових надбудов та швидке вирішення можливих помилок у роботі програми. Наявність розширень та додаткових інструментів створених спільнотою також робить це IDE дуже гнучким та підходящим не тільки для розробки вебдодатків, а й загалом для створення будь-яких рішень та для різних сфер.

Наявність гнучкості та відкритості можуть бути як перевагою так і недоліком для середовища розробки. Використання надбудов створених спільнотою може створити ризик небезпеки для розроблюваного коду. У випадку з Visual Studio Code встановлення додаткових розширень є неминучим, оскільки сама програма не має ніяких компіляторів або додаткових середовищ. У випадку цієї задачі потрібно буде встановлювати достатньо велику кількість надбудов, які можуть не тільки вплинути на ефективність роботи середовища розробки, але й створюють ризик конфліктів між розширеннями та компонентами, оскільки більшість розширень створюються незалежно один від одного та з розрахунком на стабільну роботу виключно як самостійних надбудов.

На відміну від Visual Studio Code, середовище розробки JetBrains WebStorm має набір усіх компонентів для комфортної розробки веборієнтовних додатків. Окрім наявності потрібних компіляторів одразу після встановлення та без додаткового налаштування, WebStorm також має інтегровані інструменти розробки та подальшої наладки, як наприклад локальний вебсервер або клієнт Git який можна використовувати з будь-якою

платформою контролю версій коду. Об’єктивно, користувацький інтерфейс WebStorm є більш зрозумілим, бо майже постійно зовнішній вигляд програми залишається незмінним через відсутність потреби у встановленні додаткових розширень.

Головний недолік IDE від JetBrains це те, що це середовище розробки знаходиться у закритій розробці, що робить виправлення наявних в програмі помилок більш довгим процесом. Також це створює деякі обмеження у розширенні спільноти, що спричиняє зменшення доступної документації та ускладнює розробку додаткових компонентів.

Беручи до увагу мету у вигляді наявності інструментів та орієнтовність на розробку вебдодатків найкращим варіантом буде середовище JetBrains WebStorm. Беззаперечно, з гнучкістю Visual Studio Code є можливість створити та отримати досвід еквівалентний використанню WebStorm, але ця гнучкість і є недоліком Visual Studio Code. Щоб створити цей досвід потрібно виконати велику кількість налаштувань та встановити багато додаткових компонентів, що також може бути дуже затратним у часі. Беручи до уваги, що існує Community версія IDE WebStorm, яка є так само безкоштовною як і Visual Studio Code, наявність функціоналу та зручність WebStorm робить це середовище найбільш підходящим для сфери цього завдання.

1.10 Висновки до розділу та постановка задачі на розробку IoT системи

В першому розділі було проведено аналіз компонентів для створення інформаційної системи типу IoT. Розглянуто вибір як фізичних складових так і програмних компонентів та рішень.

Серед фізичних складових було порівняно різні види сенсорів, їх типи та як вони впливають на похибку при зчитуванні даних. Також було досліджено вибір мікроконтролерів відповідно до вимог задачі. Окрім цього

було обрано серверну архітектуру яка підходила би під завантаженість та виклики в умовах агрокультурного середовища.

Щодо програмних рішень було визначено компоненти інформаційної системи які буде розроблено та проаналізовано які сервіси та інструменти будуть мати найбільш позитивний вплив під час розробки та роботи програмної частини IoT системи.

Виходячи з результатів отриманих під час аналізу виконаного у розділі, в даній роботі буде розроблено веборієнтовну інформаційну систему типу IoT, яка надає можливість контролювати фізичні показники середовища в якому ростуть різноманітні агрокультури.

Інформаційна система реалізуватиме наступний функціонал:

- зчитування вологості та температури ґрунту;
- надання рекомендованих значень вологості та температури ґрунту;
- зчитування температури, вологості, температури та освітленості навколишнього середовища рослини;
- надання рекомендованих значень температури, вологості, температури та освітленості навколишнього середовища рослини;
- зручне відображення даних будь-якого пристрою який має браузер;
- легкий доступ до даних за допомогою технології RFID.

РОЗДІЛ 2
АРХІТЕКТУРА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Топологія та послідовність системи

Точкою входу взаємодії клієнтської частини з інформаційною системою є сканування RFID-мітки за допомогою смартфону з підтримкою NFC технології. Тоді смартфон за посередництва WiFi маршрутизатора робить запит на локальний сервер. Після отримання запиту сервер повертає HTML сторінку відповідно до посилання, яке містилося у запиті смартфону та яке було отримано смартфоном з RFID-мітки.

Взаємодія системи зчитування даних, а саме сенсорів та мікроконтролерів відрізняється від попередньо описаної клієнтської частини. Спочатку мікроконтролер робить зчитування з сенсорів, після чого так само за посередництва WiFi маршрутизатора дані зібрані з сенсорів відправляться до локального серверу для подальшої обробки. Схему взаємодії в інформаційній системі зображено на рисунку 2.1.

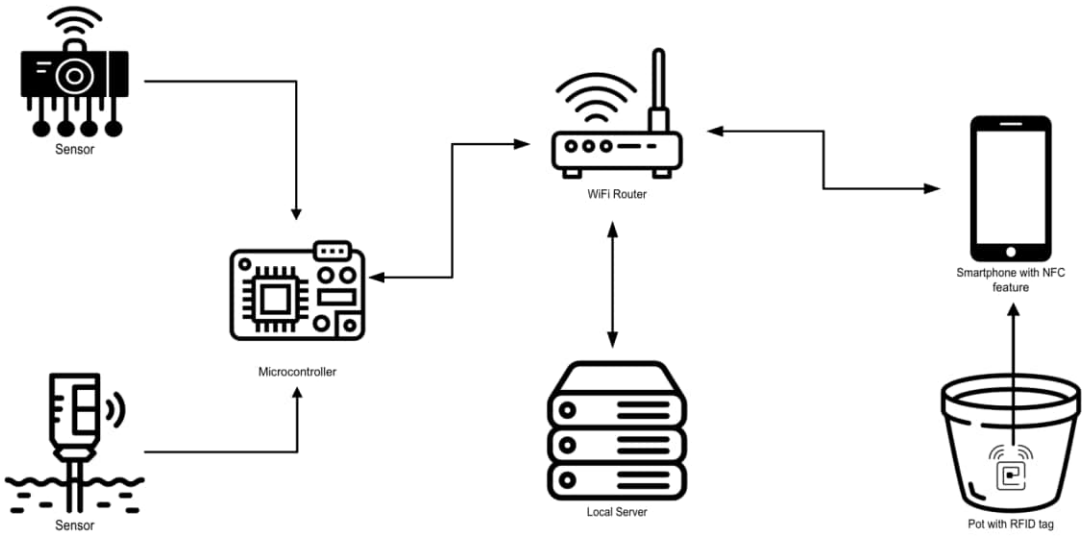

Рисунок 2.1 – Структура інформаційної системи

2.2 Структура серверної частини інформаційної системи

Зібрані дані з систем моніторингу завжди спочатку проходять через локальний API. В залежності від типу надісланих даних вони можуть бути проаналізовані та оптимізовані для подальшого зберігання та відображення безпосередньо за допомогою внутрішнього функціоналу API. Після обробки API зв’язується з базою даних та відправляє оброблені дані до відповідної таблиці бази даних. Отримання, обробка та збереження інформації з сенсорів виконується в асинхронному режимі, тобто не за запитом користувача, сервісу чи будь-якого іншого клієнта.

Коли клієнт робить запит на отримання інформації, він відправляє запит на отримання HTML сторінки з вебсерверу. Під час запиту цієї сторінки також передається ідентифікатор рослини, за допомогою якого і буде виконуватись формування запитів як до локального API так і до зовнішнього. Зв’язок між сервісними елементами відображено на рисунку 2.2.

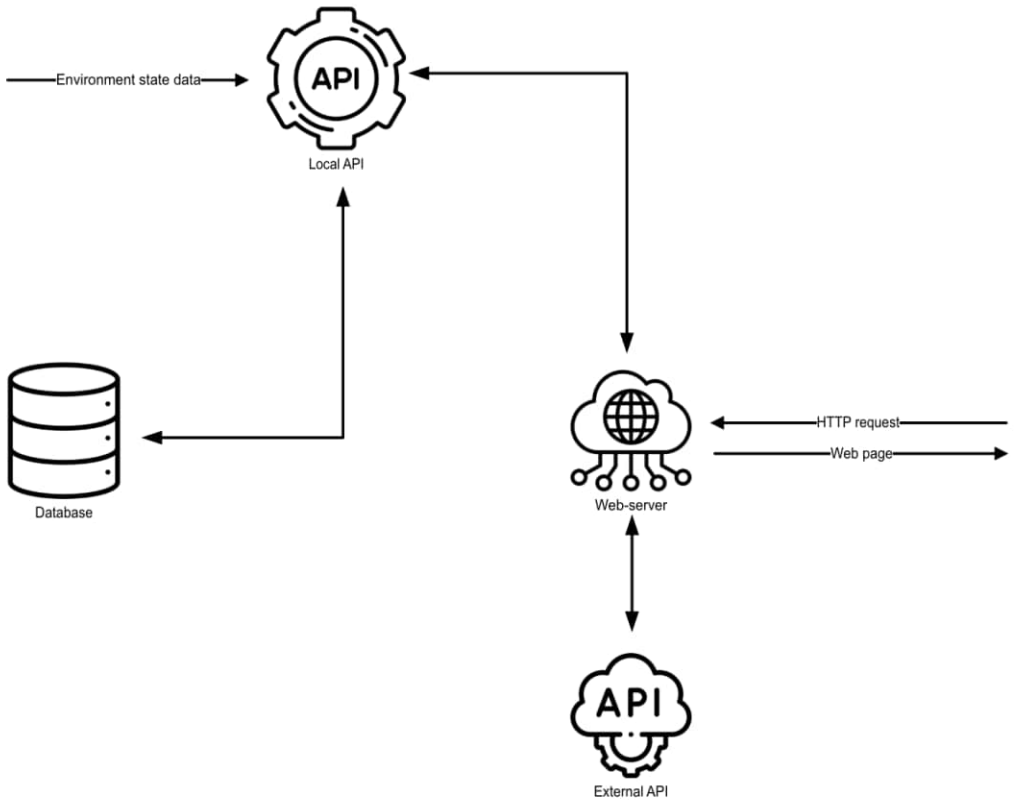

Рисунок 2.2 – Зв’язок між сервісами

2.3 Особливості встановлення операційної системи Raspbian OS

Оскільки в якості серверного обладнання було обрано Raspberry Pi 5, що є нестандартним форматом комп'ютера, для нього існують спеціально оптимізовані варіанти образів оперативних систем.

Для створення та запису образів для Raspberry Pi існує програмне забезпечення з назвою Raspberry Pi Imager (рисунок 2.3), в якому можна обрати найбільш підходящий варіант образу оперативної системи. Після вибору образу за допомогою тієї ж самої програми можна також попередньо налаштувати або навіть встановити додаткові утиліти до відповідних образів перед безпосереднім встановленням оперативної системи на одноплатний комп'ютер.

