КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій проектування та прикладної математики

(кафедра)

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «IoT система моніторингу та сезоного контролю динаміки росту тепличних рослин»

БОКАЧ РОМАН ВАЛЕРІЙОВИЧ

(прізвище, ім’я та по батькові студента повністю)

Київ 2026 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

автоматизації і інформаційних технологій

(факультет)

інформаційних технологій проектування та прикладної математики

(кафедра)

ЗАТВЕРДЖУЮ
Деканом факультету
д.т.н., професор Терентьєв О.О.
_____
„___” _____2026 року

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

на тему: «IoT система моніторингу та сезоного контролю динаміки росту тепличних рослин»

Виконав: Студент спеціальності
126 «Інформаційні системи та технології»
(шифр і назва спеціальності)

Бокач Р.В.
(прізвище та ініціали)

Керівник д.т.н., проф. Терентьєв О.О.
(прізвище та ініціали)

Рецензент доц. Рябчун Ю.В.
(прізвище та ініціали)

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: автоматизації і інформаційних технологій

Кафедра: інформаційних технологій проєктування та ПМ

Освітній рівень: «бакалавр» за ОПП

Спеціальність: 126 «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ

Деканом факультету

д.т.н., професор Терентьєв О.О.

_____

„___” _____ 2026 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІНОЇ РОБОТИ

НА ЗДОБУТТЯ ОСВІТНЬОГО РІВНЯ «БАКАЛАВР»

Бокач Роман Валеріойвич

1. Тема роботи: IoT система моніторингу та сезоного контролю динаміки росту тепличних рослин
затверджена наказом ректора КНУБА № 1857/23.6/25 від «03» 11 2025 р.
2. Керівник роботи: Терентьєв О.О., декан факультету
3. Строк подання студентом роботи до захисту: червень 2026 р
4. Зміст пояснювальної записки за розділами:
Р.1. Аналіз предметної області та постановка задачі
Р.2. Архітектура системи та моделі обробки інформації
Р.3. Розробка інформаційного забезпечення системи
Р.4. Розробка програмного забезпечення системи
Р.5. Ергономіка та користувацький інтерфейс інформаційної системи
5. Інформаційні слайди:
С.1. Характеристика проблеми та IoT-рішень у сільському господарстві
С.2. Структурно-функціональна модель системи
С.3. Концептуальна модель предметної області
С.4. Даталогічна модель бази даних
С.5. Фізична реалізація бази даних
С.6. Програмна реалізація системи та тестовий приклад роботи

6. Календарний план виконання кваліфікаційної роботи

Види робіт та їх зміст	Дата виконання
Р.1. Аналіз предметної області та постановка задачі	Січень 2026 р.
Р.2. Архітектура системи та моделі обробки інформації	Лютий 2026 р.
Р.3. Розробка інформаційного забезпечення системи	Березень 2026 р.
Р.4. Розробка програмного забезпечення системи	Квітень 2026 р.
Р.5. Ергономіка та користувацький інтерфейс інформаційної системи	Травень 2026 р.
Остаточне оформлення роботи	Червень 2026 р.
Направлення роботи на рецензування	Червень 2026 р.
Попередній захист роботи на кафедрі	Червень 2026 р.

7. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта, представника комісії	Дата	Підпис
Ергономіка інформаційних технологій	д.т.н. проф. Терентьєв О.О.		
Прийом програмного продукту	д.т.н. проф. Терентьєв О.О.		

8. Дата видачі завдання: 02 грудня 2025 року

Керівник _____ Терентьєв О.О.

(підпис) (прізвище та ініціали)

Бакалавр _____ Бокач Р.В.

(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Тема роботи: «IoT система моніторингу та сезоного контролю динаміки росту тепличних рослин»

Мета кваліфікаційної роботи бакалавра - створення і впровадження комплексної IoT системи, яка допомагає забезпечити оптимальне функціонування тепличних господарств, підвищити продуктивність, ефективність та якість виробництва, зменшити витрати ресурсів та збільшити прибутковість.

Об’єкт дослідження – система IoT рішень для моніторингу та контролю тепличних рослин.

Предмет дослідження – обробка даних з сенсорів, методи їх контролю та передачі з використанням систем IoT.

Кваліфікаційна робота бакалавра складається зі змісту, вступу, основної частини, яка включає три розділи, висновків та списку використаних джерел. Всього 103 сторінки.

КЛЮЧОВІ СЛОВА: IoT, сенсори, теплиця, моніторинг, автоматизація, контроль, Node.js, Express.js, SQLite.

SUMMARY

The topic of the bachelor’s qualification work is: “IoT system for monitoring and seasonal control of greenhouse plant growth dynamics”.

The aim of the bachelor’s thesis is to design and implement a comprehensive IoT system that ensures optimal operation of greenhouse farming systems, increases productivity, efficiency and product quality, reduces resource consumption, and improves overall profitability.

The object of the study is an IoT-based system for monitoring and controlling greenhouse plants.

The subject of the study is data processing from sensors, methods of control and transmission using IoT technologies.

The qualification work consists of an introduction, main body (which includes three chapters), conclusions, and a list of references. The total volume of the work is 103 pages.

KEY WORDS: IoT, sensors, greenhouse, monitoring, automation, control, Node.js, Express.js, SQLite.

ЗМІСТ

ВСТУП

Розвиток тепличного виробництва став актуальною темою у зв'язку з постійним зростанням попиту на свіжі овочі та зелень, особливо взимку. Проте, ефективне управління вирощуванням тепличних культур потребує постійного моніторінгу, контролю їхнього росту, тощо. В даному контексті Internet-Of-Things (далі IoT) системи моніторінгу набувають великого значення, забезпечуючи надійний аналіз та збір даних про зростання тепличних культур. Сезонний контроль динаміки росту тепличних культур стає ключовим елементом в забезпеченні їхньої оптимальної урожайності та якості. Таким чином, розробка і впровадження інноваційних IoT систем моніторінгу та сезонного контролю динаміки росту культур є важливим кроком у покращенні ефективності та якості вирощування культур в тепличних господарствах.

Мета цієї роботи полягає в розробці та впровадженні системи моніторінгу та сезонного контролю динаміки росту тепличних культур на основі IoT технологій.

Об'єктом дослідження є тепличні культури, які вирощуються в тепличних умовах для отримання якісного врожаю.

Предметом дослідження є система моніторінгу та сезонного контролю динаміки росту тепличних культур, включаючи апаратні засоби (датчики, пристрої збору даних) та програмне забезпечення (далі ПО) для збору, аналізу та візуалізації інформації про хід росту культьур.

Наукова новизна цього дослідження полягає в розробці та використанні інтегрованої системи моніторінгу, яка базується на технологіях IoT, для контролю динаміки росту тепличних культур. Дана система включає в себе сучасні сенсори, пристрої збирання даних та ПО, що дозволяє збирати, аналізувати та візуалізувати інформацію про ріст культур у реальному часі. Результати цього дослідження сприятимуть удосконаленню методів вирощування тепличних культур, що дозволить аграрним виробникам ефективніше контролювати та оптимізовувати умови росту та розвитку культур. Такий підхід відкриває нові можливості для підвищення якості та врожайності продукції у сільському господарстві. Також, це робить його важливим внеском в забезпечення продовольчої безпеки.

Теоретичне значення даної роботи полягає в розширенні розуміння принципів і методів моніторінгу та контролю росту тепличних культур за

допомогою IoT пристроїв. Результати дослідження сприятимуть поглибленню теоретичних знань у сфері агрономії, поглибить розуміння впливу різноманітних факторів на розвиток культур та дозволить вдоскуналювати методи їхнього вирощування.

Практичне значення полягає у створенні системи моніторінгу, яка дозволить аграрним виробникам ефективніше контролювати та оптимізовувати умови вирощування тепличних культур. Це сприятиме підвищенню врожайності та якості продукції, зменшенню втрат врожаю, оптимізації витрат на виробництво. Окрім цього, впровадження інтегрованої системи моніторінгу буде сприяти зменшенню негативного впливу вирощування тепличних культур на навколишнє середовище шляхом ефективного використання ресурсів та уникнення перевищення допустимих норм.

РОЗДІЛ 1.

АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Постановка задачі

Постановка задачі полягає в розробці та впровадженні системи моніторінгу та сезонного контролю динаміки росту тепличних культур з викроистанням технологій IoT. Конкретні завдання включають в себе:

- Розробка апаратно-програмного забезпечення для збору та аналізу даних про хід росту культур.
- Створення сенсорних пристроїв для вимірювання параметрів навколишнього середовища та розвитку культур.
- Розробка системи зв’язку для передачі даних з датчиків до центральної системи обробки інформації.
- Розробка алгоритмів обробки та аналізу даних для виявлення закономірностей у росту та розвитку культур.
- Імплементація системи моніторінгу та контролю в реальному тепличному середовищі.
- Проведення евсперементальних досліджень для перевірки ефективності системи та її впливу на врожайність та якість продукції.

Ці завдання спрямовані на створення практично застосованного рішення для підвищення ефективності вирощування тепличних культур а також на оптимізацію процесів сільских господарств.

Загальне завдання полягає в створенні і впровадженні комплексної системи моніторінгу та контролю динаміки росту тепличних культур з використанням IoT пристроїв. Ця система має мету забезпечити постійний нагляд за умовами росту культур, збір та аналіз даних про їх ріст та розвиток без участі людини, а також надати можливість оперативного втручання для умов вирощування. Для досягнення цієї мети необхідно розробити та реалізувати комплекс технічних та програмних рішень, які включатимуть в себе апаратні засоби для збору даних, засоби зв’язку для передачі інформації, а також алгоритми обробки та аналізу даних. Результатом виде створення системи, яка дозволить оптимізувати процес вирощування культур у теплицях, забезпечуючи підвищену врожайність та якість продукції.

1.2 Теоретичні відомості

Система моніторінгу та контролю теплиць спрямована на підвищення продуктивності та ефективності сільского господарства в умовах сучасної конкуренції на ринку агробізнесу. За дослідженнями, до 2030 року очікується зростання ринку IoT у сільськогосподарському секторі до близько 2.026 трильйонів доларів США. Прогнозується значний розвиток IoT у сільскому господарстві з річним приростом капіталу та прибутку (CAGR) +8.8%.

Технологія IoT включає автономні методи управління процесами в сільскому господарстві, що допомагають фермам виробляти високоякісну продукію та забезпечують їх найкращими та найсучаснішими засобами для керування процесами виробництва. Наприклад, фармацевтична компанія, акції якої торгуються на біржі, звернулася до Dosun IoT для вирощування гідропонних культур у теплиці, щоб задовольнити власні бізнес потреби.

Ріст та розвиток культур безпосередньо залежать від доступності фотосинтезу, який необхіден для їхньої життєдіяльності, цвітіння та плодоношення. Однак через непередбачувані зміни клімату та умов освітлення, культури часто потребують додаткової підтримки, щоб ефективно використовувати фотосинтетичні ресурси на різних етапах свого розвитку. Це особливо важливо на початкових стадіях росту, наприклад, при вирощуванні розсади.

У зусиллях поліпшити умови вирощування в непередбачуваних умовах, все більше людей звертаються до кімнатних культур. Однак вирощування таких культур у приміщеннях ускладнюється відсутністю достатньої кількості сонячного світла. Крім того, традиційні системи освітлення на основі світлодіодних ламп зазвичай використовують фіксоване співвідношення червоного та синього світла для досягнення певної нтенсивності освітлення.

Спроби задовольнити потреби різних культур у світлі або на різних етапах їхнього росту можуть призвести до недостатнього або надмірного освітлення. Тому для оптимального росту та розвитку культур у тепличних умовах потрібен науково обгрунтований і розумний спектр світла.

Для вирішення цих проблем були розроблені автоматизовані системи моніторінгу теплиць. За остінні роки зростає популярність таких

технологій, як IoT і штучний інтелект (AI), що сприяє розвитку підключених до мережі теплиць. Це дозволяє виробникам відстежувати та контролювати різні параметри, такі як яскравість, температура, стан грунту, вологість тощо, забезпечуючи оптимальні умови для росту культур.

Компанія Dusun IoT пропонує систему моніторінгу та контролю теплиць, яка базується на останньому протоколі Bluetooth 5.4. Для забезпечення інтеграції різних пристроїв, це рішення використовує два стандарти протоколу Bluetooth: мережевий стандарт Bluetooth Mesh для управління освітленням культур і трансляцію даних маяків для датчиків терміналів.

У цій системі шлюз DSGW-030 IoT виступає ядром мережі Mesh, налаштовуючи кожен модуль Bluetooth керування світлом у всій мережі. Використовуючи стандарт протоколу Bluetooth Mesh, система збирає, контролює, та передає дані. Модуль керування сітчастим освітленням Bluetooth IoT може працювати з різними лдерелами світла, що дозволяє під’єднати до мережі світильники різних типів та розмірів.

Центр автоматизації Smart gateway DSGW-030 спеціально розроблений для розробників IoT для налаштування логіки мікропрограм. Розробники можуть створювати спеціальні мікропрограми на нижньому рівні апаратного забеспечення, що допомогає прискорити розробку IoT Gateway. Шлюз DSGW-030 для домашньої автоматизації обслуговує різні галузі, включаючи розумні будинки, інтелектуальну безпеку та пенсійні послуги, також володіє універсальним набором функцій, підтримуючи кілька бездротових протоколів, таких як Wi-Fi 2.4G, ZigBee 3.0, BLE 5.2 і Z-Wave. Основны характеристики включають:

- Операційна система (ОС): Linux@OpenWrt
- Процесор: MTK7688 (MIPS24KEc на 580 МГц)
- Оперативна пам’ять: 64МБ, доступний простір: 16МБ (для користувацької розробки)
- Флеш-пам’ять: 16МБ
- Підтримка протоколів IEEE802.11n/g/b
- Підтримка Bluetooth 5.2, Zigbee 3.0, Wi-Fi 2.4G, Z-Wave

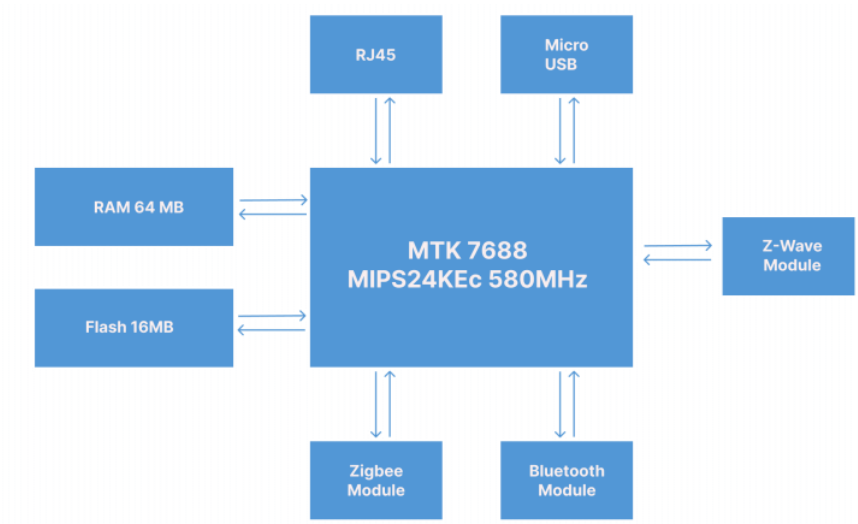


Рис. 1.1. Блок-схема обладнання

Датчик температури та вологості можна використовувати як приклад кінцевого датчика. Коли шлюз виявляє датчик поблизу, він збирає відповідні дані про температуру та вологість і надсилає їх на налаштований сервер. Після цього користувач може створювати логічні рішення та програми на серверній стороні на основі зібраних даних

Рис. 1.2. Креслення DSGW-030 Home Automation Gateway

Після налаштування мережі і шлюзу, його можна підключити до власної хмарної платформи IoT за допомогою протоколу MQTT. У цьому процесі передача даних між хмарною платформою та шлюзом є двонаправленою, що дозволяє шлюзу надсилати дані на платформу в режимі реального часу, а також дозволяє платформі контролювати обладнання для керування освітленням через інтерфейс API.

Smart Gateway виступає як центральний елемент розумного будинку, що безперервно підключається бездротовим шляхом до різноманітних розумних пристроїв і сприяє їх спільній роботі.

Основні можливості Smart Gateway включають:

- Управління освітленням, електрикою, та дрібними побутовими пристроями з будь-якого місця забезпечує зручний дистанційний доступ.
- Планування включення та вимикання пристроїв у різний час доби, автоматизуючи процеси та підвищуючи енергоефективність.
- Отримання сповіщень про несподівані події у вашому будинку, щоб завжди бути в курсі та знати про потенційні проблеми з безпекою.
- Налаштування підключеного освітлення для реагування на присутність людей, створюючи зручне та інтуїтивно зрозуміле освітлення.
- Моніторінг рівня вологості та температури для контролю умов у вашому будинку. Завдяки своїй універсальній функціональності, Smart Gateway дозволяє легко керувати розумним будинком і насолоджуватися комфортним середовищем.

Замість використання обладнання для отримання інформації встановлюються спеціальні датчики. Зібрані дані піддаються додатковій обробці, щоб отримати інформацію про стан приміщень, використання ресурсів і стан обладнання в реальному часі. Датчики дозволяють своєчасно виявляти будь-які помилки що можуть призвести до втрати потужностей.

Технологія IoT також відстежує та аналізує дані у реальному часі, щоб забезпечити контроль управління, зчитувая відпорвідня параметри (яскравість, температура, вологість, тощо.) для забезпечення кращого зростання культур

Аномалії в роботі обладнання, такі як несправність ламп освітлення, можуть призвести до перерв у процесі культивування та спричинити неправильні записи даних. Різноманітні рішення для розумніх теплиць

можуть надсилати сповіщення про виникнення проблем, що дорзволить менеджерам своєчасно реагувати та уникати втрат.

Менеджери мають змогу віддалено керувати системою за допомогою мобільного додатку. Наприклад, вони можуть швидко змінювати параметри систем для сприяння пришвидшеному росту гідропонних рослин, регулюючи різні спектри світла, яскравість, або рівень вологи. Керування водяним насосом дозволяє подавати добриво, забезпечучи коріння рослин необхідними поживними речовинами. Також можна встановити таймер для ефективного використання ресурсів.

Працівники можуть керувати повністю автоматизованою системою для вирощування за допомогою голосового керування. Достатньо надати прості команди, і система освітлення/зрошення виконають надану команду.

Шлюз DSGW-030 застосовується у різних сферах, в розумних будинках, розумних системах безпеки, сільскому господарстві, тощо. Це ядро IoT, що підтримує різні бездротові протоколи обговорені раніше. Для підключення до мережі шлюзу, доступні Wi-Fi та Ethernet.

DSM-058 Powerboard це економічне рішення для управлоіння 5CH приводом, який в основному використовується для драйверів з режимом приводу 0-10 В ШІМ. Він може забезпечувати функції дистанційного затемнення, налаштування кольору, або підключати сцени за допомогою фірмового подуля DSM-055

1.3 Сценарій використання IoT рішень для тепличних культур

IoT широко застосовується в сільському господарстві, в рослиництві, тваринництві, техніці, зрошенні, моніторингу якості води та погоди, ґрунту та боротьбі з хворобами і шкідниками, тощо. Використання IoT в базується на моніторингу, контролю, сільськогосподарській техніці, точному землеробстві та тепличному виробництві.

В сільському господарстві існує безліч факторів з навколишнього середовища, які впливають на вирощування сільськогосподарської продукції. Це такі фактори як: опади, вологість повітря, температуру, вологість ґрунту, засоленість, сонячна радіація, переміщення шкідників, та

людську діяльність. Отримання цих всіх даних допомагає розуміти процеси які відбуваються, та дозволяє знаходити та приймати оптимальні рішення для покращення якості продукції, мінімізації ризиків та максимізації продуктивності та прибутку в тому числі. Крім того, ці дані можуть знизити витрати та допомогти фермерам приймати заздалегідь заходи, з метою введення коригуючих та профілактичних заходів, на основі отриманих даних.

Ось декілька сценаріїв використання IoT рішень для різних агрокультур:

Моніторинг умов росту: Системи IoT можуть надавати інформацію в реальному часі стосовно температур, вологості, рівня CO2, рівня освітлення та інші фактори, в залежності від потреб. Це допомагає забезпечувати ідеальні умови, для зростання різних культур, доглядати за ними, а також реагувати на зміни вчасно.

- Автоматизована система поливу: Система IoT може контролювати полив культур, як в теплиці, на полі, у вазоні на основі вимірювань вологості ґрунту, та погодних умов. Це допомагає забезпечити оптимальний рівень вологості для для кожного виду рослин, запобігаючи пересиханню або ж навпаки, перенасиченню.
- Система контролю освітлення: Система IoT може контролювати, автоматично регулювати та налаштовувати освітлення в теплицях в залежності від вимог до кожної агрокультури. Вона може вмикати\вимикати світло, налаштовувати яскравість та спектр випромінення для оптимального росту.
- Система аналітики, та прогнозування врожаю: Система IoT, на основі зібраних даних, може створювати аналітику по вирощуваним культурами, та на основі цих даних, робити ймовірний прогноз на завершення процесу зростання. Це може допомагати фермерам в плануванні збору врожаю та оптимізувати ряд виробничих процесів.
- Моніторинг шкідників та хвороб: Система IoT, за допомогою датчиків, камер та машинного бачення, може виявляти шкідників та різних хвороб у рослин, та дозволить фермерам вчасно реагувати та запобігати їх поширенню.
- Віддалене керування: Система IoT дозволяє керувати теплицями, та іншими розумними системами відділено за допомоги різних

пристроїв: від телефонів, комп`ютерів, планшетів, до різних вбудованих терміналів. Це дозволяє їм моніторити та керувати процесами зростання рослин навіть не знаходячись поряд, або в самій теплиці, будівлі.

Останні п`ять років спостерігається тенденція швидкого розвитку автоматизованих систем вирощування рослин, зокрема гроубоксів (Growbox) – це комплекс спеціального обладнання для культивування рослин у штучному середовищі. Вони представляють з себе ящики, або ж шафи, різного розміру з необхідним пристосуванням для вирощування різних видів агрокультур та забезпечують високий рівень врожайності. Ця концепція дозволяє людям займатися вирощуванням не тільки на спеціалізованих фермах, а й у форматі квартири, в кімнаті, на балконі, фактично будь де. Також з початку вони були розроблені на основі дослідження у гідропоніці з метою вирощування рослин у штучному середовищі, з мінімальними витратами. Фактично це автоматизована система розумної теплиці, тільки в мініатюрі.

1.4 Огляд аналогів на ринку

Протягом багатьох століть сільське господарство вимагало від людей тяжкої фізичної, ручної праці. З часом, та розвитком технологій, дане ремесло оптимізовувалось, та на разі найоптимальнішим варіантом є використання автоматизованих ферм, теплиць. Автоматизація в даному форматі зумовлює контроль та управління кліматичними умовами, вона сприяє кращому зростанню рослини, адже немає залежності від зовнішніх факторів, а все можна налаштувати під конкретну агрокультуру, та кращій врожайності. На разі є висока потреба в автоматизації та механізації технологічних процесів. В цілому, систему моніторингу та управління можна розглядати, як взаємодію декількох управлінських процесів та об`єктів.

