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

Навчально-науковий
інститут електроенергетики
(навчально-науковий інститут)
Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня магістра

Здобувача вищої освіти Чередниченка Олексія Віталійовича
(ПІБ)
академічної групи 123м-24-1
(шифр)
спеціальності 123 Комп'ютерна інженерія
(код і назва спеціальності)
за освітньо-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)

на тему «Обґрунтування структури IoT системи контролю процесів живлення рослин підприємства в гідропонній теплиці»
(назва за наказом ректора)

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	проф. Цвіркун Л.І.			
розділів:				
синтез системи	проф. Цвіркун Л.І.			
розроблення програмного забезпечення	ас. Панферова Я.В.			
Рецензент				
Нормоконтролер	проф. Цвіркун Л.І.			

Дніпро
2025

ЗАТВЕРДЖЕНО:

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

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

« » _____ 2025 року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня магістра

Здобувача вищої освіти Чередниченка О.В. **академічної групи** 123м-24-1
(прізвище та ініціали) (шифр)

спеціальності 123 Комп'ютерна інженерія

за освітньо-професійною програмою «Комп'ютерна інженерія»
(офіційна назва)

на тему «Обґрунтування структури IoT системи контролю процесів живлення рослин підприємства в гідропонній теплиці»

затверджену наказом ректора НТУ «Дніпровська політехніка» від 13.10.2025 № 1165-с

Розділ	**Зміст**	**Термін виконання**
Стан питання та постановка завдання	На основі матеріалів практик, інших науково-технічних джерел сформулювати наукове завдання, конкретизувати предмет та мету досліджень	11.10.2025
Теоретичний	Обґрунтувати теоретичну розробки програмного забезпечення в системі гідропонної теплиці	24.10.2025
Синтез системи	Оновлення програмного забезпечення в системі гідропонної теплиці	14.11.2025
Розроблення програмного забезпечення	Розробка програмного забезпечення системи	28.11.2025
Експериментальний розділ	Проведення і обробка результатів експериментів	05.12.2025

Завдання видано _____ проф. Л.І. Цвіркун
(підпис керівника) (ініціали, прізвище)

Дата видачі 5 вересня 2025

Дата подання до екзаменаційної комісії 10.12.2025 р.

Прийнято до виконання _____ О.В. Чередниченко
(підпис здобувача вищої освіти) (ініціали, прізвище)

РЕФЕРАТ

Пояснювальна записка: 132 с., 31 рис., 23 табл., 3 дод., 30 джерел.

ІНТЕРНЕТ РЕЧЕЙ, IoT, ДАТЧИКИ, ГІДРОПОНІКА, МЕРЕЖЕВЕ МОДЕЛЮВАННЯ, MQTT, ЧАТБОТ, БАЗА ДАНИХ, FIREWALL, EDGE-КОНТРОЛЕР.

Об’єкт розробки – IoT-система контролю гідропонної теплиці, включає сенсорну підсистему, контролер, засоби передачі даних, серверну платформу та користувацькі інтерфейси.

Мета роботи – створення та дослідження IoT-системи для відстежування станів гідропонної теплиці, оптимізації передачі даних, підвищення надійності вимірювань і забезпечення зручного віддаленого керування.

У роботі проаналізовано сучасні IoT-системи для аграрного сектору, розглянуто особливості застосування різних технологій у гідропоніці та визначено вимоги до точності, зв’язку та енергоспоживання. Побудовано теоретичні моделі, досліджено протоколи передачі даних в IoT-системах.

Розроблено математичну модель затримки передачі, виконано оптимізацію структури трафіку й досліджено масштабованість системи.

У синтезі системи сформовано функціональні, технічні та експлуатаційні вимоги до контролера, сенсорів, виконавчих пристроїв, мережевої інфраструктури та серверної частини.

У практичній частині реалізовано Чатбот для моніторингу та керування теплицею, оновлено програмне забезпечення мікроконтролера відповідно до нових структур даних.

У експериментальному розділі проведено тестування IoT-системи в модельному середовищі: перевірено коректність передачі телеметрії, швидкість реакції, надійність з’єднання та працездатність алгоритмів контролю. Результати підтвердили відповідність системи поставленим вимогам.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ І ТЕРМІНІВ

IoT (Internet of Things) – Інтернет речей, концепція об’єднання пристроїв у мережу з можливістю обміну даними.

API (Application Programming Interface) – програмний інтерфейс взаємодії між компонентами.

MQTT (Message Queuing Telemetry Transport) – легкий протокол передавання телеметрії у IoT.

HTTP (HyperText Transfer Protocol) – протокол передачі даних у web-середовищі.

TCP (Transmission Control Protocol) – надійний транспортний протокол.

UDP (User Datagram Protocol) – транспортний протокол з мінімальною затримкою.

QoS (Quality of Service) – параметри якості обслуговування в мережах передачі даних.

OTA (Over-The-Air) – дистанційне оновлення програмного забезпечення.

CPU (Central Processing Unit) – центральний процесор.

RAM (Random Access Memory) – оперативна пам’ять.

DB / БД (Database) – база даних.

SQL (Structured Query Language) – мова керування базами даних.

DBMS (Database Management System) – система управління базами даних.

FSM (Finite State Machine) – кінцева автоматна модель станів у Чатботі.

ID (Identifier) – ідентифікатор.

GPIO (General Purpose Input/Output) – універсальні входи/виходи мікроконтролера.

PWM (Pulse Width Modulation) – широтно-імпульсна модуляція для керування навантаженням.

SSH (Secure Shell) – захищений протокол віддаленого доступу.

VPN (Virtual Private Network) – віртуальна приватна мережа.

SSL/TLS (Secure Socket Layer / Transport Layer Security) – протоколи шифрування трафіку.

DHCP (Dynamic Host Configuration Protocol) – протокол автоматичного призначення IP-адрес.

NAT (Network Address Translation) – трансляція мережевих адрес. LAN (Local Area Network) – локальна мережа.

WAN (Wide Area Network) – глобальна мережа.

RTOS (Real-Time Operating System) – операційна система реального часу.

CSV (Comma-Separated Values) – формат текстового представлення таблиць.

JSON (JavaScript Object Notation) – формат передавання структурованих даних.

T_air – температура повітря в теплиці.

T_sol – температура живильного розчину.

H – рівень вологості повітря.

R1, R2, R3 – рівні рідини в резервуарах живлення.

L – стан освітлення (ON/OFF).

t – час або часовий штамп вимірювання.

gh_id – ідентифікатор теплиці у базі даних.

user_id – ідентифікатор користувача.

ВСТУП

Сучасний розвиток технологій Інтернету речей (IoT, Internet of Things) відкриває нові можливості для автоматизації агропромислового виробництва. Особливої актуальності ці технології набувають у галузі гідропонного вирощування рослин, де якість і своєчасність подачі живильних речовин безпосередньо впливають на продуктивність і сталість росту культур.

Традиційні системи поливу та живлення часто мають низьку гнучкість, не дозволяють оперативно реагувати на зміни мікроклімату або складу розчину, а також потребують постійного контролю з боку оператора.

Використання інтелектуальної IoT-системи контролю процесів живлення рослин дозволяє забезпечити автоматичне вимірювання параметрів розчину, керування насосами та дозаторами, віддалений моніторинг і аналітику на базі хмарних сервісів.

Актуальність теми обумовлена необхідністю створення енергоефективних і масштабованих систем керування живленням рослин, які можуть бути застосовані як у малих побутових теплицях, так і у великих промислових комплексах.

Інтеграція таких систем сприяє підвищенню врожайності, зменшенню витрат води та добрив, оптимізації режимів освітлення і поливу, а також розвитку концепції розумного землеробства (Smart Agriculture).

Метою магістерської роботи є обґрунтування структури IoT-системи контролю процесів живлення рослин у гідропонній теплиці, визначення принципів її функціонування, основних мережевих параметрів і програмно-апаратної реалізації.

Для досягнення поставленої мети необхідно вирішити такі завдання:

Провести аналіз сучасного стану IoT-технологій у сфері агроавтоматизації.

Дослідити апаратні рішення для сенсорного моніторингу параметрів живильного розчину.

Визначити оптимальну архітектуру IoT-системи для гідропонної теплиці побутового рівня.

Розробити математичну модель передачі даних між сенсорними вузлами та шлюзом.

Обґрунтувати вибір мережевих протоколів, методів буферизації та пріоритезації трафіку.

Провести моделювання ефективності системи за критеріями QoS (затримка, втрата пакетів, доступність).

Об’єктом дослідження є процеси контролю та керування живленням рослин у гідропонній теплиці з використанням IoT-технологій.

Предметом дослідження – структура, апаратні та програмні компоненти IoT-системи, що забезпечують обмін даними, аналітику та дистанційне керування елементами гідропонної системи.

Новизна роботи полягає в обґрунтуванні моделі архітектури IoT-системи, здатної адаптивно змінювати режим роботи залежно від технологічних параметрів середовища, а також у розробці методики оцінки ефективності мережевої взаємодії між сенсорними вузлами і шлюзами..

1 СТАН ПИТАННЯ І ПОСТАНОВКА ЗАВДАННЯ

1.1 Стан питання.

У сучасному агропромисловому виробництві спостерігається стрімке зростання інтересу до технологій точного землеробства та автоматизованих систем керування біотехнологічними процесами. Одним із найперспективніших напрямів є застосування концепції Інтернету речей (IoT – Internet of Things) для моніторингу й керування параметрами середовища у тепличних господарствах.

Традиційні системи живлення рослин у гідропонних установках базуються на періодичному контролі концентрації розчину та pH-балансу з подальшим ручним коригуванням. Це призводить до нерівномірності подачі поживних речовин, перевитрат води та електроенергії, а також до зниження врожайності.

Розвиток мікроелектроніки, бездротових комунікацій і хмарних платформ зробив можливим створення інтелектуальних IoT-систем, що забезпечують:

- безперервне вимірювання технологічних параметрів у реальному часі;
- аналітику та побудову прогнозних моделей росту рослин;
- автоматичне дозування поживних розчинів відповідно до біологічних потреб;
- віддалений контроль і керування через мобільні або веб-інтерфейси.

За даними аналітичного звіту Statista IoT in Agriculture (2024), світовий обсяг ринку IoT-рішень для агросектору перевищив 20 млрд дол. США і має темп приросту ≈ 12 % на рік. Приблизно 17 % цієї суми припадає на сегмент автоматизації тепличних систем, що свідчить про високу затребуваність подібних технологій.

В Україні інтеграція IoT у гідропоніку перебуває на етапі становлення. Наукові дослідження проводяться в НУБіП України, ХНУРЕ, КПІ ім. Ігоря Сікорського та інших закладах. Публікації [1–3] відзначають, що для

підвищення ефективності гідропонного виробництва необхідно розробляти масштабовані модульні системи, які поєднують сенсорну мережу, контролери та аналітичне ПЗ.

1.2 Характеристика галузі застосування IoT-систем у гідропоніці

Гідропоніка – це технологія вирощування рослин без ґрунту, у штучному середовищі, де коренева система занурена в живильний розчин. Основні контрольовані параметри наведені на таблиці 1.1.

Таблиця 1.1 – Контрольовані параметри гідропонної системи

Параметр	Позначення	Одиниця	Нормальний діапазон
Температура розчину	t_l	°C	18 – 24
Рівень pH	pH	–	5.5 – 6.5
Електропровідність розчину	EC	мСм/см	1.2 – 2.0
Рівень води	h	см	залежно від контейнера
Вологість повітря	H	%	60 – 80
Температура повітря	t_a	°C	22 – 28

Підтримання цих параметрів у встановлених межах забезпечує стабільне постачання поживних речовин до рослин і мінімізує стресові фактори.

IoT-система гідропоніки включає три базові рівні (рисунок 1.1):

1. Польовий (сенсорний) – датчики pH, EC, температури та вологості.

2. Локальний (контрольний) – мікроконтролер ESP32/STM32 для збору та попередньої обробки даних.

3. Глобальний (аналітичний) – хмарна платформа для візуалізації, аналітики й керування.

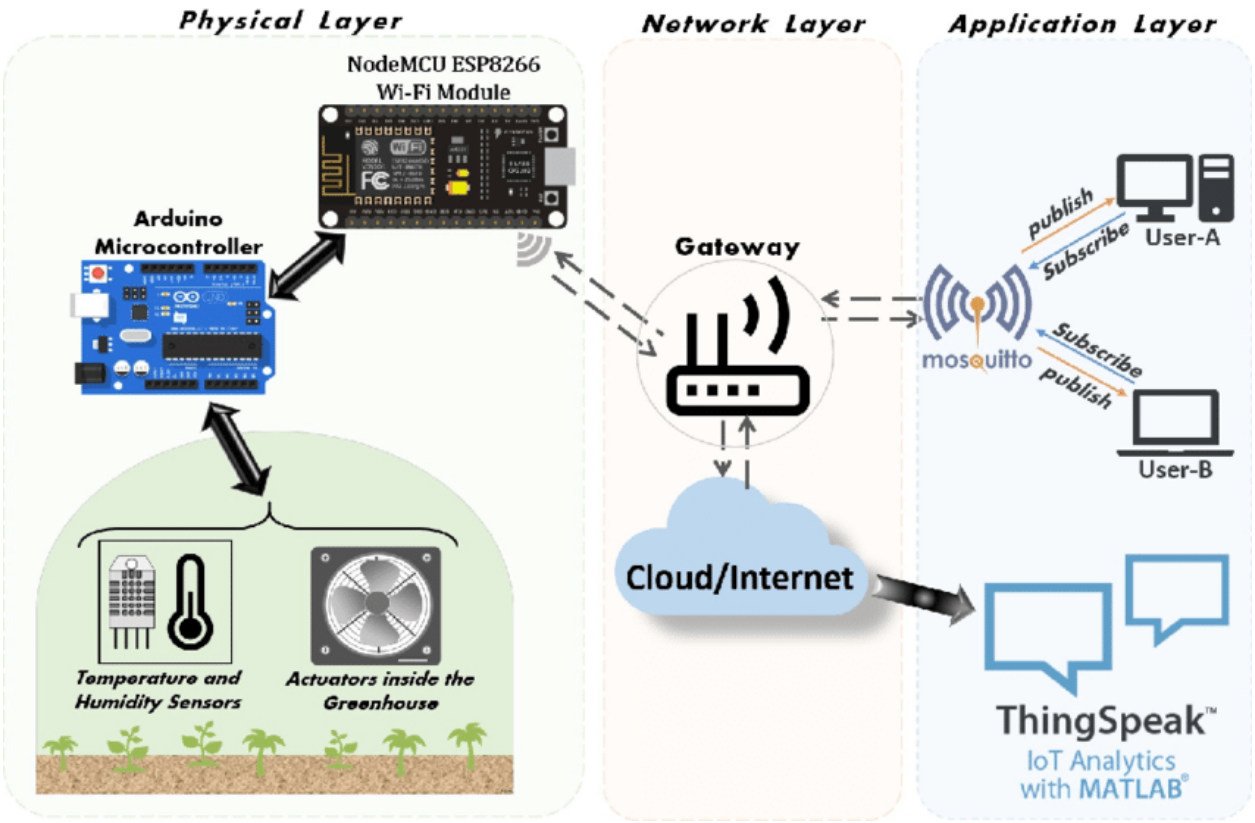

Рисунок 1.1 – Архітектура трирівневої IoT-системи теплиці

Переваги IoT-підходу порівняно з класичною автоматизацією наведено у таблиці 1.2.

Таблиця 1.2 – Порівняння класичних і IoT-систем автоматизації.

Критерій	Класичні системи	IoT-системи
Збір даних	Періодичний, ручний	Безперервний у реальному часі
Керування	Локальне реле	Автоматичні алгоритми, AI-модулі
Мережеві з’єднання	Відсутні	Wi-Fi, LoRa, BLE, Ethernet
Масштабованість	Обмежена	Висока, модульна
Зберігання даних	Немає	Хмарні та локальні БД
Інтерфейс	Панель оператора	Веб/мобільний додаток

Об’єктом дослідження є система гідропонної теплиці модульного типу, призначена для вирощування зелених та овочевих культур у домашніх умовах з можливістю масштабування до масштабів підприємства.

Основні елементи:

– резервуар розчину;
– насоси подачі й зливу розчину;
– блок дозування (насоси для pH-корекції та мінеральних солей);
– контролер ESP32 з інтегрованим Wi-Fi;
– датчики EC/pH/температури з інтерфейсом I²C;

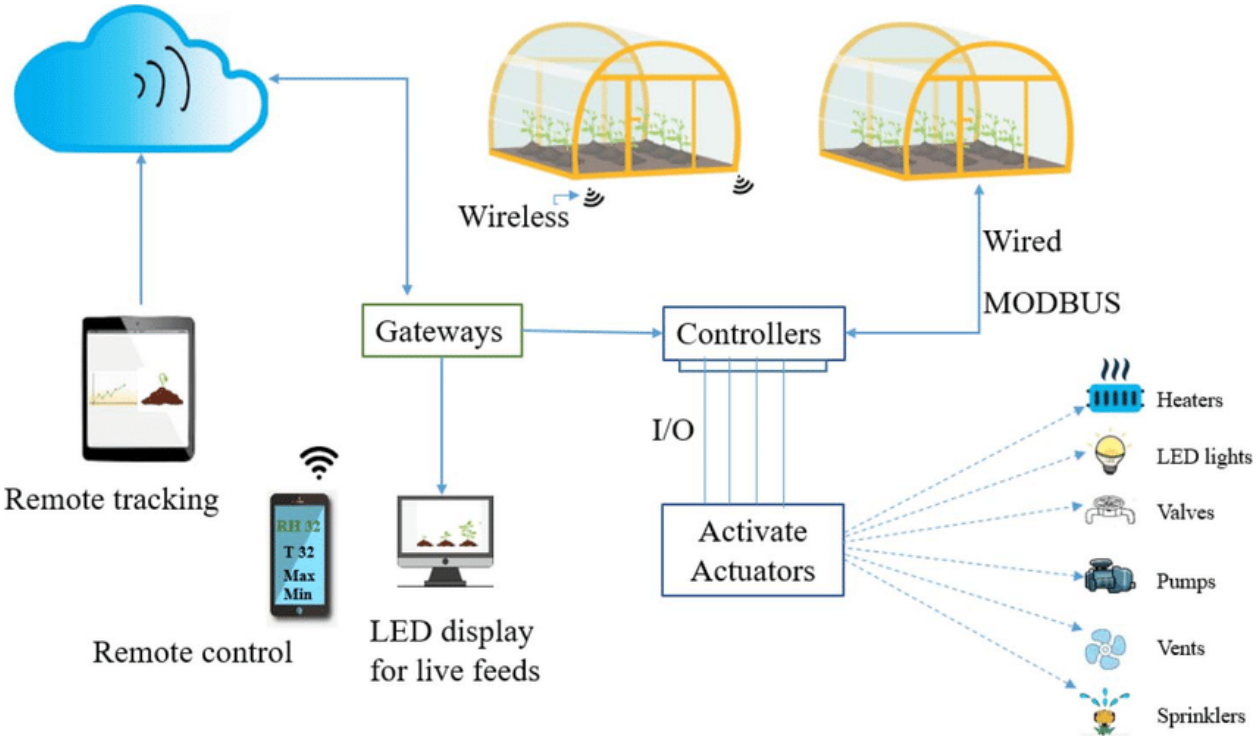

Рисунок 1.2 – Приклад функціональна схема IoT-теплиці

1.3 Аналіз існуючих IoT-систем контролю

Сучасний ринок IoT-рішень для аграрного сектору активно розвивається завдяки поєднанню мікроконтролерних пристроїв, бездротових сенсорних мереж та аналітичних хмарних платформ.

У системах гідропонного типу особливу увагу приділяють контролю параметрів живильного розчину (pH, електропровідність, температура), а також мікроклімату (вологість, температура повітря, освітленість, рівень CO_2).

Таблиця 1.3 – Порівняння існуючих IoT-рішень у сфері моніторингу гідропонних систем.