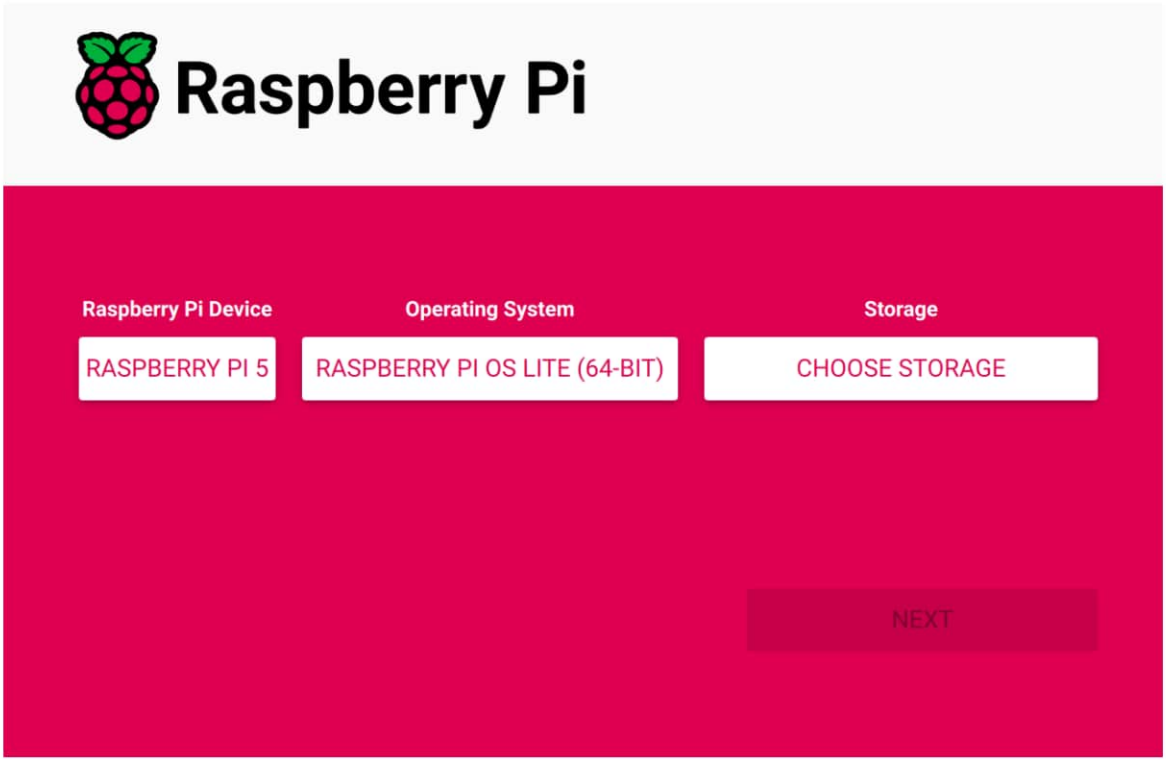

Рисунок 2.3 – Інтерфейс утиліти Raspberry Pi Imager

Для Raspberry Pi 5 та умов виконання завдання було визначено, що найбільш підходящою версією операційної системи буде Raspberry Pi OS Lite (64 bit), де Lite у назві означає що система поставляється без користувацького

інтерфейсу та виключно з командною строчкою. Також в якості додаткових конфігурацій було обрано:

- Ім’я хосту – agrocult.local
- Увімкнен SSH сервіс з автентифікацією за допомогою даних локального користувача
- Налаштовано користувача з ім’ям Volodymyr та паролем 126211
- Налаштовано приєднання до бездротової мережі
- В якості локальних параметрів встановлена часова зона Europe/Amsterdam та розкладка клавіатури us

2.4 Опис налаштувань та специфікації бази даних

Після безпосереднього встановлення PostgreSQL та допоміжних інструментів управління СУБД, було виконано налаштування файлу pg_hba.conf, який керує доступом до самої бази даних. За вимогами завдання визначено, що потрібно щоб усі вхідні підключення у локальній мережі та безпосередньо на хості були можливі. Це робиться зміненням значень полів “METHOD” на “trust” навпроти правил типу “local” та “host”. Лістинг файлу pg_hba.conf відредагованого до вимог завдання надано на рисунку 2.3 (сторінка. 31).

Коли з’єднання до бази даних було налаштовано, потрібно створити користувача який матиме доступ до бази даних, та саму базу даних де будуть зберігатися таблиці з даними. Це можна зробити за допомогою виконання SQL-скрипту від імені користувача який створюється за замовчуванням після встановлення PostgreSQL. Лістинг скрипту надано у рисунку 2.4 (сторінка. 31).

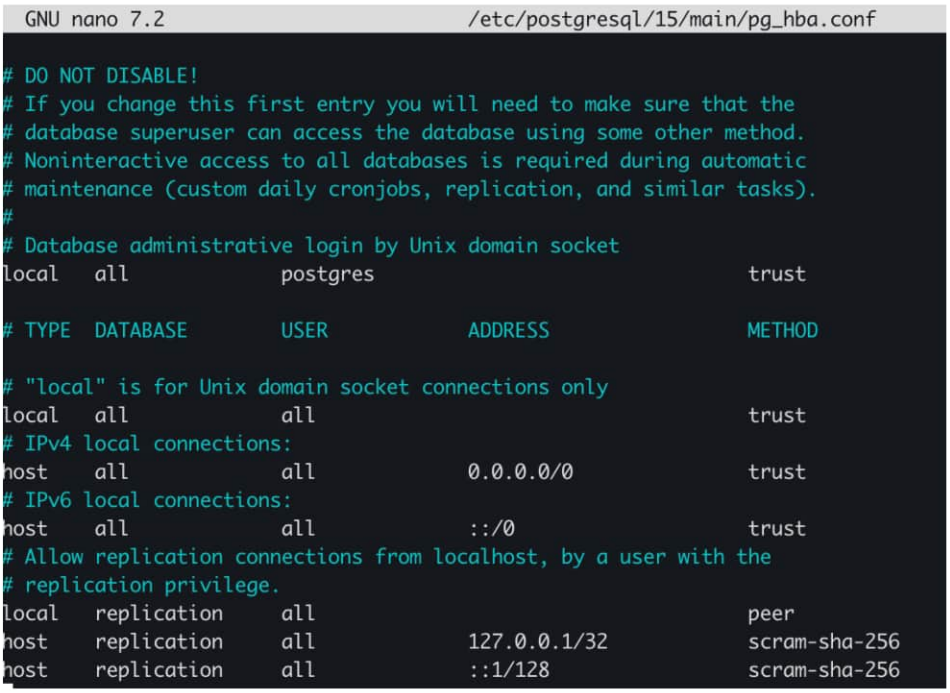

Рисунок 2.3 - Лістинг файлу pg_hba.conf

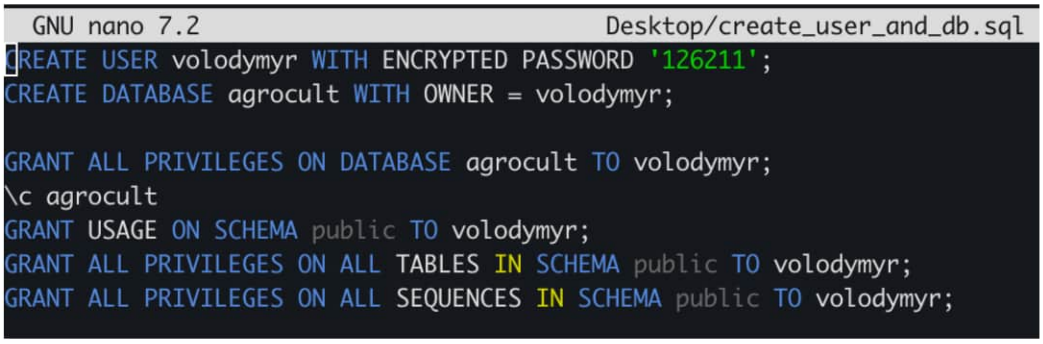

Рисунок 2.4 – Лістинг скрипту створення користувача та бази даних

Для кожного типу рослини повинна бути створена відповідна таблиця. Таким чином дані будуть більш структуровані та буде більш оптимізовано у швидкості отримання даних. Структура таблиці буде наступна:

- колонка id – первинний ключ, міститиме порядковий номер запису;
- колонка plant_id – міститиме ідентифікатор рослини за яким буде виконуватись пошук даних;
- колонка humidity – матиме дані про вологість ґрунту;
- колонка timestamp – міститиме час коли було створено вимірювання. Завдяки вбудованому функціоналу PostgreSQL поле налаштовано

таким чином, щоб час додавався автоматично коли запис буде вноситись до бази даних. Час береться безпосередньо із системи серверу;

- колонка location – вказуватиме на місцезнаходження рослини. В залежності від цього значення буде виконуватись пошук інформації про навколишнє середовище в якому росте рослина. Це може бути окрема кімната, теплиця, інше місце в якому дані навколишнього середовища можуть відрізнятися.

Приклад SQL-запиту для створення таблиці, в якій будуть зберігатися дані про вологість ґрунту різних квітів:

```
CREATE TABLE public."flowers" (
id SERIAL PRIMARY KEY,
plant_id TEXT NOT NULL,
humidity NUMERIC NOT NULL,
timestamp TIMESTAMP NOT NULL DEFAULT NOW(),
location TEXT NOT NULL
) ;
```

Також оскільки окрім фізичних властивостей ґрунту (у цьому випадку вологості ґрунту) будуть також зчитуватися дані про навколишнє середовище в якому ростуть рослини (освітленість, температура та вологість повітря та атмосферний тиск), для цих даних повинна бути створена окрема таблиця. Структура таблиці для збереження властивостей навколишнього середовища виглядатиме наступним чином:

- колонка id – первинний ключ, міститиме порядковий номер запису;
- колонка location – ідентифікатор місця де було зчитано стан середовища;
- колонка air_humidity – матиме дані про вологість повітря;
- колонка air_temperature – матиме дані про температуру повітря;
- колонка lighting – матиме дані про освітленість;

- колонка air_pressure – міститиме дані про атмосферний тиск;
- колонка timestamp – міститиме час коли було створено запис (тобто коли було виконано виміри).

Приклад SQL-запиту для створення таблиці, в якій будуть зберігатися дані про різні середовища в яких ростуть різні агрокультури:

```
CREATE TABLE public."surroundings" (
    id SERIAL PRIMARY KEY,
    location TEXT NOT NULL,
    air_humidity NUMERIC NOT NULL,
    air_temperature NUMERIC NOT NULL,
    lighting NUMERIC NOT NULL,
    air_pressure NUMERIC NOT NULL,
    timestamp TIMESTAMP NOT NULL DEFAULT NOW()
);
```

2.5 Файлова структура проєкту

Як було зазначено раніше, в якості платформи контролю файлів та версій цих файлів буде використовуватись GitHub. На рисунку 2.5 відображено структуру репозиторію agrocult користувача vshechenkov.

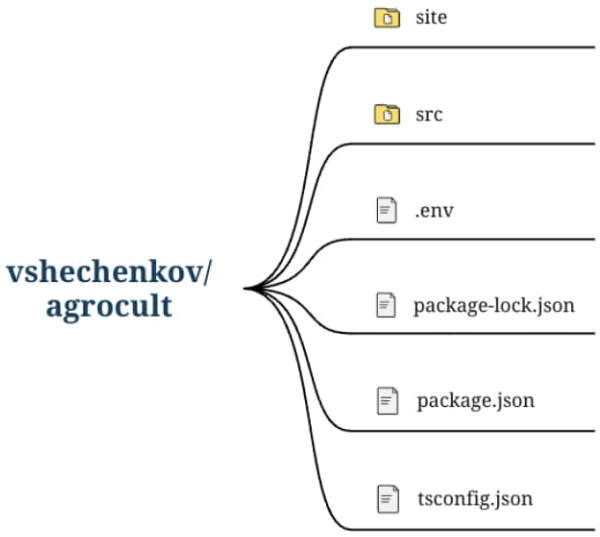

Рисунок 2.5 – Структура репозиторію agrocult

Відповідно до рисунку 2.5 можна виокремити два каталоги – site та src. Каталог site міститиме файли, які будуть використовуватись Frontend частиною інформаційної системи. Каталог src міститиме файли коду API серверу. Також окрім двох каталогів репозиторій містить файли які мають пряме відношення до API сервісу:

- .env – містить змінні середовища які містять дані для приєднання до бази даних;
- package-lock.json – автоматично згенерований файл який містить залежності використані у проєкті;
- package.json - файл маніфесту, який містить метадані про проєкт і визначає його залежності, скрипти та інші конфігурації;
- tsconfig.json - цей файл є конфігураційним файлом для компілятора TypeScript (tsc). Він визначає, як код TypeScript має бути скомпільований у JavaScript.