Загальною метою автоматизації управління є підвищення ефективності використання потенційних можливостей об`єкта управління. Їх розрізняють три основні типи систем управління мікрокліматом

1. Ручне керування. Воно включає в себе візуальний контроль росту рослин, ручний полив, вмикання та вимикання регуляторів температури, а також ручне додавання добрив та пестицидів. Цей метод вимагає

значного часу, має високий відсоток допуску людських помилок, саме це робить його менш точним та не надійним.

2. Часткова автоматизація. Вона поєднує в собі ручний контроль, а також часткову автоматизацію деяких процесів. Вона подібна до ручного керування, але зменшує рівень праці на функціях поливу та контролі параметрів.
3. Повна автоматизація. Це складні системи, які добре обладнані для реагування на будь-які зміни в підконтрольному їм середовищі. Вони працюють за принципом зворотного зв`язку, що дозволяє їм ефективно реагувати на будь які фактори. Хоч ці системи і дозволяють уникнути більшості проблем, які пов`язані з людським фактором, але такі системи можуть бути досить дорогими.

Тепер ближче перейдемо до основних компонентів таких систем, а саме найпоширенішим їх варіантів, які можемо використати в проектуванні нашої теплиці:

NodeMCU – це плата, яка була розроблена для IoT рішень компактного формату. Вона може з`єднувати фізичний об`єкт (лампочка, датчик, мотор, чайник, магнітні дверні замки, та взагалі все, що може працювати від електрики) з мережею. Дана платформа універсальна та відносно дешева. Це технічне рішення включає в себе автоматизацію процесів регуляції вологості, освітлення, температури та вентиляції нашої теплиці. Для цього всього використовуються датчики вологості, освітленості (фоторезистори), і температури відповідно.

Рис. 1.3 Плата NodeMCU ESP8266

DS3231 – це відносно дешевий, та неймовірно точний годинник реального часу (RTC) з вбудованим кварцовим генератором, з температурною компенсанцією і кристалом. Сам пристрій підтримує точне хронометрування, має вбудовану батарею, та навіть при перериванні основного живлення, продовжує працювати. Інтеграція кристалічного резонатора підвищує довгострокову точність пристрою, а також зменшує кількість використаних деталей, при виробництві.

DS3231 випускається в комерційній та промисловій сферах. RTC зберігає інформацію про секунди, хвилини, години, дні, дату, місяць та навіть рік. Дата в кінці місяця автоматично корегується на місяці які складають 31 день, в тому числі і правки на високосний рік. Модуль працює або в діапазоні 24 години, або в 12 годинному форматі з індикатором AM\PM.

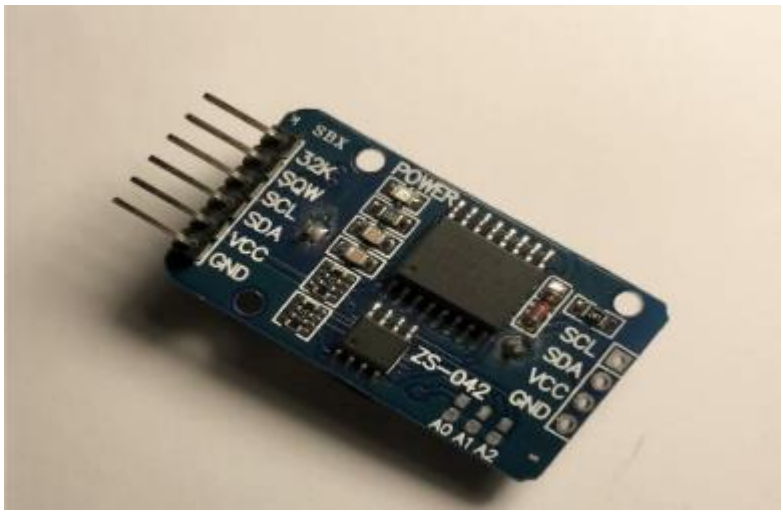


Рис. 1.4 Модуль годинника реального часу DS3231

RealTime Database – це база даних FireBase. Вона є ефективним рішенням з низькою затримкою для додатків, які потребують синхронізації станів між клієнтами в режимі реального часу. В той самий час, це хмарна база даних NoSQL, яка відома своїм оновленням інформації в реальному часі, масштабованістю та здатністю працювати в автономному режимі. Фактор реального часу є одним з найбільш унікальних його аспектів. Оновлення даних і бази даних зазвичай відбуваються тільки при виконанні HTTP-запиту, запиту GET, який оновлюється перший, та запиту POST, або PATCH, який оновлюється пізніше. По своїй логіці цей процес залежить від того, що клієнт саме виконує цей запит, оскільки в іншому випадкові оновлення даних та інформації не відбудеться.

Надалі розглянемо деякі проблеми, які пов`язані з такими системами:

1. Складність відстеження зміни кліматичних параметрів, а саме: вологість повітря, вологість ґрунту, освітленість, Ph ґрунту, температури тощо. Основна проблема саме в складності, адже від цих параметрів залежить безпосередньо робота такої системи, ефективність такої системи, ефективність росту агрокультур.
2. Високі витрати на обслуговування та потреба в високо кваліфікованому технічному персоналі. Сучасні системи використовують мобільні технології, як системи зв`язку та як бездротові системи зборку інформації, що забезпечує глобальний доступ до даних на фермах, теплицях. Проте це супроводжується різноманітними обмеженнями, а саме: складність конструкції, складність системи моніторингу, складність ремонту, складність обслуговування, високі фінансові витрати.
3. Більшість комерційних проектів з автоматизованих теплиць спрямовані на величезні тепличні комплекси з площею в декілька гектарів, тоді як ринок автоматизованих теплиць для фермерських та індивідуальних господарств залишається нерозвиненим.

1.5 Визначення вхідних даних для вирішення поставленої задачі

Провівши аналіз предметної області, можемо визначити вхідні дані для вирішення поставленої задачі, та сформувати короткий підсумок стосовно реалізації:

- Дані які ми будемо отримувати, будуть передаватись з датчиків у теплиці, це безпосередньо аналітичні дані, для формування та введення дій зі сторони системи, для оптимізації та автоматизації нашої агрокультури, створення для неї ідеальних умов зростання
- Для збору та обробки інформації створюємо базу даних, розглянемо фізичну і логічну схеми
- Розробимо функціонал, який буде емулювати роботу наших датчиків, а також емулювати ріст агрокультури, будемо розглядати на прикладі однієї агрокультури

- Побудуємо функціональну схему нашого додатку, а також концептуальну, контексту та use and case діаграму, для більш детального опису роботи нашого додатку
- На останок розглянемо методи, за допомогою яких буде реалізовано функціональну частину, базу даних, та веб інтерфейс

1.6 Висновок до першого розділу

Система моніторингу та контролю в теплицях спрямована на підвищення продуктивності та ефективності вирощування різних агрокультур в умовах розумного середовища, за дослідженнями, до 2030 року очікується зростання ринку IoT в сільськогосподарському секторі до, приблизно, 2026 мільярдів доларів США. Також прогнозується значний розвиток IoT у сільському господарстві з кількісним ростом капіталу та прибутку на рівні CAGR на 8.8%.

Технології IoT включають в себе автономні методи управління сільським господарством, що значно спрощує та допомагає фермерам виробляти якісну продукцію і забезпечує власникам ферм можливості в усіх сферах, масштабування, оптимізація, дохід.

РОЗДІЛ 2. МОДЕЛЬ ПРОЦЕСУ ОБРОБКИ ІНФОРМАЦІЇ

2.1 Проектування бази даних та проведення аналітичної діяльності

RFID - це система радіочастотної ідентифікації, вона використовується для контролю, обліку та ідентифікації об`єктів, які можуть бути співробітниками, товарами на полицях магазину, тваринами на фермах, або ж рослинами. Також RFID системи мають високу популярність, та використовуються для контролю доступу в компанія, або у системах освіти. Кожна система RFID складаються з зчитувального пристрою та також з RFID мітки, яка також називається транспондер.

В RFID мітки може бути вбудований мікропроцесор з пам`яттю, або ж навіть може мати власну оперативну систему, в залежності від потреб обраного

середовища використання. Мітки класифікуються за діапазоном зчитування: на ближню (до 20 см), на середню (від 20 см до 5 м), і на дальню (від 5 м до 300м).

Майже усі RFID мітки складаються з інтегральної схеми, для безпосереднього зберігання та обробки інформації, а також з антени для приймання та передачі сигналу.

Взагалі мітки можна також класифікувати за кількома параметрами:

- Частота.
- Джерело живлення.
- Тип пам`яті.
- Виконання.

За джерелом живлення існують пасивні та активні мітки. Пасивні мітки не мають джерела живлення, працюють вони за допомоги електричного струму, який виникає через індукцію, потужності якої достатньо, для роботи крем`яного чіпа CMOS, розміщеного в мітці, який потрібен для передачі відповідного сигналу.

Також на виробництвах пасивні мітки використовуються і вбудовуються в наліпки. Вони досить компактні, наприклад 0.5х0.5мм та можуть інтегруватись навіть в аркуш паперу. Пасивні мітки передають сигнал методом модуляції відбитого сигналу несучої частоти, що випромінюється зчитування, і приймають модульований сигнал, відбитий від мітки.

Навпроти, активні мітки вже мають власне джерело живлення, та не залежать від зчитувача, це дозволяє їм працювати на високих відстанях. Також вони можуть мати великі габарити та додатково оснащені електронікою. Оскільки активним міткам не потрібне окреме джерело енергії, вони є більш надійним варіантом, і забезпечать нам більш надійне зчитування на відстані.

Також активні мітки можуть генерувати сигнали високого рівня, аніж мітки пасивного типу, що робить їх більш придатним варіантом, для більш непридатних умов для радіосигналу. Використання таких міток дозволяє нам зберігати точну інформацію про рослину, а саме про її параметри, можна відстежувати та контролювати параметри, а також відстежувати динаміку росту рослини.

Розглянемо декілька варіацій, для використання таких міток, в проекті розумної теплиці:

1. Ідентифікація: якщо в одній теплиці ми вирощуємо не 1 агрокультуру, а декілька, в системі нам потрібно якось їх розрізняти, адже для кожної культури нам потрібні різні умови зростання. За допомоги таких міток ми можемо маркувати наші агрокультури, що дозволяє нам швидко та ефективно ідентифікувати їх, та додавати нашу інформацію у базу даних.
2. Моніторинг: RFID мітки можуть допомогти відстежувати, наскільки ефективно зростають наші рослини, доповнюючи дані про динаміку росту, а також можуть надати інформацію стосовно такого, наскільки врожайні є наші культури.
3. Управління витратами: Також RFID мітки можуть надати фермерам інформацію, стосовно витрат води, світла, добрив використовуються на 1 конкретну культуру, завдяки датчикам і даним, ми можемо визначити, скільки оптимально потрібно, для нашої рослинки, та таким чином ми можемо знизити витрати та раціонально використовувати ресурси.

Після того, як ми визначили, які дані ми збираємо про наші агрокультури, більшість з яких ми будемо отримувати з нашої RFID мітки, інформацію вирішено збирати в базі даних. Взагалі база даних – це систематизована колекція даних, яка зберігається та оновлюється в електронному вигляді, з метою отримання ефективного доступу для наших даних, та здатності маніпулювати ними. За допомоги бази даних ми можемо організувати, зберігати, та керувати великими обсягами даних, які можемо отримувати та аналізувати, збирати в статистику, за допомоги деяких запитів.

Найпоширеніша мова для бази даних, є безпосередньо SQL, за допомоги цієї мови ми можемо керувати, маніпулювати реляційними базами даних, які складаються зі зв`язаних між собою таблиць, в яких зберігається інформація про об`єкти, процеси та події. Саме в таких базах даних, інформація зберігається в вигляді таблиць, з стовпцями та рядками, де кожен стовпчик представляє певний атрибут: ім`я, дата, час. Зв`язки між нашими табличками встановлюються за допомоги індексів, та зовнішніх ключів. Індекси дозволяють нам знаходити швидко деякі значення, а зовнішні ключі відповідальні за зв`язки між таблицями.

SQL бази даних не тільки зберігають інформацію, а ще дозволяють нам виконувати деякі операції з тими даними, які зберігаються, наприклад як додавання, видалення, оновлення записів у таблицях. Також вони надають змогу створювати запит для вибору даних з однієї, або декількох таблиць, та також дозволяють об`єднувати їх, для отримання певної інформації.

Ключова перевага такої бази даних, в основі якої є мова SQL, є безпосередньо реляційний підхід до організації, формування даних. Даний підхід дозволяє не тільки зберігати дані, а і забезпечує легкий доступ до них. Самі ж зв`язки між таблицями дозволяють створювати комплексні запити та проводити аналіз даних, залежно від потреби користувача.

У SQL базах даних використовуються наступні типи зв`язків між таблицями: один до багатьох, багато до одного, один до одного, багато до багатьох. У зв`язках «один до багатьох», один запис у таблиці може мати багато суміжних записів в інших таблицях. А у зв`язку «багато до багатьох» кожен запис в таблиці може мати багато суміжних записів в іншій таблиці, і навпаки.

Для встановлення зв`язків між таблицями бази даних використовуються зовнішні ключі, які показують на значення іншої таблиці. Також індекси використовуються для підвищення швидкості пошуку та фільтрування даних, що надає нам швидкий доступ до записів у таблицях, завдяки додатковій структурі даних.

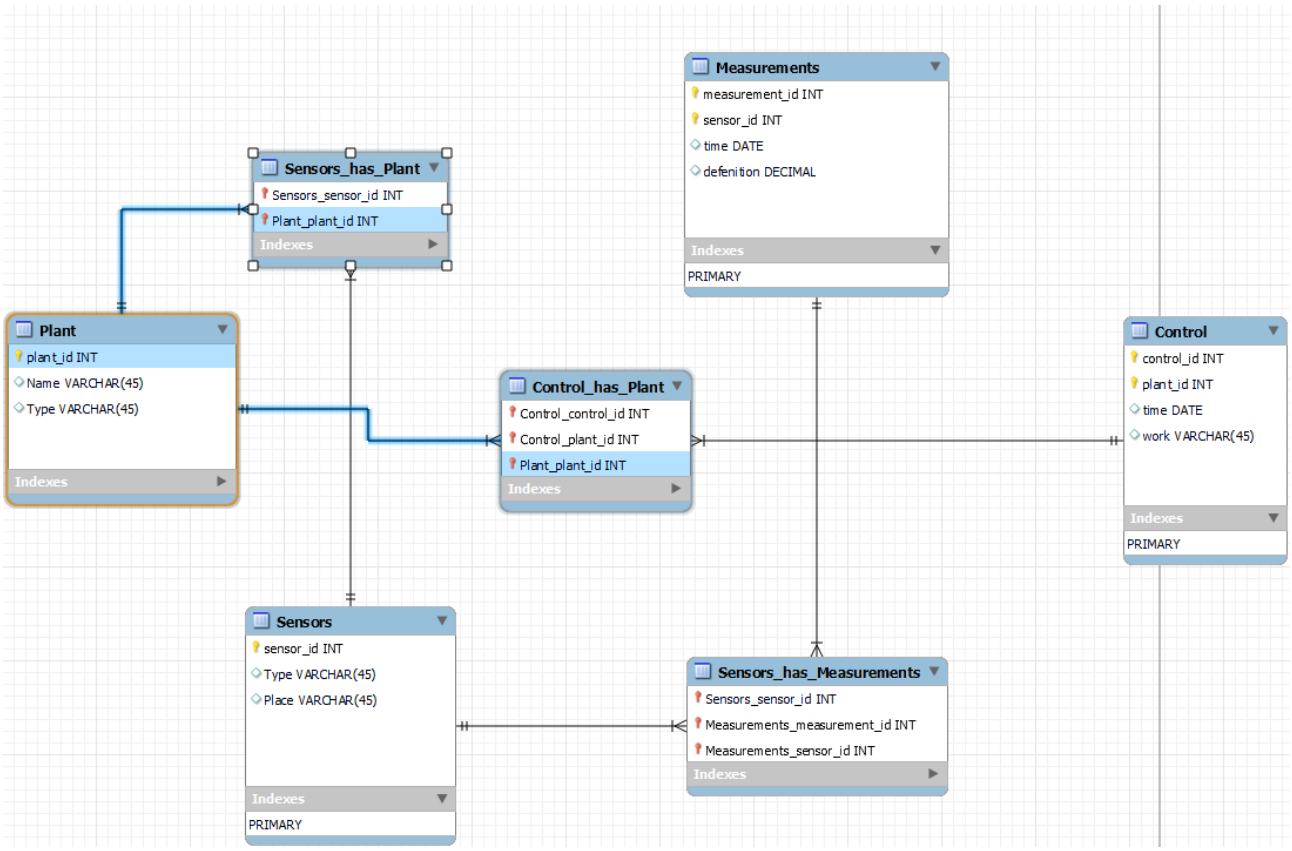


Рис 2.1 Логічна модель

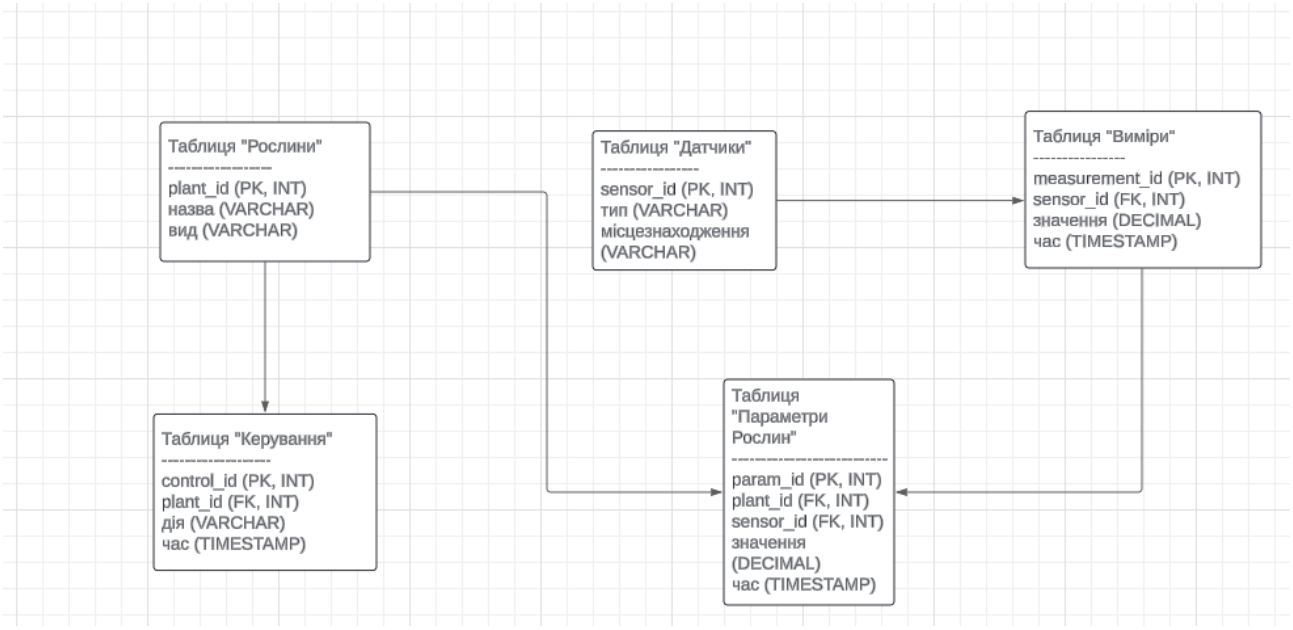


Рис 2.2 Фізична модель

База даних в додатку міститиме наступні таблиці:

1. Plant – інформація про нашу агрокультуру: айді культури, назва культури, вид культури.
2. Sensors – інформація та параметри датчиків: айді датчику, тип датчика, місце розташування датчика.
3. Measurements – інформація та збереження наших вимірів: айді виміру, айді датчика, з якого вимір взято, значення вимірювання, час протягом якого проводили вимірювання.
4. Control – айді контролюючих заходів, айді культри, над якою проводили захід, опис виконаної дії, час проведення заходу.
5. Plant settings - айді параметра культури, айді культури, айді датчика, значення параметра, який записано, час фіксації параметра.

Таблиця Plants:

- plant_id – унікальний ідентифікатор рослини.
- name – назва рослини.
- type – вид рослини.

Таблиця 2.1 Поля таблиці “Plants” з типами даних

Plant_id	INT
name	VARCHAR

type	VARCHAR

Таблиця Sensors:

- sensor_id - унікальний ідентифікатор датчика.
- type – тип датчика.
- place – місцезнаходження датчика.

Таблиця 2.2 Поля таблиці “Sensors” з типами даних

sensor_id	INT
type	VARCHAR
place	VARCHAR

Таблиця Measurements:

- measurement_id – унікальний ідентифікатор вимірювання
- sensor_id – ідентифікатор датчика (зовнішній ключ на таблицю “Sensors”)
- definition – значення вимірювання
- time – час проведення вимірювання

Таблиця 2.3 Поля таблиці “Measurements” з типами даних

measurement_id	INT
sensor_id	INT
definition	DECIMAL
time	TIMESTAMP

Таблиця Control:

- control_id – унікальний ідентифікатор контролю
- plant_id – ідентифікатор рослини (зовнішній ключ на таблицю “Plants”)
- work – опис виконаної дії
- time – час проведення контролю

Таблиця 2.4 Поля таблиці “Control” з типами даних

control_id	INT
plant_id	INT
work	VARCHAR
time	TIMESTAMP

Таблиця Plant parameters:

- param_id – унікальний ідентифікатор параметра
- plant_id – ідентифікатор рослини (зовнішній ключ на таблицю “Plants”)
- sensor_id – ідентифікатор датчика (зовнішній ключ на таблицю “Sensors”)
- definition – значення параметра
- time – час фіксації параметра

Таблиця 2.5 Поля таблиці “Plant parameters” з типами даних

param_id	INT
plant_id	INT
sensor_id	INT
definition	DECIMAL
time	TIMESTAMP

Опис зв’язків між таблицями:

1. Таблиця “Plants” та таблиця “Control”:
- У однієї рослини може бути багато записів про керування, але кожен запис про керування відноситься тільки до однієї рослини.
2. Таблиця “Plants” та таблиця “Plant parameters”:
- У однієї рослини може бути багато параметрів, але кожен параметр відноситься тільки до однієї рослини.
3. Таблиця “Sensors” та таблиця “Measurements”:
- Один датчик може робити багато вимірів, але кожен вимір робиться тільки одним датчиком.
4. Таблиця “Sensors” та таблиця “Plant parameters”:
- Один датчик може відслідковувати параметри багатьох рослин, але кожен параметр рослини відслідковується тільки одним датчиком.

5. Таблиця “Plants” та таблиця “Plant parameters”:

- У однієї рослини може бути багато параметрів, але кожен параметр відноситься тільки до однієї рослини.