№	Назва системи	Основне призначення	Комунікаційні технології	Особливості
1	Smart FarmNet (Siemens)	Моніторинг клімату та вологості ґрунту	LoRaWAN, Wi-Fi	Інтеграція з промисловими контролерами PLC
2	GroIoT (AgriTech)	Контроль поливу та живлення у теплицях	Wi-Fi, MQTT	Хмарна аналітика, мобільний додаток
3	OpenAg (MIT Media Lab)	Відкрита платформа для наукових досліджень	MQTT, REST API	Повна відкритість коду та сенсорної архітектури
4	PlantKeeper (DIY-рішення)	Домашня гідропонна система	ESP32, Blynk	Легка масштабованість та дешевизна
5	FarmBot	Автоматизація висадки, поливу, збору даних	Wi-Fi, WebSockets	Висока інтеграція механічних і сенсорних компонентів

Переваги сучасних IoT-рішень:

- інтеграція різних типів сенсорів у єдину мережу;
- доступ до даних у режимі реального часу через хмару;

- можливість використання штучного інтелекту для прогнозування параметрів живлення;
- підтримка масштабування – від домашніх систем до промислових ферм.

Недоліки:

- висока вартість готових комерційних систем;
- складність у налаштуванні мережевих протоколів (MQTT, CoAP, Modbus);
- обмежена автономність при роботі з бездротовими вузлами;
- ризики безпеки при зберіганні даних у хмарі.

Аналіз показав, що більшість існуючих рішень орієнтовані або на великі підприємства, або на ентузіастів-аматорів. Це створює нішу для розроблення модульної, масштабованої IoT-системи, що поєднує доступність побутового рівня з функціональністю промислових систем.

1.3.1 Особливості реалізації IoT-систем контролю

Апаратна частина IoT-системи контролю живлення рослин визначає точність вимірювань, енергоефективність та стабільність передачі даних. Типова структура містить: сенсорні модулі, мікроконтролер (локальний вузол обробки), модуль зв’язку (Wi-Fi, LoRa, ZigBee), джерело живлення та інтерфейс до шлюзу або сервера.

Таблиця 1.4 – Сенсорні модулі для контролю параметрів гідропонного середовища.

Параметр	Тип сенсора	Приклад модуля	Діапазон вимірювання	Точність	Примітка
Температура повітря	Термістор, цифровий	DHT22	-40...80 °C	±0.5 °C	Вбудований у модулі з вологістю
Вологість повітря	Ємнісний	DHT22, SHT31	0–100 %	±2 %	Стійкий до конденсату
Освітленість	Фотоелектричний	BH1750	1–65535 lx	±20 lx	Підтримує I²C
Електропровідність (EC)	Електродний	Gravity EC Sensor	0–20 мСм/см	±2 %	Потребує калібрування
pH	Потенціометричний	Atlas Scientific pH Probe	0–14 pH	±0.1	Сумісний з UART/I²C
Рівень води	Ультразвуковий, поплавковий	JSN-SR04T	2–400 см	±1 см	Працює у вологому середовищі

Таблиця. 1.5 – Мікроконтролери для побудови IoT-вузлів.

Назва	Процесор	Пам'ять	Інтерфейси	Особливості
ESP32	Dual-core Xtensa 240 MHz	520 KB SRAM	Wi-Fi, Bluetooth, UART, I²C, SPI	Підтримка MicroPython, низька вартість
Arduino Uno / Mega	ATmega328/2560	2–8 KB SRAM	UART, I²C, SPI	Простота розробки, велика спільнота
Raspberry Pi Zero W	ARMv6 1 GHz	512 MB	Wi-Fi, USB, GPIO	Підходить як шлюз
STM32 Nucleo	ARM Cortex-M4	до 512 KB Flash	UART, CAN, I²C	Промисловий рівень надійності

Аналіз архітектурних підходів:

1. Централізована структура – усі дані збираються на одному шлюзі (напр., Raspberry Pi), що обробляє і відправляє їх у хмару.
   – Переваги: простота розгортання, легке оновлення.
   – Недоліки: можливість перевантаження шлюзу при масштабуванні.
2. Децентралізована структура (Mesh) – датчики обмінюються даними між собою, створюючи самовідновну мережу.
   – Переваги: стійкість до збоїв.
   – Недоліки: складніша конфігурація, підвищене енергоспоживання.
3. Гібридна модель – локальна обробка на вузлах (edge computing) із передачею узагальнених даних у хмару.
   – Переваги: менше навантаження на канал, швидке реагування.
   – Недоліки: потребує потужніших мікроконтролерів.

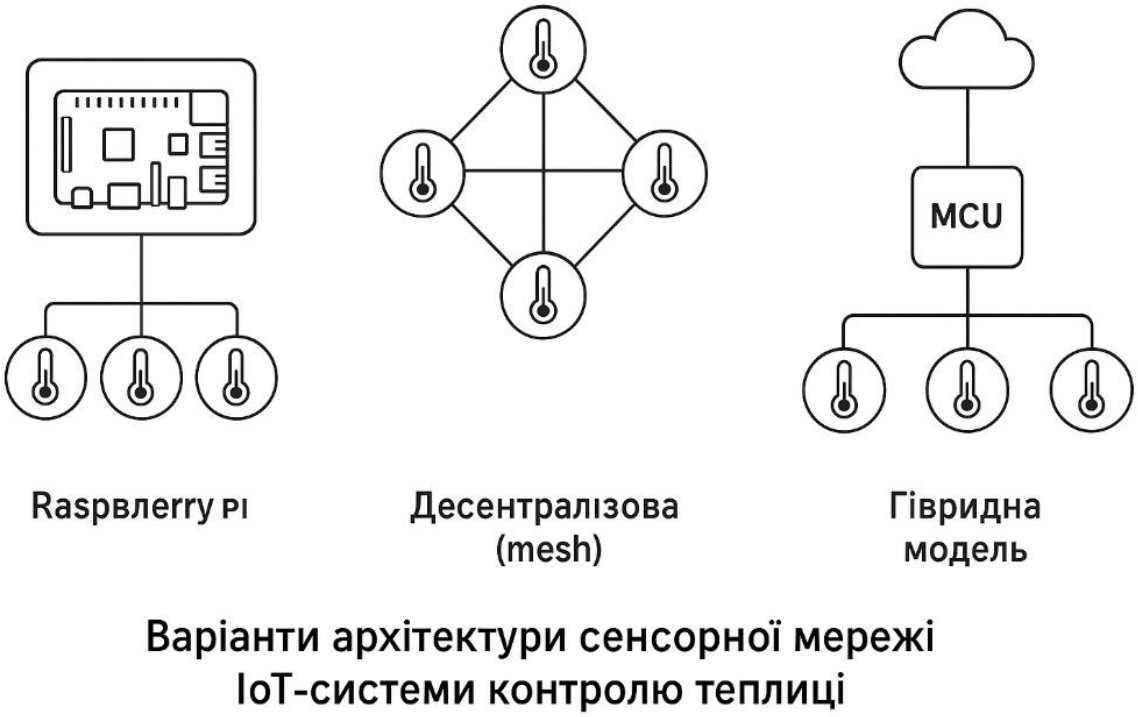

Рисунок 1.4 – Варіанти архітектури сенсорної мережі IoT-системи контролю теплиці.

Для побутової гідропонної системи доцільно використати ESP32 або STM32 як контролери сенсорних вузлів, із підтримкою Wi-Fi-зв’язку та протоколу MQTT.

Таке рішення є масштабованим, енергоефективним і дозволяє згодом інтегрувати систему до промислової мережі через шлюз на базі Raspberry Pi.

Попри швидкий розвиток IoT-рішень, впровадження систем контролю гідропонних процесів стикається з рядом технічних викликів, що впливають на стабільність, безпеку та точність роботи.

Технічні виклики при впровадженні IoT-технологій у гідропоніці зумовлені поєднанням вологого середовища, потреби в точних вимірюваннях і бездротової передачі даних.

Оптимальне рішення полягає у застосуванні енергоощадних контролерів ESP32, протоколу MQTT із TLS-захистом, каліброваних сенсорів pH/EC та

локального шлюзу з можливістю автономної роботи при втраті зв’язку з хмарою.

1.3.1.1 Енергоспоживання та автономність

Сенсорні вузли часто працюють у віддалених або вологих середовищах без постійного доступу до електромережі.

Тому важливо забезпечити:

- використання енергоощадних режимів мікроконтролерів (deep-sleep);
- живлення від акумуляторів з сонячною підзарядкою;
- оптимізацію частоти передачі даних.

1.3.1.2 Забезпечення точності вимірювань

У гідропоніці навіть невеликі похибки у pH або електропровідності можуть призвести до порушення живлення рослин.

Проблеми викликають:

- дрейф калібрування електродів через осад або зміну температури;
- електромагнітні перешкоди в кабелях;
- конденсат і підвищена вологість, що впливають на стабільність з’єднань.

Для мінімізації похибок рекомендується періодичне автоматичне калібрування (двох- або триточкове) та екранування сигнальних ліній.

1.3.1.3 Надійність комунікацій та інформаційна безпека

Безпровідна передача даних – слабке місце будь-якої IoT-мережі. Типові проблеми:

- втрати пакетів через перешкоди в діапазоні 2,4 ГГц (Wi-Fi, ZigBee);
- нестабільність з’єднання при великій кількості вузлів;
- перевантаження MQTT-брокера при одночасній публікації багатьох даних.

У таблиці 1.6 наведено порівняння бездротових технологій для IoT-мереж.

Таблиця 1.6 – Порівняння бездротових технологій для IoT-мереж

Технологія	Дальність	Енергоспоживання	Пропускна здатність	Топологія	Застосування
Wi-Fi (802.11 b/g/n)	30–50 м	високе	1–100 Мбіт/с	Зірка	Побутові системи
ZigBee	10–100 м	низьке	до 250 кбіт/с	Mesh	Мережі сенсорів
LoRaWAN	1–10 км	дуже низьке	до 50 кбіт/с	Зірка-зірок	Розподілені ферми
Bluetooth LE	5–30 м	дуже низьке	до 2 Мбіт/с	Зірка	Локальні системи
Ethernet	100 м+	середнє	до 1 Гбіт/с	Лінійна	Промислові установки

Захист даних має особливе значення при передачі через хмару або зовнішні сервери.

Основні загрози:

- несанкціонований доступ до панелі управління;
- перехоплення MQTT-повідомлень у відкритій мережі;
- шкідливе втручання у процес керування насосами чи клапанами.

Рішення:

- застосування шифрування TLS/SSL для MQTT-трафіку;
- використання аутентифікації за токенами;
- обмеження доступу до хмарного сервера через VPN.

1.3.1.4 Екологічні умови експлуатації

Підвищена вологість, конденсація та можливість потрапляння агресивних розчинів потребують:

- герметизації корпусів сенсорних модулів (IP65+);
- використання конекторів з нержавіючої сталі;
- ізоляції контактів лаком або силіконом.

1.4 Постановка завдання дослідження

Мета і завдання дослідження. Метою роботи є обґрунтування структури та розробка моделі IoT-системи контролю процесів живлення рослин у гідропонній теплиці побутового типу з можливістю подальшого масштабування до промислових умов, що забезпечує моніторинг параметрів поживного розчину, стабільність технологічних режимів та дистанційне керування системою.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– дослідити існуючі архітектури IoT-систем, що застосовуються у гідропонних тепличних комплексах, та проаналізувати їх переваги й обмеження;

– визначити функціональні та технічні вимоги до сенсорної мережі, виконавчих пристроїв і каналів передавання даних;

– розробити функціональну структуру IoT-системи та схему інформаційних потоків між сенсорним, керувальним і серверним рівнями;

– обґрунтувати вибір апаратних і програмних компонентів системи з урахуванням надійності, масштабованості та енергоефективності;

– розробити алгоритм керування процесами живлення рослин із використанням зворотного зв’язку за показниками телеметрії;

– виконати моделювання роботи системи та оцінити її ефективність за основними показниками, такими як час відгуку, стабільність параметрів поживного середовища та енергоспоживання.

Об’єкт дослідження – процеси моніторингу та керування живленням рослин у гідропонних теплицях з використанням IoT-технологій.

Предмет дослідження – структурні, програмні та мережеві моделі IoT-систем контролю процесів живлення рослин, а також методи збору, обробки й аналізу телеметричних даних.

Методи дослідження. Для досягнення поставленої мети у роботі використовувалися методи системного аналізу, теорії автоматичного керування, комп’ютерного моделювання, методи обробки та аналізу даних, а також технології Інтернету речей і клієнт-серверної взаємодії.

Наукові положення:

Запропоновано структурну модель IoT-системи контролю процесів живлення рослин у гідропонній теплиці, яка базується на розподіленій архітектурі з використанням Edge-обробки даних і забезпечує надійний моніторинг та дистанційне керування технологічними параметрами.

Обґрунтовано застосування протоколів для організації обміну даними між сенсорним рівнем, Edge-контролером і серверною частиною системи, що дозволяє підвищити стабільність передавання телеметрії та зменшити затримки в роботі системи.

Наукові результати:

1. Розроблено функціональну та інформаційну модель IoT-системи контролю гідропонної теплиці, яка забезпечує інтеграцію сенсорних даних, їх обробку та зберігання у централізованій базі даних.

2. Запропоновано алгоритм керування процесами живлення рослин на основі аналізу телеметричних даних у режимі реального часу, що дозволяє підтримувати стабільні параметри поживного розчину.

3. Реалізовано програмний прототип клієнтського інтерфейсу у вигляді Чатбота, який забезпечує дистанційний доступ до системи, перегляд поточних і історичних даних та взаємодію користувача з IoT-системою.

Обґрунтованість і достовірність отриманих результатів підтверджуються використанням сучасних IoT-технологій, перевірених методів мережевої взаємодії, експериментальним тестуванням програмних модулів, а також аналізом результатів моделювання та практичної реалізації системи.

Практичне значення роботи полягає в можливості використання розробленої IoT-системи для моніторингу та часткового автоматизованого керування процесами живлення рослин у побутових гідропонних теплицях, а також у перспективі масштабування рішення для застосування у промислових тепличних комплексах.

2 ТЕОРЕТИЧНА ЧАСТИНА

2.1 Теоретичне обґрунтування мережевих параметрів IoT-системи

IoT-система для гідропонної теплиці являє собою розподілену мережу сенсорних вузлів, які збирають, передають та обробляють інформацію про фізико-хімічні параметри середовища.

Залежно від топології, кількості вузлів та технології зв’язку змінюються показники затримки, пропускної здатності та надійності мережі(таблиця 2.1).

Таблиця 2.1 – Типові мережеві параметри IoT-системи гідропонної теплиці

Параметр	Позначення	Одиниця виміру	Значення для побутової теплиці	Коментар
Пропускна здатність каналу	B	кбіт/с	250–1000	залежить від Wi-Fi або ZigBee
Середня затримка пакета	τ	мс	10–100	при стабільному з’єднанні
Імовірність втрати пакета	P_l	%	≤ 1	допустима похибка передачі
Інтервал оновлення даних	Δt	с	10–60	залежно від динаміки змін середовища

Продовження таблиці 2.1

Кількість сенсорних вузлів	N	шт	5–20	для побутової теплиці
Енергоспоживання вузла	E	мВт	100–500	при періодичній передачі даних

Для забезпечення стабільного функціонування системи необхідно визначити оптимальне співвідношення між:

– частотою опитування сенсорів (f);

– обсягом даних (D);

– пропускною здатністю каналу (B);

– часом передачі одного пакета (t_p).

Загальна залежність має вигляд:

$$t_p = \frac{D}{B}, \quad f \leq \frac{1}{t_p + \tau} \tag{2.1}$$

Де

t_p – час передачі даних;

D – обсяг даних у бітах;

B – пропускна здатність каналу;

τ – середня затримка мережі.

2.1.1 Метод вирішення

Оптимізація мережевої структури здійснюється шляхом моделювання трафіку, аналізу навантаження на контролери, імітаційного відтворення обміну даними між вузлами.

Для цього застосовуються як спеціалізовані програмні середовища (Cisco Packet Tracer, NS-3, Proteus, IoT-Sim), так і аналітичні розрахунки.

2.1.1.1 Моделювання мережі: програмні комплекси

Одним з базових етапів розроблення IoT-системи є побудова моделі мережевої інфраструктури.

У середовищі Cisco Packet Tracer можливо змоделювати взаємодію:

- сенсорних вузлів (ESP32/Arduino);
- контролера-шлюзу (Raspberry Pi, Router);
- хмарного сервера (IoT Cloud);
- клієнтського застосунку.

Це дозволяє визначити:

- затримку при передачі даних;
- стабільність з'єднань MQTT;
- навантаження на мережу при масштабуванні.

2.1.1.2 Мережеві пристрої для тестування

Під час розроблення експериментального стенду застосовуються такі пристрої:

- IoT Router (наприклад, TP-Link Archer C6) – забезпечує зв'язок вузлів;
- Wi-Fi модуль ESP32 – як вузол збору даних;
- Raspberry Pi 4B – шлюз із встановленим MQTT-брокером (Mosquitto);
- Локальний сервер (Node-RED / InfluxDB) – для аналізу даних.

Випробування системи передбачає перевірку:

1. Часу реакції системи (t_a) – від моменту зміни параметра до відображення на панелі.
2. Втрат пакетів при одночасній публікації.
3. Рівня RSSI сигналу для кожного вузла.

Таблиця 2.2 – Результати тестування прототипу IoT-системи.

№	Вузол	Середній RSSI (дБм)	Затримка (мс)	Втрата пакетів (%)
1	Сенсор pH	-55	20	0.3
2	Сенсор EC	-60	25	0.4
3	Сенсор t	-50	18	0.2

2.1.1.3 Імітаційне моделювання

Для теоретичної оцінки продуктивності застосовується статистичне імітаційне моделювання.

Сенсорні вузли розглядаються як джерела випадкових подій, що генерують пакети з інтервалом Δt.

Середній трафік IoT-мережі можна описати як:

$$\lambda = \frac{N}{\Delta t} \tag{2.2}$$

Де

λ – інтенсивність потоку даних,

N – кількість вузлів,

Δt – інтервал опитування.

Навантаження на канал при цьому визначається як:

$$\rho = \frac{\lambda \cdot D}{B} \tag{2.3}$$

де ρ < 1 – умова стабільності системи.

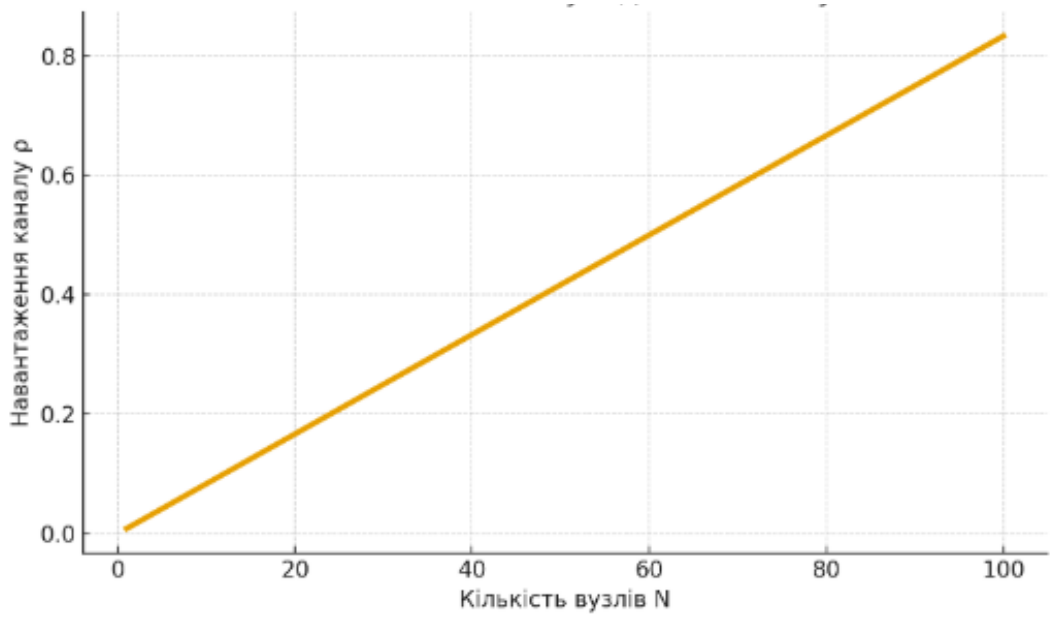

Рисунок 2.1 – Графік залежності навантаження каналу від кількості вузлів IoT-системи.