На рисунку 2.6 зображен вміст каталогу site де знаходяться усі файли та компоненти які будуть надаватися клієнту вебсервером у якості вебсайту. Цей каталог містить файл конфігурації вебсерверу .htaccess та підкаталоги відповідно з іменем відповідно до вмісту цих підкаталогів, який представляє з себе елементи з яких буде складатися вебсайт:

- каталог fonts – містить шрифти які будуть використані у файлах стилів для покращення зовнішнього вигляду вебсайту;
- каталог images – містить плейсхолдер (заглушку) зображення яка буде відображатися поки сервер отримує зображення рослини з API-серверу;
- каталог scripts – містить динамічну частину сайту яка зв’язується з двома API серверами та розміщує отримані дані на сторінці;
- каталог styles – містить налаштування та класи елементів сторінки, дає більш гнучко налаштувати дизнайн вебсторінок;

- каталог webpages – містить HTML сторінки які будуть задіювати вище перелічені елементи та які будуть надсилатися клієнту вебсервером.

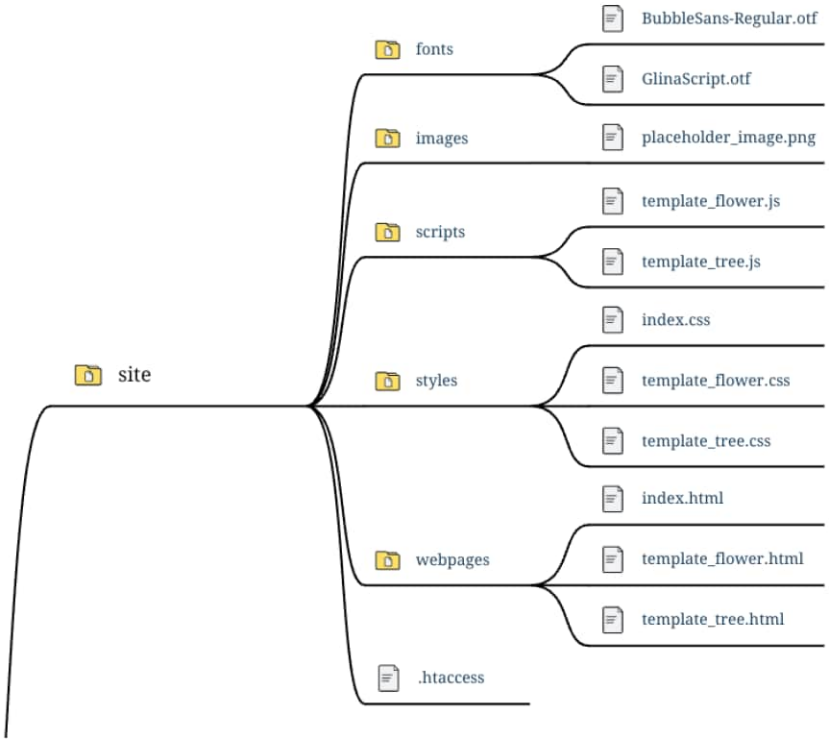

Рисунок 2.6 – Структура каталогу site

Рисунок 2.7 демонструє структуру папки src яка містить файли коду локального API серверу. Цей каталог містить 7 папок та 3 файли, більшість папок містять лише один файл призначення відповідно до назви самої папки, лише папка DAO має два файли: SoilDAO.ts який містить CRUD-функції які будуть виконуватись базою даних та SoilDAOinterface.ts який експортує функції SoilDAO.ts в якості інтерфейсу, що є фундаментальним концептом об’єктно-орієнтованого програмування, який забезпечує гарантію типів, зрозумілість та поліморфізм. Файли які представлені в каталозі мають наступне призначення:

- data-source.ts – модуль підключення до бази даних;
- index.ts – файл запуску API серверу;

- routes.ts – файл який зв’язує URL шляхи API з відповідними функціями коду.

Рисунок 2.7 – Структура каталогу src

2.6 Діаграма послідовності отримання даних від API

Діаграма послідовності на рисунку 2.7 демонструє покроковий процес взаємодії внутрішніх складових API для отримання інформації, починаючи із запиту даних з API Frontend частиною і завершуючи отриманням структурованої інформації у вигляді JSON відповіді відповідно до шляху та параметрів за якими було зроблено запит.

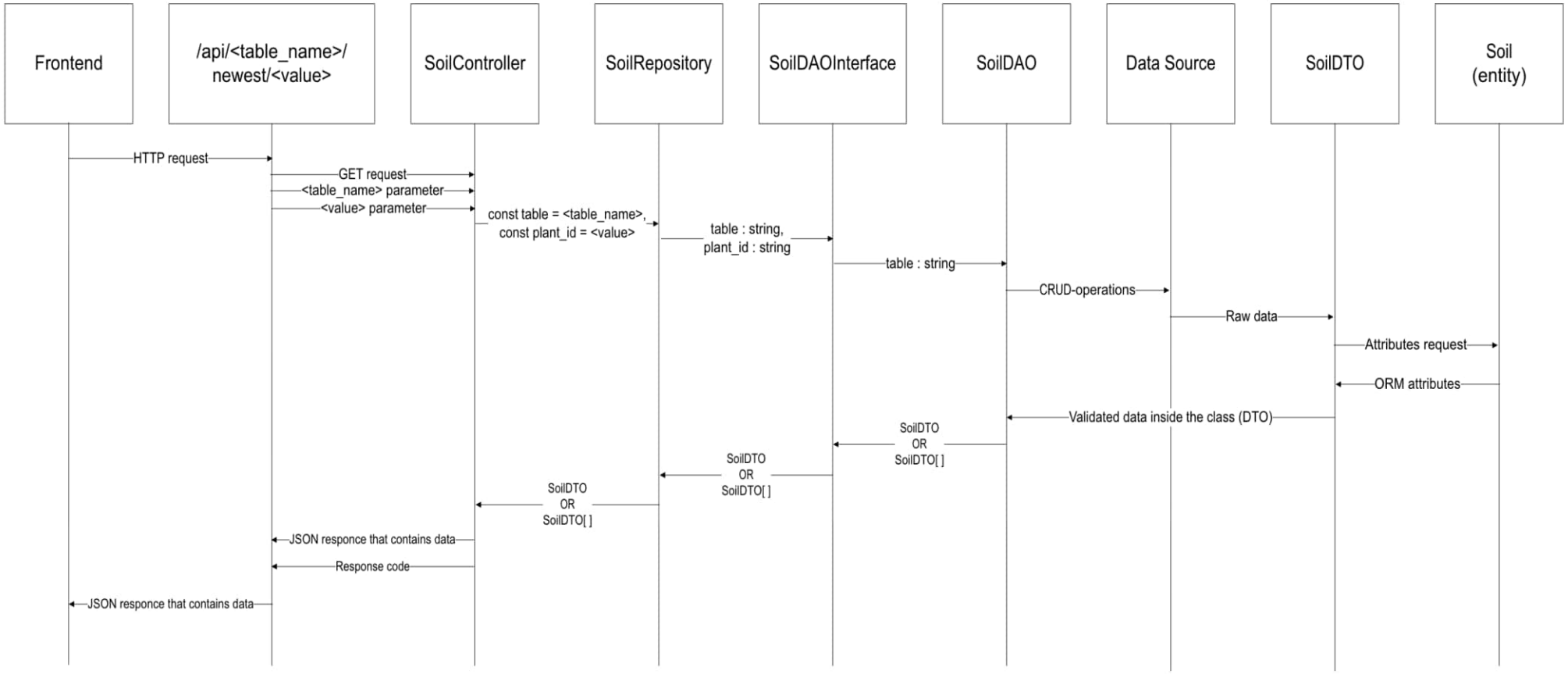

Рисунок 2.7. – Діаграма послідовності GET запиту

2.7 Принцип отримання даних з API

Як видно з рисунку 2.7 запит до API починається з Frontend частини серверу. Запит робиться безпосередньо за URL адресою за якою знаходиться API сервер. Окрім цього, Frontend також передає деякі параметри за якими будуть відбуватись сортування та пошук потрібних даних саме для запитуваної рослини. Серед параметрів можна визначити:

- ім’я таблиці – визначає тип рослини, наприклад квітка або дерево;
- формат запитуваних даних – визначає які самі дані буде повернуто; є вибірковим параметром; наприклад, параметр «newest» буде повертати найновіші дані про запитувану рослину, а відсутність параметру буде повертати усі наявні дані;
- ідентифікатор – параметр який визначає дані якої саме рослини потрібно обробити / повернути.

Відповідно до параметрів які було отримано API сервером буде визначено які подальші дії будуть виконані з даними. Ідентифікацію параметрів, витяг даних з цих параметрів та перенаправлення їх до потрібних функцій виконує контролер.

Коли дані параметрів було видобуто, їх буде перевизначено у змінні та передано до репозиторію. Репозиторій приймає на себе необроблені дані та виконує фільтрацію або інші перетворення задля повернення цих даних у вигляді такого типу, який було запитано від Frontend частини.

Перед безпосереднім перетворенням даних ці дані потрібно отримати. Операція отримання даних виконується через DAO інтерфейс який експортує сам клас DAO. Абревіатура DAO розшифровується як Data Access Object та представляє з себе шаблон який забезпечує абстрактний інтерфейс для певного типу бази даних. Простіше кажучи це точка, з якої API отримує або до якої відправляє інформацію до бази даних.

Щоб DAO мав можливість передавати запитувані дані, а також гарантував структурованість та відповідність даних які було запитано, буде використовуватись DTO що означає перші три букви слів Data Transfer Object. DTO представляє з себе шаблон який окрім того що спрощує передачу даних від бази даних до інших компонентів API, також містить функціонал валідації отриманих даних, що гарантує відповідність типів даних які будуть передані в подальше опрацювання.

В свою чергу запитувані атрибути отримуються від класу сутності. Цей клас є визначенням які саме дані будуть отримані в якості відповіді на вхідний запит. Сутність містить в собі найменування полів які повинні бути отримані та типи даних які очікуються від цих полів. Також сутності використовуються в якості моделей у Data Source (точка доступу до бази даних) для отримання даних напряму з бази даних.

Коли дані було отримано з бази даних, структуровано, відфільтровано та інкапсульовано у клас Data Transfer Object, ці дані рухаються в зворотному порядку до контролера. Коли дані доходять до контролера, відбувається форматування цих даних до вигляду JSON payload (дані які складаються з пар ключ-значення і можуть представляти різні типи даних [12]) та якщо отримання даних було виконано успішно, разом з JSON відповіддю також відправляється HTTP відповідь 200, яка свідчить про успішне виконання операції.

2.8 Діаграма послідовності надсилання даних до API

Окрім отримання даних для виводу їх для клієнта, API також передбачено запис даних до бази даних. Тобто, API яке створено для цього проєкту має подвійне призначення: зчитування та надання даних а також запис даних. В цьому випадку відправною точкою є сам мікроконтролер, який формує та відправляє дані, а кінцевою операцією є отримання коду який

підтверджує успішний запис даних. Діаграма послідовності запису виміряних значень наведена на рисунку 2.8 (сторінка 41).

2.9 Принцип відправлення даних до API та їх збереження

Для відправлення та запису даних використовується інший тип запиту – POST. Цей запит містить в собі попередньо структуровані дані у вигляді JSON payload та відправляються до API в якості тіла HTTP запиту.