2.2 Визначення необіхдних пристроїв зчитування RFID

Визначимо датчики, що потрібні для сільского господарства, та найкраще всього їй підійдуть.

Рисунок 2.3 Датчик зчитування RFID Zebra FX9600

Zebra FX9600 є високопродуктивним RFID-читачем, який може зчитувати мітки на відстані до 15 метрів із швидкістю до 1200 міток за секунду. Цей датчик оптимально підходить для роботи в умовах високої вологості та температур, що робить його ідеальним для використання у сільському господарстві, особливо в умовах ферм та складських приміщень.

Рисунок 2.4 Датчик зчитування ThingMagic M6e

ThingMagic M6e може читати RFID мітки на відстані до 10 метрів, що дозволяє фермерам ефективно відстежувати ріст рослин та інвентар у великих просторах. Цей датчик також оснащений адаптивними антенами, що дозволяє зчитувати мітки з різних кутів і орієнтацій, підвищуючи точність і швидкість зчитування. Також дозволяє працювати в різних режимах, включно з розширеним читанням та записом, що є важливим для глибокого моніторингу та управління ресурсами на фермі.

Рисунок 2.5 Датчик зчитування Honeywell IP2L

Honeywell IP2L є портативним RFID-читачем, що може зчитувати мітки на відстані до 7 метрів і підтримує Bluetooth, що дозволяє легко інтегруватися зі смартфонами та планшетами. Цей датчик ідеально підходить для сільського господарства, теплиць, де потрібна мобільність та здатність швидко сканувати мітки на культурах чи під час транспортування врожаю. Honeywell IP2L також має можливість зчитування міток в умовах різних частотних забруднень, що забезпечує точність і надійність у незвичайних умовах.

2.3 Визначення необхідних RFID міток

RFID мітки в теплицях та на фермах мають кілька основних характеристик, що будуть забезпечувати їхню надійність та ефективність в напруженому сільськогосподарському середовищі. Основні характеристики, на які потрібно звернути увагу при виборі RFID міток для використання на фермах наступні:

1. Стійкість до навколишнього середовища:
    a. Водонепроникність і пилозахист: Мітки повинні бути стійкими до води та пилу щоб витримувати зовнішні умови, особливо вологі умови теплиць та зовнішнє середовище на відкритих фермах.
    b. Температура і стійкість: Мітки повинні витримувати різкі коливання температур, що можуть виникнути через зміни погоди або контрольовану зміну середовища в теплицях.
2. Дальність читання:
    a. Мітки повинні мати достатню дальність зчитування щоб забезпечити максимально ефективне сканування на великих просторах, як наприклад у великих теплицях.
3. Живлення:
    a. Пасивні мітки зазвичай не мають власного джерела живлення, та активізуються лише коли зчитуються за допомогою RFID зчитувача, що в свою чергу зменшує частоту обслуговування міток та самовартість самих міток.
    b. Активні мітки мають вбудовані батареї, що можуть значно збільшити радіус зчитування RFID читачем, але потребують регулярного обслуговування через більш комплексну структуру та постійну заміну акумуляторів.
4. Частотний діапазон:
    a. Мітки, що працюють на (ультрависокочастотному) UHF діапазоні, краще підходять для сільскогосподарських застосунків через їхню велику дальність зчитування та високу швидкість передачі даних.

Приклади сучасних RFID міток що найбільш підійдуть для критих ферм та теплиць:

- Alien Squiggle Higgs-4:
    - Це UHF RFID мітка, яка відома своєю високою продуктивністю та високою дальністю зчитування, що робить її ідеальною для використання в сільскому господарстві. Має високу стійкість до різноманітних навколишніх впливів.
    - Діапазон 860-960MHz, 736 біт пам’яті, вироблена з полімерного матеріалу, має дальність зчитування 10-12 метрів залежно від умов

та читача, підтримує протокол EPCglobal Gen2 для високої надійності зчитування

Рисунок 2.6 UHF RFID Alien Squiggle Higgs-4

- Zebra ZT410 Silverline:
  - Спеціалізована RFID мітка для металевих поверхонь, вирізняється своєю універсальністю і стійкістю до складних умов. Її можна використовувати для ідентифікації обладнання та інвентаря на фермах і теплицях.
  - Діапазон 860-960Mhz, 512 біт пам’яті, спеціалізована конструкція для монтажу на металеві поверхні, до 6 метрів дальність зчитування, оптимізована для логістичних та індустріальних застосувань

Рисунок 2.7 UHF RFID Zebra ZT410 Silverline

- Confidex Carrier Tough II:
  - Міцна UHF RFID мітка, розроблена спеціально для використання в складних умовах. Водонепроникна та пилозахищена, ідеально підходить для використання у зовнішніх умовах.
  - Діапазон 865-928Mhz з високою чутливістю, 496 біт пам’яті, стійкий до корозії та ударів матеріал, до 11 метрів дальність зчитування, ідеальна для використання в дуже сурових умовах, умовах високої вологості та умовах екстримальних температур.

Рисунок 2.8 UHF RFID Confidex Carrier Tough II

2.4 Розрахунок площі сільского господарства та критих теплиць.

Визначення необхідних площ:

- Припустимо, що середня врожайність помідорів становить 10 кг/м², а запланований врожай — 16 тонн (16 000 кг). Таким чином, необхідна площа для вирощування огірків буде розраховуватися за формулою:

  $P = V/Y.$

  Де:
  P – необхідна площа ($м^2$)
  V – запланований врожай (кг)
  Y – середня врожайність на $м^2$ ($кг/м^2$)
  $P = 16000$ кг / 10 ($кг/м^2$)
  Таким чином, площа ля вирощування 16 тонн помідорів становить 1600$м^2$.

- До загальної площі теплиць, потрібно додати площі для додаткових приміщень, таких як:
  - Склади для зберігання добрив та інвентарю
  - Приміщення сортування та пакування врожаю
  - Офісні та технічні приміщення.

  Для складу, приміщень для пакування, офісів, та інших, можна додатково виділити 10% від площі теплиць. Підрахуємо цю площу за наступною формулою:

  $P_{дод} = 0.1 * P = 0.1 * 1600\ м^2 = 160\ м^2.$

  Таким чином загальна площа буде становити:

$$P_{заг} = P + P_{дод} = 1600\ м^2 + 160\ м^2 = 1760\ м^2$$

- Для того щоб максимально використати наявну площу та максимально збільшити врожай, потрібно оптимально розташувати теплиці та інші приміщення. При розташуванні потрібно брати до уваги орієнтацію теплиць (на південь для максимального освітлення), забезпечити зручний доступ до всіх частин теплиць, розташовувати допоміжні приміщення у безпосередній близості до теплиць.

З отриманих даних, побудуємо план сільськогосподарської ділянки:

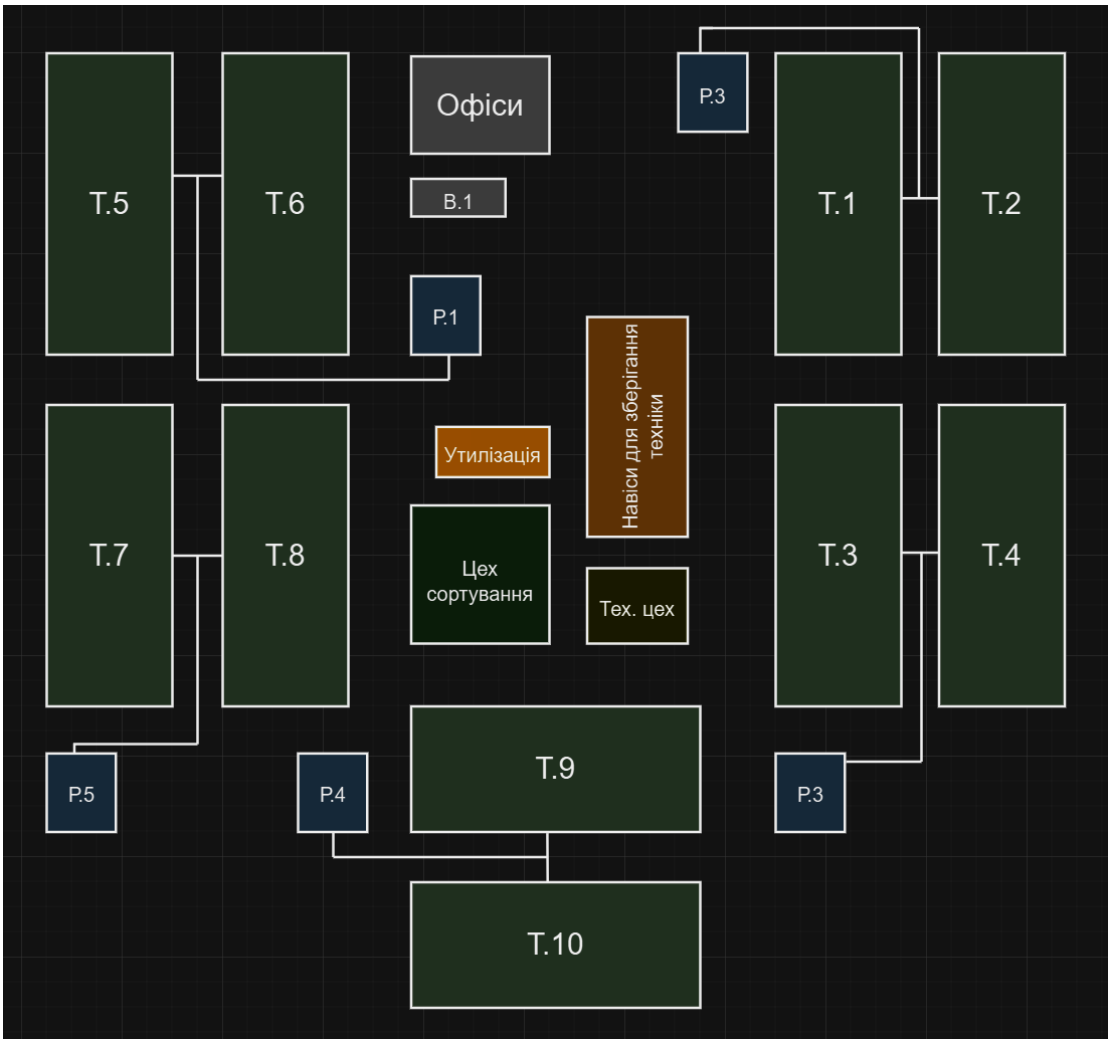


Рисунок 2.9 План сільського господарства

На плані зображено:

- Криті теплиці (Т.1, Т.2, Т.3, Т.4, Т.5, Т.6, Т.7, Т.8, Т.9, Т.10)
- Резервуари для води та добрив (Р.1, Р.2, Р.3, Р.4, Р.5)

- Цех сортування/пакування
- Технічний цех
- Навіси для зберігання обслуговуючої техніки
- Утилізаційне приміщення
- Офіси
- Зона відпочинку (В.1)

Призначення приміщень:

1. Криті теплиці
    A. Вирощування тепличних культур
    B. Кожна теплиця облаштована клімат контролем для вирощування різних культур з оптимальними умовами мікроклімату
    C. В кожній теплиці розташовані системи автоматичного орошення, освітлення, вентиляції.
2. Резервуари для води та добрив
    A. Зберігання води для зрошування теплиць та інших потреб ферми
    B. Зберігання добрив для живлення рослин
    C. Забезпечення систем автоматичного дозування добрив у зрошувальних системах
3. Цех сортування
    A. Сортування врожаю за якістю, розміром, та інш. Параметрами.
    B. Підготовка продукції до пакування та транспортування
    C. Пакування продукції/пакування
    D. Використання автоматизованих ліній сортування для підвищення ефективності процесу
4. Технічний цех
    A. Ремонт та обслуговування сільськогосподарської техніки та обладнання.
    B. Зберігання запчастин для техніки та обладнання
    C. Проведення планових технічних обслуговувань, оглядів, та робіт
5. Навіси для зберігання обслуговуючої техніки
    A. Зберігання обладнання, техніки.
    B. Захист обладнання, техніки від погодних умов
    C. Організація зручного доступу до техніки для швидкого використання в випадку потреби.

6. Утилізаційне приміщення
    A. Збір та утилізація органічних відходів
    B. Утилізація бракованої продукції
    C. Компостування органічних матеріалів для подальшого використання як добрива.
7. Офіси
    A. Адміністративне управління фермою
    B. Місце для роботи управлінського та адміністративного персоналу.
8. Зона відпочинку
    A. Місце для відпочинку працівників ферми.
    B. Їдальня та зони релаксації і відпочинку

2.5 Методи та засоби обробки інформації

Обробка інформації в системі моніторингу та котролю динаміки росту тепличних рослин, на основі IoT, є важливою складовою забезпечення ефективного вирощування рослин. Методи та засоби обробки інформації включають:

1. **Збір даних.**

Збір даних є першим етапом процесу обробки інформації. В системі використовуються різні типи сенсорів та пристроїв для збору інформації про параметри навколишнього середовища та стан рослин.

- Температурні сенсори: вимірюють температуру повітря і ґрунту.
- Вологоміри: визначають рівень вологості повітря та ґрунту.
- Сенсори освітленості: вимірюють інтенсивність світла, що потрапляє на рослини.
- Сенсори CO2: Вимірюють концентрацію вуглекислого газу в повітрі.
- RFID-мітки: Використовуються для ідентифікації рослин та відстеження їх стану.

2. **Передача даних.**

Дані зібрані за допомоги сенсорів, передаються до центральної системи обробки інформації. Для цього використовуються різні методи для передачі данних

- Бездротові мережі: Wi-Fi, Bluetooth, Zigbee забезпечують передачу даних від сенсорів, до центрального сервера.
- Провідні з`єднання: Ethernet використовується для надійної та швидкої передачі в межах теплиці.

Zigbee – це специфікація для високорівневих комунікаційних протоколів, які використовують невелику потужність, та використовуються для персональних мереж з малим радіусом дії. Основна мета даної специфікації – створити мережу для бездротових пристроїв з низьким енергоспоживання.

3. **Зберігання даних.**

Дані зберігаються в базі даних, для подальшої обробки та аналізую. В якості сховища даних використовується реляційна база даних, яка забезпечує організоване зберігання великих обсягів даних, з можливістю отримання швидкого доступу до них.

- База даних: Реалізована на основі MySQL для зберігання структурованих даних.
- Хмарні сховища: Вони використовуються для зберігання великих обсягів даних та забезпечення їх доступності з будь-якого місця.

4. **Обробка даних.**

Обробка даних включає в себе аналіз та інтерпретацію отриманих даних для прийняття рішень:

- Алгоритм обробки даних: Використовуються алгоритми машинного навчання та штучного інтелекту для аналізу даних, виявлення закономірностей та прогнозування росту рослин.
- Реальний час: Дані будуть оброблятись в реальному часі, для забезпечення оперативного реагування на зміни умов середовища.

5. Візуалізація даних.

Результати обробки даних будуть візуалізуються для користувачів через різні інтерфейси, наприклад:

- Графічні інтерфейси: Використовуються для відображення даних у вигляді графіків, діаграм та таблиць.
- Панелі управління: Інтерактивні панелі управління дозволяють користувача моніторити стан рослин та умов середовища в реальному часі.
- Мобільні додатки: такі додатки забезпечують загальний доступ до даних, з будь-якого місця та дозволяють користувачу отримувати сповіщення від нашої системи, при зміни, критичні зміни умов, стану рослин, вихід з ладу критично важливого обладнання, тощо.

6. Автоматизація управляння.

Автоматизація управління процесами в середині нашої системи забезпечить оптимальні умови для росту наших культур:

- Системи управління мікрокліматом: Автоматичне регулювання температури, вологості, освітлення та концентрації CO2.
- Системи поливу: Автоматизований полив рослин на основі даних про вологість ґрунту та погодні умови.
- Системи живлення: Автоматизоване внесення добрив та мікро елементів.

Приклад використання

Наприклад, в системі моніторингу та контролю динаміки росту тепличних рослин використовуються датчики для вимірювання вологості та вологості повітря, які, в свою чергу, передають дані до центрального серверу. Сам алгоритм обробки даних аналізує отримані дані та визначає необхідність регулювання мікроклімату в теплиці. Система автоматично вмикає та вимикає обігрів, вентиляцію та полив, з метою підтримання оптимальних умов для росту рослин. Користувач може в реальному часі переглядати стан теплиці через мобільний додаток, або ж через веб-інтерфейс та отримувати сповіщення про необхідність якихось дій, зі сторони користувача.

Таким чином методи та засоби обробки інформації в системі моніторингу та контролю тепличних культур, на основі IoT технологій забезпечать ефективне управління процесами вирощування рослин, в той самий час, підвищуючи їх врожайність та якість.

2.6 Підрахування витрат

При розрахункові витрат, на впровадження IoT системи моніторингу та контролю динаміки росту тепличних культур, врахуємо і витрати на обладнання, програмне забезпечення, розробку, установку, обслуговування та експлуатацію. Розглянемо ці витрати більш детально, з використанням плану сільського господарства, де червоні позначки вказують на RFID зчитувачі, а сині – на зчитувачі різноманітних параметрів мікроклімату теплиць.

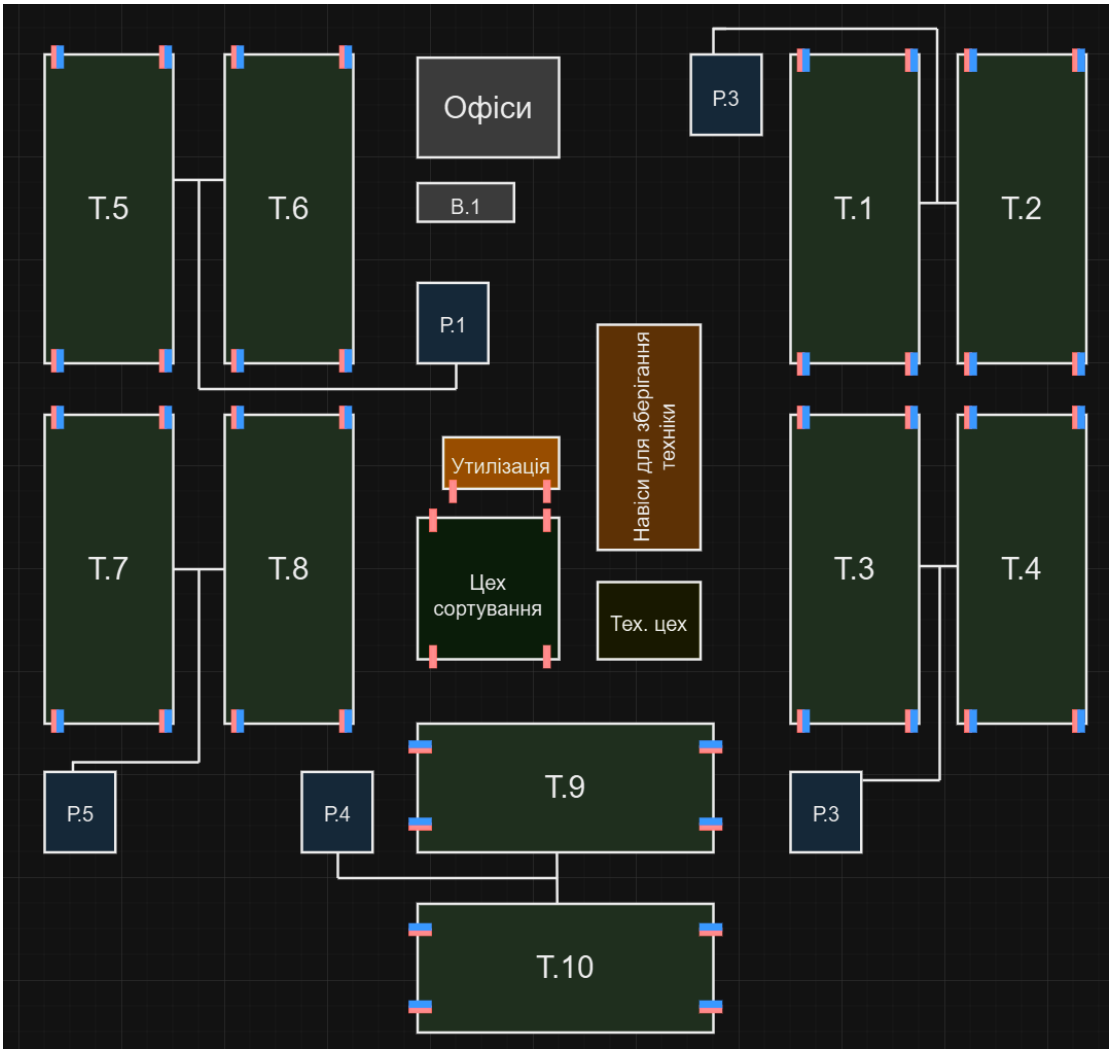


Рисунок 2.10 План сільського господарства з датчиками (червоні – RFID зчитувачі, сині – зчитувачі різноманітних параметрів мікроклімату теплиць)

1. Вартість обладнання.

Сенсори та RFID мітки:

- Температурні сенсори: Встановлюються в кожній теплиці для моніторингу температури. Вартість одного сенсора становить 30 доларів США. Для кожної теплиці потрібно 2 сенсори.
    - 2 сенсори на 10 теплиць, по 30 доларів = 600 доларів США.
- Вологоміри: Встановлюються в кожній теплиці для вимірювання вологості. Вартість одного сенсора становить 40 доларів США. Для кожної теплиці також потрібно по 2 сенсори.
    - 2 сенсори на 10 теплиць, по 40 доларів = 800 доларів США.
- Сенсори освітленості: Встановлюються для контролю освітленості в теплиці. Вартість одного сенсора в районі 50 доларів США. Для кожної теплиці – по одному сенсору.
    - 1 сенсор на 10 теплиць, по 50 доларів = 500 доларів США.
- Сенсори CO2: Встановлюються в кожній теплиці для вимірювання рівня CO2. Вартість одного – 200 доларів США. Для кожної теплиці – 1 сенсор.
    - 1 сенсор на 10 теплиць, по 200 доларів = 2000 доларів США.
- RFID-мітки: Встановлюються на кожній культурі. Для кожної теплиці це орієнтовно 100 міток. Вартість однієї мітки, 1 долар США.
    - 100 міток на 10 теплиць, по 1 долару = 1000 доларів США.

Зчитувачі RFID:

- Zebra FX9600: Даний пристрій встановлюється для зчитування міток. Вартість одного такого – 1750 доларів США. Потрібно 10 зчитувачів для покриття всієї площі.
    - 10 зчитувачів, по 1750 доларів = 17500 доларів США.

Системи обробки даних:

- Сервери: Вартість сервера, для зберігання та обробки даних складає близько 4000 доларів США.

- Комунікаційне обладнання: Включає маршрутизатори та комутатори. Вартість закупівлі становить приблизно 1000 доларів США.

2. Вартість впровадження системи.

Розробка так інтеграція:

- Розробка програмного забезпечення: Вартість розробки, приблизна – 15000 доларів США.
- Інтеграція сенсорів та RFID-міток: Вартість налаштування та інтеграціїї обладнання становить біль 7000 доларів США.