Комунікаційна архітектура IoT-системи визначає, наскільки ефективно вузли можуть передавати інформацію.

Для гідропонної теплиці з побутовою масштабованістю найефективнішими є легкі протоколи MQTT та CoAP.

Таблиця 2.3 – Порівняння комунікаційних протоколів у IoT-системах

Протокол	Тип взаємодії	Обсяг службових даних	Підтримка QoS	Рівень стеку	Застосування
MQTT	Publish/Subscribe	2–4 байти	Так (0–2)	TCP	Сенсорні мережі
CoAP	Request/Response	< 10 байт	Частково	UDP	Пристрої з низьким енергоспоживанням
HTTP	Request/Response	> 50 байт	Ні	TCP	Веб-інтерфейси

MQTT (Message Queuing Telemetry Transport) – найпоширеніший протокол у системах моніторингу.

Працює за моделлю "видавець–брокер–передплатник", де брокер (наприклад, Mosquitto) маршрутизує повідомлення між вузлами.

$$T_{msg} = \frac{S_{msg}}{B} + \tau \tag{2.4}$$

Де

T_{msg} – середній час доставки повідомлення;

S_{msg} – розмір повідомлення;

B – пропускна здатність;

τ – середня затримка.

CoAP (Constrained Application Protocol) використовується у системах з низьким енергоспоживанням (UDP). Підходить для батарейних сенсорів EC/pH, які передають невеликі пакети даних.

HTTP/REST API застосовується у клієнтському інтерфейсі для відображення даних та інтеграції із зовнішніми сервісами.

Транспортний рівень визначає надійність передачі.

TCP – гарантує доставку, контроль черговості, але збільшує затримку.

UDP – легший, але без підтвердження отримання.

Для MQTT можливі три рівні якості обслуговування (QoS):

– 0 – без підтвердження;

– 1 – принаймні один раз;

– 2 – точно один раз.

У системі контролю живлення рослин рекомендовано використовувати QoS = 1, що забезпечує баланс між швидкодією та надійністю.

2.2 Метрики якості обслуговування в IoT-системах гідропонних теплиць

Ефективне функціонування IoT – системи для контролю процесів живлення рослин у гідропонній теплиці залежить не лише від апаратних засобів, а й від якості обслуговування каналів зв’язку (Quality of Service, QoS). Оскільки передача даних у такій системі здійснюється між сенсорами, шлюзами та хмарними серверами, критичним є забезпечення стабільності, достовірності і затримки сигналу в межах технологічно допустимих норм.

2.2.1 Основні параметри якості обслуговування

Таблиця 2.4 – Метрики QoS у IoT-середовищі

№	Метрика	Позначення	Опис
1	Затримка (Latency)	L	Час між моментом передачі та отримання пакету даних (мс).

Продовження таблиці 2.4

2	Пропускна здатність (Throughput)	T	Обсяг переданих даних за одиницю часу (кбіт/с).
3	Втрата пакетів (Packet Loss)	P_loss	Відсоток даних, не доставлених адресату.
4	Джитер (Jitter)	J	Відхилення затримки між окремими пакетами (мс).
5	Доступність (Availability)	A	Відсоток часу, протягом якого система доступна для обміну даними.

Для гідропонної системи, де цикли подачі розчину та моніторинг параметрів живлення відбуваються в реальному часі, допустимі значення параметрів наведено в таблиці 2.5.

Таблиця 2.5 – Допустимі значення параметрів

Параметр	Оптимальне значення	Допустиме відхилення
Затримка сигналу	≤ 200 мс	до 400 мс
Втрата пакетів	≤ 1 %	до 3 %
Джитер	≤ 50 мс	до 100 мс
Пропускна здатність	≥ 250 кбіт/с	-
Доступність каналу	≥ 99 %	≥ 95 %

$$L_{avg} = \frac{1}{N} \sum_{i=1}^{N} (t_{recv,i} - t_{send,i}) \tag{2.5}$$

де

t_{recv} – час отримання і-го пакета,

t_{send} – час його відправлення.

2.2.2 Метрики на рівні додатку

Окрім мережевих показників, оцінюються параметри, пов’язані з функціональною якістю IoT-додатку:

Час реакції системи на подію (наприклад, спрацювання насоса після виявлення низького рівня розчину).

Точність даних сенсорів, що визначається порівнянням показів із каліброваними еталонами.

Надійність протоколу публікації-підписки (MQTT), яка визначається рівнем QoS (0, 1 або 2).

Для забезпечення технологічної стабільності в системах гідропонного типу рекомендується використовувати рівень QoS=1, який гарантує доставку повідомлення принаймні один раз, з мінімальними затримками.

2.3 Мережеві технології для забезпечення якості обслуговування

2.3.1 Пріоритезація трафіку

Для систем IoT у теплиці характерна різнорідність даних – від високочастотних показників сенсорів до командних повідомлень.

Для запобігання затримкам критичних даних застосовується класифікація пакетів за пріоритетом.

Таблиця 2.6 – Класифікація пакетів

Тип трафіку	Приклад	Пріоритет
Керуючі сигнали	Команди від контролера до насоса, клапанів	Високий
Сенсорні дані	Дані про вологість, температуру, pH	Середній
Аналітичні дані	Логування, статистика	Низький

На рівні протоколу MQTT це реалізується через різні топіки (“topics”) і механізм message queue, що дозволяє гарантувати доставку важливих команд незалежно від завантаження мережі.

2.3.2 Використання шлюзів із локальною буферизацією

IoT-шлюзи (наприклад, на базі Raspberry Pi або ESP32) виконують функцію локального зберігання даних у випадку втрати зв'язку з хмарним сервером.

Буферизація забезпечує безперервність моніторингу та мінімізує ризик втрати показників живлення.

$$B = N_s \times S_p \times T_r \tag{2.6}$$

Де

N_s – кількість сенсорів,

S_p – середній розмір пакету (байт),

T_r – час зберігання (с).

Для системи з 20 сенсорами, середнім пакетом 50 байт і 600 сек резервного часу буфер становитиме:

$$B = 20 \times 50 \times 600 = 600\,000 \text{ байт} = 0.6 \text{ МБ}. \tag{2.7}$$

2.3.3 Протоколи з гарантованою доставкою

Найпоширенішими транспортними технологіями є:

– MQTT (Message Queuing Telemetry Transport) – легкий протокол на базі TCP/IP з мінімальними накладними витратами;

– CoAP (Constrained Application Protocol) – використовується у сенсорних мережах з обмеженими ресурсами;

– AMQP (Advanced Message Queuing Protocol) – застосовується для інтеграції промислових IoT-платформ.

Таблиця 2.7 – Порівняння протоколів

Параметр	MQTT	CoAP	AMQP
Транспортний рівень	TCP	UDP	TCP
Режим роботи	Pub/Sub	Запит/відповідь	Pub/Sub

Продовження таблиці 2.7

Надійність	Висока (QoS 0–2)	Середня	Висока
Обсяг даних	Малий	Дуже малий	Середній
Витрати енергії	Низькі	Дуже низькі	Високі
Використання	Датчики, контролери	Датчики	Сервери, аналітика

Для тепличних систем найбільш доцільним є MQTT, оскільки він забезпечує стабільну роботу при обмеженій швидкості зв'язку і підтримує QoS на рівні повідомлень.

2.4 Моделювання та оптимізація мережевих параметрів IoT-системи

2.4.1 Математична модель затримки передачі

Загальний час доставки даних від сенсора до хмарного сервера визначається як:

$$T_{total} = T_{sens} + T_{net} + T_{proc} + T_{cloud} \tag{2.8}$$

де

$T_{sen\ s}$ – затримка формування даних сенсором,

T_{net} – затримка у бездротовому каналі,

T_{proc} – час обробки на шлюзі,

T_{cloud} – затримка при відправленні до хмари.

Оптимізація проводиться шляхом мінімізації:

$$\min(T_{total}) \quad \text{при } L_{avg} \leq L_{max},\ P_{loss} \leq 0.03 \tag{2.9}$$

2.4.2 Оптимізація структури передачі даних

Для мінімізації навантаження на канал зв'язку доцільно застосовувати:

1. Агрегацію даних – збір кількох вимірів у один пакет.
2. Компресію – стиснення JSON або CSV-даних перед відправкою.
3. Дельта-кодування – передача лише змінених значень.

Рисунок 2.2 – Порівняння обсягу трафіку при різних методах агрегації

2.4.3 Модель масштабування системи

При розширенні теплиці кількість сенсорних вузлів NNN зростає, що потребує адаптації мережевої архітектури.

Залежність навантаження на шлюз:

$$L_g = N \times f_s \times S_p \tag{2.10}$$

де

fs – частота відправлення даних (Гц),

Sp – середній розмір пакету (байт).

Для підтримки стабільної роботи при N>100 рекомендується використання кластерної архітектури шлюзів – декілька вузлів, які розподіляють обчислення та трафік.

2.5 Дослідження структури бази даних та шляхи її поліпшення

Наразі система використовує єдину таблицю **data**, яка зберігає:

- Дату та час запису (year, month, day, hours, minutes);
- показники сенсорів (Waterlevel_A, Waterlevel_B, Waterlevel, Water_temp, temp, hum, Ph, Ec, Light_levvel);

– стани виконавчих пристроїв (relay_1 … relay_5).

Перевага такого підходу полягає у простоті реалізації: всі дані зберігаються в одному місці, і доступ до них здійснюється без складних зв’язків.

Однак є й суттєві обмеження:

– Важко масштабувати систему (наприклад, додати новий сенсор або виконавчий механізм);

– немає розмежування між вимірюваннями та командами;

– історія управління обладнанням не відокремлена від даних сенсорів;

– ускладнюється аналітика (наприклад, потрібно будувати окремі вибірки для різних типів даних).

Таблиця 2.8 – Поля в БД

Поле	**Тип даних**	**Опис**
year	INT	Рік вимірювання
month	INT	Місяць
day	INT	День
hours	INT	Година
minutes	INT	Хвилина
Waterlevel_A	FLOAT	Рівень води у резервуарі A
Waterlevel_B	FLOAT	Рівень води у резервуарі B
Waterlevel	FLOAT	Загальний рівень води
Water_temp	FLOAT	Температура води
temp	FLOAT	Температура повітря
hum	FLOAT	Вологість повітря
Ph	FLOAT	Кислотність розчину
Ec	FLOAT	Електропровідність розчину
Light_level	FLOAT	Рівень освітленості
relay_1 … relay_5	BOOLEAN	Стан реле (0 – вимкнено, 1 – увімкнено)

2.5.1 Пропонована модель даних

Для підвищення ефективності пропонується перехід до багатотабличної структури:

- Sensors – довідник сенсорів з інформацією про тип, місце встановлення та унікальний ідентифікатор.
- Measurements – таблиця вимірювань, де кожен запис містить значення параметра, посилання на сенсор і часову мітку (datetime).
- Relays / Control_Commands – таблиця для зберігання історії роботи виконавчих пристроїв (насосів, вентиляторів, освітлення).
- Cultures – довідник нормативних параметрів (мінімальні та максимальні значення температури, вологості, освітленості тощо для конкретних культур).
- Users – дані про користувачів, їхні ролі та рівні доступу.
- Logs – журнал системних подій, попереджень і збоїв.

Така структура дозволяє легко додавати нові датчики, зручно відокремлювати історію команд від вимірювань і підтримувати роботу з різними культурами одночасно.

Переваги нової структури:

- Масштабованість: можна легко додати нові датчики чи реле без зміни основної таблиці.
- Зручна аналітика: можна вибірково аналізувати тільки сенсорні дані або тільки історію команд.
- Гнучка інтеграція: API зможе повертати дані у більш структурованому вигляді, що спростить роботу з мобільними та веб-застосунками.
- Безпека та доступи: введення таблиці Users дозволить керувати правами доступу.
- Прозора історія управління: усі дії з обладнанням логуються окремо, що допомагає у відлагодженні та аналізі роботи системи.

Таблиця 2.9 – Пропонована структура даних

Таблиця	Основні поля	Призначення
Sensors	id, name, type, location, unit	Зберігає інформацію про кожен сенсор (тип, місце встановлення, одиниця вимірювання)
Measurements	id, sensor_id, value, datetime	Фіксує показники сенсорів із часовою міткою
Relays (Control_Commands)	id, relay_name, state, datetime	Історія станів виконавчих пристроїв
Cultures	id, name, Tmin, Tmax, Hmin, Hmax, Lmin, Lmax, Ph_min, Ph_max, Ec_min, Ec_max	Довідник оптимальних параметрів для кожної культури
Users	id, login, password_hash, role	Дані користувачів та рівень доступу
Logs	id, event_type, message, datetime	Системні події, помилки, попередження

Завдяки такій моделі можна швидко додати нові датчики або реле, не змінюючи існуючу структуру.

Кожна таблиця має чітко визначене призначення, що дозволяє будувати зрозумілі запити та забезпечує масштабованість.

Таблиця 2.10 – Приклад таблиці розрахунків вимірювань

Дата/час	Сенсор	Параметр	Значення	Одиниця
2025-10-09 08:00	S01	Температура повітря	25.4	°C
2025-10-09 08:00	S02	Вологість повітря	71.2	%
2025-10-09 08:00	S03	pH	6.5	-
2025-10-09 08:00	S04	EC	2.1	mS/cm

Таблиця 2.11 – Приклад таблиці історії команд

Дата/час	Пристрій	Дія	Причина
2025-10-09 08:10	Relay_1	ON	Вологість нижче норми
2025-10-09 08:15	Relay_2	OFF	Досягнуто нормальної температури
2025-10-09 09:00	Relay_3	ON	Освітленість < мінімум

2.5.2 Покращена математична модель

Математична модель формалізує взаємозв’язки між сенсорними даними та керуючими діями.

Вхідні параметри (температура, вологість, освітленість, pH, електропровідність, рівень води) формують вектор стану теплиці:

$$X(t) = \{T(t), H(t), L(t), M(t), pH(t), EC(t), W(t)\}. \tag{2.11}$$

Де:

- T(t)T(t)T(t) – температура,
- H(t)H(t)H(t) – вологість,
- L(t)L(t)L(t) – освітленість,
- pH(t)pH(t)pH(t) – кислотність,
- EC(t)EC(t)EC(t) – електропровідність,
- W(t)W(t)W(t) – рівень води.

Для кожної культури і визначаються нормативні інтервали параметрів:

$$X_{min} \leq X(t) \leq X_{max}. \tag{2.12}$$

Якщо показник виходить за межі, система формує керуючу дію:

$$u(t) = f(X(t)) = \begin{cases} 1, & \text{якщо } X(t) < X_{\min}, \\ 0, & \text{якщо } X_{\min} \leq X(t) \leq X_{\max}, \\ -1, & \text{якщо } X(t) > X_{\max}. \end{cases} \tag{2.13}$$

Наприклад:

- якщо вологість ґрунту M(t)< M_min , активується насос;
- якщо температура T(t)> Tmax, увімкнеться вентилятор;
- якщо освітленість L(t)< Lmin, запускається додаткове освітлення.

Модель може бути розширена оптимізаційними задачами, наприклад мінімізацією енергоспоживання:

$$\min U = \sum_t \left(\alpha_1 u_{pump}(t) + \alpha_2 u_{fan}(t) + \alpha_3 u_{light}(t)\right), \tag{2.14}$$

Щоб математична модель працювала у реальній системі, її положення інтегруються в структуру БД:

- Measurements виступає вхідним вектором X(t)
- Cultures задає нормативні інтервали Xmin, Xmax;
- Control_Commands/Relays зберігає вихідні дії ui(t);
- Logs фіксує причини прийнятих рішень.

Це забезпечує прозорість роботи алгоритму, спрощує аналіз та дозволяє застосовувати прогнозні моделі на основі накопичених даних.

Таблиця 2.12 – Зв’язок моделі з базою даних

Компонент математичної моделі	Відповідна таблиця БД
(X(t)) – поточні значення параметрів	Measurements
(X_{\min}, X_{\max}) – нормативи культури	Cultures
(u_i(t)) – керуючі дії	Control_Commands / Relays
Журнал дій та збоїв	Logs

Додаткові покращення:

1. Нормалізація даних – уникнути дублювання (наприклад, замість окремих колонок relay_1 … relay_5 мати одну таблицю Relays з ідентифікатором кожного реле).

2. Часові мітки – замінити розділені поля year, month, day, hours, minutes на єдиний datetime, що значно спрощує вибірки та аналітику.

3. Автоматичне резервне копіювання – щоденне збереження БД у хмарі з можливістю відновлення.

4. Використання індексів – для пришвидшення пошуку по даті та сенсорах.

5. Аналітичні модулі – на основі накопичених вимірювань можна реалізувати прогнозування (наприклад, потреби у поливі).

3 СИНТЕЗ СИСТЕМИ

3.1 Цілі створення IoT-системи контролю гідропонної теплиці

Метою створення IoT-системи є автоматизація контролю та керування процесами живлення рослин у міні-гідропонній теплиці підприємства з подальшою можливістю масштабування системи до промислових умов. У рамках проєкту передбачається побудова технологічної інфраструктури, здатної забезпечити безперервний моніторинг параметрів розчину та мікроклімату, автоматичне дозування поживних речовин, ведення журналу даних, формування аналітичних звітів та сповіщення оператора в режимі реального часу.

Основними цілями створення системи є:

– реалізація автоматичного циклу живлення рослин на основі виміряних параметрів pH, EC, температури та вологості повітря;

– підвищення точності дозування поживних речовин завдяки використанню перистальтичних насосів та алгоритмів корекції;

– забезпечення віддаленого моніторингу через телеграм-бот, включно з можливістю отримання графіків за різні часові інтервали;

– централізоване зберігання даних на Linux-сервері та можливість подальшої інтеграції з системами агроаналітики;

– зменшення людського фактору та уникнення помилок при підживленні;

– створення основи для масштабованої IoT-архітектури, яка дозволяє розширити кількість сенсорів та резервуарів без змін у базовій логіці.

Система має забезпечувати автономність роботи, високу точність сенсорних вимірювань, енергоефективність, відмовостійкість та адаптивність до змін технологічних параметрів.

3.2 Розробка вимог до IoT системи

3.2.1 Функціональні вимоги

Функціональні вимоги описують поведінку системи та взаємодію між її компонентами. Для міні-гідропонної теплиці IoT система повинна забезпечувати:

1. Моніторинг параметрів середовища:

– температура повітря $T_{air}(t)$, °C;

– відносна вологість)RH(t), %;

– освітленість L(t), люкси;

– електропровідність поживного розчину EC(t), мСм/см;

– кислотність розчину pH(t).

Параметри вимірюються за допомогою відповідних сенсорів і передаються на контролер в режимі реального часу. Формально, поточний стан середовища можна описати як вектор стану:

$$D(t)=[T_{air}(t),RH(t),L(t),EC(t),pH(t)] \quad (3.1)$$

2. Автоматизоване управління насосами та подачею поживних речовин: Для забезпечення оптимального росту рослин система повинна регулювати подачу поживних розчинів. Керування можна представити як функцію від вектора стану:

$$U(t)=f(D(t),S_{desired}) \quad (3.2)$$

де $U(t)=[Q_A(t),Q_B(t),Q_W(t)]$ – обсяги подачі поживних речовин A, B та води, а $S_{desired}$ – бажані параметри середовища.

3. Збір та збереження даних:

– логування даних у базі даних Linux-сервера;

– можливість збереження даних у CSV та графічному вигляді;

– архівація даних щоденно та формування резервних копій (backups).

4. Інтерфейс користувача через Чатбот:

– отримання актуальних показників середовища;

– відображення графіків за різні періоди: рік, місяць, тиждень, день, година;

– можливість ручного керування насосами;

– сигналізація при аварійних станах (наприклад, EC або pH виходять за допустимий діапазон).