Загалом, послідовність операцій для запису інформації дуже схожа на процедуру отримання інформації. Коли дані форматуються з JSON запису до змінної типу SoilDTO, вони проходять такий самий шлях від репозиторію до Data Source. Єдина відмінність це те, що дані одразу подаються інкапсульованими у клас SoilDTO, що не потребує додаткових запитів до сутностей та перевірки типів даних.

Ще одною відмінністю є зворотна відповідь від Data Access Object: коли відбувалось зчитування даних, у якості відповіді надсилався клас який містив запитувані дані, а у випадку з записом даних в якості відповіді повертається логічне значення true або false, яке інформує про результат спроби запису даних до бази даних.

Окрім цього також і відрізняється HTTP код відповіді: під час запиту на отримання даних якщо операція пройшла успішно, то клієнт (Frontend) отримував HTTP код відповіді 200 OK, у випадку запиту на запис даних при успішному виконанні запиту HTTP кодом буде 201 Created, що свідчитиме про створення запису в базі даних з інформацією яка була направлена до API.

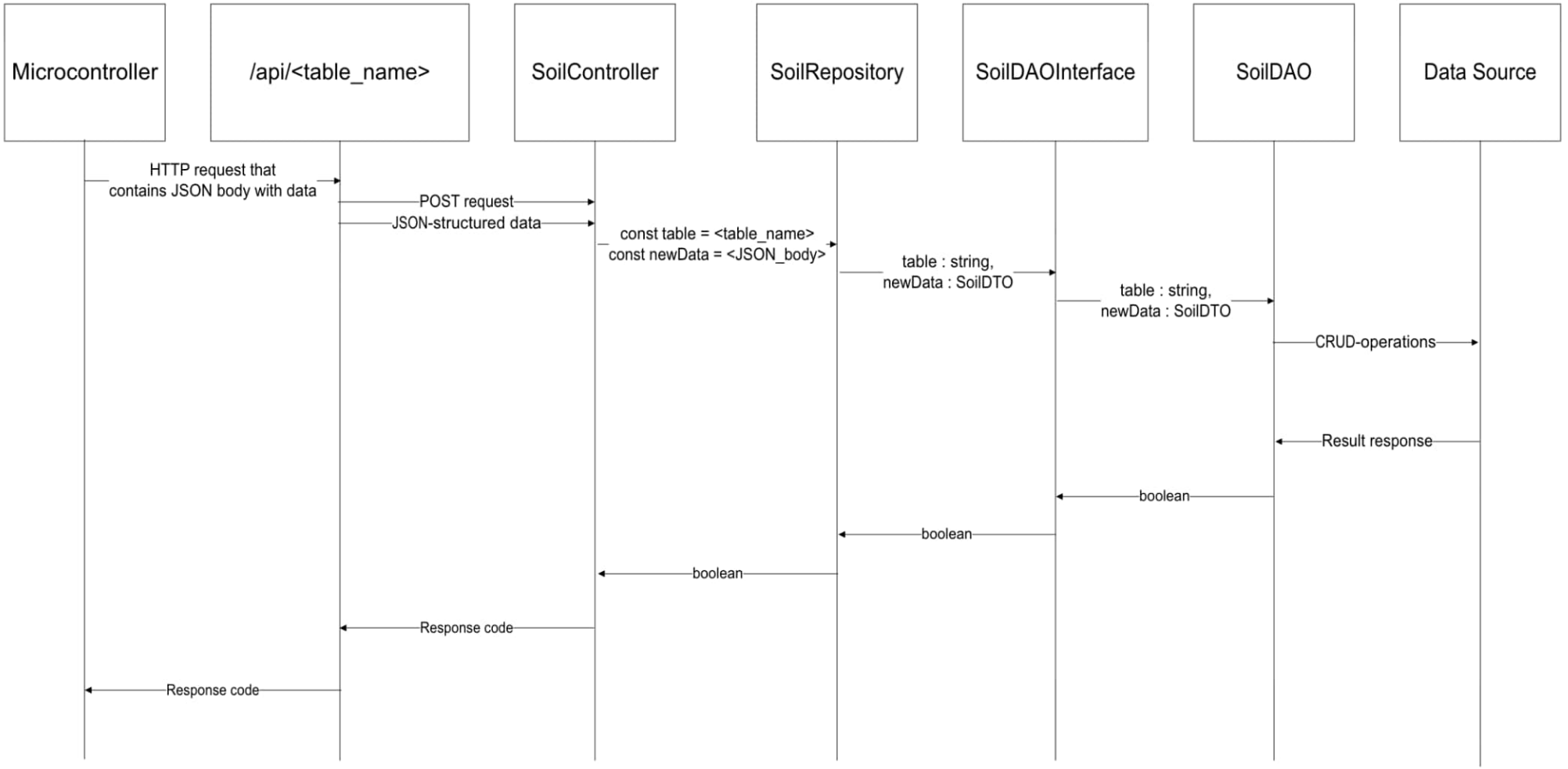

Рисунок 2.8 – Діаграма послідовності POST запиту

2.10 Додаткові компоненти API

Протягом розробки API також виконуватиметься тестування та діагностика помилок які можуть виникнути. Задля швидшого знаходження та виправлення цих помилок існують різні middleware (проміжне програмне забезпечення) компоненти.

Middleware – функція, що може знаходитися між запитом, який зробив клієнт, та відповіддю сервера для нього. Декілька middleware функцій можна об'єднати для виконання конкретних дій в залежності від запиту клієнта.

У випадку API яке було розроблено в цій роботі, використовується поєднання двох функцій об’єднаних в middleware – логер запитів реального часу та функція запису даних у локальний файл. Поєднання цих функцій дозволить визначити запити за якими шляхами було створено, в який момент часу, який був результат та характер запиту який було зроблено.

Окремо логер запитів також враховується як middleware-сервіс, оскільки він перехоплює вище згадану інформацію про запити та виводить її у консоль API серверу. Приклад log-записів записаних до локального файлу API серверу:

```
[19.05.2025 17:29:26]  Server running on port http://localhost:3000
[19.05.2025 17:29:31]  GET / 404 1ms
[19.05.2025 17:30:11]  Server shutting down...
```

2.11 Діаграма послідовності роботи мікроконтролера

Діаграма послідовності з рисунку 2.9 (сторінка 38) показує покрокову роботу мікроконтролера від приєднання до бездротової мережі WiFi до відправлення HTTP запиту на API сервер. Сама діаграма розділена на дві області a та b, які відповідають за попереднє налаштування мікроконтролера

перед виконанням основного блоку коду та основний код який виконуватиме безпосередньо затребувані дії відповідно.

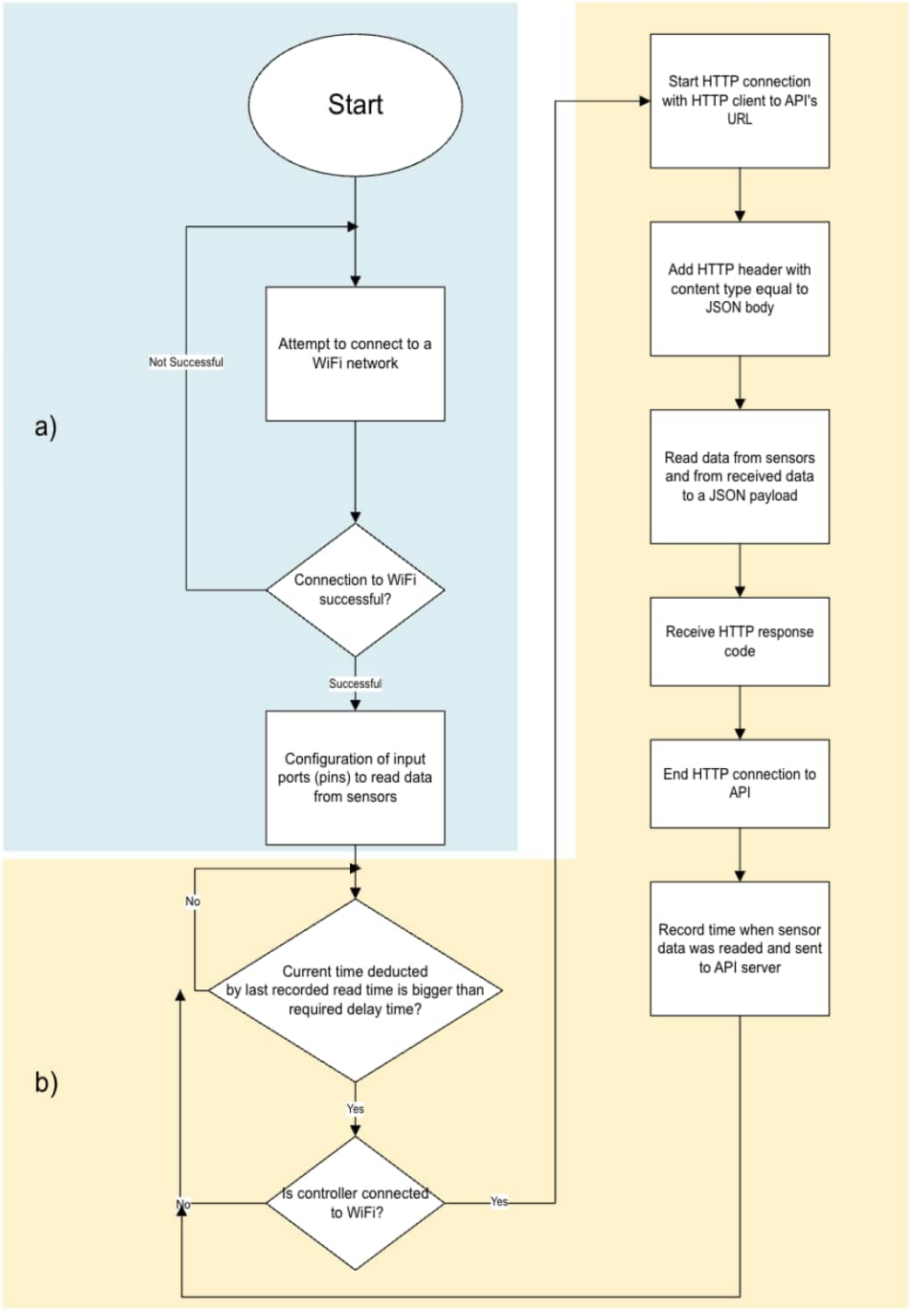

Рисунок 2.9 – Діаграма послідовності роботи мікроконтролера: a) блок підготовки; b) основний блок коду

2.12 Принцип роботи моніторингової інформаційної системи

Код який виконуватиме мікроконтролер починається з оголошення бібліотек які будуть використані для полегшення роботи з кодом та констант які міститимуть усі дані потрібні для коректного налаштування та подальшого зчитування і передачі даних. Серед констант можна визначити: порти (піни) до яких приєднані сенсори, найменування та пароль бездротової мережі до якої також під'єднаний сервер (оскільки зв'язок між системами відбуватиметься в межах локальної мережі), частота зчитування даних у мілісекундах та URL адреса за якою знаходиться точка входу API сервера для надсилання даних до API.

Після оголошення усіх констант починається виконання блоку підготовки та налаштування мікроконтролера до виконання основного блоку коду. Блок підготовки включає в себе налаштування серійного порту для наладки коду та перевірки правильності його виконання. Коли серійний порт було налаштовано починається процес приєднання до WiFi мережі. Це виконується за допомогою бібліотеки WiFi.h яка містить методи приєднання WiFi модуля мікроконтролера ESP32-C3 до бездротових мереж. Після виклику методу приєднання йде блок перевірки з'єднання який буде гарантувати що доки мікроконтролер не буде приєднано до мережі не почнеться виконання основного блоку коду. Окрім цього, останньою операцією є переведення портів до яких приєднані сенсори у режим зчитування даних.