Інсталяція обладнання:

- Встановлення сенсорів та комунікаційного обладнання: Вартість проведення робіт становить біля 5000 доларів США.
- Налаштування системи управління: Вартість налаштування алгоритму обробки даних становить близько 4000 доларів США.

3. Операційні витрати.

Обслуговування обладнання:

- Регулярне обслуговування: Вартість становить біля 1500 доларів США, на рік.
- Заміна компонентів: Вартість становить біля 1000 доларів США, на рік.

Енерговитрати:

- Електроенергія: Вартість електроенергії, для роботи усього обладнання, це приблизно 300 доларів США на місяць, в ту ж чергу це 3600 доларів США, на рік.

4. Додаткові витрати:

Навчання персоналу:

- Тренінги та навчальні матеріали: Вартість навчання персоналу становить приблизно 3000 доларів США.

Резервні системи:

- Резервні копії даних: вартість створення резервних копій даних, та їх зберігання приблизно становить 2000 доларів США.

Таблиця 2.6 Загальний підрахунок витрат:

Категорія	**Вартість (USD)**
Сенсори та RFID-мітки	5900
Зчитувачі RFID Системи обробки даних	17500 5000
Розробка ПЗ та інтеграція	22000
Інсталяція обладнання	9000
Обслуговування обладнання	2500 (Щорічно)
Енерговитрати	3600 (Щорічно)
Навчання персоналу	3000
Резервні системи	2000
Загальна вартість:	70500

При розрахункові витрат, ми оцінили загальну вартість впровадження IoT системи моніторингу та контролю динаміки росту тепличних рослин, що буде спряти прийняттю обґрунтованих рішень щодо інвестування в інноваційні технології для підвищення ефективності сільського господарства.

2.7 Вибір мови програмування

При розробці програмного забезпечення для системи моніторингу та контролю динаміки росту тепличних рослин, нам необхідно обрати відповідне середовище для розробки, яке забезпечить максимальну ефективність, надійність та зручність під час роботи. Також середовище розробки повинно підтримувати

всі необхідні технології та мови програмування, які будуть використовуватись під час створення проекту.

Вимоги до середовища розробки

1. Підтримка мови програмування:

- Обране середовище розробки повинно підтримувати JavaScript та Node.js, оскыльки основна функціональна частина буде написана за використання даних мов програмування.

2. Інтеграція з інструментами обробки:

- Середовище повинно підтримувати інтеграцію з Git для контролю версій, а також з різними інструментами, такими як Replit, який також буде використовуватись під час розробки.

3. Підтримка бібліотек та фреймворків:

- Середовище повинно мати підтримку використання Express.js, EJS для шаблонізації, bcrypt для хешування паролів, connect-flash для сповіщень, і також sqlite3 для роботи з базою даних.

4. Зручний інтерфейс:

- Середовище розробки повино мати інтуїтивно зрозумілий інтерфейс для полегшення роботи.

5. Підтримка тестування та відлагодження:

- Середовище розробки повинно мати вбудовані інструменти для тестування та дебагінгу коду, що дозволить нам швидко виявляти баги, та виправляти їх.

2.8. Вибір середвища розробки

Для розробки даної проектної роботи, було обрано Replit. Replit – це інтегроване середовище для розробки, яке дозволяє писати, тестувати та реалізовувати код, без встановлення додаткового програмного забезпечення. Дане середовище підтримує цілий ряд різних мов програмування, включаючи і те, яке ми будемо використовувати, а саме Node.js та Javascript.

Основні особливості Replit:

1. Підтримка багатьох мов програмування:

- Replit підтримує різні мови програмування, такі як Javascript, Python, C++, Ruby тощо. Для нашого проекту важливими є саме підтримка Node.js та Javascript.

2. Розширення та інтеграція:

- Replit дозволяє нам інтегрувати різні бібліотеки та фреймворки. У нашому проекті ми використовували Express.js для створення серверної частини, EJS для шаблонізації, bcrypt для хешування паролів, connect флеш для сповіщень, а також SQLite3 для роботи з базою даних

```
const express = require("express");
const path = require("path");
const session = require("express-session");
const flash = require("connect-flash");
const bcrypt = require("bcryptjs");
const sqlite3 = require("sqlite3").verbose();
const app = express();
const PORT = process.env.PORT || 3000;
```

Рисунок 2.11 інтеграція бібліотек та фреймворків

3. Зручний інтерфейс:

- Replit має дуже зручний інтерфейс, який значно полегшує процес розробки. Це середовище ідеально підходить як для індивідуальної розробки, так і для роботи в групі, оскільки є можливість спільного редагування коду одночасно.

4. Інструменти для тестування та відлагодження:

- Replit надає вбудовані інструменти для тестування коду, його дебагінгу. Це надає нам змогу швидко знаходити та усувати різні помилки та баги

5. Хмарне зберігання та доступність.

- Оскільки Replit працює як хмарний сервіс для розробки, то ми можемо отримати доступ до свого проекту з будь-якого місця та пристрою. Що робить процес розробки дуже зручним та гнучким.

Впровадження системи на основі Replit

1. Створення проекту в Replit:

- Ініціалізація нового проекту на платформі Replit.
- Налаштування середовища для роботи з Node.js.

2. Налаштування залежностей:

- Встановлення нам необхідних пакетів, для роботи з проектною роботою, а саме Express.js, EJS, bcrypt, connect-flash, sqlite3
  Виконуємо це за допомоги команди: **npm install express ejs bcrypt connect-flash sqlite3**

3. Розробка основного функціоналу:

- Реалізація основної серверної частини за допомоги Express.js
- Налаштування маршрутизації задля обробки запитів і відповідей.
- Реалізація механізмів автентифікації та авторизації користувачів.

```
const app = express();
const PORT = process.env.PORT || 3000;
```

```
app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

Рисунки 2.12 та 2.13 Реалізація основної серверної частини.

```
// Middleware setup
app.set("view engine", "ejs");
app.use(express.static(path.join(__dirname, "public")));
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(
    session({
        secret: "secret",
        resave: true,
        saveUninitialized: true,
    }),
);
app.use(flash());
```

Рисунок 2.14 Налаштування маршрутизації та обробки запитів користувачів.

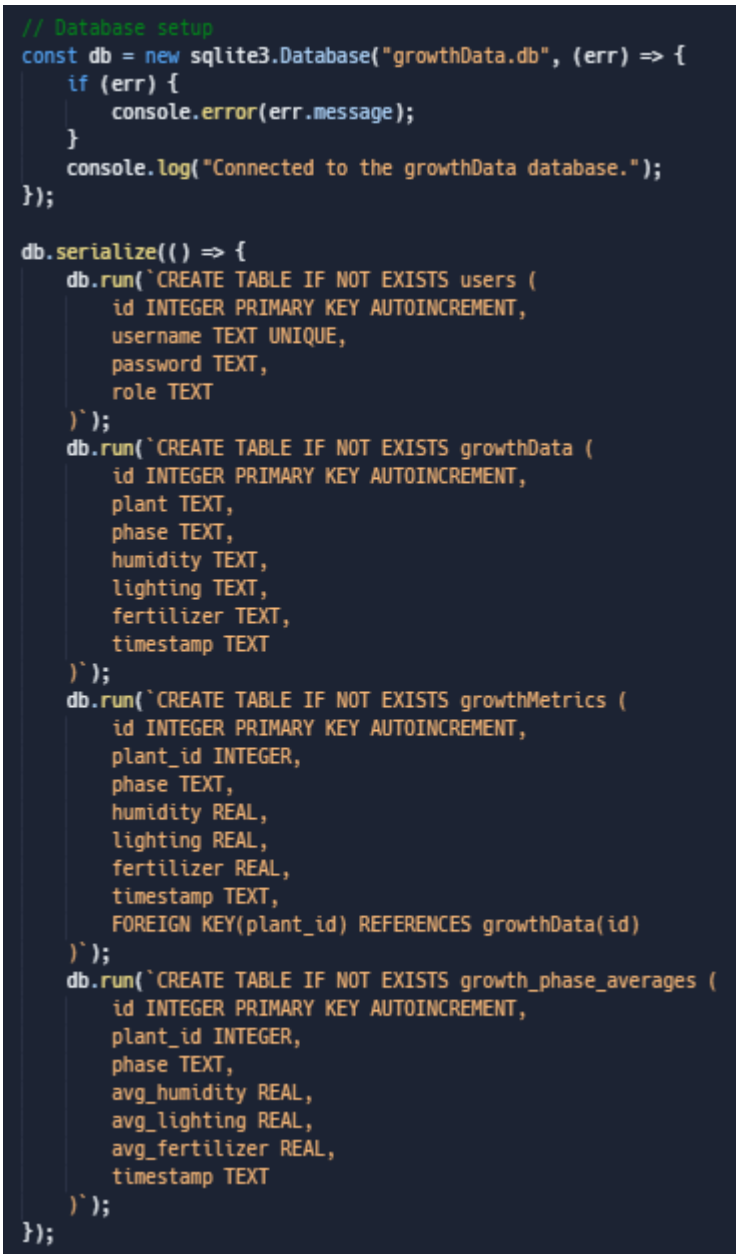
```
// Authentication middleware
function ensureAuthenticated(req, res, next) {
    if (req.session.user) {
        return next();
    }
    res.redirect("/login");
}

function ensureAdmin(req, res, next) {
    if (req.session.user && req.session.user.role === "admin") {
        return next();
    }
    res.redirect("/");
}
```

Рисунок 2.15 Реалізація механізму автентифікації користувача.

4. Інтеграція з базою даних SQLite3:

- Налаштування підключення до бази даних SQLite.
- Реалізація моделей для зберігання даних про користувачів, росту рослин, та інших параметрів.



```
// Database setup
const db = new sqlite3.Database("growthData.db", (err) => {
    if (err) {
        console.error(err.message);
    }
    console.log("Connected to the growthData database.");
});

db.serialize(() => {
    db.run(`CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT,
        role TEXT
    )`);
    db.run(`CREATE TABLE IF NOT EXISTS growthData (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        plant TEXT,
        phase TEXT,
        humidity TEXT,
        lighting TEXT,
        fertilizer TEXT,
        timestamp TEXT
    )`);
    db.run(`CREATE TABLE IF NOT EXISTS growthMetrics (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        plant_id INTEGER,
        phase TEXT,
        humidity REAL,
        lighting REAL,
        fertilizer REAL,
        timestamp TEXT,
        FOREIGN KEY(plant_id) REFERENCES growthData(id)
    )`);
    db.run(`CREATE TABLE IF NOT EXISTS growth_phase_averages (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        plant_id INTEGER,
        phase TEXT,
        avg_humidity REAL,
        avg_lighting REAL,
        avg_fertilizer REAL,
        timestamp TEXT
    )`);
});
```

Рис 2.16 Налаштування бази даних та

Ініціалізація моделей для зберігання даних

5. Розробка інтерфейсу користувача.

- Використання EJS для створення шаблону сторінок.
- Реалізація основних сторінок для відображення даних про стан росту культур та статистики.

6. Тестування та дебагінг.

- Використання вбудованих інструментів для тестування та дебагінгу коду.

- Виконання різних тестів, функціональних та модульних, для перевірки коректності роботи системи.

Висновок

Вибір середовища розробки є важливим етапом при створені IoT системи моніторингу та контролю динаміки росту тепличних рослин. Обране середовище на 100% задовольняє наші потреби, забезпечує зручність та підтримку різних інструментів, що дозволяє нам швидко та якісно реалізувати задумане рішення.

2.9. Опис клієнт серверної архітектури Replit

Для розробки, нами було обрано клієнт-серверну архітектуру Replit, яка в свою чергу базується на принципах REST. Дана архітектура є оптимальною для побудови веб-додатку, вона забезпечує масштабованість, гнучкість під час використання. Також вона дозволяє нам відокремити клієнтську частину від серверної, що підвищує надійність та безпекові заходи системи.

Основні принципи REST API:

1. Клієнт-серверна архітектура:

- Клієнт та сервер мають бути окремими компонентами, і не повинні залежити один від одного. Клієнт надсилає запити на сервер, і отримує відповіді, при всьому цьому не має інформації про реалізацію самого сервера.

2. Без стану:

- Кожен запит клієнта має містити в собі всю необхідну інформацію для сервера, а сервер в свою чергу не має зберігати інформацію про клієнта, між запитами. Такий механізм роботи забезпечує простоту в системі, та масштабованість.

3. Кешування:

- Сервер повинен повертати заголовок: Cache-Control, вын потрібен для підказки клієнту, чи можемо ми кешувати відповідь на запит. Це зменшує навантаженість на сервер, і прискорює процес обробки запитів.

4. Єдиний інтерфейс:

- Інтерфейс між клієнтом та сервером має бути єдиними, та стандартизованим, аби зменшити та виключити проблеми взаємодії між компонентами.

5. Шарована система:

- Система повинна бути розділена на різні шари, що дозволяє нам спростити розробку та тестування окремих компонентів системи. Кожен шар виконує свої функції і взаємодіє через стандартизований інтерфейс.

Висновок

Використання клієнт-серверної архітектури REST дозволяє нам побудувати гнучку та масштабовану систему для моніторингу та контролю динаміки росту тепличних рослин. Інтеграція з платформою Replit забезпечує зручність у розробці, тестуванні та впроваджені додатку, який дозволить швидко та якісно реалізувати наше рішення.

2.10 Висновок до другого розділу.

У другому розділі ми більш ретельно розглянули процес створення IoT системи моніторингу та контролю динаміки росту тепличних рослин. Дослідили деякі методи та засоби обробки інформації, а також провели розрахунок витрат на впровадження системи.

Нами було визначено, що основними складовими IoT системи є сенсори для збору даних про параметри навколишнього середовища, RFID-мітки для ідентифікації рослин, сервери для зберігання та обробки даних, а також комунікаційне обладнання для передачі даних. Особливу увагу було приділено саме вибору відповідних сенсорів та RFID зчитувачів, а також визначено необхідну кількість та типи обладнання для забезпечення повного покриття тепличного комплексу.

Використання сенсорів та RFID-міток дозволяє нам виконувати моніторинг умов вирощування в реальному часі, що також забезпечує оптимальні умови для вирощування рослин. Автоматизація управління мікрокліматом у теплицях

допомагає підтримувати необхідні параметри температури, вологості, освітленості та рівня CO2. Це сприяє підвищенню врожайності та якості продукції, а також зменшенню витрат на виробництво.

Було проведено детальний підрахунок витрат на впровадження системи, в тому числі вартість на обладнання, розробки програмного забезпечення, інсталяції та обслуговування. Загальна вартість впровадження IoT системи склала приблизно 70500 доларів США. Також враховані витрати на навчання персоналу, енерговитрати та резервні системи.

При розробці програмного забезпечення для системи моніторингу та контролю динаміки росту тепличних рослин, ми обрали відповідне середовище, яке забезпечує максимальну ефективність , надійність та зручність, під час роботи. Обране середовище розробки підтримує всі необхідні нам мови програмування, а саме JavaScript та Node.js, що є основними для нашого проекту.

Також для розробки проекту обрали середовище Replit, яке підтримує інтеграцію з Git, для контролю версій, а також з іншими інструментами, які критично необхідні для нашого проекту. Replit забезпечує дуже зручний інтерфейс, широку підтримку бібліотек та фреймворків, а також має вбудовані інструменти для дебагінгу коду.

Для забезпечення ефективної розробки нашого додатку, та його функціонування, ми обрали ряд технологій та інструментів, серед яких:

- Express.js для побудови серверної частини
- EJS для шаблонізації
- Bcrypt для хешування паролів
- Connect-flash для сповіщень
- SQLite3 для роботи з базою даних

Дані технології дозволяють створити нам гнучку, та масштабовану систему, яка відповідає всім вимогам проекту.

Таким чином, ми завершили другий розділ, він містить детальну інформацію про вибір технологій, мови програмування та середовища розробки, для розробки IoT системи моніторингу та контролю динаміки росту тепличних рослин.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ МЕТОДІВ ОБРОБКИ ДАНИХ У РЕАЛЬНОМУ ЧАСІ

3.1 Алгоритм роботи системи

Алгоритм роботи системи IoT – це комп`ютерні програми, які визначають послідовність дій, які необхідні для сбору інформації, передачі, обробки та аналізу даних, отриманих з підключених до мережі IoT пристроїв. Вони забезпечать безперебійну роботи нашої системи, та дозволять отримувати точну та актуальну інформацію для прийняття рішень.

Основні етапи алгоритму:

1. Збір даних з датчиків

- Опис: Датчики, встановлені в теплиці, вимірюють параметри: Вологість ґрунту, вологість повітря, температуру, освітленість, та показник добрив за методикою NPK (N – натрій, P – фосфор, K – калій)
- Процес: Датчики регулярно, 24\7 знімають показники та передають їх на локальний контролер.

2. Передача даних до центрального сервера

- Опис: Дані з локальних контролерів передаються на центральний сервер для обробки.
- Процес: Використовуємо бездротові засоби передачі інформації, а саме Wi-Fi чи Zigbee, або ж дротові, для окремих випадків, Ethernet.

3. Збереження даних

- Опис: Дані, які ми отримуємо з датчиків, зберігаємо в нашу базу даних, для подальшої їх обробки.
- Процес: Дані зберігаються в реляційну базу даних, в нашому випадку SQLite3, що надає нам здатність зручно зберігати, та отримувати швидкий доступ.

4. Обробка даних

- Опис: Обробка даних включає в себе аналіз та інтерпретацію отриманих показників.
- Процес: Використовуємо наші показники, для розрахунку динаміки росту в відсотковому відношенні, на основі якої можемо впровадити деякі дії зі сторони системи, для підвищення ефективності росту.

5. Візуалізація даних

- Опис: Результат обробки даних візуалізується, для сприйняття користувачем.
- Процес: Дані відображаються у вигляді графіків, діаграм та таблиці, у веб інтерфейсі, або мобільному додаткові.

6. Прийняття рішень та автоматизація

- Опис: На основі отриманих даних, система приймає рішення, стосовно необхідності внесення змін умов в теплиці
- Процес: Автоматичне регулювання мікроклімату(температури, вологості, освітленості) та поливу, а також сповіщення користувача про критичні зміни або несправності системи, для негайного втручання та вирішення можливих проболем

Блок-схема роботи системи:

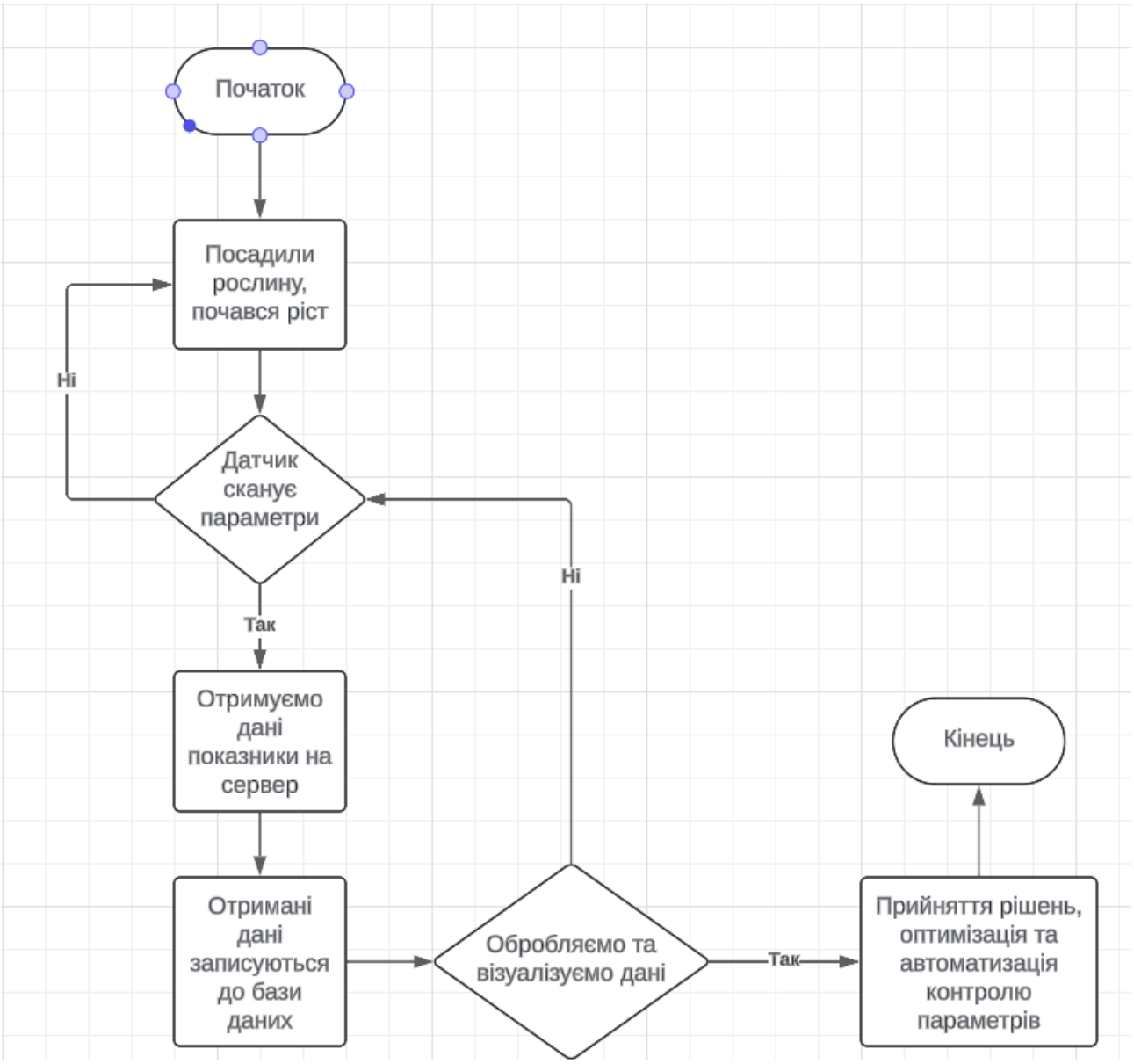


Рисунок 3.1 Блок-схема роботи системи

Діаграма станів роботи системи:

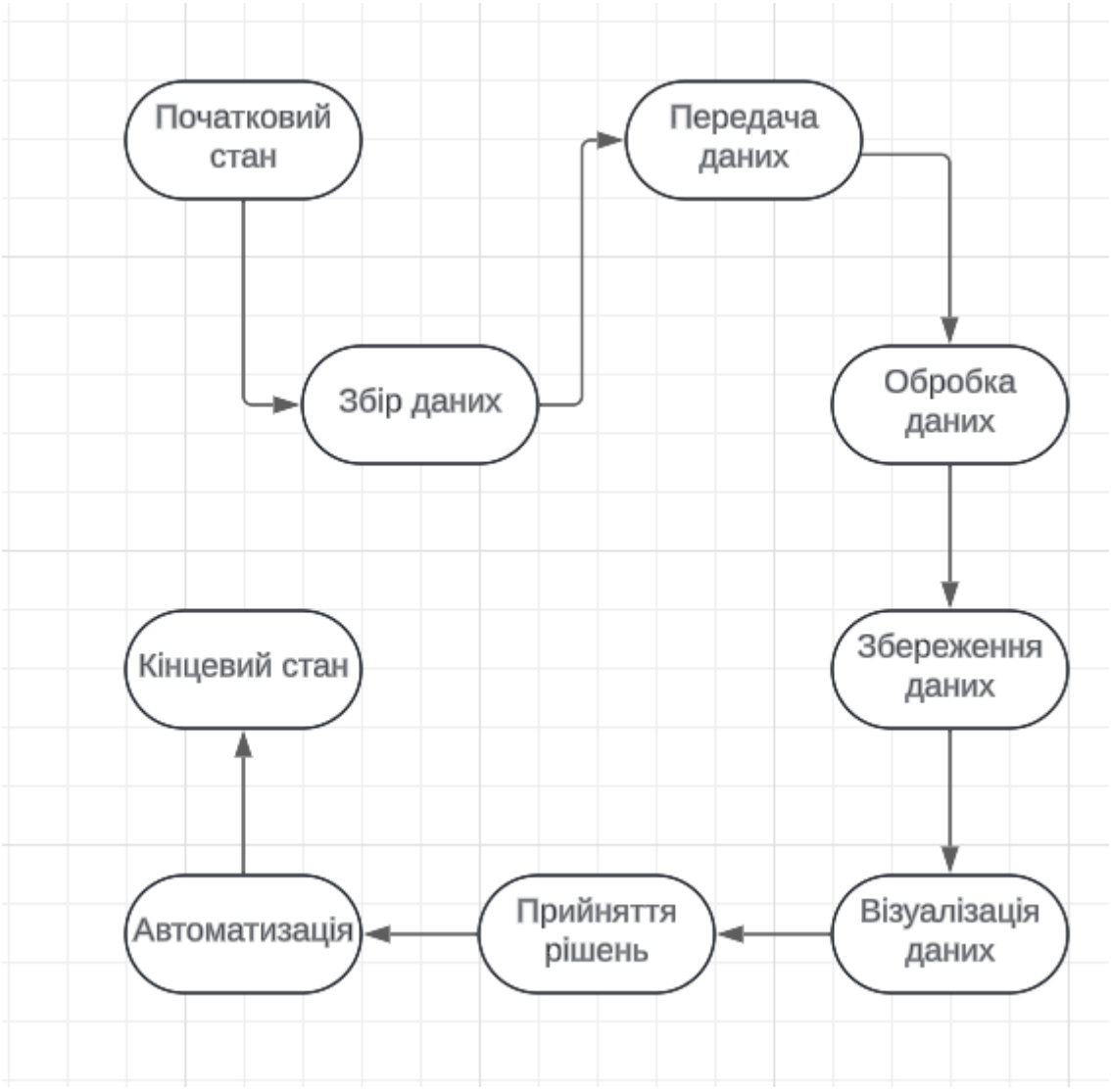


Рисунок 3.2 Діаграма станів роботи програми

Приклад роботи:

Кожна рослина з прикріпленою RFID міткою, яка знаходиться в теплиці, передає показники через RFID зчитувач до сервера. Після того, як датчик зчитує інформацію з мітки на рослині, інформація передається на сервер, де вона обробляється, записується в базу даних, та після цього візуалізується в веб-додаткові.

Висновок:

Алгоритм роботи системи моніторингу та контролю динаміки росту тепличних рослин на основі IoT технологій, забезпечує ефективний збір та аналіз даних, що дозволяє створити оптимальні умови для вирощування культур, та підвищення їх врожайності, та якості.

3.2 Шаблон роботи веб-додатку

Веб-додаток розроблений для проекту IoT система моніторингу та сезоного контролю динаміки росту тепличних рослин. Основні компоненти які були реліазовані включають серверну частину, клієнтську частину та базу даних. Веб додаток побудований з використанням Node.js, Express.js, EJS, bcrypt, connect-flash, та SQLite3.

Архітектура веб-додатку

1. Серверна частина:

- Node.js та Express.js: Серверна частина обробляє запити клієнтів, керує сесіями та маршрутизацією.
- Middleware: Використовується для обробки запитів, автентифікації користувачів та обробки помилок.

2. База дних:

- SQLite3: Є легкою реляційною базою даних, для зберігання інформації про рослини, сенсори, вимірювання та користувачів.
- Структура бази даних
    - Таблиця “Users”: зберігає інформацію про користувачів.
    - Таблиця “Plants”: зберігає інформацію про рослини.
    - Таблиця “Sensors”: зберігає інформацію про сенсори.
    - Таблиця “Measurements”: зберігає дані вимірювань.
    - Таблиця “ControlActions”: зберігає дані про контрольні дії.

3. Клієнтська частина:

- EJS: Шаблонізатор для генерації динамічних веб-сторінок
- CSS та JavaScript: Використовується для стилізації сторінок та додавання інтерактивності.

Основні функції веб-додатку

1. Реєстрація та вхід користувачів:

- Bcrypt: Використовується для хешування паролів користувачів перед зберіганням у базі даних.
- Connect-flash: Використовується для відображення повідомлень про успішну реєстрацію або помилки при вході

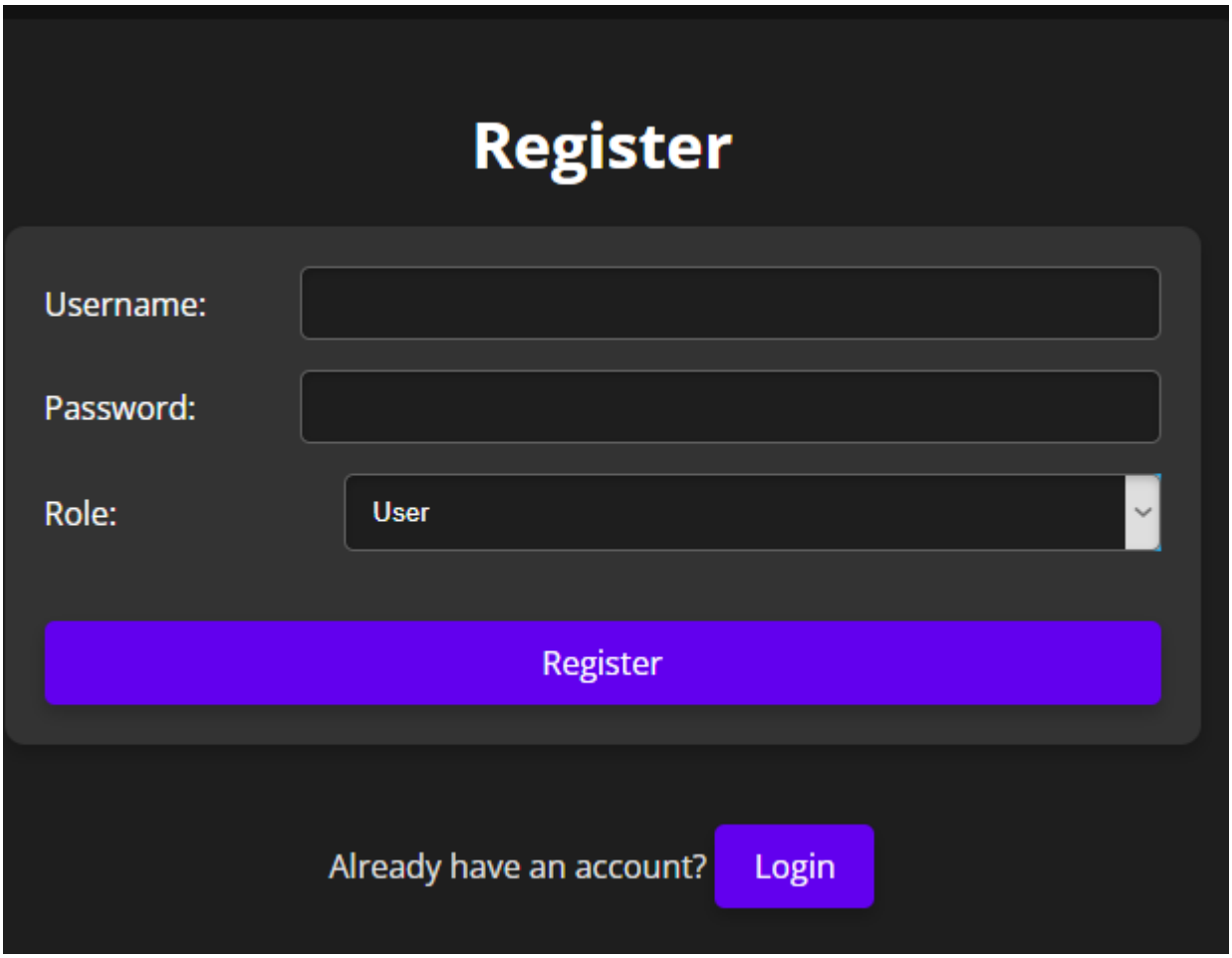


Рисунок 3.3 Вікно реєстрації користувача

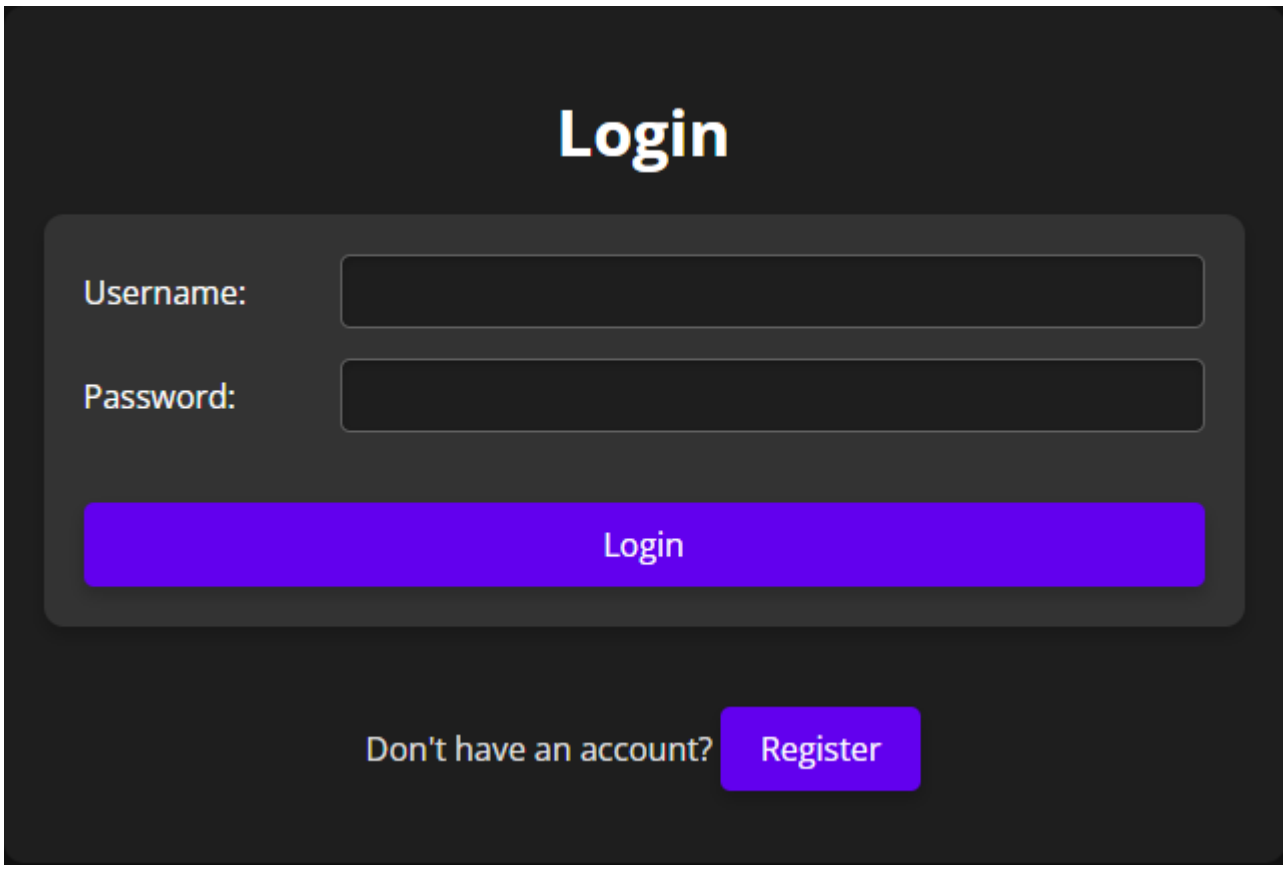


Рисунок 3.4 Вікно логіну до додатку

2. Збір даних:

- API для сенсорів: Сервер обробляє дані, що надходять від сенсорів, та зберігає їх в базі даних.