5. Масштабованість системи:

– додавання додаткових резервуарів та насосів;

– підключення додаткових сенсорів та контролерів;

– можливість об’єднання кількох теплиць у єдину мережу.

3.2.2 Технічні вимоги

Технічні вимоги визначають параметри обладнання та технологічні аспекти роботи IoT системи контролю процесів живлення рослин у міні-гідропонній теплиці. Вони забезпечують коректну роботу системи, стабільність роботи датчиків, точність контролю параметрів середовища та безпечну передачу даних до серверної частини.

3.2.2.1 Контролер

В якості центрального контролера системи обрано Raspberry Pi 4 з мінімум 4 ГБ оперативної пам’яті. Основні вимоги до контролера:

– підтримка MicroPython або іншої легковажної мови програмування для реалізації логіки керування.

– GPIO порти: мінімум 26 портів для підключення сенсорів та реле; порти повинні підтримувати PWM, I^2C, SPI, UART.

– обробка даних: контролер повинен забезпечувати обробку даних сенсорів у реальному часі та приймати рішення щодо включення насосів, обігрівачів, вентиляторів.

– захист від перенапруги та коротких замикань: вбудовані засоби захисту на рівні GPIO і реле.

– енергоспоживання: не більше 15 Вт у активному режимі, що дозволяє використовувати автономні джерела живлення у малих і середніх установках.

Для забезпечення надійності роботи рекомендується резервне живлення контролера через UPS модуль на 5–10 Вт, що дозволяє уникнути втрати даних у випадку короткочасного відключення електроенергії.

3.2.2.2 Датчики

Датчики забезпечують збір точних даних про стан середовища та складу поживного розчину наведено в таблиці 3.1

Таблиця 3.1 – Датчики

№	Пристрій	Призначення	Діапазон вимірювань	Точність
1	DS18B20	Температура води	-55…+125 °C	±0.5 °C
2	DHT22	Температура та вологість повітря	0…100% RH / -40…+80 °C	±2% RH / ±0.5 °C
3	BH1750	Освітленість	1…65535 lx	±10%
4	EC сенсор	Провідність поживного розчину	0…20 mS/cm	±0.1 mS/cm
5	pH сенсор	Кислотність/лужність	0…14 pH	±0.1 pH

Додатково можуть бути встановлені датчики рівня рідини в резервуарах (ультразвукові або поплавкові) для контролю запасів поживних розчинів та води.

Контролер обчислює середнє значення вимірювань з сенсорів для усунення шуму:

$$T_{\text{avg}} = \frac{1}{n}\sum_{i=1}^{n} T_i \tag{3.3}$$

де T_i– температура, зафіксована сенсором i, n – кількість вимірів за інтервал часу Δt.

Аналогічно обчислюється середнє значення вологості RH_{avg}, EC_{avg} та pH_{avg}.

3.2.2.3 Виконавчі пристрої

Система керування передбачає автоматизацію подачі поживних розчинів та підтримання оптимальних умов:

– перистальтичні насоси для розподілу поживних речовин А і Б та води; продуктивність насосів підбирається за формулою:

$$Q = V_{\text{reservoir}} / t_{\text{feed}} \tag{3.4}$$

де Q – продуктивність насоса, $V_{reservoir}$ – об’єм резервуара, t_{feed} – час подачі.

– реле для включення/виключення насосів, обігрівачів та вентиляторів. Кожне реле повинно витримувати навантаження відповідного виконавчого пристрою.

– додаткові вентилятори або обігрівачі, якщо температура або вологість виходить за межі оптимальних значень (за алгоритмом контролера).

3.2.2.4 Комунікації та мережа

Для передачі даних до серверної частини передбачено:

– підключення до локальної мережі та Інтернету через Ethernet або Wi-Fi.

– протоколи передачі: MQTT для реального часу та HTTP для архівованих даних.

– безпека даних: TLS/SSL шифрування для всіх MQTT-з’єднань та API-викликів.

– періодичність передачі даних: від 1 хвилини (для температури та вологості) до 1 години (для історичних графіків).

Для масштабування системи до промислових розмірів передбачена можливість підключення до IoT-шлюзів, які агрегують дані від декількох Raspberry Pi, зменшуючи навантаження на сервер.

3.2.2.5 Сервер та база даних

Для зберігання та обробки даних передбачено використання Linux-сервера з реляційною базою даних (PostgreSQL або MySQL). Основні вимоги:

- обробка вхідних даних від декількох контролерів одночасно.
- API для взаємодії з Чатботом та веб-інтерфейсом: REST або GraphQL.
- резервне копіювання та архівування: щоденне створення бекапів бази даних, щотижневе – архівування історичних даних.
- обчислювальна потужність сервера повинна забезпечувати побудову графіків за рік, місяць, тиждень, день та годину без затримок.

Система будує графіки зміни параметрів середовища та поживного розчину. Наприклад, інтеграл для сумарного обсягу води за день:

$$V_{\text{day}} = \int_0^{24h} Q(t)\,dt \tag{3.5}$$

де Q(t) – миттєва продуктивність насоса в літрах за годину.

3.2.2.6 Додаткові вимоги

- Масштабованість системи: підтримка до 10–50 теплиць одночасно.
- Модульність: можливість додавати нові датчики або реле без переробки основної логіки.
- Розширення функціоналу: можливість інтеграції з зовнішніми системами контролю (Smart Home, SCADA).
- Енергетична ефективність: система повинна споживати мінімум електроенергії при автономній роботі.

3.2.3 Експлуатаційні вимоги

Експлуатаційні вимоги визначають умови надійної та безпечної роботи IoT системи для міні-гідропонної теплиці, включаючи умови довкілля, швидкість реакції, безпеку обладнання та зручність користування.

3.2.3.1 Температурний режим та середовище експлуатації

Система повинна функціонувати в діапазоні температур від +5°C до +40°C, що відповідає умовам домашніх та офісних приміщень, а також невеликих теплиць.

- Контролер та сервер повинні працювати без перегріву. Для цього передбачено природне охолодження та/або невеликі вентилятори.
- Температурні датчики повинні бути калиброванні для точності ±0.5°C.
- Температурна компенсація для сенсорів pH та EC реалізується за формулою:

$$EC_{\text{corr}} = EC_{\text{meas}} \times [1 + \alpha \cdot (T - 25)] \tag{3.6}$$

де EC_{meas} – виміряне значення електропровідності, α – температурний коефіцієнт (≈0.019/°C), T – температура розчину в °C.

3.2.3.2 Захист від вологи та пилу

Міні-гідропонна теплиця містить резервуари з водою та насоси, тому необхідний захист електроніки:

- Ступінь захисту IP65 для сенсорів, насосів та реле, що забезпечує захист від пилу та бризок води.
- Контролер розміщується в окремому корпусі з вентиляцією та гідроізоляцією.
- Всі кабелі повинні мати герметичні конектори RJ45, а при проході через корпус – ущільнення.

Це забезпечує довговічність компонентів і зменшує ймовірність короткого замикання.

3.2.3.3 Мінімізація часу відгуку

Система повинна реагувати на зміну параметрів середовища та розчину менш ніж за 1 хвилину, що дозволяє оперативно коригувати умови для рослин.

3.2.3.4 Автоматичне повідомлення про аварійні стани

Система повинна повідомляти користувача про критичні стани через Чатбот, SMS або email:

- підвищення або зниження температури за межі допустимого діапазону;
- рівень вологості або освітленості нижче мінімального;
- критичні значення pH або EC;
- відмова насоса або датчика.

Повідомлення надсилається через сервер, де дані контролюються та аналізуються за алгоритмом:

$$\text{alert} = \begin{cases} 1, & \text{якщо } |P_{\text{current}} - P_{\text{set}}| > \Delta P_{\max} \\ 0, & \text{інакше} \end{cases} \tag{3.7}$$

де $P_{current}$ – поточне значення параметра (температура, вологість, pH, EC)

P_{set} – встановлене оптимальне значення,

ΔP_{max} – допустиме відхилення.

3.2.3.5 Резервне живлення

Для запобігання збоїв при відключенні електроенергії передбачено UPS або літієві батареї:

- Контролер та сенсорні модулі повинні мати автономне живлення на 1–2 години;
- Насоси – резервне живлення на 15–30 хвилин, достатньо для завершення циклу подачі поживного розчину;
- Алгоритм управління включає автоматичне призупинення циклу подачі рідини, щоб уникнути переливу або висихання системи.

3.2.3.6 Дистанційне оновлення програмного забезпечення (OTA)

Система повинна підтримувати онлайн оновлення прошивки та програмного забезпечення:

- безпечне підключення до сервера оновлень через TLS;
- збереження резервної копії попередньої версії прошивки для відкату у випадку помилки;
- можливість масштабування до декількох теплиць одночасно.

Алгоритм оновлення:

1. Контролер отримує інформацію про наявність нової версії;
2. Перевірка контрольної суми для цілісності файлу;
3. Оновлення прошивки сенсорів та виконавчих пристроїв;
4. Перезавантаження системи;
5. Повідомлення користувача про успішне завершення процесу.

3.2.3.7 Додаткові експлуатаційні вимоги

- надійність: MTBF (Mean Time Between Failures) для насосів – не менше 5000 годин, для контролера – 50 000 годин.
- легкість обслуговування: модульна конструкція, легка заміна сенсорів та насосів без втручання в основну логіку.
- інтерфейс користувача: можливість віддаленого моніторингу та керування через веб-інтерфейс або мобільний додаток.
- масштабованість: можливість підключення додаткових модулів контролю освітлення, вентиляції або поживних розчинів.

3.3 Логічна схема системи

Логічна схема IoT-системи відображає взаємодію апаратних та програмних компонентів, порядок обміну даними, логіку прийняття рішень та принципи автоматизованого керування гідропонною теплицею. У межах системи реалізовано багаторівневу архітектуру, що поєднує сенсорний шар,

контролер на базі Raspberry Pi, IoT-шлюз, серверну інфраструктуру, інтерфейси користувача та виконавчі механізми.

Схема забезпечує безперервний моніторинг параметрів середовища, аналіз отриманих даних і формування керуючих сигналів для підтримання оптимальних умов росту рослин.

3.3.1 Загальна структура схеми

Функціональна структура складається з таких логічних блоків(рис. 3.1):

Рисунок 3.1 – Логічна структура логічних блоків

1. Шар збору даних (сенсорні модулі)
2. Шар локальної обробки (Edge-контролер)
3. Комунікаційний шар (MQTT/HTTP-шлюз)
4. Серверний шар (обробка, аналітика, зберігання даних)
5. Шар взаємодії з користувачем (Чатбот, веб-інтерфейс)
6. Шар керування виконавчими пристроями

Функціональна схема передбачає циклічний збір телеметрії, її обробку та формування управляючих впливів відповідно до заданих режимів роботи.

3.3.2 Рівень збору даних

Сенсорна підсистема відповідає за вимірювання ключових параметрів мікроклімату та хімічних характеристик живильного розчину:

- температура повітря та розчину (ds18b20)
- вологість повітря (dht22/sht31)
- освітленість (bh1750)
- електропровідність (ec-сенсор, аналоговий ацп)
- ph (електрод із підсилювачем сигналу)
- рівень води / поживного розчину (ультразвук або поплавкові датчики)

Датчики працюють з інтервалом 5–60 секунд залежно від типу параметра. Зчитування здійснюється Raspberry Pi через шини:

- I2C (BH1750, EC контролери),
- 1-Wire (DS18B20),
- UART або аналогові модулі для pH/EC,
- GPIO для поплавків та реле.

Датчики формують первинні виміряні сигнали, які передаються в блок локальної обробки.

3.3.3 Локальна обробка та прийняття рішень (Edge-контролер)

Raspberry Pi виконує роль польового контролера, який реалізує такі функції:

1. Первинна обробка сигналів:

- корекція та калібрування вимірів (pH, EC, температура розчину);
- фільтрація шуму (медіанний фільтр, ковзне середнє);
- компенсація температури для EC.

2. Логіка контролю процесів

Контролер аналізує поточний стан та порівнює його із заданими цільовими значеннями (setpoint):

- Температура розчину → керування нагрівачем/охолодженням;
- pH → дозування розчинів pH Up / pH Down;
- EC → дозування живильного концентрату;
- Рівень → автоматичне доливання води;
- Освітленість → керування додатковим освітленням (якщо передбачено).

Управління здійснюється з використанням простих порогових алгоритмів або ПІ-регулятора.

3. Формування керуючих сигналів. Контролер передає команди виконавчим пристроям через GPIO/реле:

- ввімкнути насос/вентилятор;
- дозувати певний об'єм;
- зупинити або скинути аварію.

3.3.4 Комунікаційний рівень (передача телеметрії та команд)

Комунікації між контролером та сервером здійснюються через:

- MQTT (основний протокол),
- HTTP(S) (резервний канал або API для Чатбота).

3.3.5 Серверний шар (БД, API)

Сервер виконує ключові функції централізованої обробки інформації:

Дані з MQTT-брокера або REST API надходять у модуль обробки:

- перевірка коректності;
- видалення некоректних значень;
- нормалізація.

Формуються таблиці:

- sensor_readings,
- pump_logs,

- device_state,
- events,
- user_actions,
- configs,
- ota_updates.

Система підтримує архівацію, індексацію за timestamp та агрегування даних.

- побудова часових графіків (EC, pH, температура),
- визначення трендів зростання/споживання,
- статистичні моделі для рекомендацій.

При отриманні критичної події:

- перевіряється відповідність правилам оповіщення,
- формується повідомлення в Чатбот;
- активується сценарій аварійного керування.

3.3.6 Інтерфейси користувача (Чатбот)

- отримання знімку поточного стану (EC, pH, температура);
- перегляд графіків за період;
- отримання аварійних повідомлень;
- ручне дозування або керування насосами/

3.3.7 Виконавча підсистема

- перистальтичних насосів (живлення A/B, вода);
- вентиляторів та клапанів;
- обігрівача або охолоджуючого елемента (за потреби);
- реле та драйверів.

3.3.8 Алгоритм роботи системи

1. Збір даних

Датчики через інтервал передають виміряні параметри контролеру.

2. Локальна обробка

Виконується:

- фільтрація,
- перевірка на вихід за межі,
- підготовка даних до аналізу.

3. Передача телеметрії:

Контролер публікує дані на MQTT-брокер.

4. Серверна обробка:

API приймає та зберігає дані в БД.

5. Аналіз:

Перевірка виходу параметрів за норми:

- pH вище норми, увімкнути насос pH Down,
- EC нижче, дозування концентрату,
- низький рівень води, доливання.

6. Формування керуючих дій:

Сервер або Edge-контролер (залежно від режиму) відправляє команду.

7. Виконання команди:

Насоси працюють відповідно до встановленого алгоритму.

8. Зворотний зв’язок:

Статус операції повертається у БД, дублюється у Чатбот.

3.3.9 Аварійні та спеціальні режими

Передбачено такі режими роботи:

1. Normal Mode: повна автоматизація.
2. Safe Mode: вимкнення дозування при серйозних помилках датчиків.
3. Offline Mode: робота без сервера, локальні алгоритми.
4. Maintenance Mode: ручне керування, тест насосів.
5. Critical Mode: зупинка системи та надсилання термінових повідомлень.

3.4 Аналіз функціоналу та обладнання

3.4.1 Відповідність обладнання функціям системи

Обрані апаратні компоненти формують комплекс, який дозволяє покривати всі ключові функції IoT-рішення:

- Моніторинг мікроклімату: температурні, вологісні й рівневі датчики забезпечують неперервний збір даних у реальному часі. Таким чином система має актуальні параметри для прийняття управлінських рішень;
- автоматичне керування: реле, MOSFET-модулі та виконавчі механізми реалізують алгоритми автоматизації – полив, вентиляція, перемикання клапанів, керування освітленням;
- lистанційний доступ і сповіщення: модулі зв’язку забезпечують інтеграцію з хмарною платформою, мобільним застосунком та Чатботом;
- захист і безперервність роботи: резервне живлення та корпуси із захистом IP65 гарантують стабільність функціонування навіть у змінних умовах довкілля.

Таким чином обладнання підібрано таким чином, що дозволяє повністю реалізувати як базовий функціонал (контроль параметрів), так і розширений (аналітика, сценарне керування, аварійні режими).

3.4.2 Аналіз взаємодії та сумісності обладнання

Компоненти обрано з урахуванням принципів сумісності:

- Електрична сумісність. Використання стандартних робочих напруг 3.3V/5V (датчики, контролер) та 12/24V (насоси, клапани) мінімізує потребу в додаткових перетворювачах і стабілізаторах;
- Логічна/протокольна сумісність. Датчики підтримують поширені протоколи – I2C, OneWire, UART, що забезпечує можливість підключення без конфліктів.
- Модульність. Мікроконтролер має достатню кількість цифрових/аналогових входів для підключення всієї необхідної кількості датчиків і виконавчих механізмів.

– Масштабованість .У разі потреби система підтримує підключення додаткових модулів зв’язку, сенсорів або реле без зміни базової архітектури.

3.4.3 Оцінка продуктивності та можливостей обраного контролера

Центральний контролер забезпечує:

– оперативну обробку даних сенсорів з високою частотою опитування;

– достатню обчислювальну потужність для виконання алгоритмів керування;

– низьке енергоспоживання у режимах очікування;

– інтегровані засоби бездротового зв’язку (Wi-Fi/BLE);

– можливість OTA-оновлень (Over-The-Air) програмного забезпечення.

Ці параметри значно перевищують мінімально необхідні, що робить контролер достатньо продуктивним для системи середньої складності з можливістю подальшого розширення.

3.4.4 Аналіз датчиків

Вибрані моделі DS18B20 або аналогічні забезпечують:

– похибку ±0.5°C;

– стійкість до вологи (у герметичному виконанні);

– можливість роботи у довгих лініях зв’язку.

Використання цифрових сенсорів (DHT22/SHT31) дає:

– цифровий вихід без калібрування;

– сталу точність вимірювання.

Поплавкові або ультразвукові датчики дозволяють:

– виявляти критично низький рівень;

– захищати насос від сухого ходу,

3.4.5 Аналіз виконавчих пристроїв

У структурі IoT-системи контролю процесів живлення рослин у гідропонній теплиці виконавчі пристрої відіграють ключову роль, оскільки

вони забезпечують безпосереднє втручання в технологічний процес відповідно до команд, сформованих контролером на основі даних сенсорів. Ефективність та надійність функціонування гідропонної установки значною мірою залежать від правильного вибору, продуктивності та стійкості таких пристроїв.

1. Насоси циркуляції поживного розчину

Циркуляційні насоси забезпечують переміщення робочого розчину між баком, живильними каналами та фільтрами.

Їх вибір визначається:

– необхідною продуктивністю (л/год), залежно від площі теплиці;

– напором;

– безшумністю роботи;

– можливістю роботи в автоматизованих циклах;

– довговічністю при контакті з мінеральними добривами.

У системі IoT насос вмикається автоматично за сигналами контролера за рівнем заповнення, потребою в аерації чи корекції складу розчину.

2. Дозатори та перистальтичні насоси

Дозуючі пристрої використовуються для автоматичного введення мінеральних добрив, pH-коректорів та мікроелементів.

Переваги перистальтичних насосів:

– висока точність дозування при малих об’ємах;

– відсутність контакту рідини із рухомими частинами;

– можливість тривалої безперервної роботи;

– стійкість до агресивних хімічних реагентів.

У системі гідропоніки такі дозатори забезпечують автоматичну корекцію EC (електропровідності) та pH відповідно до заданих параметрів.

3. Вентиляційні та кліматичні виконавчі пристрої

Хоча основною функцією системи є контроль процесів живлення, стабільне середовище важливе для поглинання поживних речовин. Виконавча частина включає:

– вентилятори, що регулюють циркуляцію повітря;

– серводвигуни або електроприводи для відкриття кватирок/вікон;

– зволожувачі чи осушувачі для підтримання мікроклімату;

– клапани дозованої подачі CO_2 (за наявності системи CO_2-підживлення).

Ці пристрої працюють синхронно з модулями контролю розчину, забезпечуючи оптимальні умови для абсорбції поживних речовин кореневою системою.