За успішним виконанням блоку налаштування йде виконання основного блоку програми. Починається цей блок з порівняння чи є різниця значення вбудованої функції millis() та змінної яка відповідає за збереження часу коли було виконано останнє зчитування даних з сенсорів більша за значення константи яка була задекларована на початку та містить інтревал через який будуть виконані вимірювання.

Функція millis() – одна з фундаментальних функцій в IoT. Ця функція повертає кількість мілісекунд, що пройшли з моменту коли плата мікроконтролеру почала виконувати код поточної програми. [13] Під час програмування планованих IoT систем або IoT систем реального часу, реалізація коду для цих типів інформаційних систем без використання функції millis() є неможливою.

Якщо умови першої перевірки було виконано код переходить до другої умови – успішного приєднання до WiFi мережі. Якщо обидві умови задоволено програма переходить до виконання основної частини коду.

За використання бібліотеки HTTPClient.h встановлюється з'єднання з API сервером URL адреса якого була оголошена в якості константи на початку програми. Після успішного з'єднання з API починається формування самого запиту на запис інформації.

Формування починається зі створення HTTP заголовку який дозволяє мікроконтролеру передати додаткову інформацію разом з повідомленням у запиті, в цьому випадку заголовок міститиме інформацію що разом з запитом також повинна надійти додаткова інформація у вигляді JSON payload.

Сам JSON payload формується за допомогою бібліотеки ArduinoJson.h, яка полегшує структуризацію та формування даних для надсилання у вигляді JSON. Під час формування цього payload визначається ключ та значення цього ключа. Ключами в цьому випадку будуть ідентифікатор колонки бази даних до якої будуть записані дані, а значенням буде інформація яку буде записано до вище згаданої колонки.

Під інформацією яка буде передана в якості значення відповідно до ключа у JSON payload мається на увазі значення які будуть зчитані напряму з сенсорів. В залежності від типу сенсору, існує два режими зчитування: цифрове це звичайний булевий вираз true або false та аналогове зчитування – коли значення це цифровий або символьний вираз, простіше кажучи будь-яке не булеве значення.

Після того як payload сформовано, інформація поміщається у тіло запиту - частину запиту, яка передає інформацію на сервер. [14]

У результаті ми отримуємо повністю сформований HTTP POST запит з заголовком що вказує на наявність інформації та тілом яке містить цю інформацію. Цей запит відправляється із оголошенням змінної яка міститиме функцію надсилання цього запиту, і яка повертатиме код відповіді до цієї змінної. Після цього для наладки коду, код відповіді буде виведено в консоль через серійний порт (якщо мікроконтролер під'єднано до комп'ютера або іншого девайсу який має встановлене середовище розробки ArduinoIDE).

В якості процедури завершення виконання основної частини коду виконується запис поточного значення функції millis() задля подальшого порівняння та відтермінування наступного зчитування та передачі значень сенсорів відповідно до встановленої константи.

Таким чином відбувається повний цикл роботи моніторингової інформаційної системи, яка складається з мікроконтролера та сенсорів. Повний лістинг коду який виконує мікроконтролер можна знайти в Додатку А.

2.13 Опис налаштувань вебсерверу

Як було зазначено у першому розділі, в якості вебсерверу для цього проєкту виступатиме Apache 2. Цей сервіс має дуже велике різноманіття налаштувань які допоможуть гнучко налаштувати безпеку та доступність Frontend частини проєкту.

Перш за все, оскільки джерелом файлів та версій коду загалом є репозиторій на сервісі GitHub, файли для Frontend частини які будуть запитані та завантажені з цього джерела будуть скоріш за все розташовані за шляхом відмінним від шляху за замовчуванням після встановлення вебсерверу Apache. Задля змінення шляху розташування файлів вебсайту, потрібно виконати наступну команду:

```
sudo nano /etc/apache2/sites-available/000-default.conf
```

Ця команда відкриє файл конфігурації в якому визначені налаштування віртуальних хостів за замовчуванням. Окрім цього, там також визначається шлях до кореня каталогу де знаходяться файли для вебсерверу, який можна вказати після значення DocumentRoot. Строка зі зміненим шляхом до файлів Frontend частини буде виглядати наступним чином:

```
DocumentRoot /home/volodymyr/Desktop/agrocult/site/
```

Після встановлення шляху за замовчуванням, потрібно виконати налаштування самого вебсерверу. Щоб відкрити файл де знаходяться конфігурації Apache 2 треба виконати команду:

```
sudo nano /etc/apache2/apache2.conf
```

Далі, у розділі де описані налаштування каталогів (строки які починаються з <Directory>), в кінець конфігурації було додано наступний запис:

```
<Directory /home/volodymyr/Desktop/agrocult/site/>
    Options -Indexes
    AllowOverride All
    Require all granted
</Directory>
```

Відповідно до цього запису для каталогу де знаходяться файли Frontend частини було встановлено наступні налаштування:

- Options -Indexes – запобігає відображенню списку файлів (функція вебсерверу за замовчуванням) якщо у директорії не міститься файлу з назвою index.html;
- AllowOverride All – визначає пріоритет налаштувань які знаходяться у файлі .htaccess вищими за кофігурації з поточного файлу. Це

дозволяє більш гнучко налаштувати доступ до файлів Frontend частини;

- Require all granted – дозволяє усім хостам локальної мережі мати доступ до цього вебсерверу.

Оскільки планується створення файлу .htaccess для додаткових конфігурацій доступу, можна заздалегідь зазначити ім’я файлу з додатковими налаштуваннями та створити правило яке заборонить перегляд фалйу .htaccess напряму з клієнтського веббраузера. Ці налаштування декларуються наступними строками коду в тому ж самому файлі apache2.conf:

```
AccessFileName .htaccess
<FilesMatch "^\.ht">
        Require all denied
</FilesMatch>
```

Файл .htaccess буде розміщено у корені каталогу де знаходитимуться файли Frontend, та міститиме наступне налаштування:

```
RedirectMatch ^/$ /webpages/
```

Це налаштування за спроби отримання доступу до вебсервера за URL адресою без шляху (наприклад: agrocult.local) буде автоматично перенаправляти до каталогу де знаходитимуться HTML файли (наприклад: agrocult.local/webpages).

2.14 Діаграма послідовності роботи Frontend частини

Рисунок 2.10 (сторінка 51) представляє діаграму послідовності роботи Frontend частини інформаційної IoT системи що розробляється в цьому завданні. Початок взаємодії з Frontend означається коли клієнт (веббраузер) робить вихідний запит до вебсерверу на отримання вебсторінки із запитуваними даними, а фіналом роботи Frontend частини є повернення

вебсторінки вебсервером, вміст якої залежить від того чи була операція отримання даних успішною.

2.15 Принципи роботи компонентів Frontend

Коли клієнт робить запит на отримання вебсторінки, спочатку цей запит оброблюється вебсервером. На цій стадії вебсервер перевіряє чи є в клієнта дозвіл на отримання запитуваного шляху або сторінки.

Якщо клієнт має доступ до сторінки, вебсервер починає пошук запитуваної сторінки за шляхом за яким було виконано запит. В умовах цього проєкту буде існувати стільки вебсторінок скільки буде існувати типів рослин.

Коли потрібну вебсторінку було знайдено, завантажується файл стилів відповідно до кожної сторінки. Ці файли міститимуть властивості елементів вебсторінки та окремі класи-набори властивостей, які можуть бути застосовані до окремих елементів. Також стилі містять декларації інших елементів дизайну, як наприклад шрифтів.

Варто зазначити, що сторінки передаються з плейсхолдерами (заглушками) замість самих даних які було запитано користувачем. Все тому що коли сторінка передається до клієнта, разом з цим виконується JavaScript скрипт, який приєднується до API серверів та за успішного отримання даних заміщає плейсхолдери запитуваними даними. Приклад програмного коду виконуваного для отримання даних з API та розміщення їх на сторінці відображено у Додатку Б. Плейсхолдери розміщуються в усіх полях для яких дані повинні бути отримані з API серверів.

Скрипти виконують запити послідовно на два API сервери: на локальний API сервер для отримання даних про властивості середовища в яких ростуть агрокультури та на зовнішній API сервер який містить базу даних з рекомендованими показниками в яких повинні рости агрокультури та

навіть зображення як повинні виглядати рослини в залежності від запитуваної агрокультури.

В деяких випадках щоб отримати дані з API серверу потрібно виконати попередню авторизацію. У випадку з локальним сервером жодного методу авторизації не було додано, оскільки він не відкритий до загальної мережі. На відміну від локального, перед надсиланням відповіді на вхідний запит зовнішній API сервер повинен отримати API ключ, який ідентифікував би того хто виконує запит даних. Цей ключ передається в якості HTTP заголовка. Приклад запиту який виконується скриптом для отримання даних із зовнішнього API серверу та який містить заголовок з ключем авторизації:

```
fetch(`https://open.plantbook.io/api/v1/plant/detail/${plantId}/?format=json`, { headers: { 'Authorization': 'Token <auth_token>' }})
```

Після отримання відповіді від обох API серверів, якщо ця відповідь містить відрізняється від успішної відповіді, замість заміщення елементів сторінки пустими місцями буде показано повідомлення про помилку зі значенням помилки для більш легкої діагностики. Код умови перевірки статусу запиту виглядає так:

```
if (!response.ok) {
            throw new Error(`Помилка сервера: ${response.status}`);
        }
        return response.json();
```

Також на елементі коду вище можна побачити, що якщо запит було виконано успішно, отриманий JSON елемент буде передано у подальшу обробку, а саме:

- Дані будуть зібрані та розміщені у попередньо визначені поля;
- До даних також будуть додані додаткові символи (умовні позначки, нотатки і т.п.) для зручності відображення.