```
app.post('/api/measurements', (req, res) => {
    const { sensor_id, value } = req.body;
    const timestamp = new Date().toISOString();
    db.run("INSERT INTO Measurements (sensor_id, value, timestamp) VALUES (?, ?, ?)", [sen
        if (err) {
            res.status(500).send("Error inserting measurement");
        } else {
            res.status(201).send({ id: this.lastID });
        }
    });
});

app.get('/api/measurements', (req, res) => {
    db.all("SELECT * FROM Measurements", [], (err, rows) => {
        if (err) {
            res.status(500).send("Error fetching measurements");
        } else {
            res.status(200).json(rows);
        }
    });
});

app.listen(3000, () => {
    console.log('Server is running on port 3000');
});
```

Рисунок 3.5 Отримання показників сенсорів через API

3. Візуалізація даних:

- Графіки та діаграми: Дані вимірювань відображаються у вигляді графіків та діаграм для зручного аналізу.
- Сторінки моніторингу: Веб-додаток надає інтерфейс для перегляду поточних даних про умови в теплиці.

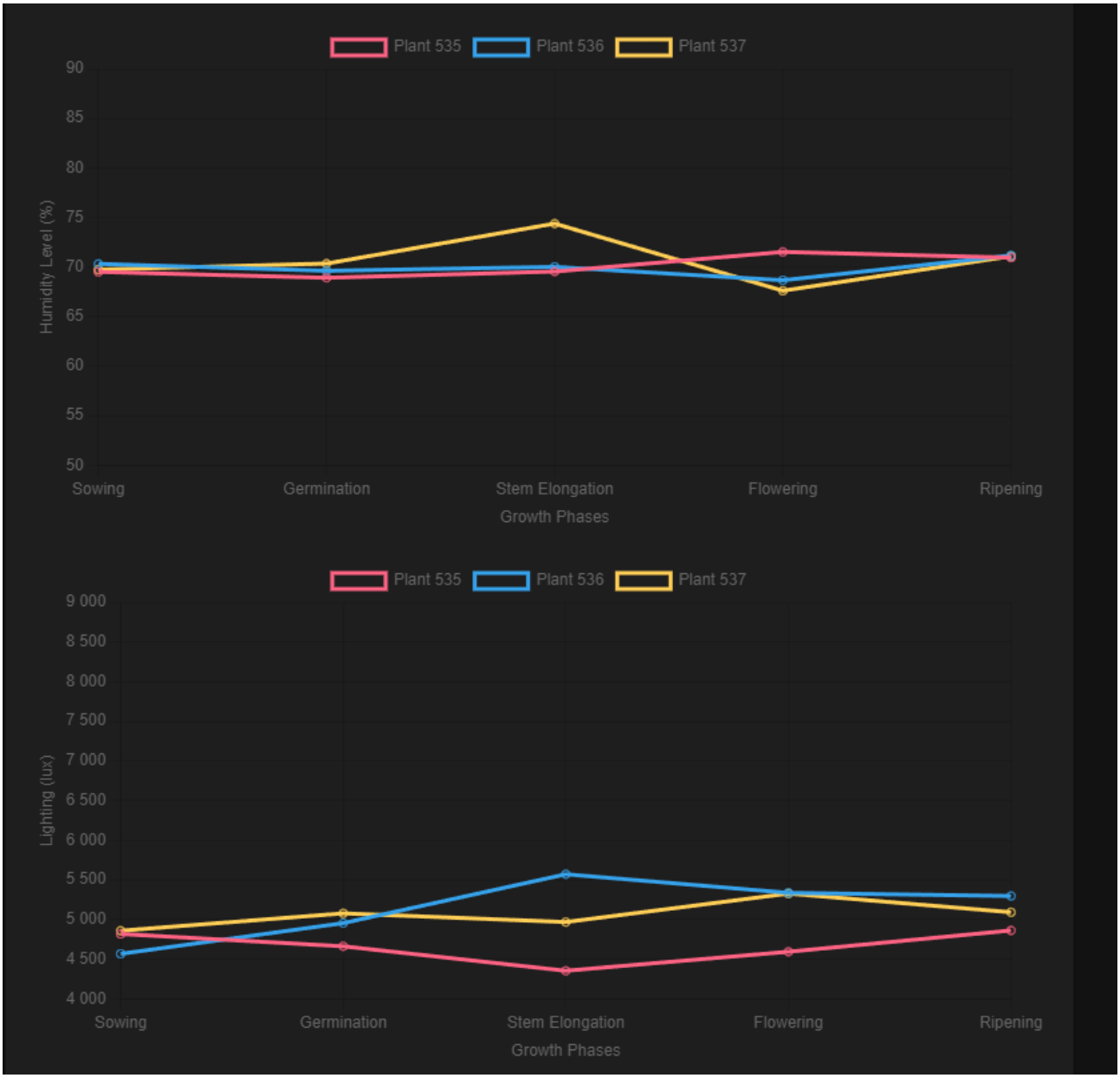


Рисунок 3.6 Візуалізація показників даних

4. Автоматизація:

- Скрипти для автоматичного контролю: Автоматизація дій, таких як полив, або регулювання освітлення, на основі зібраних даних