4. Освітлювальні модулі

LED-світильники важливий виконавчий елемент, що впливає на інтенсивність фотосинтезу та, відповідно, на потреби рослини в поживному розчині.

Вимоги до світильників у IoT-системі:

– можливість ШІМ-керування інтенсивністю;

– стабільна робота при високій вологості;

– підтримка таймерів добових циклів;

– енергоефективність.

5. Електронні реле та модулі живлення

Для кожного виконавчого елемента передбачено силові модулі:

– реле з гальванічною розв’язкою;

– MOSFET-модулі для LED-освітлення і насосів 12/24V;

– захист від перенавантаження та короткого замикання.

Ці модулі гарантують стабільне живлення виконавчих пристроїв та безпечне переключення високих струмів.

3.5 Розробка принципової схеми IoT системи контролю процесів живлення рослин

Основним елементом системи є мікроконтролер, який повинен мати достатню обчислювальну потужність для обробки даних у реальному часі,

підтримувати роботу з декількома сенсорами та виконавчими пристроями одночасно, а також забезпечувати стійке мережеве підключення. Мінімально необхідний контролер повинен підтримувати Wi-Fi або Ethernet, мати щонайменше 8 аналогових/цифрових входів для сенсорів та 4 керовані виходи для виконавчих модулів. Окрім того, важливою вимогою є наявність захисту від перенапруги та можливість роботи у вологому середовищі при температурі від 0 до +50°C.

Датчики, що застосовуються для вимірювання ключових параметрів — температури повітря, температури розчину, рівня вологості, EC та pH — повинні забезпечувати достатню точність, щоб контролер міг формувати коректні керуючі дії. Для температурних датчиків достатньою є точність ±0.5°C, а для датчиків вологості — похибка не більше ±3 %. Модулі вимірювання pH та електропровідності повинні бути призначені для довготривалої роботи в агресивних середовищах та мати можливість регулярного калібрування.

Виконавчі пристрої, такі як електромагнітні клапани, циркуляційні насоси та перистальтичні дозатори, повинні працювати від стандартних низьковольтних джерел живлення 12–24 В та мати номінальний ресурс не менше 30 000 циклів ввімкнення, що гарантує надійну роботу протягом усього періоду експлуатації теплиці. Також важливо, щоб вони могли керуватися реле чи MOSFET-модулями без додаткових перетворювачів сигналу.

Комунікаційна інфраструктура системи має забезпечувати стабільний канал передачі даних. Мінімально достатньою є підтримка Wi-Fi 2.4 ГГц із рівнем сигналу, що дозволяє працювати на відстані до 20–30 метрів у закритому приміщенні, або дротове Ethernet-підключення, якщо теплиця розташована на віддаленій ділянці. Передбачається, що контролер зможе надсилати дані на сервер з інтервалом не більше однієї хвилини без втрати пакету.

Джерело живлення повинно гарантувати безперервну роботу всієї системи. Мінімально потрібний блок живлення — 12 В або 24 В потужністю

не менше 60 Вт, із захистом від короткого замикання та температурним контролем. Для підвищення надійності також доцільно передбачити резервне живлення, наприклад невеликий акумулятор або UPS, який дозволить системі продовжувати роботу протягом щонайменше 30 хвилин.

Окремим аспектом є захист обладнання. Усі компоненти мають відповідати класу захисту не нижче IP44, що дозволить експлуатувати їх у середовищі з підвищеною вологістю та можливістю прямого контакту з краплями води. Корпуса контролера, сенсорів та виконавчих модулів повинні бути виготовлені з матеріалів, стійких до корозії та хімічного впливу поживних розчинів.

Таким чином, мінімальні вимоги до обладнання формуються з урахуванням умов експлуатації гідропонної теплиці, необхідності стабільного збору даних, можливості точного керування процесами живлення рослин та забезпечення довготривалої роботи системи без значного обслуговування. Їх дотримання гарантує сумісність пристроїв між собою, надійність функціонування IoT-інфраструктури та можливість подальшого розширення системи без повної модернізації обладнання(таблиця 3.2).

Таблиця 3.2 – Мінімальні вимоги до компонентів IoT-системи гідропонної теплиці

Компонент	Мінімальні технічні характеристики	Примітка (роль у системі)
Мікроконтролер IoT	- 32-біт CPU, ≥ 160 МГц - RAM ≥ 320 КБ - Wi-Fi 2.4 ГГц або Ethernet - ≥ 8 GPIO для датчиків - ≥ 4 керовані виходи (реле/MOSFET)	Центральний вузол збору та обробки даних, керування насосами та клапанами
Датчик температури повітря	- Точність ≥ ±0.5°C - Діапазон 0…50°C - Цифровий інтерфейс (I2C/1-Wire)	Контроль мікроклімату та виявлення

Продовження таблиці 3.2

Датчик температури розчину	- Точність ≥ ±0.3°C - Водонепроникний корпус - Діапазон 0…50°C	Вимірювання температури поживного розчину
Датчик вологості повітря	- Точність ≥ ±3 %RH - Діапазон 20–90 %RH	Використовується для управління вентиляцією
Датчик рівня розчину (3 шт.)	- Цифровий або аналоговий - Робоча напруга 5–12 В - Захист від вологи	Резервуари 1/2/3 – контроль рівня води та розчинів
Датчик pH	- Точність ±0.1 pH - Калібрування 4.0 / 7.0 / 10.0	Контроль кислотності
Датчик EC (електропровідності)	- Діапазон 0–20 mS/cm - Температурна компенсація	Контроль концентрації розчину
Циркуляційний насос	- Продуктивність 300–600 л/год - Живлення 12/24 В DC - Захист від сухого ходу	Забезпечення циркуляції поживного розчину
Перистальтичні дозатори	- Живлення 12 В - Дозування 10–100 мл/хв	Додавання добрив (A/B/pH-коректор)
Модуль керування (реле)	- 4-канальне реле або MOSFET - Навантаження ≥ 10 А - Оптоізоляція	Керування насосами та клапанами
Комунікаційний модуль (при потребі)	- Wi-Fi 2.4 ГГц або Ethernet RJ-45 - Підтримка MQTT/HTTP	Передача телеметрії на сервер
Блок живлення	- Вихід 12–24 В - Потужність ≥ 60 Вт - Захист від КЗ та перегрів	Основне живлення системи

Продовження таблиці 3.2

Корпус обладнання	- Клас захисту не нижче IP44 - Ударостійкий пластик або метал	Захист електроніки від вологи
Резервне живлення (опціонально)	- Акумулятор 12 В 4–7 А·год або UPS	Підтримка роботи при зникненні електрики

3.5.1 Вибір елементної бази

Основним компонентом розробки системи обрано мікроконтролер Raspberry Pi Pico W за його компактність, низьке енергоспоживання та наявність вбудованого Wi-Fi модуля, що дозволяє реалізувати бездротовий моніторинг та управління. Нижче наведено опис обраних датчиків та інших компонентів системи, що використовуватимуться для побудови корпоративної мережі та контролю процесів живлення в системі гідропонної теплиці [8].

Далі обрано такі датчики та компоненти як:

1. Датчик температури DS18B20 обраний для вимірювання температури розчину завдяки його високій точності та широкому діапазону вимірювань (-55°C до +125°C). Він підтримує інтерфейс 1-Wire, що дозволяє підключати декілька датчиків до одного порту мікроконтролера, що зменшує кількість необхідних портів введення/виведення.

2. Модуль мультиплексора використовується для розширення кількості доступних входів/виходів Raspberry Pi Pico W. Це дозволяє підключити більше датчиків та інших пристроїв, ніж дозволяє обмежена кількість GPIO на мікроконтролері.

5. OLED-дисплей використовується для виведення інформації про стан системи, включаючи показники датчиків та статус системи. Дисплей забезпечує високу контрастність та чіткість зображення, що дозволяє легко зчитувати дані навіть при яскравому освітленні.

6. Годинник реального часу DS3231 обрано для забезпечення точного часу та дати. Це необхідно для синхронізації процесів у системі та ведення логів. DS3231 має високу точність завдяки вбудованому кварцовому генератору.

7. Датчик наявності води використовується для визначення наявності або відсутності води у резервуарах системи. Це важливо для запобігання пересиханню та забезпечення якісного живильного розчину.

8.EC-метр DFR0300 використовується для вимірювання електропровідності (EC) розчину в гідропонній системі. Це дозволяє контролювати концентрацію поживних речовин у воді, що є критичним параметром для забезпечення оптимальних умов росту рослин.

9. Датчик pH SEN0161-V2 використовується для вимірювання рівня pH розчину в гідропонній системі. Це важливий параметр для підтримки оптимальних умов росту рослин. Датчик забезпечує точні вимірювання та має тривалий термін служби.

10. 4-х канальне реле використовується для керування кількома пристроями одночасно. Це дозволяє централізовано керувати різними аспектами системи, такими як полив, освітлення та вентиляція.

У якості виконавчих пристроїв обрано такі компоненти як:

Перистальтичний насос 5В який призначений для перекачування рідин з високою точністю і контролем. Завдяки своїй конструкції, такий насос забезпечує рівномірну подачу живильного розчину без забруднення, що є важливим для гідропонних систем. Він працює від напруги 5В і має продуктивність 160 мл/хв. Перистальтичні насоси відомі своєю надійністю і здатністю перекачувати різні типи рідин, включаючи агресивні хімічні розчини, оскільки рідина не контактує з механічними частинами насоса.

Занурювальна помпа 5В пристрій, що використовуються для переміщення рідин у системах з низьким енергоспоживанням. Такі помпи можуть бути використані для циркуляції води в невеликих гідропонних системах або для подачі води в інші частини системи. Вони працюють від

напруги 5В, що дозволяє їх легко інтегрувати з мікроконтролерами та іншими низьковольтними системами. Помпи 5В забезпечують надійність і довговічність, а також можуть мати різну продуктивність залежно від конкретної моделі та виробника, що дозволяє вибрати оптимальний варіант для конкретних завдань у системі гідропоніки [9].

Таблиця 3.3 – Специфікація компонентів

№	Назва компоненту	Кількість	Входи/Виходи	Живлення
1	Raspberry Pi Pico W	1	26 GPIO	5 DC
2	Датчик температури DS18B20	1	1-Wire	3.3 DC
3	Модуль мультиплексора	1	I2C	3.3 DC
4	Перистальтичний насос	3	постійний струм	5 DC
5	Занурювальна помпа	1	постійний струм	5 DC
6	OLED-дисплей	1	I2C	3.3/5 DC
7	Годинник реального часу DS3231	1	I2C	5 DC
8	Датчик наявності рідин	3	Аналоговий	3.3/5 DC
9	EC-метр DFR0300	1	Аналоговий	5 DC
10	Датчик pH SEN0161-V2	1	Аналоговий	5 DC
11	4-х канальне реле	1	Цифровий	3.3/ DC

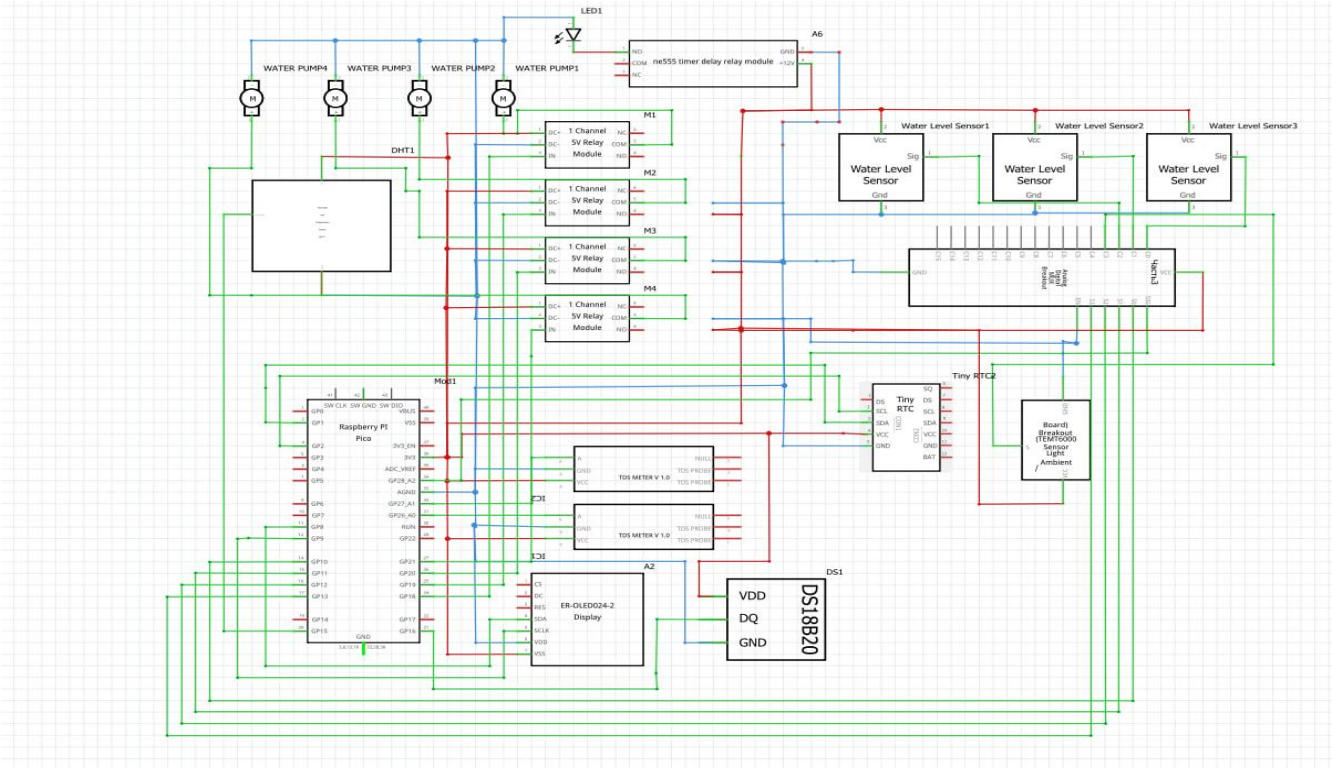

Рисунок 3.1 – Принципова схема

Комплектація IoT системи пропонується така (таблиця 3.4).

Таблиця 3.4 – Комплектація системи

№	Компонент	Кількість	Призначення
1	Raspberry Pi Pico / Pico W	1	Центральний контролер, збір даних та керування виконавчими пристроями.
2	DS18B20	1	Вимірювання температури живильного розчину.
3	TDS Sensor V1.0	2	Визначення концентрації солей у розчині.
4	Датчики рівня води	3	Контроль рівня води в резервуарах.
5	DHT22	1	Контроль температури та вологості повітря.

Продовження таблиці 3.4

6	Датчик освітленості BH1750	1	Моніторинг освітленості в теплиці.
7	RTC-модуль Tiny RTC	1	Зберігання точного часу для логування та таймерів.
8	Водяні насоси 12 В	1	Подача живильного розчину до кореневої системи рослин.
9	Релейний модуль 5 В (4-канальний)	1	Керування насосами та іншими навантаженнями.
10	Модуль реле/SSR для освітлення	1	Керування світильниками теплиці.
11	OLED дисплей 0.96" I2C	1	Локальне відображення показників датчиків.
12	Wi-Fi модуль ESP8266 / Pico W	1	Передача телеметрії на сервер або у хмару.
13	Блок живлення 12 В 10 А	1	Живлення насосів та реле.
14	DC-DC Step-Down 12→5 В 3 А	1	Стабілізоване живлення датчиків та контролера.
15	UTP Cat5e/Cat6 кабель	~10–20 м	Передача сигнальних ліній між компонентами.
16	Багатожильний силовий кабель 0.75–1 мм²	~10–15 м	Живлення насосів та реле.
17	Клемники, Dupont/Molex роз’єми	комплект	Модульне підключення датчиків й модулів.
18	Корпус (IP65)	1	Захист

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ІОТ СИСТЕМИ КОНТРОЛЮ ПРОЦЕСІВ ЖИВЛЕНН

4.1 Створення модифікованої БД для системи контролю гідропонних теплиць

Оновлена база даних забезпечує:

Безпечний доступ до даних:

1. Користувачі та теплиці ідентифікуються через логін/пароль.
2. Кожен користувач бачить лише свої теплиці та їхні дані.

Валідацію та контроль даних:

1. Значення температури, вологості та станів резервуарів перевіряються на коректність.
2. Використання CHECK-констрейнтів забезпечує відповідність фізичним та логічним межам.

Журналювання та аудит:

1. Кожна дія користувача або теплиці фіксується у таблиці audit_log з IP-адресою та часовою міткою.
2. Це відповідає вимогам ГОСТ щодо контролю доступу та аудиту.

Для розгортання системи було обрано середовище PostgreSQL Для його підготовки було перевірено наявність сервера та встановлено необхідні пакети на Debian/Ubuntu за допомогою команд sudo apt update та sudo apt install postgresql postgresql-contrib. Пакет postgresql забезпечує сам сервер баз даних, тоді як postgresql-contrib надає додаткові утиліти та розширення, що використовуються для шифрування, роботи зі статистикою та JSON. Стан сервера було перевірено командою sudo systemctl status postgresql, що показало активний статус (active running), а для керування службою використовували sudo systemctl start, stop або restart відповідно до потреб адміністратора.

Підключення до сервера здійснювалося під системним користувачем PostgreSQL за допомогою sudo -u postgres psql, що надає права адміністратора

для створення баз даних, користувачів та виконання SQL-команд. Спершу було створено нового користувача greenhouse_user з комплексним паролем СкладнийПароль123! командою CREATE USER greenhouse_user WITH PASSWORD 'СкладнийПароль123!';. Пароль відповідає сучасним стандартам безпеки: він має більше 12 символів і містить великі та малі літери, цифри та спеціальні символи. Далі було створено базу даних greenhouse_db з власником greenhouse_user, кодуванням UTF8 та шаблоном template0командою:

```
CREATE DATABASE greenhouse_db
OWNER greenhouse_user
ENCODING 'UTF8'
TEMPLATE template0;
```

Таке налаштування гарантує чистоту бази та підтримку всіх символів, що особливо важливо для назв теплиць та логів. Користувачу greenhouse_user було надано повні права на базу даних через GRANT ALL PRIVILEGES ON DATABASE greenhouse_db TO greenhouse_user;, що дозволяє створювати та модифікувати таблиці, додавати та видаляти записи. Після цього підключення до бази здійснювалося командою \c greenhouse_db, що перемикає виконання SQL-команд на новостворену базу.

У базі даних було створено таблицю users, де кожен користувач ідентифікується автоматично генерованим числовим ідентифікатором SERIAL PRIMARY KEY, а поле username обмежене форматом через CHECK, що гарантує коректність введення. Паролі зберігаються у вигляді хешів у полі password_hash, забезпечуючи безпечне зберігання та уникнення відкритих паролів. Для контролю доступу користувачам присвоюються ролі user або admin. Таблиця greenhouses зв’язує кожну теплицю з користувачем через зовнішній ключ user_id з каскадним видаленням (ON DELETE CASCADE), а поле access_key зберігає хешований ключ контролера для безпечної аутентифікації IoT-пристроїв.

Для збору телеметрії було створено таблицю telemetry, де кожен запис прив’язаний до конкретної теплиці.

Поля air_temperature, solution_temperature та humidity використовують тип NUMERIC(5,2) з обмеженням через CHECK, що забезпечує фізично коректні значення. Булеві поля reservoir1_level, reservoir2_level, reservoir3 _level та lighting_state зберігають стани обладнання, а поле timestamp автоматично фіксує час запису.

Для контролю безпеки було створено таблицю audit_log, яка фіксує всі дії користувачів із зазначенням IP-адреси (INET) та часу події. Це дозволяє здійснювати аудит та розслідування інцидентів. Додатково були застосовані обмеження доступу: усім користувачам окрім greenhouse_user доступ до бази був відкликаний через REVOKE ALL ON DATABASE greenhouse_db FROM PUBLIC;, а права на таблиці надані тільки необхідні для роботи сервісу командою GRANT USAGE, SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO greenhouse_user;.