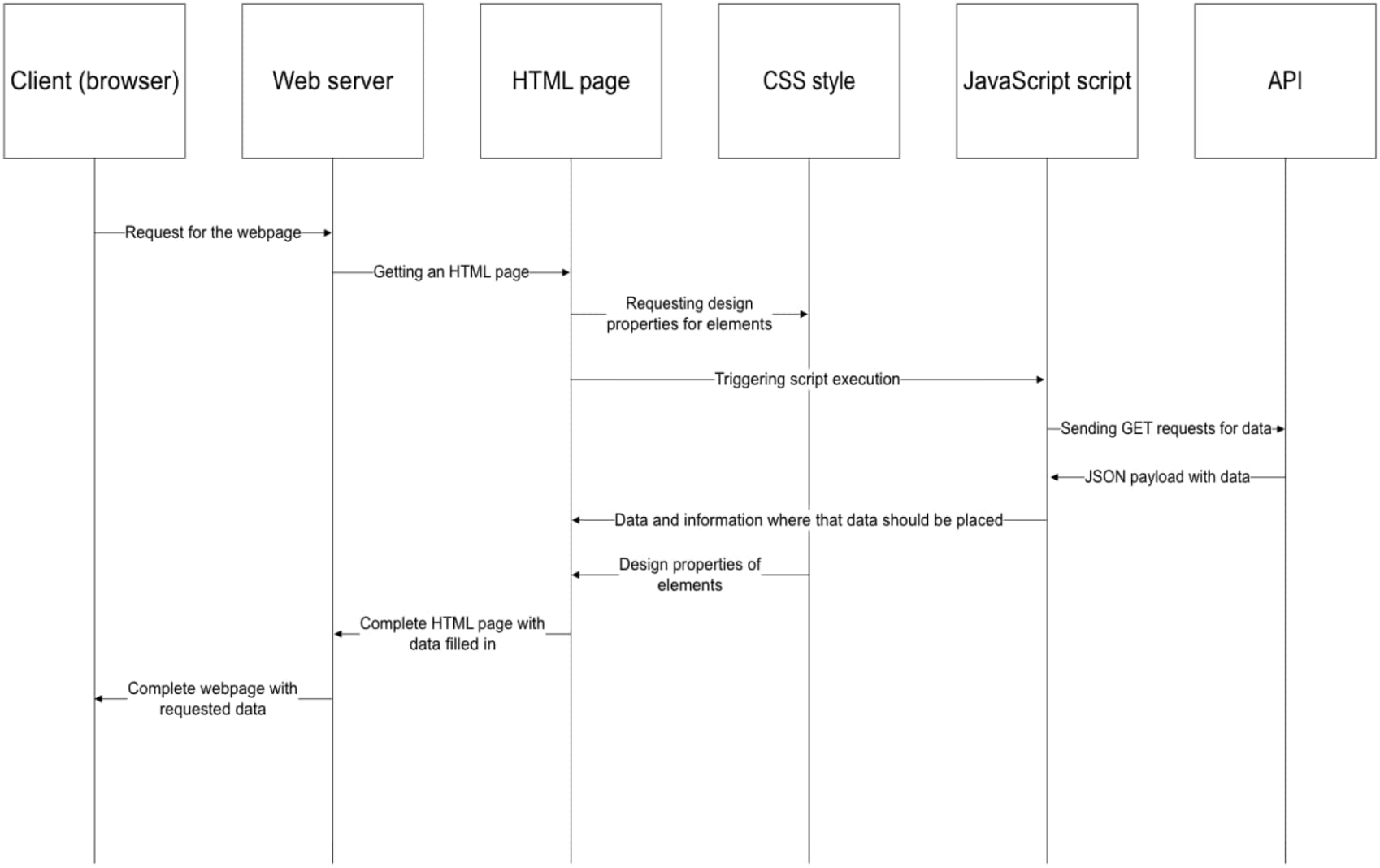

Рисунок 2.10 – Діаграма послідовності роботи Frontend

2.16 Специфіка налаштування RFID міток

Щоб зробити отримання даних більш зручним та швидким, в якості носія точки доступу до вебсторінки рослини було обрано RFID мітки. Це буде найзручнішим варіантом збереження та отримання посилань на вебсторінки рослин, особливо якщо кількість рослин та сторінок для них перевищує 20 найменувань.

Початок запису посилання на сторінку рослини в RFID мітку відбувається з формування самого посилання. Посилання формуватиметься за наступним чином:

agrocult.local/webpages/template_<plant_type>.html?plant_id=<id>_<number>, де:

- plant_type – тип рослини (дерево, квітка, т.п.);
- id – найменування рослини (наприклад, anthurium%2520andraeanum);
- number – порядковий номер рослини, якщо рослин однакового найменування декілька.

Коли посилання було сформовано, воно копіюється до мобільного додатку NFC Tools встановленого на телефоні з підтримкою технології NFC у розділ Write. Там посилання додається в якості URL-запису який потім буде записано до самої мітки. Приклад сформованого та готового до запису URL-елементу в додатку NFC Tools зображено на рисунку 2.11.

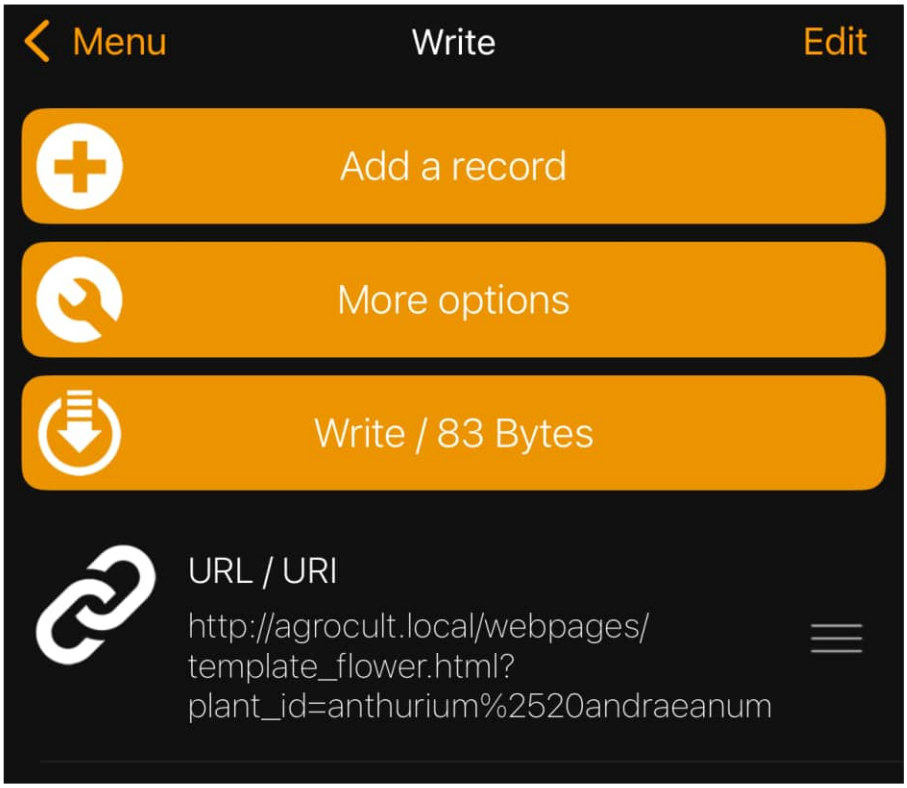

Рисунок 2.11 – Готовий до запису елемент

Коли елемент було додано, натискається кнопка Write та телефон прикладається верхньою частиною до пустої RFID мітки. Якщо запис було виконано успішно, на телефоні з’явиться повідомлення про успішний запис інформації. Приклад повідомлення про успішний запис наведено на рисунку 2.12.

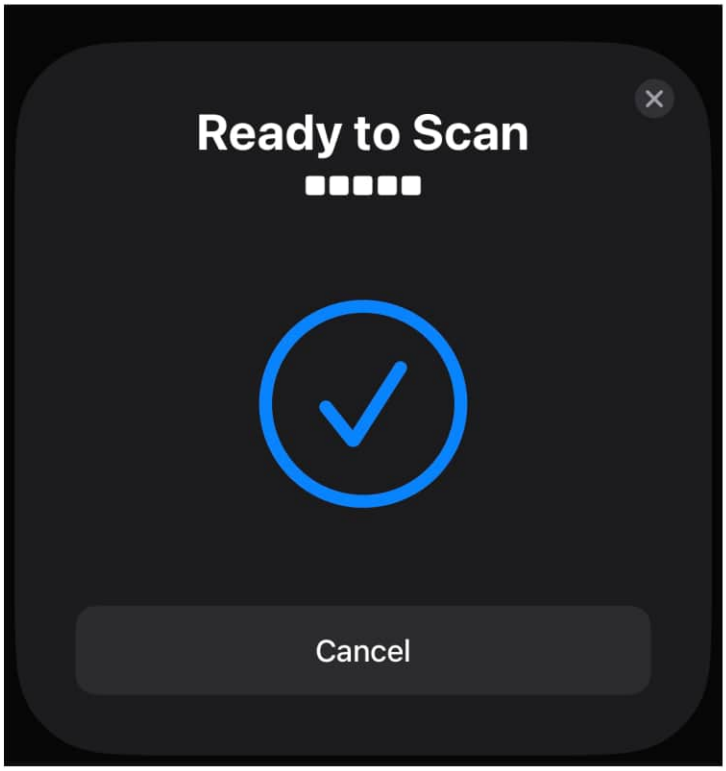

Рисунок 2.12 – Підтвердження запису інформації

Задля контролю коректності запису URL-посилання до RFID мітки додаток NFC Tools також має функціонал зчитування специфікацій та інформації вже записаної на мітку. Це виконується за допомогою активації функції Read головного меню додатку та піднесення верхньої частини телефону до RFID мітки. Якщо зчитування інформації було успішним, висвітиться подібне повідомлення до зображеного на рисунку 2.12.

На рисунку 2.13 зображено результат зчитування RFID мітки, де зображено як специфікації сканованої мітки, так і попередньо записане посилання на вебсторінку відповідної рослини.

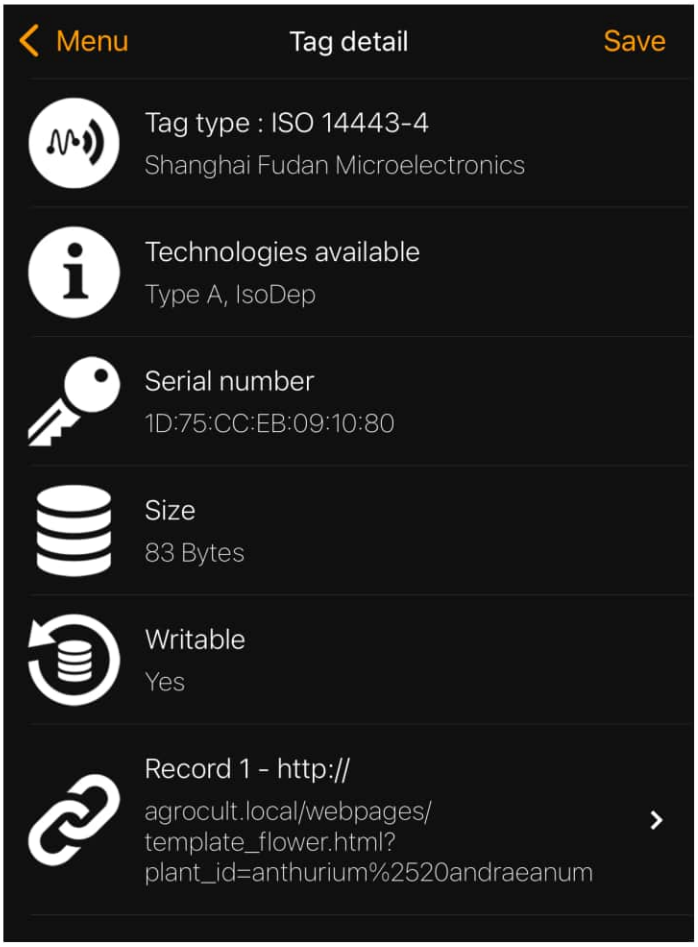

Рисунок 2.13 – Результат зчитування RFID мітки

2.17 Висновки до другого розділу

В цьому розділі було описано взаємодію між компонентами створюваної веборієнтованої інформаційної системи. Окрім окремих компонентів ця інформаційна система має у своєму складі окрему інформаційну систему у вигляді комбінації мікроконтролерів та сенсорів.

Було розглянуто послідовність отримання інформації з API серверу та передачі цієї інформації до Frontend частини для відображення. Було покроково описано рух запиту через складові API для отримання даних з бази даних та повернення відформатованих та перевірених даних у зворотному напряму до клієнта.

Окрім отримання даних так само було розглянуто відправлення даних від моніторингової системи до API серверу та рух цих даних до збереження у базі даних та надсилання відповіді про успішний запис цих даних.

Також цей розділ містить інформацію про специфіку налаштування окремих компонентів для коректної взаємодії з іншими компонентами цієї системи.

Було описано процес встановлення та налаштування бази даних PostgreSQL та структуру таблиць які знаходяться в цій базі даних та зберігатимуть дані про навколишнє середовище зібрані з сенсорів. Налаштування доступу до бази даних, найменування та кількість стовбців таблиць – це все було розглянуто в другому розділі.

Окремо було описано створення образу операційної системи для одноплатного комп’ютера Raspberry Pi який виступає в якості серверу в цій інформаційній системі. Створення та налаштування файлу образу виконувалось за допомогою програмного забезпечення під назвою Raspberry Pi Imager. Під час створення образу було внесено зміни до стандартних налаштувань; зміни включали в себе: створення локального користувача, попереднє встановлення сервісів віддаленого доступу, налаштування приєднання до бездротової мережі та інше.