```
function checkAndPerformControlActions(sensor_id, value) {
    db.get("SELECT name FROM Sensors WHERE id = ?", [sensor_id], (err, row)
        if (err || !row) return;
        const sensorName = row.name;
        if (sensorName.includes("Moisture") && value < 30) {
            performAction("Watering plants");
        } else if (sensorName.includes("Light") && value < 200) {
            performAction("Turning on lights");
        }
    });
}
```

Рисунок 3.7 Скрипт для автоматичного регулювання світла

Шаблон роботи веб-додатку

Система використовує MVC. MVC – це патерн проектування програмного забезпечення, який дозволяє розділити логіку програми на окремі компоненти.

- Модель: Містить дані та логіку їх обробки. Веб-додаток взаємодіє з базою даних через модель.
- Представлення: Відповідає за відображення даних користувача. Шаблонізатор генерує динамічні веб-сторінки.
- Контролер: Обробляє запити користувачів, взаємодіє з моделлю, та оновлює представлення.

Робота програми

Користувач заходить у веб-додаток, реєструється або входить в систему під існуючим користувачем, переглядає поточні умови в теплиці на сторінці моніторингу, аналізує дані на графіках та приймає рішення, про необхідність зміни умов або вмикає автоматизацію, для підтримання оптимальних умов.

3.3 Візуалізація роботи додатку

При відкритті додатку, користувач попадає на сторінку реєстрації

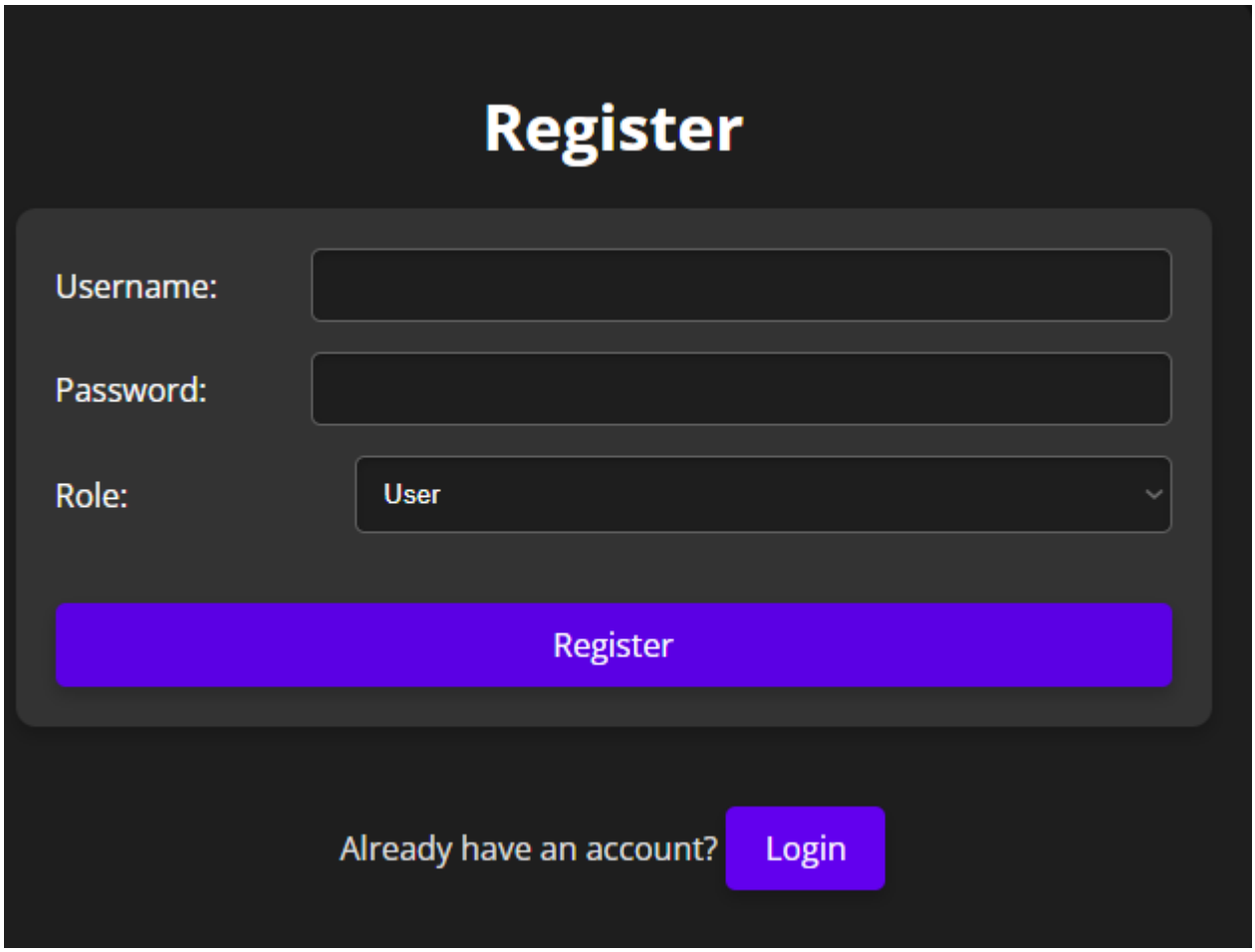


Рисунок 3.8 Вікно реєстрації

Після успішної реєстрації, користувач переходить до сторінки логіну, вводить дані та попадає до основної сторінки.

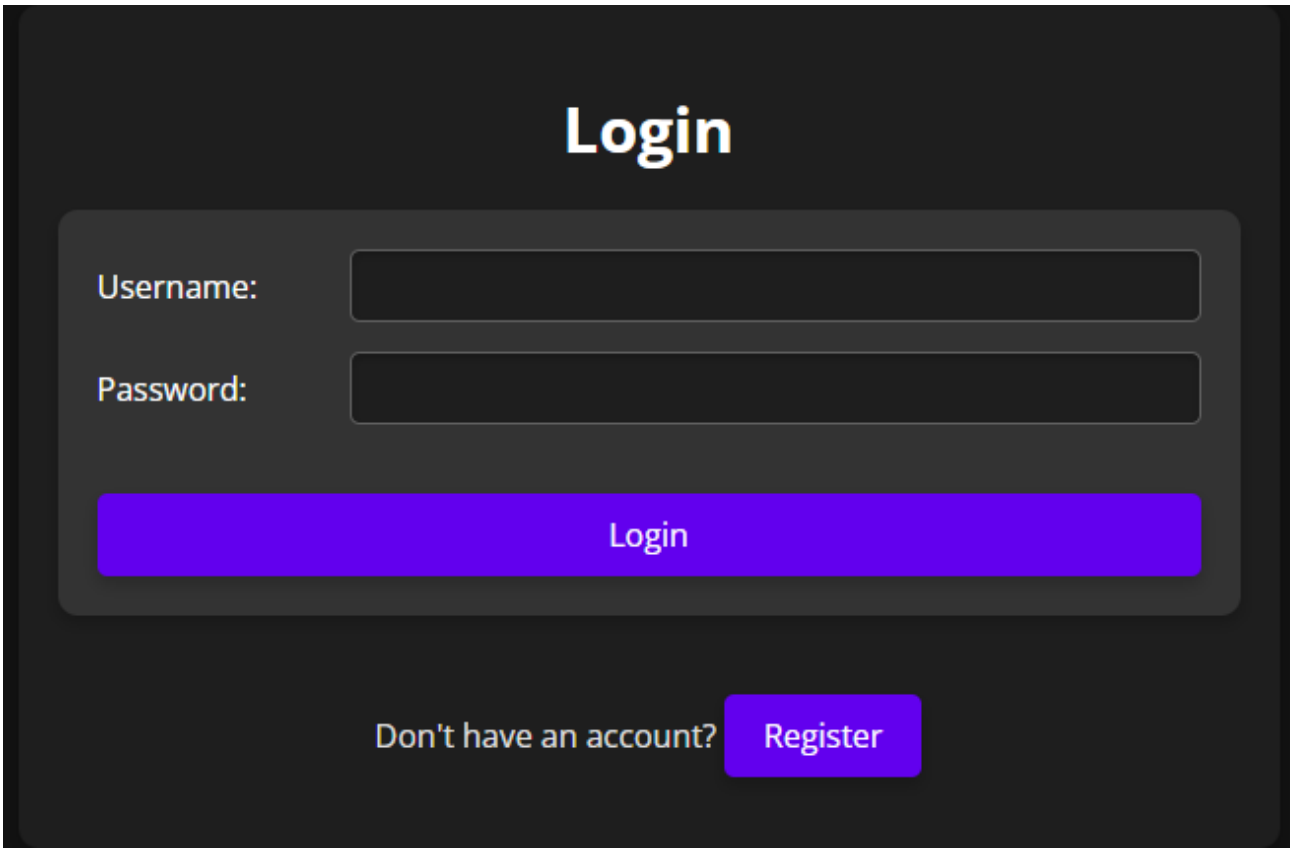


Рисунок 3.9 Вікно логіну

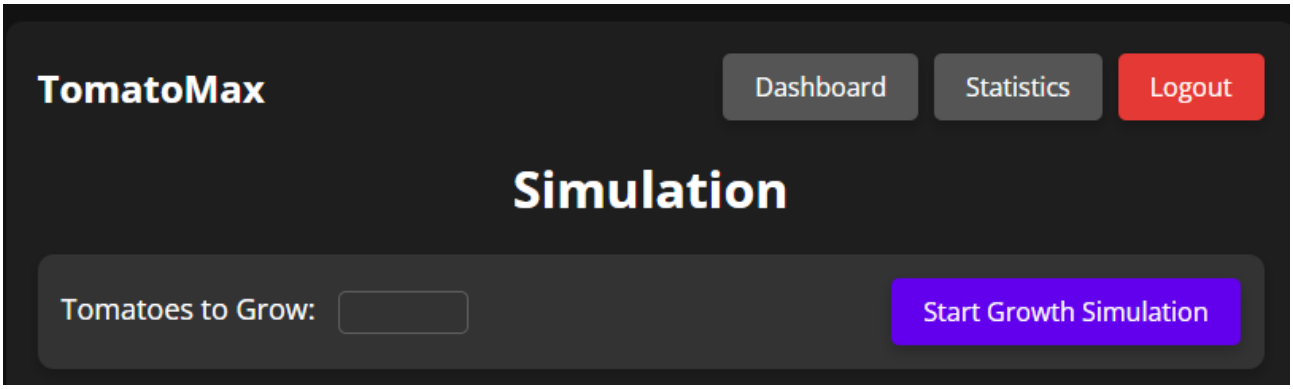


Рисунок 3.10 Основна сторінка

Після того як користувач успішно зареєструвався, користувач може проводити симуляцію росту наших агрокультур, у нашому випадку була обрана одна агрокультура – Помідор. На основній сторінці може ввести значення від 1-го до 5-ти та запустити симуляцію росту наших рослин.

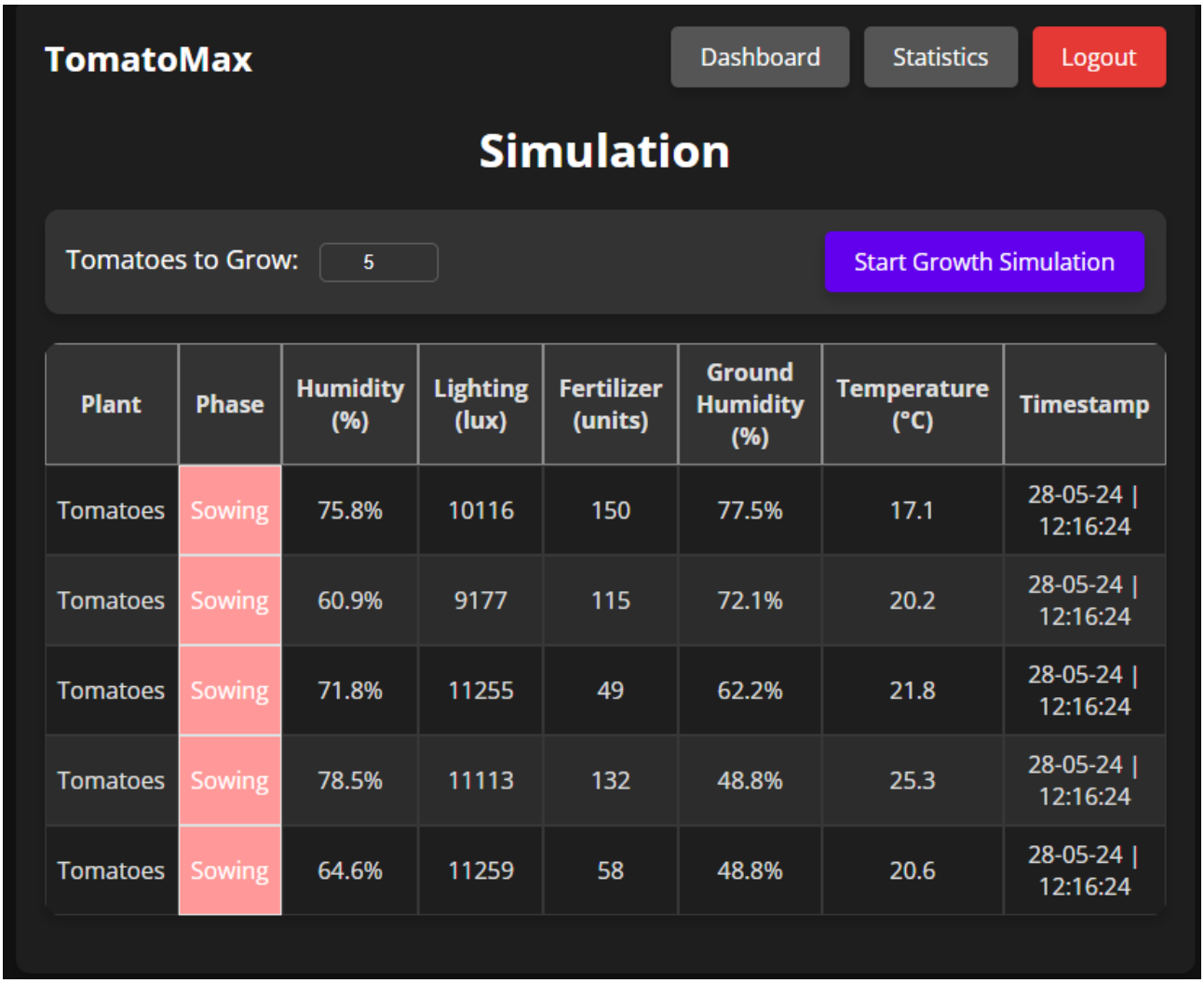


Рисунок 3.11 Основна сторінка, обрання кількості рослин для симуляції

Проведення самої симуляції.

Під час симуляції, користувачеві доступно для перегляду назва рослини, саме помідори, на далі поточна фаза росту, загалом їх маємо 5: Sowing, Germination, Stem elongation, Flowering, Ripening. Також можна переглянути показники, для поточної фази, а саме: Вологість повітря, освітленість, добрива (За методикою суміші NKP, натрій-калій фосфорного добрива), вологість грунту та температуру. Також можемо переглядати час початку симуляції.

Також на основній сторінці нам доступні наступні кнопки Statistic та кнопка виходу з облікового запису Logout

В вкладці статистика, ми можемо переглядати лінійні графіки, на яких видно середнє значення обраного параметру, на кожній фазі росту, а також корегування даного параметру, протягом всіх 5-ти фаз.

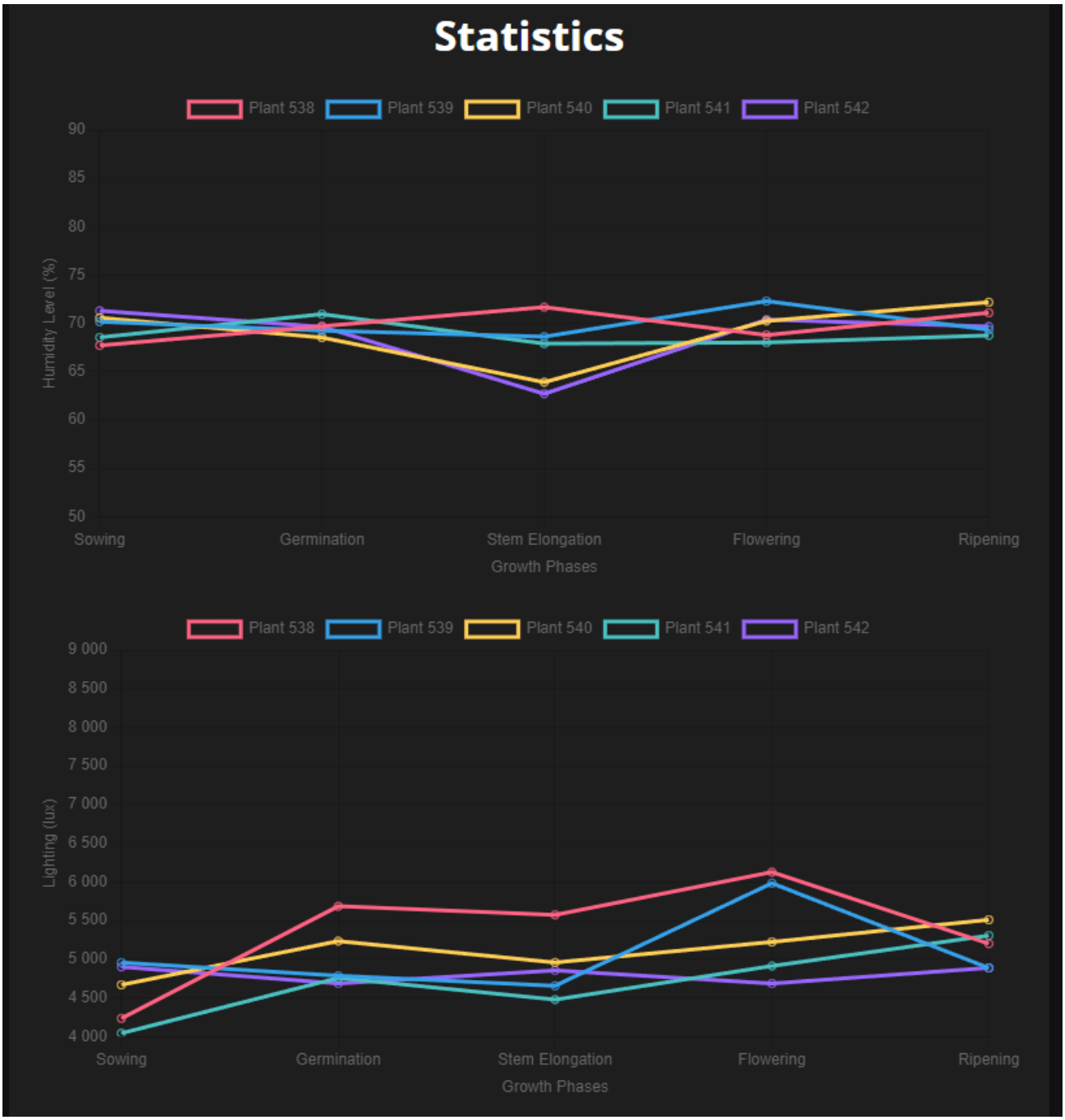


Рисунок 3.12 Візуалізація отриманих даних

Функціональна схема роботи програми:

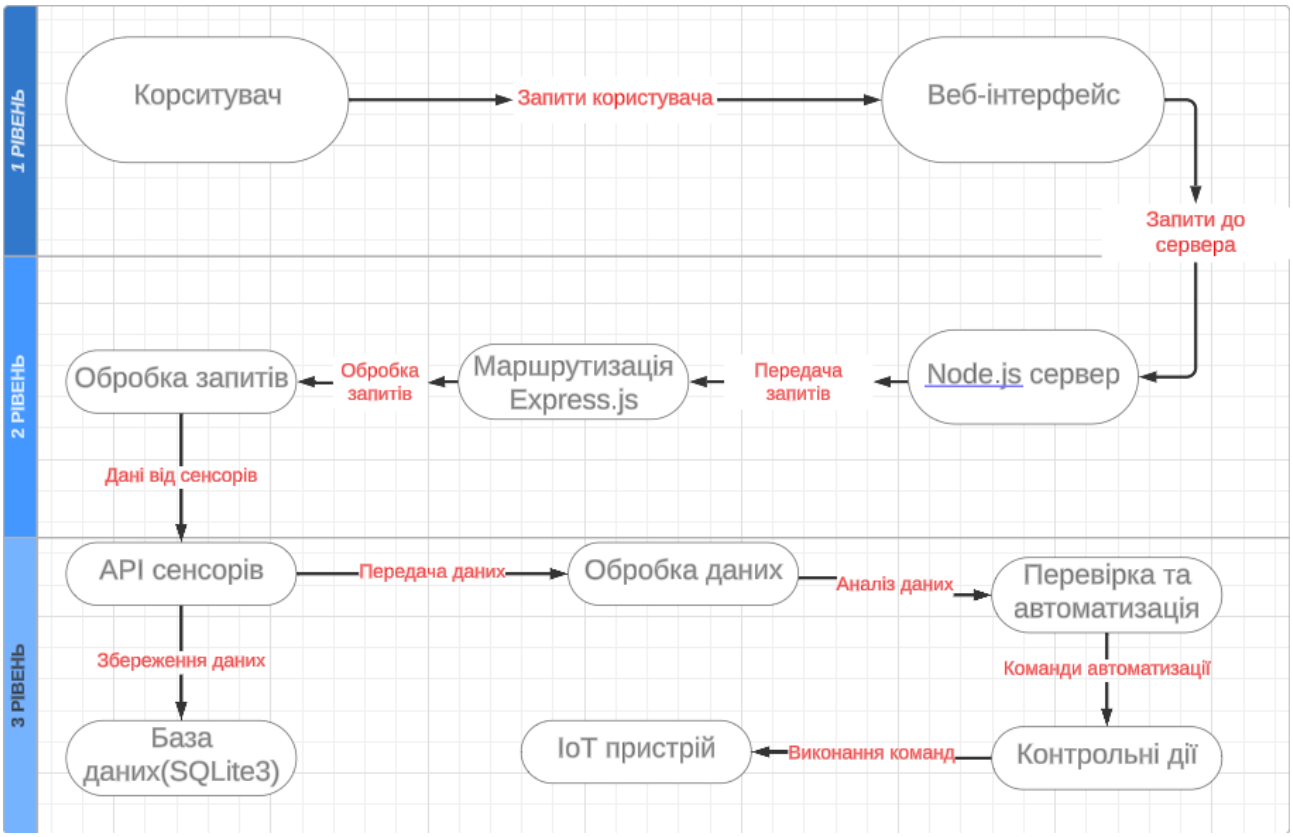


Рисунок 3.13 Функціональна схема

3.4 Алгоритм роботи додатку

Алгоритм роботи додатка можна описати наступним чином: Він включає в себе кілька основних етапів: Взаємодія користувача з інтерфейсом, обробка запитів, збір та обробка даних із сенсорів, а також автоматизація дій на основі отримання даних.

Алгоритм роботи веб-додатку забезпечує повний цикл взаємодії користувача до автоматизації дій на основі аналізу даних, отриманих з сенсорів. Такий підхід дозволяє створити ефективну IoT систему для моніторингу та контролю теплиць, що підвищує врожайність та якість рослин.

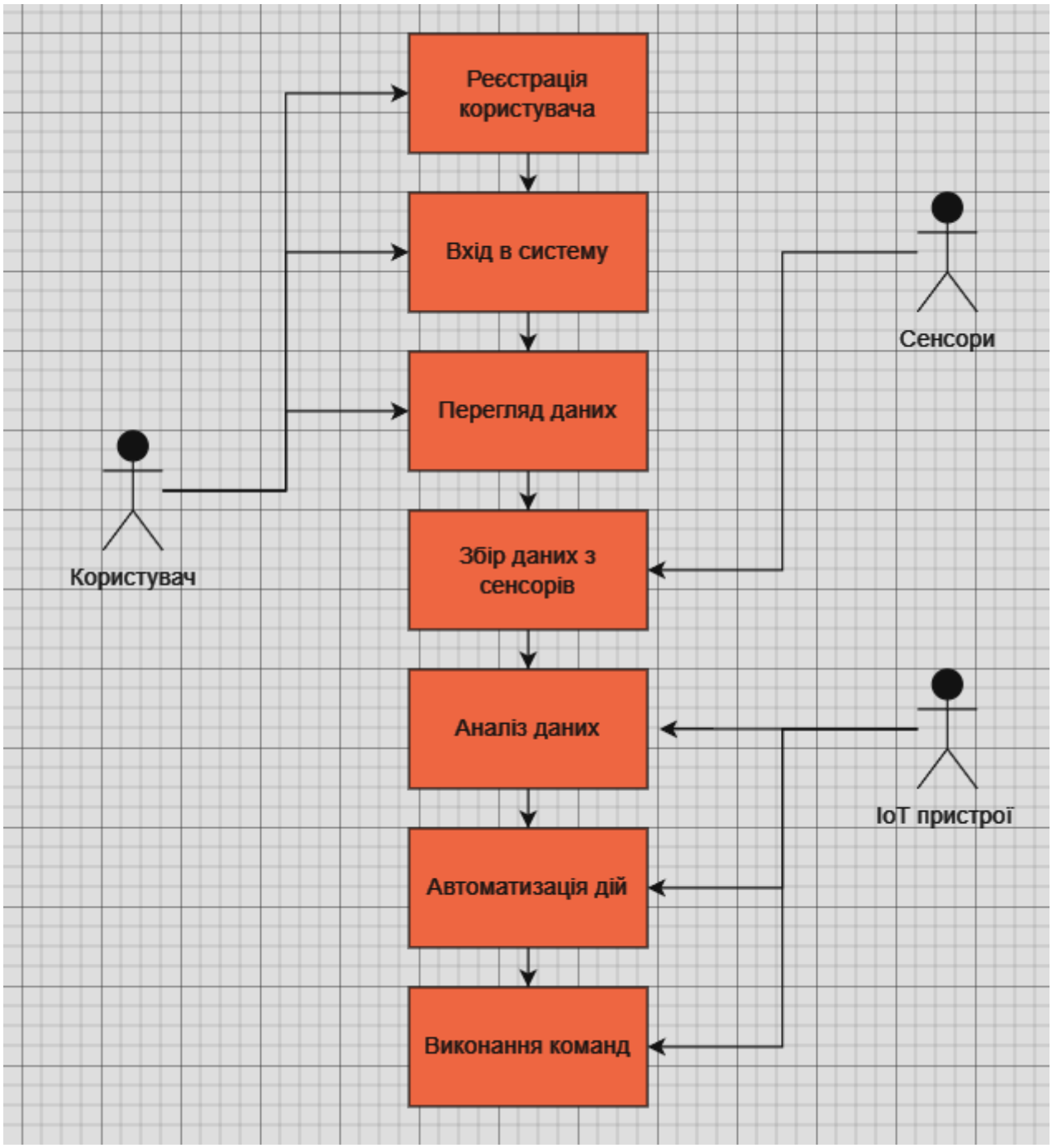


Рисунок 3.14 use-case діаграма

Основні модулі роботи додакту:

1. Головна сторінка додатку

Головна сторінка додатку відображає інтерфейс користувача, де користувач може переглядати інформацію про стан рослин

2. Сторінка статистики

На цій сторінці користувач може дізнатися більш розширену та детальну інформацію стосовно стану рослин, та даних сенсорів

3. Закриття додатку

Користувач закриває додаток, якщо він більше не потрібен, після закриття додаток завершує свою роботу та звільнює використані ресурси.

Додаткові модулі:

4. Сторінка входу та реєстрації

Сторінка яка потрібна для створення облікового запису користувача, або входу вже під існуючим обліковим записом.

5. Адміністративна панель

Сторінка адміністратора, де можна керувати налаштуваннями додатку та моніторити його роботу

3.5 Висновок до третього розділу

По ходу розділу ми розглянули основні етапи алгоритму роботи додатків, включаючи взаємодію користувача з інтерфейсом, обробку запитів сервером, збір та обробку даних з сенсорів, а також автоматизацію дій на основі отриманих даних.

Розробили трьохрівневу функціональну схему та Use-Case діаграму, які забезпечили нам візуальне представлення взаємодії між різними компонентами системи та основними сценаріями використання. Це дозволило зрозуміти повний цикл роботи додатку, від взаємодії користувача, до автоматизації дії.

Таким чином, розроблений веб-додаток забезпечує ефективний моніторинг та контроль динаміки росту тепличних культур, що сприяє підвищенню врожайності та якості продукції, а також оптимізації витрат на вирощування тепличних культур.

ВИСНОВКИ

У ході виконання бакалаврської роботи, було створено IoT систему для моніторингу та сезонного контролю динаміки росту тепличних рослин, яка включає в себе веб додаток для зручної взаємодії з користувачами. Система дозволяє ефективно відстежувати умови в теплиці, в режимі реального часу, що забезпечує своєчасне виявлення та реагування на зміни. Веб-додаток має інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам легко взаємодіяти з системою, переглядати дані з сенсорів та налаштовувати параметри автоматизації.

Інтеграція IoT пристроїв дозволила зібрати дані про умови теплиці і зберегти їх у базі даних для подальшого аналізу. Система автоматично аналізує отриманні дані і приймає рішення про необхідність виконання автоматичних дій, таких як полив або регулювання освітлення. Це значно знижує потребу в постійному ручному контролі та підвищує ефективність управління теплицею.

Розроблена система дозволяє підвищити врожайність і якість продукції, а також знизити витрати на обслуговування теплиць. Завдяки цьому, сільськогосподарські підприємства можуть стати більш конкурентно-спроможними та ефективними. Загалом, створена IoT система є ефективним інструментом для оптимізації процесів вирощування рослин в теплицях.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Інтернет речей: Принципи та парадигми / Р. Буя, А. В. Дастджерді. – Київ: Видавництво «Техніка», 2016.
2. Building the Internet of Things: Implement New Business Models, Disrupt Competitors, Transform Your Industry / M. Kranz. – Wiley, 2016.
3. Шаблони проектування Node.js / М. Касчіаро, Л. Мамміно. – Харків: Фоліо, 2016.
4. Express у дії: написання, створення та тестування додатків на Node.js / Е. Хан. – Київ: Видавництво «Старий Лев», 2016.
5. Вступ до Node.js / М. Кіслінг. – Київ: Видавництво «Наукова думка», 2012.
6. JavaScript: The Good Parts / D. Crockford. – O'Reilly Media, 2008.
7. JavaScript: Вичерпний довідник / Д. Фланаган. – Київ: Видавництво «КМ-Букс», 2011.
8. Learning IoT Development / P. Waher. – Packt Publishing, 2015.
9. JavaScript та JQuery: Інтерактивна фронт-енд розробка / Дж. Дакетт. – Львів: Видавництво «Магнолія», 2014.
10. Розробка повного стеку JavaScript з MEAN / К. Іріг, А. Бретц. – Київ: Видавництво «Альфа», 2014.
11. Eloquent JavaScript: A Modern Introduction to Programming / M. Haverbeke. – No Starch Press, 2018.

12. Вивчення SQLite для iOS / Г. Да Роша. – Київ: Видавництво «Молодь», 2015.
13. The Definitive Guide to SQLite / G. Allen, M. Owens. – Apress, 2010.
14. SQL для чайників / А. Тейлор. – Львів: Видавництво «Літера», 2013.
15. JavaScript Patterns: Build Better Applications with Coding and Design Patterns / S. Stefanov. – O'Reilly Media, 2010.
16. MongoDB: The Definitive Guide / K. Chodorow, M. Dirolf. – O'Reilly Media, 2013.
17. Програмування Інтернету речей: вступ до побудови інтегрованих IoT рішень / А. Кінг. – Львів: Видавництво «Старий Лев», 2017.
18. Побудова IoT проектів з ESP32 / А. Сантос, Р. Феррейра, М. Пімента. – Київ: Видавництво «Молодь», 2020.
19. Hands-On Internet of Things with MQTT: Build connected IoT devices with MQTT / T. Pulver. – Packt Publishing, 2019.
20. Інтернет речей: як ваша компанія може використовувати IoT для перемоги в економіці результатів / Б. Сінклер. – Львів: Видавництво «АртЕк», 2017.
21. Основи IoT та дизайну носимих технологій / Х. Раад. – Київ: Видавництво «Знання», 2020.
22. IoT Solutions in Microsoft's Azure IoT Suite: Data Acquisition and Analysis in the Real World / S. Klein. – Apress, 2017.
23. Проектування даноінтенсивних додатків / М. Клеппманн. – Київ: Видавництво «Наукова думка», 2017.
24. Практичний Node.js: Побудова масштабованих веб-додатків / А. Мардан. – Львів: Видавництво «Магнолія», 2014.
25. Node.js 8 правильний шлях: Практичний, серверний JavaScript, що масштабується / Дж. Р. Вілсон. – Київ: Видавництво «Альфа», 2018.
26. Інтернет речей з Raspberry Pi та Arduino / Р. Сінгх, А. Гелот. – Харків: Фоліо, 2019.
27. Вивчення розробки веб-додатків / С. Пуревал. – Київ: Видавництво «Техніка», 2014.

28. Clean Code: A Handbook of Agile Software Craftsmanship / R. C. Martin. – Prentice Hall, 2008.
29. Довідник по Bash / А. Роббінс. – Харків: Фоліо, 2016.
30. JavaScript: Добрі частини / Д. Крокфорд. – Львів: Видавництво «Рік-М», 2008.

ДОДАТОК А – КОД ПРОГРАМИ

Головна сторінка