Для захисту даних під час передачі клієнтами (Чатбот, теплична система) було включено SSL у конфігурації PostgreSQL (ssl = on), що забезпечує шифрування каналів і запобігає перехопленню даних. Резервне копіювання бази здійснювалося щоденно за допомогою pg_dump -U greenhouse_user -F c greenhouse_db > greenhouse_db_backup_$(date +%F).dump, а відновлення виконується через pg_restore, що гарантує збереження даних та можливість масштабування системи як вертикально, так і горизонтально за допомогою PostgreSQL Replication.

Таким чином, було налаштовано повноцінну безпечну базу даних, створено користувачів, таблиці з ключами та констрейнтами, реалізовано безпечний доступ, шифрування з’єднань та механізми резервного копіювання, що забезпечує стабільну і надійну роботу всієї системи тепличного моніторингу. Лістинги створення та налаштувань наведено у Додатку А.

Таблиця 4.1 – Таблиця user

Поле	Тип	Призначення
user_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор користувача
username	VARCHAR(50) UNIQUE NOT NULL	Логін користувача, контроль формату через CHECK
password_hash	TEXT NOT NULL	Захищений хеш пароля (bcrypt/SCRAM-SHA-256)
чатбот_id	BIGINT UNIQUE	ID користувача Чатбот
role	VARCHAR(20) DEFAULT 'user'	Розмежування ролей: 'admin' або 'user'
created_at	TIMESTAMP DEFAULT NOW()	Дата та час створення облікового запису

Таблиця 4.2 – Таблиця greenhouse

Поле	Тип	Призначення
greenhouse_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор теплиці
user_id	INT NOT NULL	Посилання на власника (users.user_id)
name	VARCHAR(100) NOT NULL	Назва теплиці
location	TEXT	Розташування теплиці
access_key	TEXT NOT NULL	Захищений хеш ключа доступу IoT-контролера
created_at	TIMESTAMP DEFAULT NOW()	Дата та час створення

Таблиця 4.3 – Telemetry

Поле	Тип	Призначення
telemetry_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор запису
greenhouse_id	INT NOT NULL	Посилання на теплицю
air_temperature	NUMERIC(5,2)	Температура повітря (-50…+80°C)
solution_temperature	NUMERIC(5,2)	Температура розчину (-10…+60°C)
reservoir1_level	BOOLEAN	Стан резервуара 1

reservoir2_level	BOOLEAN	Стан резервуара 2
reservoir3_level	BOOLEAN	Стан резервуара 3
lighting_state	BOOLEAN	Стан освітлення
humidity	NUMERIC(5,2)	Вологість (0…100%)
timestamp	TIMESTAMP DEFAULT NOW()	Час вимірювання

Таблиця 4.4 – Audit log

Поле	**Тип**	**Призначення**
log_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор запису
user_id	INT	Посилання на користувача
action	VARCHAR(255)	Опис дії (вхід, відправка даних, помилка)
ip_address	INET	IP-адреса джерела дії
timestamp	TIMESTAMP DEFAULT NOW()	Час дії

4.2 Розробка чат боту системи комплексу гідропонної теплиці

4.2.1 Архітектура та структура бота

Чатбот реалізовано на основі Python із використанням бібліотек Aiogram та asyncpg для асинхронної роботи з базою даних PostgreSQL. Архітектура бота передбачає:

1. Клієнтська частина: Чатбот-клієнт користувача.

2. Серверна частина: Бот на Python, що обробляє повідомлення, кнопки та виклики бази даних.

3. База даних: PostgreSQL, яка зберігає користувачів, теплиці, телеметрію та журнали дій.

```
hydroponics_bot/
│
├─ bot.py            # Основний файл запуску бота
├─ db.py             # Підключення до PostgreSQL та запити
├─ config.py         # Токени, параметри БД
├─ utils.py          # Генерація графіків
└─ requirements.txt  # Необхідні пакети Python
```

Рисунок 4.1 – Файлова структура бота

4.2.2 Авторизація користувачів

Авторизація є критично важливою складовою кіберфізичної системи теплиць, оскільки вона гарантує, що лише уповноважені особи отримують доступ до даних телеметрії та керування обладнанням. В нашій системі реалізовано багаторівневий підхід до авторизації, що включає перевірку логіну та пароля, а також прив’язку Чатбот-ID користувача.

Процес авторизації складається з трьох основних кроків:

Користувач надсилає команду /start у Чатбот. Бот реагує на цю команду, запитуючи логін користувача. Введене значення перевіряється на відповідність регулярному виразу, що визначає допустимі символи та довжину (3–50 символів, лише латинські літери, цифри та підкреслення).

Після введення логіну бот запитує пароль. Пароль у базі даних зберігається у вигляді хешу з використанням алгоритму bcrypt, що забезпечує стійкість до атак методом «грубої сили» та запобігає зчитуванню оригінального пароля навіть у випадку компрометації бази.

Перевірка здійснюється шляхом порівняння введеного пароля з хешем у базі:

```
if bcrypt.checkpw(entered_password.encode(), stored_hash.encode()):
    # Авторизація успішна
else:
    # Пароль невірний
```

Після успішної перевірки логіну та пароля бот зберігає чатбот_id користувача у таблиці users. Це дозволяє надалі ідентифікувати користувача

за його акаунтом у Чатбот і обмежити доступ до даних лише для власника теплиці.

Таблиця users у базі даних має наступні поля:

Таблиця 4.5 – Структура таблиці users

Поле	Тип	Опис
user_id	SERIAL PRIMARY KEY	Унікальний ідентифікатор користувача
username	VARCHAR(50) UNIQUE NOT NULL	Логін користувача
password_hash	TEXT NOT NULL	Хеш пароля користувача
чатбот_id	BIGINT UNIQUE	Чатбот-ID користувача, для прив’язки акаунта

Продовження табилці 4.5

role	VARCHAR(20) DEFAULT 'user'	Роль користувача (admin, user)
created_at	TIMESTAMP DEFAULT NOW()	Час створення акаунта

1. PRIMARY KEY (user_id) гарантує унікальність кожного користувача.

2. UNIQUE (username, чатбот_id) забезпечує неможливість дублювання логінів або прив’язки одного Чатбот акаунта до кількох користувачів.

3. role дозволяє реалізувати різні рівні доступу (наприклад, адміністратор може переглядати дані всіх теплиць, звичайний користувач — лише свої).

Безпека авторизації:

– використання хешування паролів та регулярних виразів запобігає зберіганню та використанню небезпечних або слабких паролів;

– прив’язка Чатбот-ID дозволяє ідентифікувати користувача без додаткових паролів для кожної сесії, але тільки на підтвердженому акаунті;

– усі дії авторизації можуть логуватися у таблиці audit_log, що забезпечує аудит та контроль доступу до системи.

Алгоритм роботи авторизації:

1. Користувач надсилає /start.

2. Бот запитує логін, користувач вводить, перевірка формату.

3. Бот запитує пароль, користувач вводить, перевірка хешу у базі.

4. Якщо пароль вірний, запис Чатбот-ID, користувач отримує доступ до даних теплиці.

5. Якщо пароль невірний, бот повідомляє про помилку та блокує доступ до даних.

Цей алгоритм гарантує, що до телеметрії теплиці мають доступ тільки авторизовані користувачі та що інформація захищена від несанкціонованого доступу.

4.2.3 Розробка інтерфейсу чатботу

Основна мета бота – забезпечити користувачу дистанційний моніторинг кіберфізичної системи гідропонних теплиць. Функціонал бота реалізовано через командний та кнопковий інтерфейси, які взаємодіють із базою даних PostgreSQL.

Користувач отримує актуальні показники з теплиці:

1. Температура повітря (air_temperature).

2. Температура розчину (solution_temperature).

3. Рівень резервуарів (reservoir1_level, reservoir2_level, reservoir3_level).

4. Стан освітлення (lighting_state).

5. Вологість повітря (humidity).

Ці дані надходять із таблиці telemetry через унікальний greenhouse_id, прив’язаний до користувача.

Користувач обирає команду /status. Бот отримує Чатбот-ID користувача та знаходить відповідний user_id у таблиці users. Визначається greenhouse_id користувача у таблиці greenhouses. Виконується SQL-запит:

```
SELECT air_temperature, solution_temperature, reservoir1_level, reservoir2_level, reservoir3_level, lighting_state, humidity, timestamp
FROM telemetry
WHERE greenhouse_id = $1
ORDER BY timestamp DESC
LIMIT 1;
```

Дані відправляються користувачу у текстовому або графічному вигляді.

Графіки дозволяють користувачу аналізувати динаміку показників теплиці за різні періоди:

– година;

– день;

– тиждень;

– місяць;

– рік.

Також можна обрати всі показники одночасно або окремо кожен параметр.

Запити формуються динамічно залежно від періоду та показника:

```
SELECT timestamp, air_temperature, solution_temperature, humidity, reservoir1_level, reservoir2_level, reservoir3_level, lighting_state
FROM telemetry
WHERE greenhouse_id = $1
AND timestamp >= NOW() - INTERVAL '1 week'
ORDER BY timestamp ASC;
Інтервал змінюється для 1 hour, 1 day, 7 days, 1 month, 1 year.
```

Для окремих показників вибирається лише потрібне поле.

Для побудови графіків використовується matplotlib.

Кожен показник має свій колір та легенду.

При виборі одного показника будується окремий графік.

Для групового перегляду (Всі) графіки об’єднуються на одному полотні.

Графік зберігається у буфері пам’яті та надсилається користувачу у форматі PNG.

```
import matplotlib.pyplot as plt
from io import BytesIO
fig, ax = plt.subplots()
ax.plot(timestamps, air_temp, label='Температура повітря')
ax.plot(timestamps, solution_temp, label='Температура розчину')
ax.legend()
buf = BytesIO()
plt.savefig(buf, format='png')
buf.seek(0)
bot.send_photo(chat_id=user_id, photo=buf)
```

Для зручності користувача використовується InlineKeyboardMarkup:

Перший рівень: вибір періоду (Година, День, Тиждень, Місяць, Рік).

Другий рівень: вибір показника (Всі, Температура повітря, Температура розчину, Вологість, Рівень резервуарів, Освітлення).

Алгоритм:

1. Користувач натискає кнопку періоду.

2. Бот зберігає вибір у сесії користувача.

3. Відображаються кнопки показників.

4. Користувач обирає показник – бот генерує графік.

Після отримання команди /start бот розпочинає процес автентифікації. На першому етапі бот визначає Чатбот-ID користувача та звіряє його з таблицею users, щоб визначити, чи пов’язаний користувач із системою. Якщо Чатбот-ID відсутній, бот надсилає запит на введення логіну та пароля.

Користувач вводить логін, після чого бот запитує пароль. Введені дані передаються до сервера, де пароль проходить процедуру перевірки через порівняння формованого хешу з хешем у полі password_hash таблиці users. У разі успіху бот прив’язує Чатбот-ID до користувача (якщо він ще не був прив’язаний), тим самим завершуючи авторизацію. Невдалі спроби

записуються в таблицю audit_log із зазначенням часу, IP-адреси та дії — для подальшого аудиту та кібербезпеки.

Після успішної авторизації користувач отримує доступ до функцій бота відповідно до ролі (звичайний користувач чи адміністратор).

Команда /status забезпечує швидкий перегляд останніх показників роботи теплиці. Після її отримання бот визначає, які саме теплиці закріплені за авторизованим користувачем (через таблицю greenhouses). Далі бот формує SQL-запит до таблиці telemetry, вибираючи останній запис (за максимальним значенням timestamp).

Бот відображає такі параметри:

– температура повітря;

– температура поживного розчину;

– вологість;

– стани всіх резервуарів (порожній/нормальний);

– стан освітлення;

– точний час надходження вимірювання.

Команда /graph активує кнопковий інтерфейс, який допомагає користувачу обрати тип аналізу. Бот надсилає набір кнопок (InlineKeyboardMarkup), що включають:

вибір періоду: рік, місяць, тиждень, день, година;

вибір параметра: загальний графік або окремий показник (температура повітря, розчину, вологість, резервуари, освітлення).

Це спрощує навігацію та мінімізує кількість текстових команд, забезпечуючи зручну взаємодію з ботом.

Бот обробляє всі вибори користувача за допомогою callback-запитів. Кожна кнопка передає callback-дані у структурованому вигляді, наприклад:

– period_month

– metric_air_temperature

– metric_all

– period_week;metric_humidity

Після отримання callback-події бот:

– зчитує дані вибору;

– зберігає проміжні значення (наприклад, вибраний період) у внутрішньому стані користувача або у Redis/словнику;

– перевіряє, чи вибрав користувач уже і період, і параметр;

– у разі готовності викликає функцію генерації графіка.

Функція графіка надсилає SQL-запит до таблиці telemetry, обмежуючи його за періодом та greenhouse_id користувача. Дані передаються в модуль побудови графіків (matplotlib), після чого готове зображення відправляється користувачу як файл.

Такий механізм дозволяє реалізувати складну логіку вибору в інтерактивному режимі.

4.2.4 Робота з базою даних

При запуску бота відбувається підключення до сервера бази даних PostgreSQL за допомогою бібліотеки psycopg2. Параметри підключення включають назву бази даних, ім’я користувача, пароль та адресу хоста. Використання параметризованих запитів дозволяє забезпечити безпечне виконання SQL-команд та запобігти SQL-ін’єкціям. Після встановлення з’єднання бот готовий обробляти запити користувачів та передавати дані теплиць.

```
import psycopg2
conn = psycopg2.connect(
    dbname="hydroponics",
    user="bot_user",
    password="strongpass123",
    host="localhost"
)
cursor = conn.cursor()
```

Валідація користувача починається з перевірки наявності його Чатбот-ID у таблиці users. У разі успішної перевірки бот запитує пароль, який

порівнюється з хешованим паролем у базі даних. Використання хешування забезпечує, що паролі зберігаються у безпечному вигляді і не можуть бути прочитані у разі компрометації сервера. Прив’язка Чатбот-ID до конкретного користувача гарантує, що кожен користувач бачить лише дані, які належать його власній теплиці або теплицям, якщо їх декілька. Невдалі спроби входу автоматично фіксуються в таблиці audit_log для подальшого аудиту та аналізу безпеки.

```
    cursor.execute("SELECT id, password_hash FROM users WHERE чатбот_id = %s", (чатбот_id,))
    user = cursor.fetchone()
    import bcrypt
    if bcrypt.checkpw(input_password.encode(), user[1].encode()):
        # авторизація успішна
```

Після успішної авторизації бот отримує список теплиць користувача з таблиці greenhouses. Таблиця greenhouses містить унікальний ідентифікатор теплиці, прив’язку до користувача через user_id, назву теплиці, її розташування та ключ доступу. Ключ доступу використовується для підтвердження легітимності теплиці при надсиланні даних на сервер. Всі подальші операції бота з даними телеметрії виконуються тільки для вибраної теплиці, що гарантує ізоляцію даних між користувачами та дотримання принципів конфіденційності.

```
    cursor.execute("""
        SELECT id, name FROM greenhouses WHERE user_id = %s
    """, (user_id,))
    greenhouses = cursor.fetchall()
```

Дані сенсорів надходять на сервер у вигляді структурованих повідомлень, що включають температуру повітря та розчину, вологість, стан трьох резервуарів та освітлення. Перед записом у таблицю telemetry бот перевіряє відповідність ключа доступу та значень параметрів допустимим діапазонам, що реалізується через обмеження CHECK у базі даних. Кожен запис містить часову мітку, яка дозволяє проводити аналітику за будь-який період, включаючи рік, місяць, тиждень, день або годину. Таблиця telemetry

забезпечує зв’язок з таблицею greenhouses через зовнішній ключ greenhouse_id, що гарантує правильне відображення даних конкретної теплиці та користувача.

```
cursor.execute("""
    INSERT INTO telemetry(greenhouse_id, air_temp, solution_temp, humidity, tank1, tank2, tank3, light, timestamp)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, NOW())
""", (gh_id, air, sol, hum, t1, t2, t3, light))
conn.commit()
```

Однією з ключових функцій бота є надання графічного аналізу даних телеметрії. Після отримання запиту від користувача бот формує SQL-запит до таблиці telemetry, обмежуючи вибірку значеннями greenhouse_id та часовим інтервалом. Отримані дані обробляються у вигляді графіків, які відображають зміни температури, вологості, рівнів резервуарів та освітлення за обраний період. Користувач може переглядати загальний графік для всіх показників або окремо по кожному параметру. Така організація дозволяє забезпечити детальний аналіз стану теплиці та своєчасне виявлення аномалій у роботі системи.

```
cursor.execute("""
    SELECT timestamp, air_temp
    FROM telemetry
    WHERE greenhouse_id = %s AND timestamp >= NOW() - INTERVAL %s
    ORDER BY timestamp
""", (gh_id, interval))
data = cursor.fetchall()
```

Всі дії користувача та системні події логуються в таблиці audit_log, включаючи успішні та невдалі спроби входу, запити телеметрії та побудову графіків. Це дозволяє відстежувати активність користувачів, виявляти підозрілі дії та забезпечувати відповідність стандартам кібербезпеки, зокрема вимогам ГОСТ та ISO 27001. Логування також сприяє підтримці історії взаємодій, що необхідно для аудиту та аналізу ефективності системи.

Безпека при роботі з базою даних забезпечується використанням хешованих паролів, обмеженням доступу користувачів лише до власних

теплиць, параметризованими SQL-запитами та захищеним з’єднанням. Адміністратори мають розширені права для управління таблицями та проведення аудиту, що дозволяє ефективно контролювати роботу всієї системи та забезпечує високий рівень кібербезпеки.

4.3 Доопрацювання програми мікроконтроллера

4.3.1 Розробка формату пакета телеметрії для гідропонной теплиці

Зміна структури бази даних вимагала додати до програми мікроконтролера можливість передавати у кожному пакеті телеметрії унікальний ідентифікатор теплиці. Для цього в коді з’явився новий параметр greenhouse_id, який зберігається як звичайна змінна у прошивці. Його значення задається вручну під час встановлення системи або вшивається у прошивку конкретного контролера. Фрагмент коду, який реалізує цю зміну, виглядає так:

```
greenhouse_id = 44  # Унікальний ID теплиці
```

Під час формування пакету даних цей параметр додається на перше місце, щоб сервер міг чітко визначити джерело телеметрії. Додавання ID у структуру пакету необхідне для взаємодії з новою таблицею telemetry, де поле greenhouse_id є обов’язковим.

Попередня структура даних була нерегламентованою, тому сервер змінювався разом з прошивкою. Після оновлення використовується чітко визначений формат, де дані упаковуються через struct.pack(). Тепер першим полем є ID теплиці, потім час, стан рівнів води та сенсорні показники.

Нова структура пакета передається так:

```
data = [
    greenhouse_id,
    current_time[0], current_time[1], current_time[2],
    current_time[4], current_time[5],
    water_level_1,
    temp, hum, wtemp,
    ph, conductivity,
    light_level
```

```
]
bytes_data = struct.pack('i6i6f', *data)
```

Таким чином, сервер отримує уніфікований пакет і не потребує додаткових умов перевірки параметрів.

4.3.2 Уніфікація збирання телеметрії перед формуванням пакета

Раніше окремі датчики — зокрема pH, EC та рівні води — зчитувалися у різний час. Це могло призводити до відхилень між параметрами одного пакету. Програма була переписана так, що всі сенсорні значення знімаються послідовно перед формуванням пакета.

Наприклад, зчитування основних параметрів тепер виглядає так:

```
temp, hum = measure_temperature()
wtemp = read_wtemperature()
ph = read_ph(slope, intercept)
conductivity = voltage_to_mS_cm(read_voltage_ec())
light_level = convert_to_lumens(read_sensor(0))
water_level_1 = read_sensor(4)
```

Усі ці значення утворюють один «знімок стану теплиці», що робить телеметрію більш точною.

4.3.3 Оновлення алгоритму відправлення даних на сервер

Алгоритм відправлення був перероблений так, щоб забезпечити передбачувану логіку та зменшити ймовірність розривів TCP-з’єднання. У попередній версії передача була розкидана по умовних блоках, і час відправки залежав від стану інших процесів.