РОЗДІЛ 3
ДЕМОНСТРАЦІЯ РОБОТИ РОЗРОБЛЕНОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Сценарії до демонстрації

Після закінчення етапу базової розробки веборієнтованої інформаційної системи, в якості готового продукту буде система моніторингу стану навколишнього середовища з виведенням даних про статус складових середовища у вигляді вебсторінок та інформації про рекомендовані показники цих елементів середовища. Таким чином в залежності від типу та найменування рослини клієнт зможе контролювати показники середовища в яких ростуть ці рослини відповідно до рекомендованих. Це вважатиметься першим та основним сценарієм роботи системи.

Під час розробки компонентів інформаційної системи було передбачено виникнення помилок, які повинні бути перехвачено, збережено та в подальшому виправлено розробником. Задля більш ефективного виправлення помилок, варіант виведення типів помилок та їх можливі причини буде гарною практикою. Це можливо реалізувати на двох етапах: спроба запиту вебсторінки та спроба запиту вебсторінкою даних з API серверів.

3.2 Демонстрація успішно виконаного запиту

Якщо було створено запит на отримання вебсторінки та цей запит було виконано успішно, в якості результату буде відображено вебсторінку на якій буде зображено ідентифікатор рослини, її зображення, інформація про вологість ґрунту в якому росте рослина (поточна, час вимірювання, рекомендована), а також окремим розділом інформація про стан

навколишнього середовища в якому росте рослина (температура повітря, вологість, освітленість, тиск), рекомендовані діапазони значень за яких повинна рости агрокультура та її фізичне місцезнаходження. Приклад вебсторінки за успішного виконання запиту наведено на рисунках 3.1 та 3.2.

Рисунок 3.1 – Зразок результату успішно виконаного запиту для рослини Tulpa ‘Groenland’

Рисунок 3.2 – Зразок результату успішно виконаного запиту для рослини Anthurium Andraeanum

3.3 Демонстрація запиту неіснуючої HTML сторінки

За спроби запиту неіснуючої вебсторінки, вебсервер було налаштовано на відображення користувацької сторінки про помилку з кодом 404 та причиною яка могла викликати появу цієї сторінки. Сторінка яка з’являється при спробі запиту неіснуючої вебсторінки продемонстрована на рисунку 3.3.

Рисунок 3.3 – Сторінка помилки 404

3.4 Демонстрація запиту без ідентифікатора рослини

Коли робиться запит існуючої сторінки, але обов’язковий параметр plant_id не вказано, скрипт передає елементи які будуть розміщені на сторінці та які сповіщатимуть про помилку та її можливу першопричину. Приклад сторінки яка сповіщатиме про відсутність параметру plant_id відображено на рисунку 3.4.

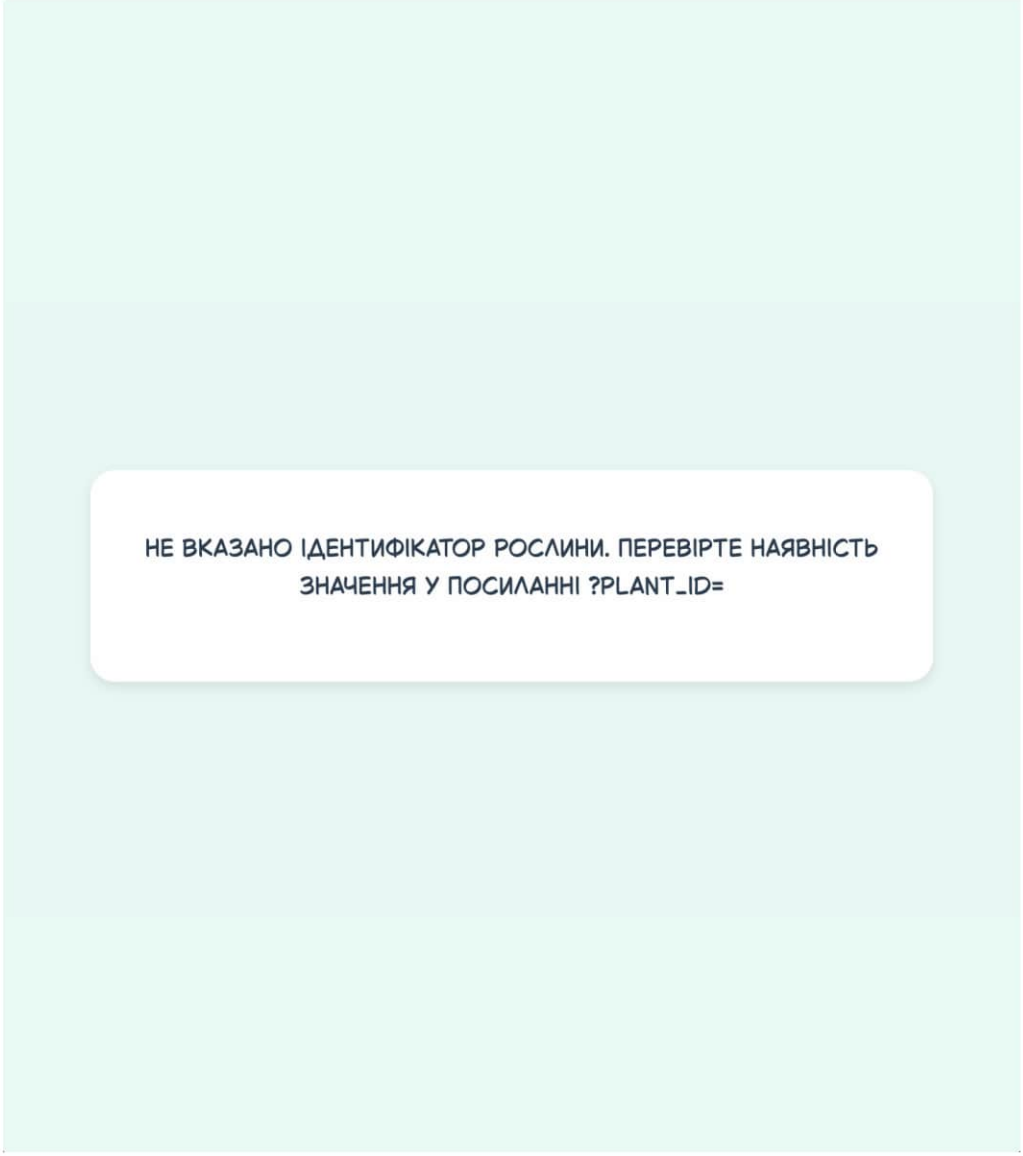

Рисунок 3.4 – Помилка про відсутність параметру plant_id

3.5 Демонстрація виникнення помилки серверної частини

Якщо умови для створення запиту виконані з боку клієнта, існує вірогідність виникнення помилок з боку сервера. Наприклад, якщо запит до API було не виконано через відсутність інтернету або якщо ідентифікатор рослини не було знайдено у базі даних. Виведення помилки відбувається за таким самим принципом як і для помилки запиту без ідентифікатора рослини, але щоб розрізнити що це серверна помилка, текст відображено червоним кольором. Рисунок 3.4 (сторінка 62) демонструє спробу передачі

API серверам неіснуючий ідентифікатор рослини, а рисунок 3.5 (сторінка 63) демонструє помилку зв’язку з API сервером.

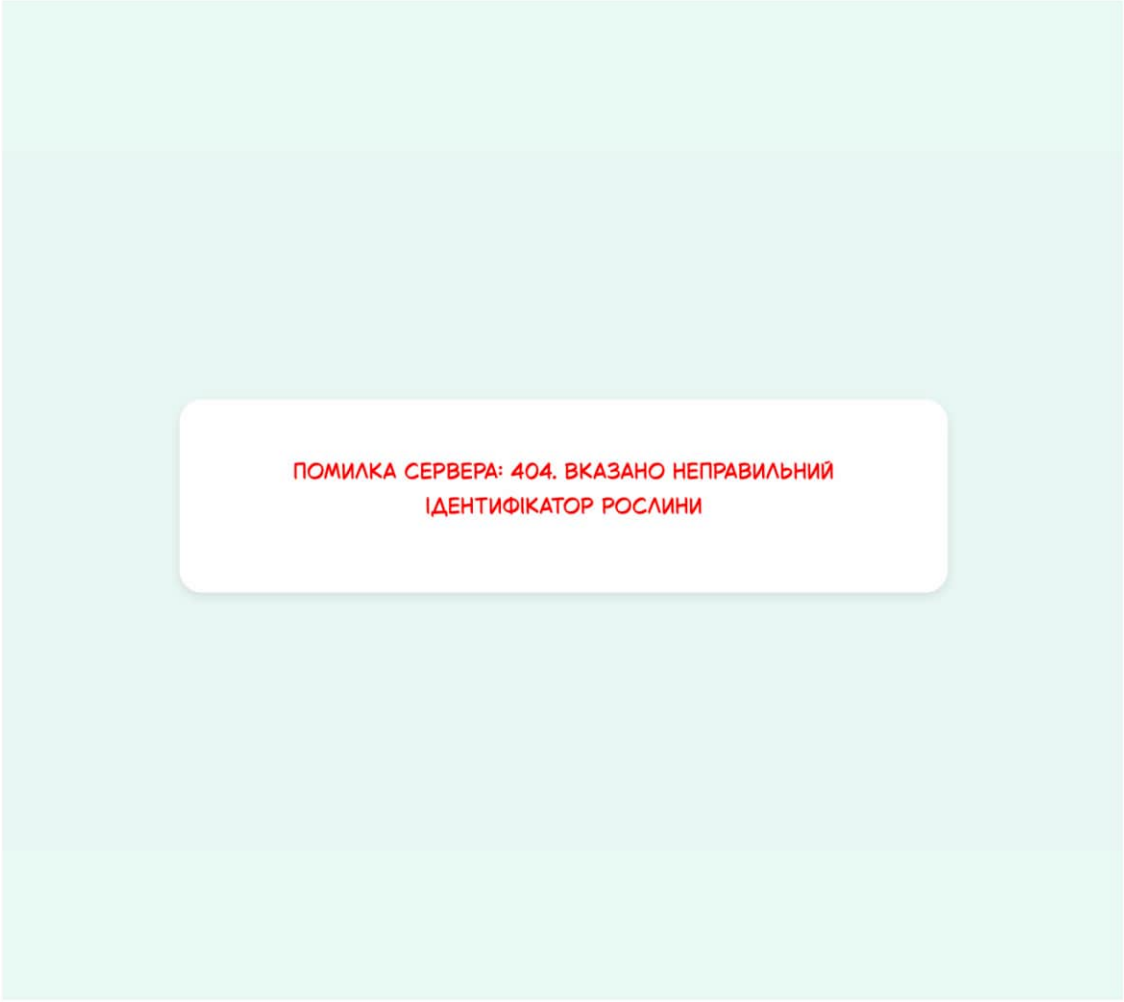

Рисунок 3.4 – Демонстрація помилки серверної частини 404

3.6 Висновки до третього розділу

В цьому розділі було розглянуто виникнення можливих сценаріїв під час використання веборієнтованої інформаційної системи а також кінцевий результат виконання цих сценаріїв.

При виникненні сценаріїв які викликані появою помилок було передбачено коротке пояснення та можливе вирішення помилки. Це дозволить швидше визначити та виправити збої в роботі інформаційної системи.