```
const express = require("express");
const path = require("path");
const session = require("express-session");
const flash = require("connect-flash");
const bcrypt = require("bcryptjs");
const sqlite3 = require("sqlite3").verbose();
const app = express();
const PORT = process.env.PORT || 3000;

// Middleware setup
app.set("view engine", "ejs");
app.use(express.static(path.join(__dirname, "public")));
app.use(express.urlencoded({ extended: true }));
app.use(express.json());
app.use(
  session({
    secret: "secret",
    resave: true,
    saveUninitialized: true,
  }),
```

```
);
app.use(flash());

// Database setup
const db = new sqlite3.Database("growthData.db", (err) => {
    if (err) {
        console.error(err.message);
    }
    console.log("Connected to the growthData database.");
});

db.serialize(() => {
    db.run(`CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE,
        password TEXT,
        role TEXT
    )`);
    db.run(`CREATE TABLE IF NOT EXISTS growthData (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        plant TEXT,
        phase TEXT,
        humidity TEXT,
        lighting TEXT,
        fertilizer TEXT,
        ground_humidity REAL,
        temperature REAL,
        timestamp TEXT
    )`);
```

```
    db.run(`CREATE TABLE IF NOT EXISTS growthMetrics (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        plant_id INTEGER,
        phase TEXT,
        humidity REAL,
        lighting REAL,
        fertilizer REAL,
        ground_humidity REAL,
        temperature REAL,
        timestamp TEXT,
        FOREIGN KEY(plant_id) REFERENCES growthData(id)
    )`);
    db.run(`CREATE TABLE IF NOT EXISTS growth_phase_averages (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        plant_id INTEGER,
        phase TEXT,
        avg_humidity REAL,
        avg_lighting REAL,
        avg_fertilizer REAL,
        avg_ground_humidity REAL,
        avg_temperature REAL,
        timestamp TEXT
    )`);
});

// Authentication middleware
function ensureAuthenticated(req, res, next) {
    if (req.session.user) {
        return next();
```

```
    }
    res.redirect("/login");
}

function ensureAdmin(req, res, next) {
    if (req.session.user && req.session.user.role === "admin") {
        return next();
    }
    res.redirect("/");
}

// User management
const loadUser = (username, callback) => {
    db.get("SELECT * FROM users WHERE username = ?", [username], (err, row) =>
{
        if (err) {
            callback(err);
        } else {
            callback(null, row);
        }
    });
};

const saveUser = (user, callback) => {
    db.run(
        "INSERT INTO users (username, password, role) VALUES (?, ?, ?)",
        [user.username, user.password, user.role],
        (err) => {
            callback(err);
```

```
        },
    );
};

// Utility function to format timestamp
const formatTimestamp = (timestamp) => {
    const date = new Date(timestamp);
    const day = String(date.getDate()).padStart(2, "0");
    const month = String(date.getMonth() + 1).padStart(2, "0");
    const year = String(date.getFullYear()).slice(-2);
    const hours = String(date.getHours()).padStart(2, "0");
    const minutes = String(date.getMinutes()).padStart(2, "0");
    const seconds = String(date.getSeconds()).padStart(2, "0");
    return `${day}-${month}-${year} | ${hours}:${minutes}:${seconds}`;
};

// Routes
app.get("/", ensureAuthenticated, (req, res) => {
    res.render("index", { user: req.session.user });
});

app.get("/dashboard", ensureAuthenticated, (req, res) => {
    res.render("dashboard", { user: req.session.user });
});

app.get("/statistics", ensureAuthenticated, (req, res) => {
    res.render("statistics", { user: req.session.user });
});
```

```
app.get("/login", (req, res) => {
   res.render("login", { message: req.flash("message") });
});

app.post("/login", (req, res) => {
   const { username, password } = req.body;
   loadUser(username, (err, user) => {
      if (user && bcrypt.compareSync(password, user.password)) {
         req.session.user = user;
         res.redirect("/");
      } else {
         req.flash("message", "Invalid credentials");
         res.redirect("/login");
      }
   });
});

app.get("/logout", (req, res) => {
   req.session.destroy();
   res.redirect("/login");
});

app.get("/register", (req, res) => {
   if (req.session.user) {
      res.redirect("/");
   } else {
      res.render("register", { message: req.flash("message") });
   }
});
```

```
app.post("/register", (req, res) => {
    const { username, password, role } = req.body;
    loadUser(username, (err, user) => {
        if (user) {
            req.flash("message", "Username already exists");
            res.redirect("/register");
        } else {
            const hashedPassword = bcrypt.hashSync(password, 10);
            const newUser = { username, password: hashedPassword, role };
            saveUser(newUser, (err) => {
                if (err) {
                    req.flash("message", "Error creating user");
                } else {
                    req.flash(
                        "message",
                        "Registration successful, please log in",
                    );
                }
                res.redirect("/login");
            });
        }
    });
});

app.get("/growth_phases", ensureAuthenticated, (req, res) => {
    res.render("index", { user: req.session.user });
});
```

```
app.get("/api/growth_metrics", (req, res) => {
  db.all(
    "SELECT plant_id, phase, AVG(humidity) as avg_humidity, AVG(lighting) as avg_lighting, AVG(fertilizer) as avg_fertilizer, AVG(ground_humidity) as avg_ground_humidity, AVG(temperature) as avg_temperature FROM growthMetrics GROUP BY plant_id, phase",
    [],
    (err, rows) => {
      if (err) {
        res.status(500).json({ error: err.message });
        return;
      }
      res.json(rows);
    },
  );
});

app.get("/api/growth_data", (req, res) => {
  db.all(
    "SELECT * FROM growthData WHERE plant = 'tomatoes'",
    [],
    (err, rows) => {
      if (err) {
        throw err;
      }
      res.json({ tomatoes: rows });
    },
  );
});
```

```
app.post("/api/start_growth", (req, res) => {
    const { tomatoesCount } = req.body;
    const plantCount = Math.min(parseInt(tomatoesCount), 5);
    const phases = [
        "Sowing",
        "Germination",
        "Stem Elongation",
        "Flowering",
        "Ripening",
    ];
    const baseDuration = 20 * 1000; // 20 seconds
    const startTime = Date.now();
    let finished = false;

    // Clear existing data from tables
    db.run("DELETE FROM growthMetrics", [], (err) => {
        if (err) {
            return res.status(500).json({ error: err.message });
        }
    });

    db.run("DELETE FROM growth_phase_averages", [], (err) => {
        if (err) {
            return res.status(500).json({ error: err.message });
        }
    });

    db.run("DELETE FROM growthData", [], (err) => {
        if (err) {
```

```
            return console.error(err.message);
        }
    });

    setTimeout(() => {
        initializeGrowthData();
        simulateGrowthData();
    }, 1000);

    function initializeGrowthData() {
        for (let i = 0; i < plantCount; i++) {
            const humidity = (Math.random() * 20 + 60).toFixed(1); // 60.0% to 80.0%
            const lighting = Math.round(Math.random() * 3000 + 6500 + 2000); // 8500 to
11500 lux
            const fertilizer = Math.round(Math.random() * 140 + 20); // 20 to 160 units
            const groundHumidity = (Math.random() * 60 + 30).toFixed(1); // 30.0% to
90.0%
            const temperature = (Math.random() * 20 + 10).toFixed(1); // 10.0°C to 30.0°C
            const timestamp = formatTimestamp(new Date());

            db.run(
                "INSERT INTO growthData (plant, phase, humidity, lighting, fertilizer,
ground_humidity, temperature, timestamp) VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
                [
                    "tomatoes",
                    phases[0],
                    humidity,
                    lighting,
                    fertilizer,
                    groundHumidity,
```

```
            temperature,
            timestamp,
          ],
        );
      }
    }

    function simulateGrowthData() {
      const intervalId = setInterval(() => {
        if (finished) {
          clearInterval(intervalId);
          return;
        }

        const currentTime = Date.now();

        db.all(
          "SELECT * FROM growthData WHERE plant = 'tomatoes'",
          [],
          (err, rows) => {
            if (err) {
              return console.error(err.message);
            }

            let allFinished = true;

            rows.forEach((row) => {
              const randomFactor = Math.random() * 2000 + 4000; // randomize
between 4 to 6 seconds
```

```
            const timeElapsed =
              (currentTime - startTime + randomFactor) %
              (baseDuration * phases.length);
            const phaseIndex = Math.floor(
              timeElapsed / baseDuration,
            );
            const currentPhase =
              phases[Math.min(phaseIndex, phases.length - 1)];

            const humidity = (Math.random() * 20 + 60).toFixed(1); // 60.0% to
80.0%
            const lighting = Math.round(
              Math.random() * 6000 + 2000,
            ); // 6000 to 8000 lux
            const fertilizer = Math.round(Math.random() * 140 + 20); // 20 to 160
units
            const groundHumidity = (
              Math.random() * 60 +
              30
            ).toFixed(1); // 30.0% to 90.0%
            const temperature = (Math.random() * 20 + 10).toFixed(
              1,
            ); // 10.0°C to 30.0°C
            const timestamp = formatTimestamp(new Date());

            db.run(
              "INSERT INTO growthMetrics (plant_id, phase, humidity, lighting,
fertilizer, ground_humidity, temperature, timestamp) VALUES (?, ?, ?, ?, ?, ?, ?, ?)",
              [
                row.id,
```

```
                currentPhase,
                humidity,
                lighting,
                fertilizer,
                groundHumidity,
                temperature,
                timestamp,
            ],
        );

        db.run(
            "UPDATE growthData SET phase = ?, humidity = ?, lighting = ?, fertilizer = ?, ground_humidity = ?, temperature = ?, timestamp = ? WHERE id = ?",
            [
                currentPhase,
                humidity,
                lighting,
                fertilizer,
                groundHumidity,
                temperature,
                timestamp,
                row.id,
            ],
        );

        if (currentPhase !== phases[phases.length - 1]) {
            allFinished = false;
        }
    });
```

```
                if (allFinished) {
                    setTimeout(() => {
                        finished = true;
                    }, 5000);
                }
            },
        );
    }, 1000);
}

res.json({ message: "Growth simulation started" });
});

// Calculate and store averages
function calculateAndStoreAverages() {
    const phases = [
        "Sowing",
        "Germination",
        "Stem Elongation",
        "Flowering",
        "Ripening",
    ];

    phases.forEach((phase) => {
        db.all(
            `SELECT
                plant_id,
                AVG(humidity) AS avg_humidity,
```

```
    AVG(lighting) AS avg_lighting,
    AVG(fertilizer) AS avg_fertilizer,
    AVG(ground_humidity) AS avg_ground_humidity,
    AVG(temperature) AS avg_temperature
  FROM growthMetrics
  WHERE phase = ?
  GROUP BY plant_id`,
 [phase],
 (err, rows) => {
    if (err) {
      return console.error(err.message);
    }

    rows.forEach((row) => {
      const timestamp = formatTimestamp(new Date());
      db.run(
        `INSERT INTO growth_phase_averages (plant_id, phase, avg_humidity, avg_lighting, avg_fertilizer, avg_ground_humidity, avg_temperature, timestamp)
         VALUES (?, ?, ?, ?, ?, ?, ?, ?)`,
        [
          row.plant_id,
          phase,
          row.avg_humidity,
          row.avg_lighting,
          row.avg_fertilizer,
          row.avg_ground_humidity,
          row.avg_temperature,
          timestamp,
        ],
```

```
                );
            });
        },
    );
});
}

// Call this function periodically, e.g., every 3 seconds
setInterval(calculateAndStoreAverages, 3000);

app.listen(PORT, () => {
    console.log(`Server is running on port ${PORT}`);
});
```

Сторінка статистики:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Statistics</title>
    <link rel="stylesheet" href="/css/style.css">
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <div class="container">
        <header>
```

```
        <div class="app-name">TomatoMax</div>
        <nav>
            <button onclick="location.href='/dashboard'">Dashboard</button>
            <button onclick="location.href='/'">Simulation</button>
            <button class="logout" onclick="location.href='/logout'">Logout</button>
        </nav>
    </header>

    <h1>Statistics</h1>

    <div class="form-container hidden-form">
        <form id="startGrowthForm">
            <label for="tomatoesCount">Tomatoes to Grow:</label>
            <input type="number" id="tomatoesCount" name="tomatoesCount" min="1" max="5" required>
        </form>
    </div>

    <div id="graphsContainer" style="display: flex; flex-direction: column; align-items: center; gap: 20px;">
        <canvas id="humidityChart" width="400" height="200"></canvas>
        <canvas id="lightingChart" width="400" height="200"></canvas>
        <canvas id="fertilizerChart" width="400" height="200"></canvas>
        <canvas id="groundHumidityChart" width="400" height="200"></canvas>
        <canvas id="temperatureChart" width="400" height="200"></canvas>
    </div>
</div>

<script>
```

```
    document.getElementById('startGrowthForm').addEventListener('submit', async function (event) {
        event.preventDefault();
        const tomatoesCount = document.getElementById('tomatoesCount').value;

        const response = await fetch('/api/start_growth', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({ tomatoesCount }),
        });

        const result = await response.json();
        alert(result.message);

        clearCharts();
        fetchGrowthData(); // Initial fetch
        fetchGrowthMetrics(); // Initial fetch
    });

    async function fetchGrowthData() {
        try {
            const response = await fetch('/api/growth_data');
            const data = await response.json();
            const tbody = document.getElementById('growthData');
            tbody.innerHTML = ''; // Clear existing data

            data.tomatoes.forEach(record => {
```

```
            const row = document.createElement('tr');
            row.innerHTML = `
                <td>Tomatoes</td>
                <td class="${record.phase.replace(/\s+/g, '-')}">${record.phase}</td>
                <td>${record.humidity}%</td>
                <td>${record.lighting}</td>
                <td>${record.fertilizer}</td>
                <td>${record.timestamp}</td>
            `;
            tbody.appendChild(row);
        });
    } catch (error) {
        console.error('Error fetching growth data:', error);
    }
}

let humidityChart, lightingChart, fertilizerChart, groundHumidityChart, temperatureChart;

async function fetchGrowthMetrics() {
    try {
        const response = await fetch('/api/growth_metrics');
        const data = await response.json();

        const phases = ["Sowing", "Germination", "Stem Elongation", "Flowering", "Ripening"];
        const plantData = {};

        data.forEach(record => {
            if (!plantData[record.plant_id]) {
```

```
                plantData[record.plant_id] = {
                    humidity: [],
                    lighting: [],
                    fertilizer: [],
                    groundHumidity: [],
                    temperature: []
                };
            }

            plantData[record.plant_id].humidity.push(record.avg_humidity);
            plantData[record.plant_id].lighting.push(record.avg_lighting);
            plantData[record.plant_id].fertilizer.push(record.avg_fertilizer);

plantData[record.plant_id].groundHumidity.push(record.avg_ground_humidity);
            plantData[record.plant_id].temperature.push(record.avg_temperature);
        });

        const datasetsHumidity = [];
        const datasetsLighting = [];
        const datasetsFertilizer = [];
        const datasetsGroundHumidity = [];
        const datasetsTemperature = [];
        const fixedColors = ["#FF6384", "#36A2EB", "#FFCE56", "#4BC0C0",
"#9966FF"];

        Object.keys(plantData).forEach((plantId, index) => {
            const color = fixedColors[index % fixedColors.length];
            datasetsHumidity.push({
                label: `Plant ${plantId}`,
                data: plantData[plantId].humidity,
```

```
            borderColor: color,
            backgroundColor: 'rgba(0,0,0,0)',
        });
        datasetsLighting.push({
            label: `Plant ${plantId}`,
            data: plantData[plantId].lighting,
            borderColor: color,
            backgroundColor: 'rgba(0,0,0,0)',
        });
        datasetsFertilizer.push({
            label: `Plant ${plantId}`,
            data: plantData[plantId].fertilizer,
            borderColor: color,
            backgroundColor: 'rgba(0,0,0,0)',
        });
        datasetsGroundHumidity.push({
            label: `Plant ${plantId}`,
            data: plantData[plantId].groundHumidity,
            borderColor: color,
            backgroundColor: 'rgba(0,0,0,0)',
        });
        datasetsTemperature.push({
            label: `Plant ${plantId}`,
            data: plantData[plantId].temperature,
            borderColor: color,
            backgroundColor: 'rgba(0,0,0,0)',
        });
    });
```

```
const humidityData = {
    labels: phases,
    datasets: datasetsHumidity
};

const lightingData = {
    labels: phases,
    datasets: datasetsLighting
};

const fertilizerData = {
    labels: phases,
    datasets: datasetsFertilizer
};

const groundHumidityData = {
    labels: phases,
    datasets: datasetsGroundHumidity
};

const temperatureData = {
    labels: phases,
    datasets: datasetsTemperature
};

if (humidityChart) {
    humidityChart.data = humidityData;
    humidityChart.update();
} else {
```

```
            humidityChart = new Chart(document.getElementById('humidityChart'),
{
                type: 'line',
                data: humidityData,
                options: {
                    responsive: true,
                    animation: {
                        duration: 0 // Disable animation
                    },
                    scales: {
                        x: {
                            title: {
                                display: true,
                                text: 'Growth Phases'
                            }
                        },
                        y: {
                            min: 50,
                            max: 90,
                            title: {
                                display: true,
                                text: 'Humidity Level (%)'
                            }
                        }
                    }
                }
            });
        }
```

```
if (lightingChart) {
    lightingChart.data = lightingData;
    lightingChart.update();
} else {
    lightingChart = new Chart(document.getElementById('lightingChart'), {
        type: 'line',
        data: lightingData,
        options: {
            responsive: true,
            animation: {
                duration: 0 // Disable animation
            },
            scales: {
                x: {
                    title: {
                        display: true,
                        text: 'Growth Phases'
                    }
                },
                y: {
                    min: 4000,
                    max: 9000,
                    title: {
                        display: true,
                        text: 'Lighting (lux)'
                    }
                }
            }
        }
```

```
    });
}

if (fertilizerChart) {
    fertilizerChart.data = fertilizerData;
    fertilizerChart.update();
} else {
    fertilizerChart = new Chart(document.getElementById('fertilizerChart'), {
        type: 'line',
        data: fertilizerData,
        options: {
            responsive: true,
            animation: {
                duration: 0 // Disable animation
            },
            scales: {
                x: {
                    title: {
                        display: true,
                        text: 'Growth Phases'
                    }
                },
                y: {
                    min: 0,
                    max: 200,
                    title: {
                        display: true,
                        text: 'Fertilizer (units)'
                    }
```

```
                }
            }
        }
    });
}

if (groundHumidityChart) {
    groundHumidityChart.data = groundHumidityData;
    groundHumidityChart.update();
} else {
    groundHumidityChart = new Chart(document.getElementById('groundHumidityChart'), {
        type: 'line',
        data: groundHumidityData,
        options: {
            responsive: true,
            animation: {
                duration: 0 // Disable animation
            },
            scales: {
                x: {
                    title: {
                        display: true,
                        text: 'Growth Phases'
                    }
                },
                y: {
                    min: 30,
                    max: 90,
```

```
                    title: {
                        display: true,
                        text: 'Ground Humidity (%)'
                    }
                }
            }
        }
    });
}

if (temperatureChart) {
    temperatureChart.data = temperatureData;
    temperatureChart.update();
} else {
    temperatureChart = new Chart(document.getElementById('temperatureChart'), {
        type: 'line',
        data: temperatureData,
        options: {
            responsive: true,
            animation: {
                duration: 0 // Disable animation
            },
            scales: {
                x: {
                    title: {
                        display: true,
                        text: 'Growth Phases'
                    }
```

```
                    },
                    y: {
                        min: 0,
                        max: 50,
                        title: {
                            display: true,
                            text: 'Temperature (°C)'
                        }
                    }
                }
            }
        });
    }
  } catch (error) {
    console.error('Error fetching growth metrics:', error);
  }
}

function clearCharts() {
  if (humidityChart) {
    humidityChart.destroy();
    humidityChart = null;
  }
  if (lightingChart) {
    lightingChart.destroy();
    lightingChart = null;
  }
  if (fertilizerChart) {
    fertilizerChart.destroy();
```

```
            fertilizerChart = null;
        }
        if (groundHumidityChart) {
            groundHumidityChart.destroy();
            groundHumidityChart = null;
        }
        if (temperatureChart) {
            temperatureChart.destroy();
            temperatureChart = null;
        }
    }

    setInterval(fetchGrowthData, 1000); // Fetch data every 1 second
    setInterval(fetchGrowthMetrics, 3000); // Fetch metrics every 3 seconds

    fetchGrowthData(); // Initial fetch
    fetchGrowthMetrics(); // Initial fetch
  </script>
</body>
</html>
```