Нова реалізація виглядає так:

```
try:
    s = usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM)
    s.connect((server_ip, server_port))
    s.send(bytes_data)
    print("Дані успішно відправлені")
except Exception as e:
    print("Помилка передачі:", e)
```

```
finally:
    s.close()
```

Через це передавання стало стабільнішим, а сервер тепер завжди отримує повний пакет, незалежно від внутрішніх станів мікроконтролера.

4.3.4 Розширення сумісності з оновленою базою даних

Оскільки таблиця telemetry була оновлена, код мікроконтролера повинен надавати дані у правильних форматах — від числових температурних значень до плаваючих значень pH та EC. Програма приводить усі величини до відповідних типів, які очікує сервер.

Наприклад, обчислення pH виконується так:

```
voltage = read_voltage_ph()
ph = slope * voltage + intercept
А електропровідність:
voltage_ec = read_voltage_ec()
conductivity = voltage_to_mS_cm(voltage_ec)
```

Ці параметри без додаткових перетворень заносяться у пакет, що відповідає формату таблиці PostgreSQL та дозволяє серверу відразу зберегти їх без повторної обробки.

Всі зміни в коді мікроконтролера спрямовані на те, щоб зробити передачу телеметрії структурованою, стандартизованою та сумісною з новою моделлю бази даних і Чатботом. Додавання ідентифікатора теплиці, новий формат пакета, уніфікований збір даних та стабільний алгоритм передачі забезпечують коректну роботу системи в умовах масштабування та багатокористувацького доступу.

5 ЕКСПЕРИМЕНТАЛЬНИЙ РОЗДІЛ

5.1 Мета і завдання експерименту

Цей етап досліджень орієнтований на виявлення проблемі помилок, які можуть впливати на взаємодію користувачів з ботом. Основна мета − перевірити, чи правильно бот виконує свої функції для кінцевих користувачів. Перевірити, чи відповідає клієнтський функціонал технічним вимогам і чи всі функції працюють відповідно до специфікації. Методи перевірки клієнтського функціоналу: − виконання типових сценаріїв використання бота; − перевірка всіх доступних команд, кнопок, меню.

Ціль перевірки клієнтського функціоналу: Виявити помилки у роботі бота, такі як некоректні відповіді, зависання, пропуск повідомлень або не зрозуміле повідомлення користувачу.

Наступне тестування механізму аутентифікації користувачів

Мета — перевірити коректність роботи системи аутентифікації, яка визначає, чи має користувач право взаємодіяти з Чатботом і отримувати дані про гідропонну теплицю. Аутентифікація є базовим етапом, що забезпечує захист від несанкціонованого доступу до IoT-системи, параметрів живлення та телеметрії.

Першим завданням є перевірка механізму звірки користувача за його Чатбот ID.

Методи перевірки аутентифікації:

– додавання користувача в базу як дозволеного абонента та перевірка доступу;

– спроба запуску бота з Чатбот-акаунтів, яких немає в системі;

– перевірка коректності обробки невідомих або заблокованих ID;

– перевірка повідомлень, що надсилаються при відмові у доступі;

– аналіз логування подій аутентифікації.

Ціль тестування аутентифікації:

– переконатися, що бот дозволяє взаємодію тільки авторизованим користувачам;

– перевірити захист телеметрії, журналів подій та команд керування від сторонніх осіб;

– гарантувати, що Чатбот не передає дані теплиці та не виконує команди для неавтентифікованих користувачів.

Таким чином, експеримент з аутентифікації охоплює повну перевірку доступу користувачів до бота, коректність процесу перевірки Чатбот ID, логування спроб доступу та механізм відмови у взаємодії. Це забезпечує базовий рівень безпеки перед виконанням клієнтських та адміністративних функцій бота.

5.2 Опис умов експерименту

5.2.1 Тестове середовище

Тестування Чатбота виконувалося у середовищі, що повністю відповідає реальним умовам експлуатації IoT-системи гідропонної теплиці. Для тестування застосовувалося таке середовище:

– Чатбот працював у режимі розробки, використовуючи офіційний Чатбот Bot API та фреймворк Aiogram 3.22.0;

– базою даних для зберігання користувачів, теплиць та телеметрії слугувала PostgreSQL, до якої доступ здійснювався через консольні утиліти psql та ДБ-клієнт (PgAdmin або TablePlus);

– взаємодія між ботом і базою здійснювалася через Python-бібліотеку asyncpg.

Таке тестове середовище дозволило змоделювати реальні умови функціонування системи, включаючи роботу з телеметрією, таблицею користувачів та механізмом аутентифікації.

5.2.2 Інструменти та методи тестування

Тестування Чатбота проводилося вручну з використанням мобільного Чатбот-клієнта. Перевірка виконувалася шляхом надсилання текстових команд, натискання інтерактивних кнопок і аналізу відповідей бота.

Основні інструменти тестування:

– Чатбот-клієнт (Android/Windows);

– PostgreSQL (psql, PgAdmin, TablePlus);

– базові SQL-запити для перевірки фактичних змін у БД під час роботи бота.

Метод тестування полягав у виконанні послідовних команд бота та аналізі:

– відповіді бота;

– дій у базі даних (створення користувача, прив’язування теплиці, запис телеметрії);

– роботи механізму аутентифікації за Чатбот ID;

– коректності внутрішньої логіки обробки помилок.

5.2.3 Тестові сценарії

Для перевірки коректності роботи Чатбота було проведено тестування всіх функцій клієнтської частини, а також механізму аутентифікації користувачів.

Основні сценарії включали:

– запуск бота й перевірка реакції на некоректні та порожні команди; тестування аутентифікації: спроби доступу зареєстрованого та незареєстрованого користувача;

– перевірка прив’язки теплиці до користувача через access_key; отримання телеметрії з бази даних та перевірка коректності відображення даних;

– обробка помилок при недоступності теплиці або невірних параметрах; аналіз поведінки бота в умовах інтенсивного використання (часті запити).

5.3 Хід експерименту

1. Перевірка стартової логіки /start

Бот використовує метод get_user_by_чатбот(). Було протестовано два сценарії:

– користувач не має прив'язки до Чатбот-ID. Бот відображає меню гостя із кнопкою – «надіслати логін (username)» та вимагає встановлений Чатбот-username;

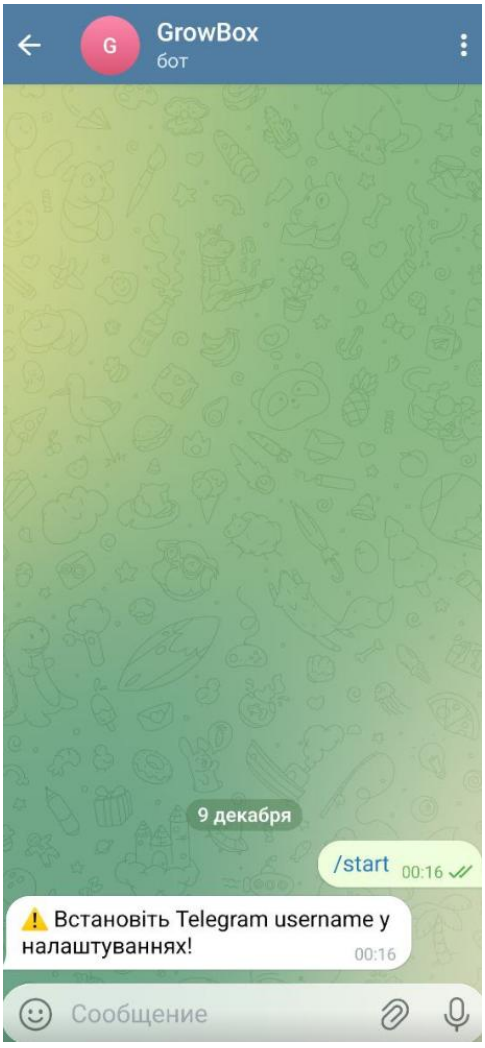

Рисунок 5.1 – Користувач не має Чатбот-username

– створюємо аккаунт;

```
greenh=# SELECT * FROM users
greenh-# ;
 user_id |   username    |                        password_hash                         | telegram_id | role |         created_at
---------+---------------+--------------------------------------------------------------+-------------+------+----------------------------
       1 | user1         | $2b$12$examplehash1                                          |   111111111 | user | 2025-12-03 22:26:58.159557
       2 | user2         | $2b$12$examplehash2                                          |   222222222 | user | 2025-12-03 22:26:58.159557
       4 | Yers          | $2b$12$oPRHiV7W2nSX6ROtfgEys.26gVX6JhVvMMvuSm6cgS6Vq3gJIeQUK |             | user | 2025-12-03 23:39:10.611976
       6 | Lav_Alina     | $2b$12$xphNRrlf9Hc9cbJeuZjv1Oe7nRVgL1SByAVObIPfl8NQeTB5OgJ4G |   486445069 | user | 2025-12-04 00:54:21.606223
       9 | cheredniche0  | $2b$12$guTGcD48zy6/argaOeFWCOawD7JsJNMF6QP7lyv0fHlVDJipjtAvG |             | user | 2025-12-04 23:00:00.9749
(5 rows)
```

Рисунок 5.2 – Поточні користувачі БД

Рисунок 5.3 – Надсилання логіну та створення паролю

```
greenh=# SELECT * FROM users;
 user_id |   username    |                        password_hash                         | telegram_id | role |         created_at
---------+---------------+--------------------------------------------------------------+-------------+------+----------------------------
       1 | user1         | $2b$12$examplehash1                                          |   111111111 | user | 2025-12-03 22:26:58.159557
       2 | user2         | $2b$12$examplehash2                                          |   222222222 | user | 2025-12-03 22:26:58.159557
       4 | Yers          | $2b$12$oPRHiV7W2nSX6ROtfgEys.26gVX6JhVvMMvuSm6cgS6Vq3gJIeQUK |             | user | 2025-12-03 23:39:10.611976
       6 | Lav_Alina     | $2b$12$xphNRrlf9Hc9cbJeuZjv1Oe7nRVgL1SByAVObIPfl8NQeTB5OgJ4G |   486445069 | user | 2025-12-04 00:54:21.606223
       9 | cheredniche0  | $2b$12$guTGcD48zy6/argaOeFWCOawD7JsJNMF6QP7lyv0fHlVDJipjtAvG |             | user | 2025-12-04 23:00:00.9749
      10 | ok_chered     | $2b$12$Fq.rOz9ymQyktyjtuQ8uOOW1aWOol4NdG34P3rHO.vItRIvKMoKNS |  1658358866 | user | 2025-12-09 00:24:55.161519
(6 rows)
```

Рисунок 5.4 – БД після додавання нового користувача

– користувач був авторизован раніше.

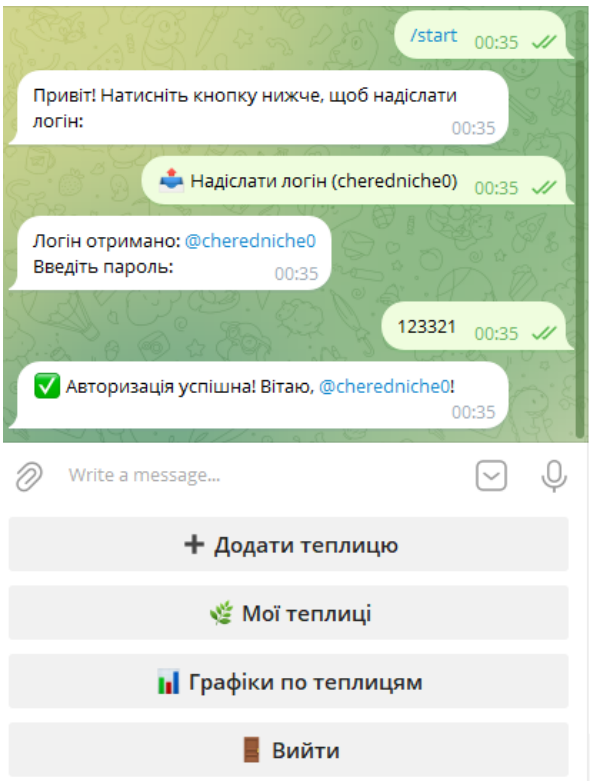

Рисунок 5.5 – Користувач був авторизован раніше

Результат: стартовий стан визначається коректно.

2. Перевірка надсилання логіну

Після натиснення кнопки « 📩 Надіслати логін» бот:

читає Чатбот-username,

перевіряє його у БД через get_user_by_login(username),

переводить користувача або в стан логіну (LoginState.waiting_password), або в стан реєстрації (RegisterState.waiting_password).

Перевірено 2 сценарії (рис. 5.1-5.5):

логін існує — користувача просять ввести пароль;

логін не існує — бот пропонує створити новий акаунт.

3. Перевірка автентифікації користувача

При введенні пароля в стані LoginState бот:

– отримує хеш із БД;

– перевіряє пароль через bcrypt.checkpw();

– у разі успіху викликає bind_чатбот(), зберігаючи Чатбот-ID у таблиці users.

Результат: логіка входу працює згідно очікувань (рис. 5.1-5.5).

4. Перевірка реєстрації нового користувача (рис. 5.1-5.5)

У стані RegisterState.waiting_password:

– бот створює запис у БД через register_user();

– одразу прив’язує Чатбот-ID;

– переводить користувача в основне меню.

5. Перевірка роботи з теплицями

Перегляд своїх теплиць

Команда «Мої теплиці» викликає get_user_greenhouses().

Було перевірено:

користувач без теплиць – відповідне повідомлення;

Рисунок 5.6 – користувач без теплиць

користувач з кількома теплицями – коректний список.

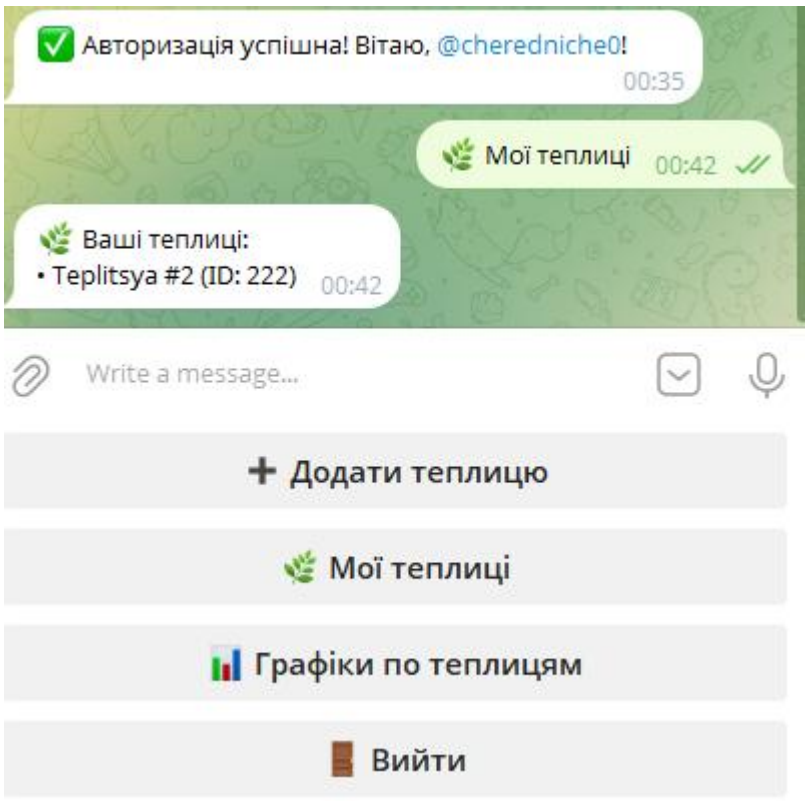

Рисунок 5.7 – користувач з теплицями

Додавання теплиці

Процес включає:

введення ID теплиці;

перевірку на існування в БД (get_greenhouse_by_user);

перевірку чи теплиця вже прив’язана;

запис зв’язку через bind_greenhouse_to_user().

Перевірено:

нечисловий ID – помилка введення;

неіснуючий ID – «Теплиця не знайдена»;

теплиця зайнята іншим користувачем;

успішне додавання.

```
greenh=# SELECT * FROM greenhouses;
 greenhouse_id | user_id |       name       |           location           |      access_key       |         created_at
---------------+---------+------------------+------------------------------+-----------------------+----------------------------
             1 |       1 | Greenhouse Alpha | Kyiv, Ukraine                | $2b$12$greenhousekey1 | 2025-12-03 22:26:58.165266
             2 |       2 | Greenhouse Beta  | Lviv, Ukraine                | $2b$12$greenhousekey2 | 2025-12-03 22:26:58.165266
             3 |       1 | Greenhouse Alpha | Kyiv, Ukraine                | $2b$12$greenhousekey1 | 2025-12-03 22:27:28.172696
             4 |       2 | Greenhouse Beta  | Lviv, Ukraine                | $2b$12$greenhousekey2 | 2025-12-03 22:27:28.172696
           123 |         | Teplitsya #1     | Kyiv, vul. Lesi Ukrainky, 10 | KEY123                | 2025-12-04 01:11:42.68968
           222 |       9 | Teplitsya #2     | Lviv, vul. Shevchenka, 25    | KEY456                | 2025-12-04 01:11:42.68968
(6 rows)
```

Рисунок 5.8 – БД до додавання новим користувачем існуючої таблиці

Рисунок 5.9 – Перевірка на коректність вводу та додавання таблиці

```
greenh=# SELECT * FROM greenhouses;
 greenhouse_id | user_id |       name       |          location          |      access_key       |         created_at
---------------+---------+------------------+----------------------------+-----------------------+----------------------------
             1 |       1 | Greenhouse Alpha | Kyiv, Ukraine              | $2b$12$greenhousekey1 | 2025-12-03 22:26:58.165266
             2 |       2 | Greenhouse Beta  | Lviv, Ukraine              | $2b$12$greenhousekey2 | 2025-12-03 22:26:58.165266
             3 |       1 | Greenhouse Alpha | Kyiv, Ukraine              | $2b$12$greenhousekey1 | 2025-12-03 22:27:28.172696
             4 |       2 | Greenhouse Beta  | Lviv, Ukraine              | $2b$12$greenhousekey2 | 2025-12-03 22:27:28.172696
           222 |       9 | Teplitsya #2     | Lviv, vul. Shevchenka, 25  | KEY456                | 2025-12-04 01:11:42.68968
           123 |      10 | Teplitsya #1     | Kyiv, vul. Lesi Ukrainky, 10 | KEY123              | 2025-12-04 01:11:42.68968
(6 rows)


greenh=# SELECT * FROM users;
 user_id |   username   |                        password_hash                         | telegram_id | role |         created_at
---------+--------------+--------------------------------------------------------------+-------------+------+----------------------------
       1 | user1        | $2b$12$examplehash1                                          |   111111111 | user | 2025-12-03 22:26:58.159557
       2 | user2        | $2b$12$examplehash2                                          |   222222222 | user | 2025-12-03 22:26:58.159557
       4 | Yers         | $2b$12$oPRHiV7W2nSX6ROtfgEys.26gVX6JhVvMMvuSm6cgS6Vq3gJIeQUK |             | user | 2025-12-03 23:39:10.611976
       6 | Lav_Alina    | $2b$12$xphNRrlf9Hc9cbJeuZjv1Oe7nRVgL1SByAVObIPfl8NQeTB5OgJ4G |   486445069 | user | 2025-12-04 00:54:21.606223
      10 | ok_chered    | $2b$12$Fq.rOz9ymQyktyjtuQ8uOOW1aWOol4NdG34P3rHO.vItRIvKMoKNS |  1658358866 | user | 2025-12-09 00:24:55.161519
       9 | cheredniche0 | $2b$12$guTGcD48zy6/argaOeFWCOawD7JsJNMF6QP7lyv0fHlVDJipjtAvG |   541819521 | user | 2025-12-04 23:00:00.9749
(6 rows)
```

Рисунок 5.10 – БД після внесення змін

6. Отримання графіків телеметрії

Після вибору «Графіки по теплицям» бот:

показує список теплиць через Inline-кнопки;

пропонує період (1h, 24h, 7d, 30d);

завантажує телеметрію get_telemetry_range();

будує графік Matplotlib:

температура повітря

вологість

Також тестувалась логіка форматування осі X для різних періодів:

годинний – інтервал 5 хв

добовий – інтервал 2 години

тижневий – інтервал 12 годин

місячний – позначення дня.