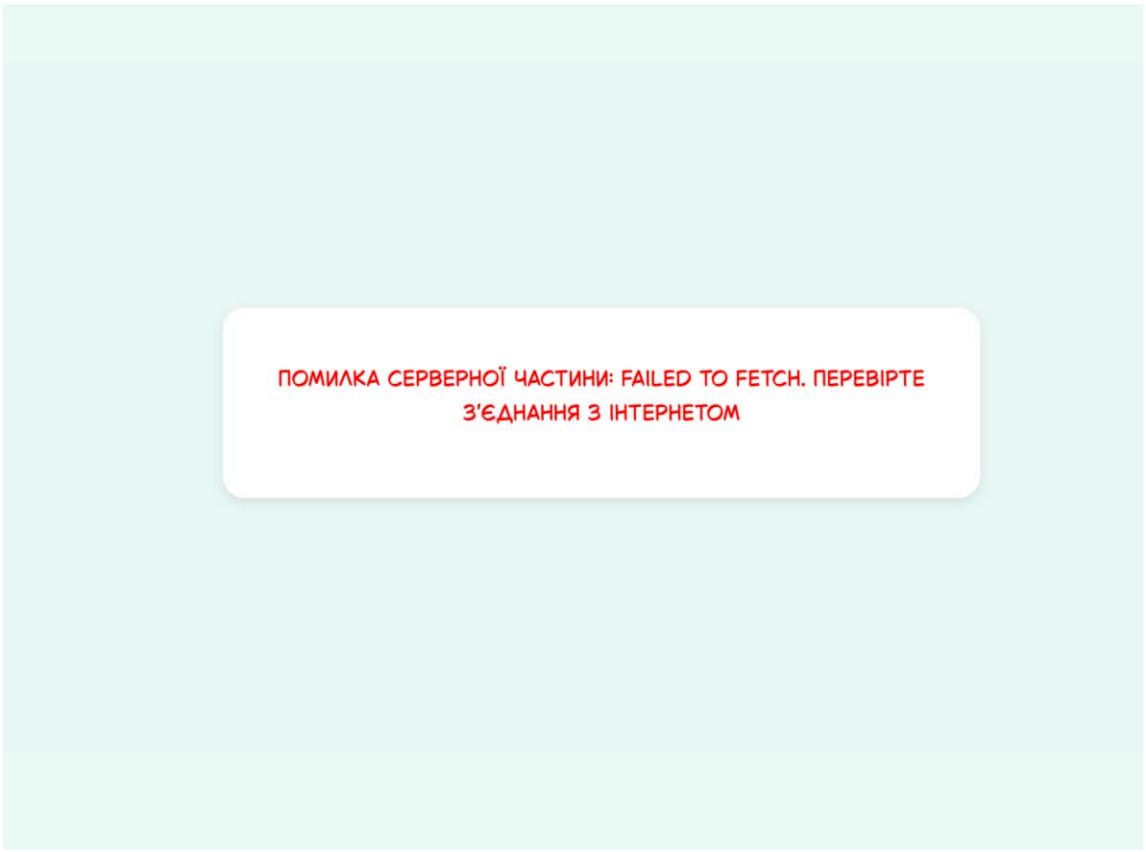

Рисунок 3.5 – Демонстрація помилки серверної частини 404

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було створено моніторингову веборієнтовану інформаційну систему яка містить елементи IoT.

Було проаналізовано стандартні компоненти типових інформаційних систем та обрано найбільш підходящі елементи саме для поставленої задачі в цій роботі. Створена архітектура оптимізована як для потреб потужності інформаційної системи, так і щодо фізичних потреб, як от компактності та масштабованості.

Також було окремо розроблено програмні продукти які є складовими інформаційної системи, наприклад API, яке прийматиме запити на запис або читання даних про різні рослини, або Frontend частина у вигляді вебсторінок, які отримують дані з API серверів та передбачають виникнення можливих помилок під час роботи інформаційної системи. Компоненти архітектури налаштовані таким чином, щоб безперебійна комунікація між цими компонентами була стабільною та гарантованою. Окрім всього структура та налаштування окремих компонентів дозволяють комфортно розширяти IoT інформаційну систему.

Результат виконання атестаційної роботи може бути використаний на агрокультурних підприємствах будь-яких розмірів для контрольованого вирощування агрокультур, що підвищить ефективність цього процесу, оскільки створена веборієнтовна інформаційна система окрім локальних значень навколишнього середовища також містить рекомендовані значення, дотримання яких сприятиме ефективності росту агрокультур.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 Internet of Things [Електрон. ресурс]. – Спосіб доступу: URL: https://www.britannica.com/science/Internet-of-Things

2 Rahman, Md. N., Hangs, R., & Schoenau, J. (2020). Influence of soil temperature and moisture on micronutrient supply, plant uptake, and biomass yield of wheat, pea, and canola. Journal of Plant Nutrition, 43(6), 823–833. – Спосіб доступу: URL: https://doi.org/10.1080/01904167.2020.1711941

3 Capacitive v/s Resistive Soil Moisture Sensor [Електрон. ресурс]. – Спосіб доступу: URL: https://www.dfrobot.com/blog-1156.html

4 Singh, D., Sandhu, A., Thakur, A. and Priyank, N. (2020). An Overview of IoT Hardware Development Platforms. International Journal on Emerging Technologies, 11(5): 155–163. – Спосіб доступу: URL: https://www.researchgate.net/publication/344207338_An_Overview_of_IoT_Hardware_Development_Platforms

5 B. M. Adam, A. Rachmat Anom Besari and M. M. Bachtiar, "Backend Server System Design Based on REST API for Cashless Payment System on Retail Community," 2019 International Electronics Symposium (IES), Surabaya, Indonesia, 2019, pp. 208-213, doi: 10.1109/ELECSYM.2019.8901668. – Спосіб доступу: URL: https://ieeexplore.ieee.org/abstract/document/8901668

6 SQLite vs PostgreSQL: A Detailed Comparison [Електрон. ресурс]. – Спосіб доступу: URL: https://www.datacamp.com/blog/sqlite-vs-postgresql-detailed-comparison

7 PostgreSQL vs. MySQL: Choosing the Right Database for Your Project [Електрон. ресурс]. – Спосіб доступу: URL: https://www.datacamp.com/blog/postgresql-vs-mysql

8 TypeScript and Express: Building Robust APIs [Електрон. ресурс]. – Спосіб доступу: URL: https://medium.com/@abdeljalil-salhi/typescript-and-express-building-robust-apis-81aef124d405

9 Web server [Електрон. ресурс]. – Спосіб доступу: URL: https://en.wikipedia.org/wiki/Web_server

10 Using Apache Web Server: Advantages and Disadvantages [Електрон. ресурс]. – Спосіб доступу: URL: https://www.cybexhosting.net/using-apache-web-server

11 Wat is RFID? [Електрон. ресурс]. – Спосіб доступу: URL: https://nfcw.nl/blog/wat-is-rfid/#wat-is-rfid

12 How do I send JSON payload to the server? [Електрон. ресурс]. – Спосіб доступу: URL: https://reqbin.com/req/2xhbguy8/json-payload-example

13 millis() – Arduino Documentation [Електрон. ресурс]. – Спосіб доступу: URL: https://docs.arduino.cc/language-reference/en/functions/time/millis/

14 HTTP Messages - HTTP | MDN [Електрон. ресурс]. – Спосіб доступу: URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Messages

15 Git [Електрон. ресурс]. – Спосіб доступу: URL: https://en.wikipedia.org/wiki/Git

ДОДАТОК А
ПРОГРАМНИЙ КОД МОНІТОРИНГОВОЇ ІНФОРМАЦІЙНОЇ СИСТЕМИ

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#define moist_pin A0
unsigned long lastTime = 0;
const char* ssid = "SVA_126-21-1";
const char* password = "PuntdakGoor";
unsigned long timerDelay = 900000;
const char* serverName = "http://agrocult.local:3000/api/flowers";

void setup(void)
{
  Serial.begin(9600);
  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
     delay(500);
     Serial.print(".");
  }
  pinMode(moist_pin, INPUT);
}
void loop() {
  if ((millis() - lastTime) > timerDelay) {
    if(WiFi.status()== WL_CONNECTED){
      WiFiClient client;
      HTTPClient http;

      http.begin(client, serverName);

      http.addHeader("Content-Type", "application/json");
      StaticJsonDocument<200> jsonDoc;
      jsonDoc["plant_id"] = "phalaenopsis 'rome'";
      jsonDoc["humidity"] = map(analogRead(moist_pin), 3088, 1241, 0,
100);

      String requestBody;
      serializeJson(jsonDoc, requestBody);

      int httpResponseCode = http.POST(requestBody);

      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);

      http.end();
    }
    else {
      Serial.println("WiFi Disconnected");
    }
    lastTime = millis();
  }}
```

ДОДАТОК Б
ПРОГРАМНИЙ КОД JAVASCRIPT СКРИПТУ

```
      function getQueryParam(param) {
          const urlParams = new URLSearchParams(window.location.search);
          return urlParams.get(param);
      }

      const plantId = getQueryParam('plant_id');
      if (!plantId) {
          document.body.innerHTML = '<h1 style="color:red;">Не вказано
ідентифікатор рослини.</h1>';
          throw new Error('plant_id не вказано у URL.');
      }

      const authHeader = {
          'Authorization': 'Token d691daac8e71d5997a490ef9f2965cc63606b802'
      };

      fetch(`http://172.16.3.104:3000/api/flowers/newest/${plantId}`)
          .then(response => {
              if (!response.ok) {
                  throw new Error(`Помилка сервера: ${response.status}`);
              }
              return response.json();
          })
          .then(data => {
              // Відображення отриманих даних на сторінці
              document.getElementById('plant-id').textContent =
data.plant_id;
              document.getElementById('soil-humidity').textContent =
`${data.humidity}%`;
              document.getElementById('measurement-time').textContent = new
Date(data.timestamp).toLocaleString();
              document.getElementById('location').textContent =
data.location;
          })
          .catch(error => {
              document.body.innerHTML = `<h1 style="color:red;">Сталася
помилка: ${error.message}</h1>`;
          });

      const location =
fetch(`http://172.16.3.104:3000/api/flowers/newest/${plantId}`)

      fetch(`http://172.16.3.104:3000/api/surroundings/${location.data.loca
tion}`)
          .then(response => {
              if (!response.ok) {
                  throw new Error(`Помилка сервера: ${response.status}`);
              }
              return response.json();
          })
          .then(data => {
              document.getElementById('air-temp').textContent =
`${data.air_temperature}C`;
```

```
            document.getElementById('air-humidity').textContent =
`${data.air_humidity}%`;
            document.getElementById('luminocity').textContent =
`${data.lighting}`;
            document.getElementById('atm-pressure').textContent =
`${data.air_pressure} мм.рт.ст.`;
        })
        .catch(error => {
            document.body.innerHTML = `<h1 style="color:red;">Сталася
помилка: ${error.message}</h1>`;
        });

    fetch(`https://open.plantbook.io/api/v1/plant/detail/${plantId}/?form
at=json`, { headers: authHeader })
        .then(response => {
            if (!response.ok) {
                throw new Error(`Помилка сервера: ${response.status}`);
            }
            return response.json();
        })
        .then(data => {
            // Відображення отриманих даних на сторінці
            document.getElementById('plant-name').textContent =
`${data.display_pid}`;
            document.getElementById('plant-image').src = data.image_url;
            document.getElementById('recom-humidity').textContent = `Від
${data.max_soil_moist}% до ${data.min_soil_moist}%`;
            document.getElementById('air-temp').textContent += ` / Від
${data.max_temp}C до ${data.min_temp}C`;
            document.getElementById('air-humidity').textContent += ` /
Від ${data.max_env_humid}% до ${data.min_env_humid}%`;
            document.getElementById('luminocity').textContent += ` / Від
${data.max_light_lux} lux до ${data.max_light_lux} lux`;
        })
        .catch(error => {
            document.body.innerHTML = `<h1 style="color:red;">Сталася
помилка: ${error.message}</h1>`;
        });
```