Структура веб-сторінок:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Growth Phases</title>
  <link rel="stylesheet" href="/css/style.css">
```

```
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
</head>
<body>
    <div class="container">
        <header>
            <div class="app-name">TomatoMax</div>
            <nav>
                <button onclick="location.href='/dashboard'">Dashboard</button>
                <button onclick="location.href='/statistics'">Statistics</button>
                <button class="logout" onclick="location.href='/logout'">Logout</button>
            </nav>
        </header>

        <h1>Simulation</h1>

        <div class="form-container">
            <form id="startGrowthForm">
                <label for="tomatoesCount">Tomatoes to Grow:</label>
                <input type="number" id="tomatoesCount" name="tomatoesCount" min="1" max="5" required>
            </form>
            <button type="submit" form="startGrowthForm">Start Growth Simulation</button>
        </div>

        <table>
            <thead>
                <tr>
                    <th>Plant</th>
                    <th>Phase</th>
```

```
                <th>Humidity (%)</th>
                <th>Lighting (lux)</th>
                <th>Fertilizer (units)</th>
                <th>Ground Humidity (%)</th>
                <th>Temperature (°C)</th>
                <th>Timestamp</th>
            </tr>
        </thead>
        <tbody id="growthData">
            <!-- Data will be populated by client-side script -->
        </tbody>
    </table>
</div>

<script>
    document.getElementById('startGrowthForm').addEventListener('submit', async function (event) {
        event.preventDefault();
        const tomatoesCount = document.getElementById('tomatoesCount').value;

        const response = await fetch('/api/start_growth', {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json',
            },
            body: JSON.stringify({ tomatoesCount }),
        });

        const result = await response.json();
```

```
    alert(result.message);

    fetchGrowthData(); // Initial fetch
});

async function fetchGrowthData() {
    const response = await fetch('/api/growth_data');
    const data = await response.json();
    const tbody = document.getElementById('growthData');
    tbody.innerHTML = ''; // Clear existing data

    data.tomatoes.forEach(record => {
        const row = document.createElement('tr');
        row.innerHTML = `
            <td>Tomatoes</td>
            <td class="${record.phase.replace(/\s+/g, '-')}">${record.phase}</td>
            <td>${record.humidity}%</td>
            <td>${record.lighting}</td>
            <td>${record.fertilizer}</td>
            <td>${record.ground_humidity}%</td>
            <td>${record.temperature}</td>
            <td>${record.timestamp}</td>
        `;
        tbody.appendChild(row);
    });
}

// Fetch table data every 1 second
setInterval(fetchGrowthData, 1000); // Fetch data every 1 second
```

```
        fetchGrowthData(); // Initial fetch
    </script>
</body>
</html>
```

ДОДАТОК Б – ІНСТРУКЦІЯ КОРИСТОВАЧУ

При відкритті додатку, користувач попадає на сторінку реєстрації

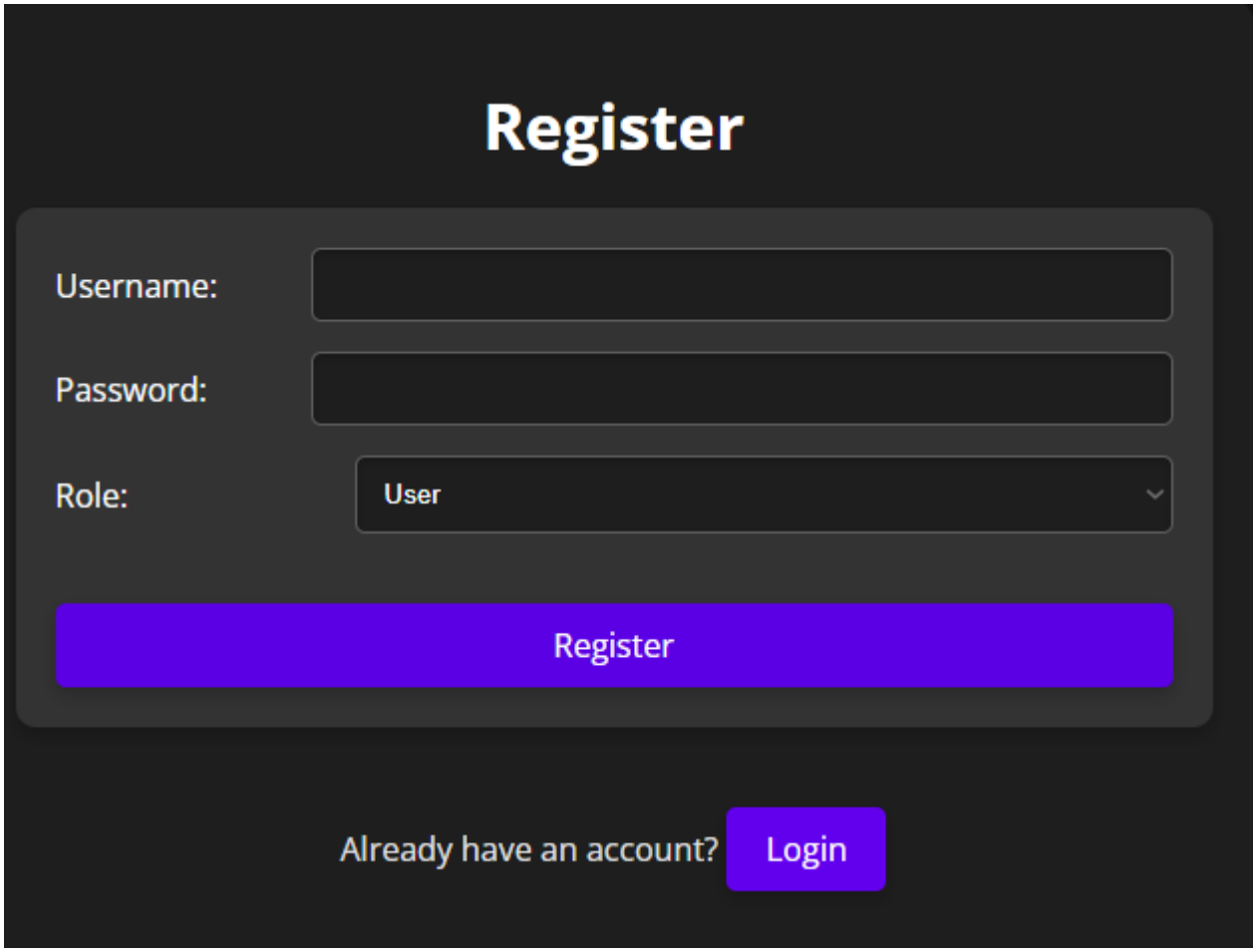


Рисунок. Вікно реєстрації

Після успішної реєстрації, користувач переходить до сторінки логіну, вводить дані та попадає до основної сторінки.

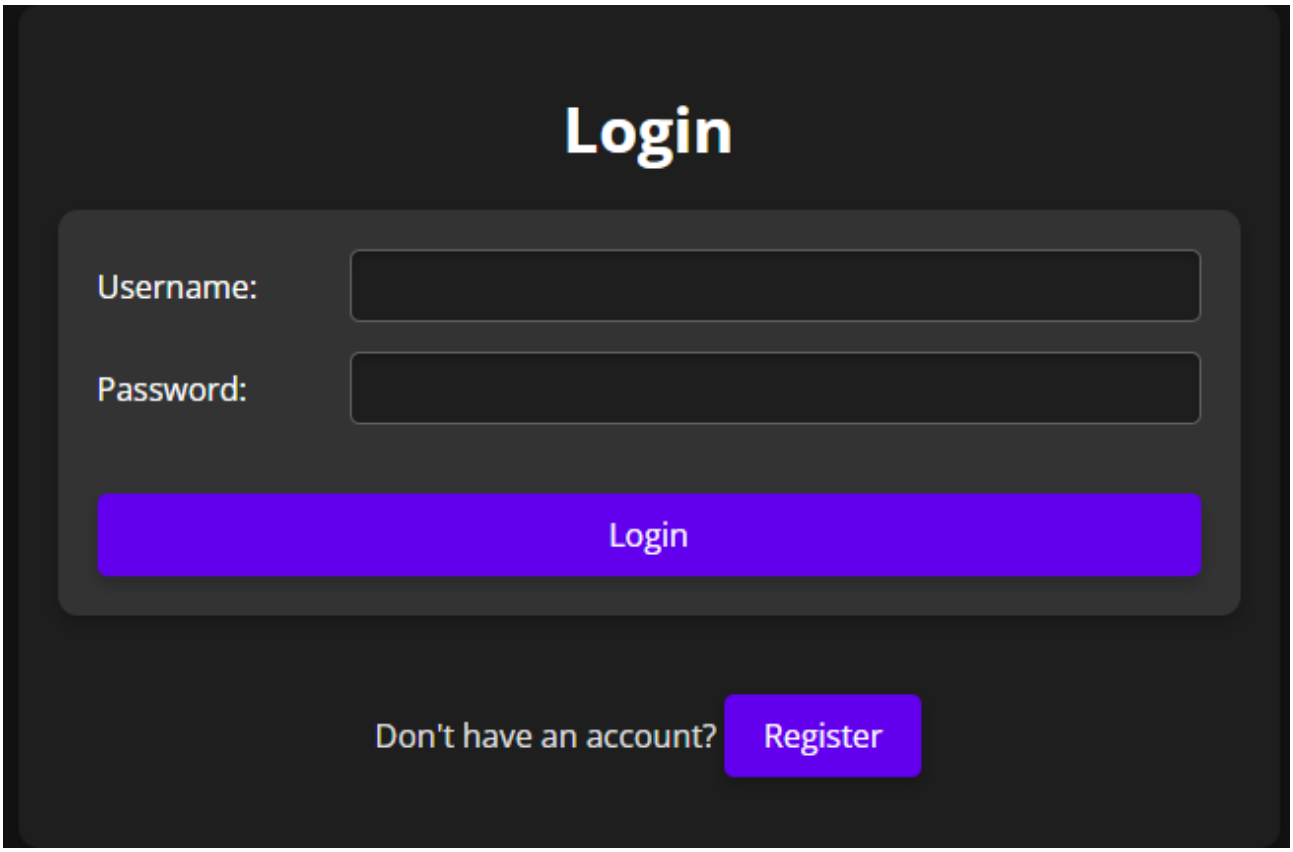


Рисунок. Вікно логіну

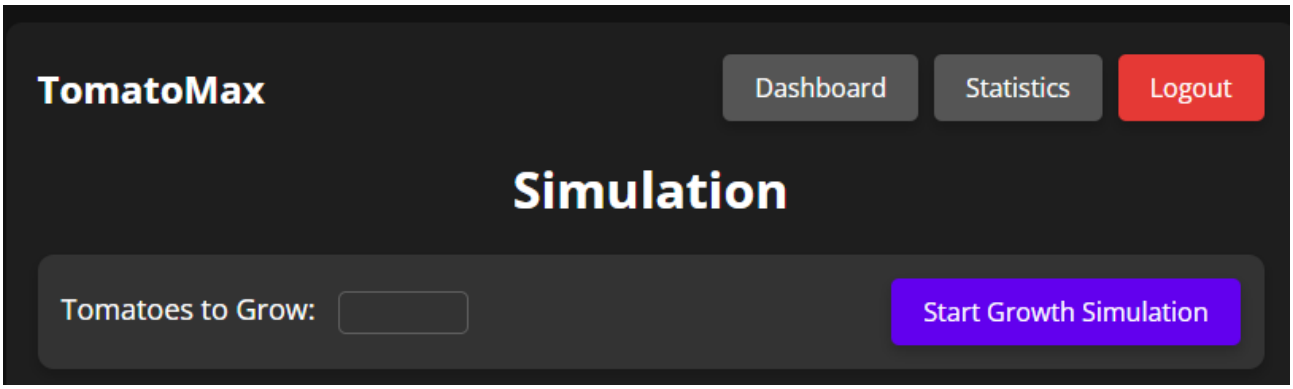


Рисунок. Основна сторінка

Після того як користувач успішно зареєструвався, користувач може проводити симуляцію росту наших агрокультур, у нашому випадку була обрана одна агрокультура – Помідор. На основній сторінці може ввести значення від 1-го до 5-ти та запустити симуляцію росту наших рослин.

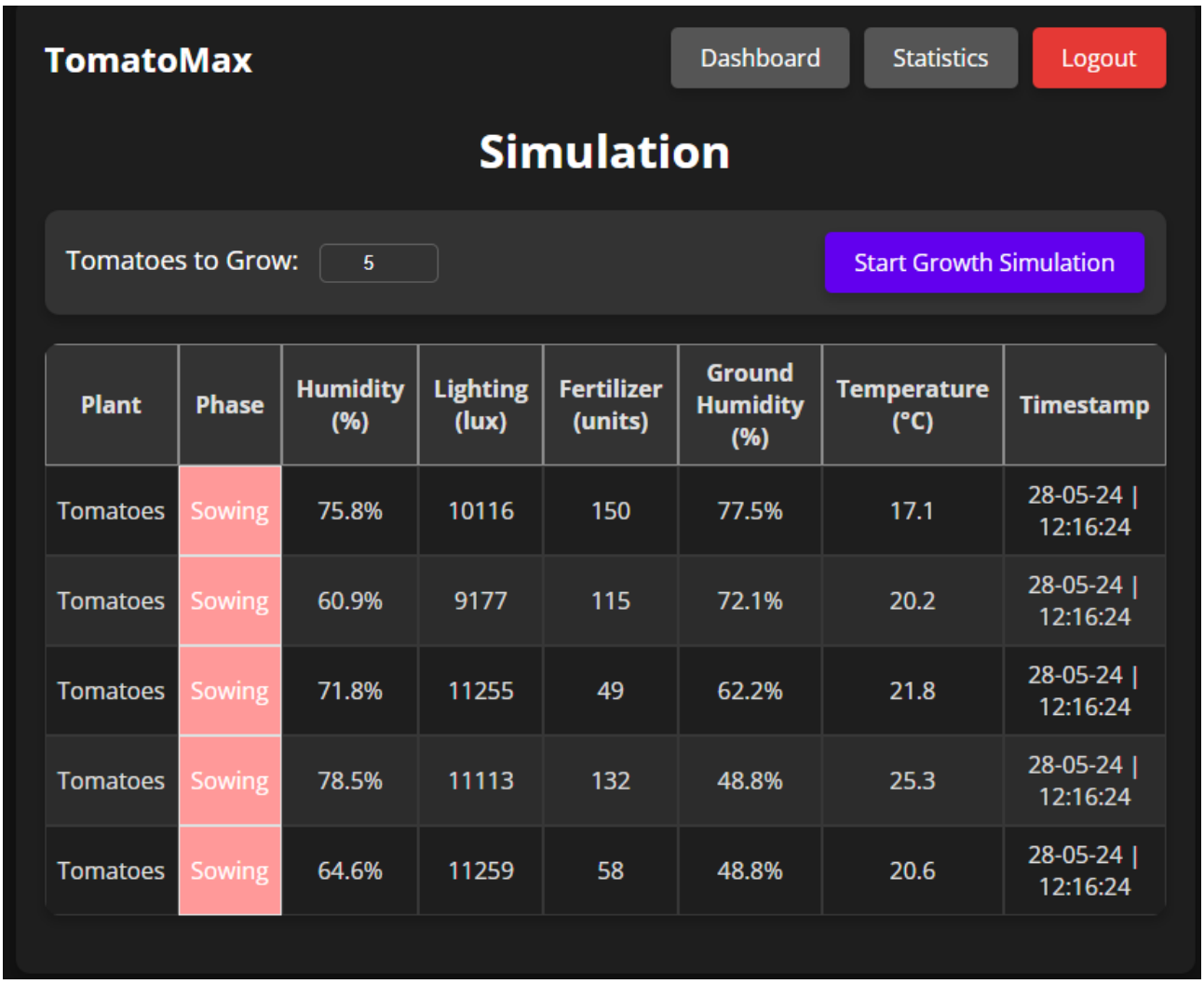


Рисунок. Основна сторінка, обрання кількості рослин для симуляції

Проведення самої симуляції.

Під час симуляції, користувачеві доступно для перегляду назва рослини, саме помідори, на далі поточна фаза росту, загалом їх маємо 5: Sowing, Germination, Stem elongation, Flowering, Ripening. Також можна переглянути показники, для поточної фази, а саме: Вологість повітря, освітленість, добрива (За методикою суміші NKP, натрій-калій фосфорного добрива), вологість грунту та температуру. Також можемо переглядати час початку симуляції.

Також на основній сторінці нам доступні наступні кнопки Statistic та кнопка виходу з облікового запису Logout

В вкладці статистика, ми можемо переглядати лінійні графіки, на яких видно середнє значення обраного параметру, на кожній фазі росту, а також корегування даного параметру, протягом всіх 5-ти фаз.

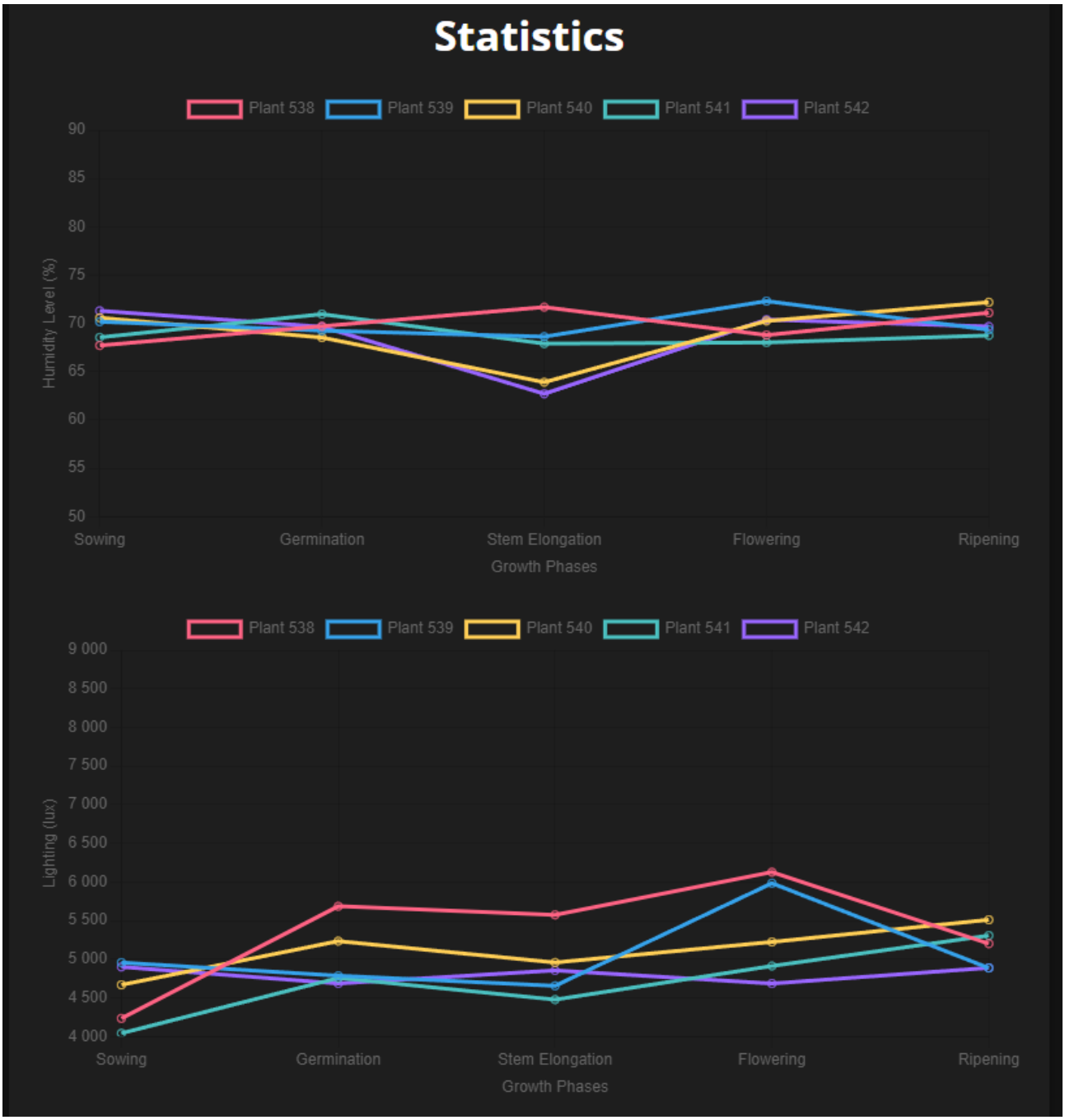


Рисунок. Візуалізація отриманих даних