Також перевірено:

відсутність даних – бот відповідає «Немає даних».

Графік відправляється у вигляді BufferedInputFile.

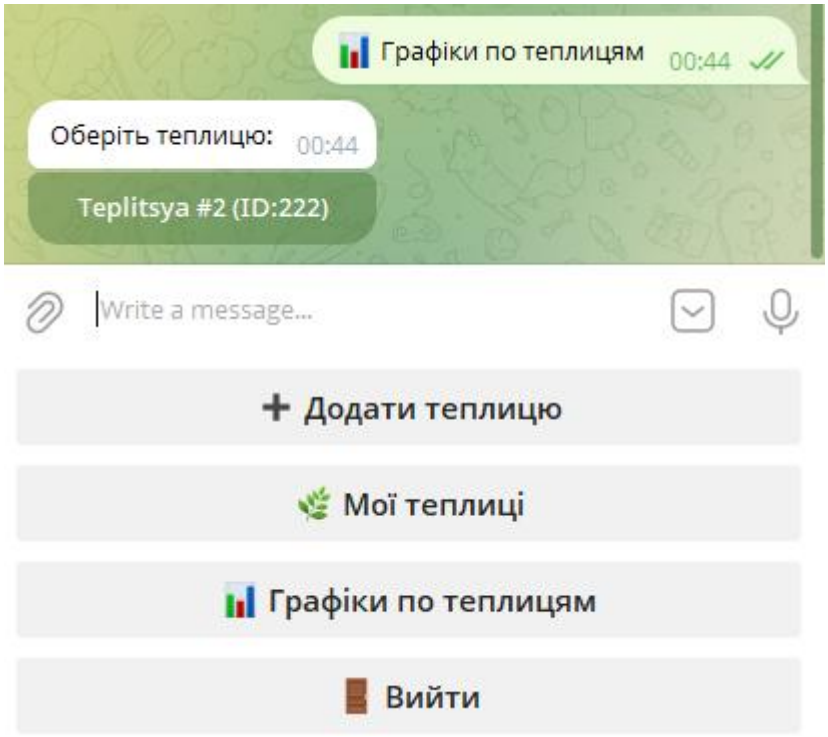

Рисунок 5.11 – Обрання між теплицями по inline кнопкам

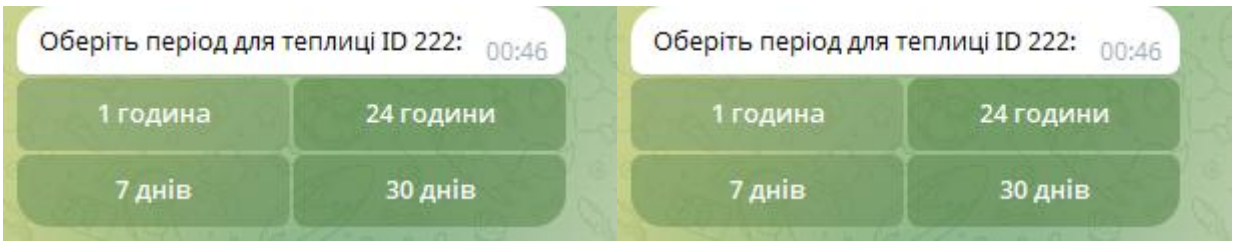

Рисунок 5.11 – Вибір проміжку часу

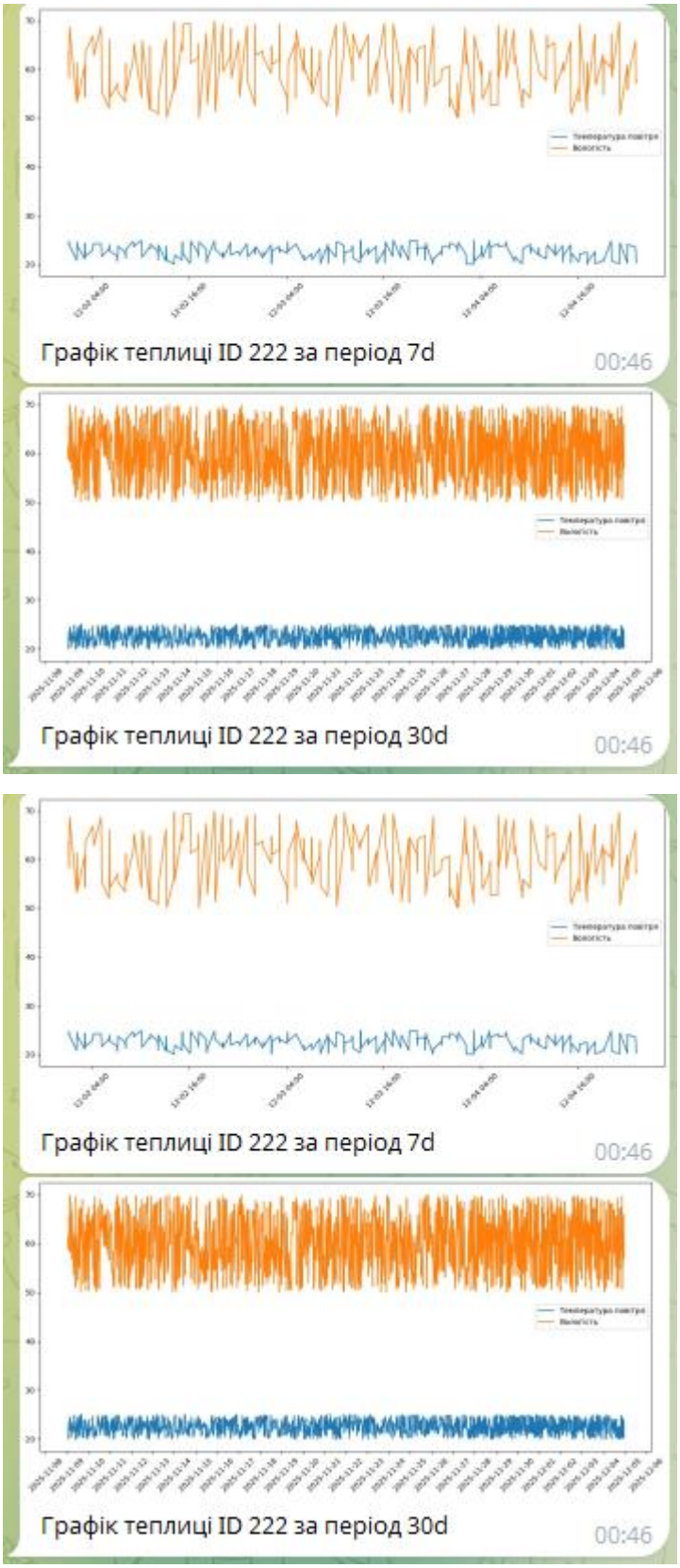

Рисунок 5.12 – Графіки за обраний період

Якщо за обраний період немає даних виводиться наступне повідомлення:

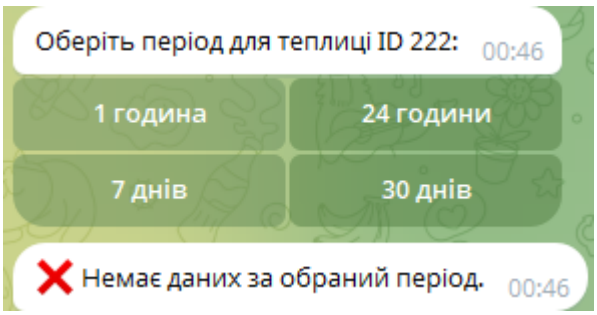

Рисунок 5.13 – Немає даних у таблиці

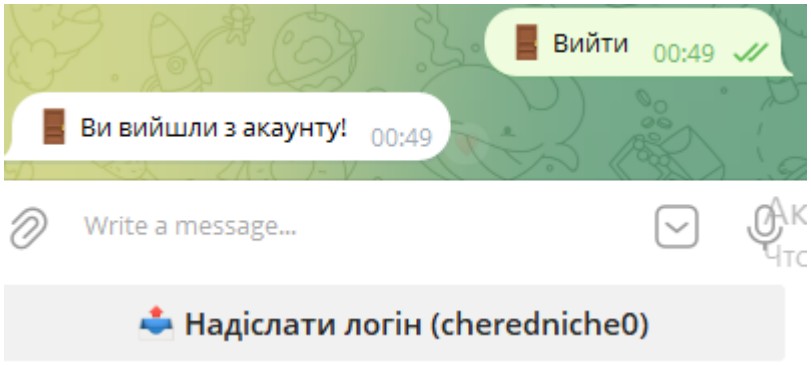

Рисунок 5.14 – Вихід з акаунту

При повторному надсиланні логіку вже зареєстрованого користувача, бот попросить пароль для підтвердження особистості:

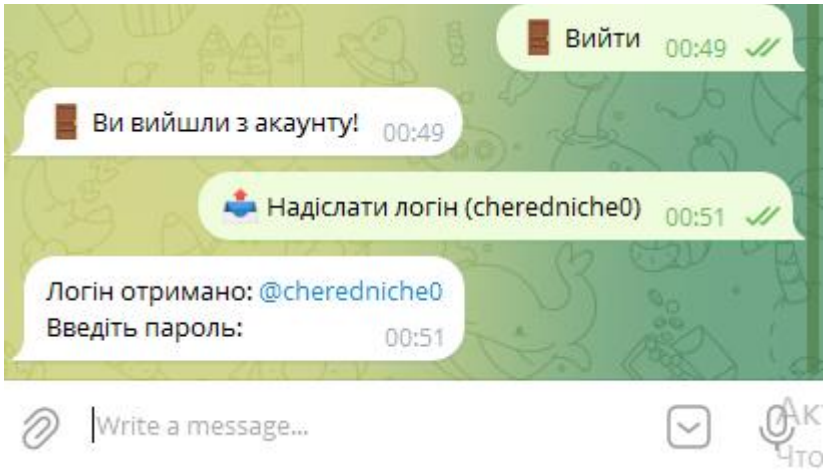

Рисунок 5.15 – Повторний вхід зареєстрованого користувача

5.4 Висновок по розділу

У межах експериментального розділу було проведено всебічне тестування Чатбота для взаємодії з IoT-системою гідропонної теплиці. Основною метою експерименту стало підтвердження працездатності розробленого клієнтського функціоналу, механізму аутентифікації

користувачів, а також перевірка правильності обробки команд та коректності взаємодії з базою даних.

У результаті тестування встановлено, що Чатбот стабільно обробляє основні сценарії взаємодії: стартову ініціалізацію, автентифікацію користувачів, реєстрацію нових облікових записів, виконання запитів до бази даних та отримання телеметрії. Механізм аутентифікації за Чатбот-ID працює відповідно до технічних вимог: бот коректно визначає авторизованих користувачів, блокує доступ невідомим абонентам та забезпечує належний рівень безпеки при роботі з конфіденційними даними системи.

Тестові сценарії підтвердили правильність реалізації логіки роботи зі списком теплиць, включаючи прив’язування теплиці до користувача, перевірку доступності, обробку некоректних введень та унікальності прив’язування. Система відображення графіків телеметрії продемонструвала стабільність, коректність побудови графіків та адекватну реакцію на відсутність даних у вибраний період.

Результати експериментального дослідження дозволили встановити, що розроблений Чатбот забезпечує надійну комунікацію з IoT-підсистемою, коректно працює з базою даних, підтримує різні стани користувача та коректно реалізує обробку помилок. Усі ключові функції відповідають визначеним у технічних вимогах параметрам, а логіка роботи системи підтверджує можливість її практичного застосування в системах моніторингу та управління гідропонними теплицями.

Отже, експериментальний розділ підтверджує працездатність, функціональну повноцінність та надійність створеного програмного забезпечення, а також демонструє відповідність розробленої системи вимогам безпеки, точності та стабільності.

ВИСНОВОК

Кваліфікаційна робота є завершеною науково-практичною роботою, в якій вирішено задачу розробки, обґрунтування та експериментальної перевірки інтелектуальної IoT-системи контролю та моніторингу гідропонної теплиці, що забезпечує збір, обробку, зберігання та віддалене керування технологічними параметрами вирощування рослин із застосуванням сучасних інформаційних, мережевих та програмних технологій Інтернету речей.

Основні висновки і результати роботи полягають у наступному:

1. Проведено аналіз сучасних технологій Інтернету речей у сфері гідропоніки та автоматизованих систем керування мікрокліматом. Визначено основні функціональні, технічні та інформаційні вимоги до IoT-системи контролю гідропонної теплиці, що дозволило обґрунтувати доцільність використання розподіленої архітектури з Edge-рівнем обробки даних і серверною частиною.

2. Розроблено структурну архітектуру IoT-системи, яка включає сенсорний рівень, рівень локальної обробки даних, комунікаційний рівень, серверний рівень та користувацький інтерфейс. Запропонована архітектура забезпечує безперервний збір телеметрії, надійне передавання даних і можливість дистанційного керування об’єктом.

3. На основі теоретичних досліджень сформовано модель мережевої взаємодії системи та обґрунтовано вибір протоколів MQTT та HTTP для передавання даних. Розглянуто питання забезпечення якості обслуговування (QoS), надійності обміну повідомленнями, інформаційної безпеки та оптимізації трафіку між Edge-контролером і сервером.

4. Здійснено синтез системи, що включав розробку функціонального опису, вибір сенсорного обладнання, виконавчих пристроїв, контролера, побудову функціональної схеми та структури кабельного живлення. Визначено мінімальні вимоги до апаратних компонентів та забезпечено можливість масштабування системи.

5. Розроблено та модернізовано структуру бази даних для зберігання телеметричних даних, інформації про користувачів і теплиці. Запропонована база даних забезпечує цілісність, масштабованість та швидкий доступ до даних, а також підтримує інтеграцію з клієнтськими застосунками.

6. У практичній частині реалізовано клієнтський інтерфейс у вигляді Чатбота на базі Aiogram 3.22.0, який забезпечує авторизацію користувачів, прив’язку теплиць, перегляд списку об’єктів та відображення історичних графіків телеметрії (температури, вологості та параметрів поживного розчину). Чатбот інтегрований з PostgreSQL, використовує механізм FSM, кнопковий інтерфейс і систему логування подій.

7. Проведено експериментальне дослідження працездатності системи, яке включало тестування клієнтського функціоналу, механізмів авторизації, роботи з базою даних, графічних модулів та стійкості до помилок. Оцінено коректність обробки команд, поведінку системи при інтенсивному навантаженні та валідацію вхідних даних.

8. Результати експериментальних досліджень підтвердили стабільність роботи розробленої IoT-системи, коректність реалізованих механізмів безпеки, а також точність обробки та відображення телеметричної інформації в режимі реального часу та в історичному вигляді.

Таким чином, у кваліфікаційній роботі досягнуто поставленої мети та виконано всі поставлені завдання. Розроблена IoT-система може бути використана для моніторингу та часткового керування гідропонною теплицею, а отримані результати є основою для подальшої модернізації системи, впровадження автоматичного регулювання умов вирощування та масштабування рішення до промислових тепличних комплексів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сайт компанії «ГФІ» – Education [Електронний ресурс] – Режим доступу: https://greenfuture.com.ua/ (дата звернення 10.10.2025р.)

2. Структура управління організацією. Лекція з навчальної дисципліни «Менеджмент організацій» / Павленчик А. О. – ЛДУФК, 2020. – 16 с. [Електронний ресурс] – Режим доступу: https://ldufk.edu.ua (дата звернення 10.10.2025р.)

3. Методичні рекомендації до виконання кваліфікаційної роботи бакалавра / Цвіркун Л.І. та ін. – НТУ «ДП», 2024. – 62 с. [Електронний ресурс] – Режим доступу: https://nmu.org.ua (дата звернення 10.10.2025р.)

4. Налаштування NAT – Education [Електронний ресурс] – Режим доступу: http://surl.li/hsyzr (дата звернення 10.10.2025р.)

5. Налаштування VLAN – Education [Електронний ресурс] – Режим доступу: http://surl.li/hrjwk (дата звернення 10.10.2025р.)

6. ACL списки – Education [Електронний ресурс] – Режим доступу: http://surl.li/hszbq (дата звернення 10.10.2025р.)

7. Налаштування VPN – Education [Електронний ресурс] – Режим доступу: http://surl.li/hrjwr (дата звернення 10.10.2025р.)

8. Інтернет речей – Education [Електронний ресурс] – Режим доступу: http://surl.li/dcsqu (дата звернення 10.10.2025р.)

9. Офіційна документація Raspberry Pi – [Електронний ресурс] – Режим доступу: https://www.raspberrypi.com/documentation (дата звернення 10.10.2025р.)

10. Датчики та їх компоненти – hwlibre [Електронний ресурс] – Режим доступу: https://www.hwlibre.com/ (дата звернення 10.10.2025р.)

11. Компоненти кіберфізичних систем – DFRobot Community [Електронний ресурс] – Режим доступу: https://community.dfrobot.com/ (дата звернення 10.10.2025р.)

12. Мова програмування MicroPython – [Електронний ресурс] – Режим доступу: https://micropython.org/ (дата звернення 10.10.2025р.)

13. Документація ESP32 – Espressif Systems [Електронний ресурс] – Режим доступу: https://www.espressif.com/en/support/documents/technical-documents (дата звернення 10.10.2025р.)

14. MQTT Version 5.0. OASIS Standard – [Електронний ресурс] – Режим доступу: https://docs.oasis-open.org/mqtt (дата звернення 10.10.2025р.)

15. CoAP RFC 7252 – IETF [Електронний ресурс] – Режим доступу: https://datatracker.ietf.org/doc/html/rfc7252 (дата звернення 10.10.2025р.)

16. IEEE 802.11 Wi-Fi Standard – IEEE [Електронний ресурс] – Режим доступу: https://standards.ieee.org (дата звернення 10.10.2025р.)

17. Cisco Networking Academy. Packet Tracer IoT – [Електронний ресурс] – Режим доступу: https://www.netacad.com (дата звернення 10.10.2025р.)

18. Документація PostgreSQL – [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs (дата звернення 10.10.2025р.)

19. Документація MySQL – [Електронний ресурс] – Режим доступу: https://dev.mysql.com/doc (дата звернення 10.10.2025р.)

20. Гідропоніка: Сучасні технології вирощування – AgroPortal [Електронний ресурс] – Режим доступу: https://agroportal.ua (дата звернення 10.10.2025р.)

21. Hydroponic Systems and IoT Monitoring – ResearchGate [Електронний ресурс] – Режим доступу: https://www.researchgate.net (дата звернення 10.10.2025р.)

22. LoRaWAN Specification v1.1 – LoRa Alliance [Електронний ресурс] – Режим доступу: https://lora-alliance.org (дата звернення 10.10.2025р.)

23. Bluetooth Low Energy v5.3 – Bluetooth SIG [Електронний ресурс] – Режим доступу: https://www.bluetooth.com/specifications (дата звернення 10.10.2025р.)

24. NASA Plant Growth Laboratory – NASA NTRS [Електронний ресурс] – Режим доступу: https://ntrs.nasa.gov (дата звернення 10.10.2025р.)

25. FAO Hydroponic Farming Guidelines – FAO [Електронний ресурс] – Режим доступу: https://www.fao.org (дата звернення 10.10.2025р.)

26. IoT Data Analytics in Agriculture – SpringerLink [Електронний ресурс] – Режим доступу: https://link.springer.com (дата звернення 10.10.2025р.)

27. Grafana Dashboards Documentation – [Електронний ресурс] – Режим доступу: https://grafana.com/docs (дата звернення 10.10.2025р.)

28. Prometheus Monitoring System – [Електронний ресурс] – Режим доступу: https://prometheus.io/docs (дата звернення 10.10.2025р.)

29. OpenWeather API – [Електронний ресурс] – Режим доступу: https://openweathermap.org/api (дата звернення 10.10.2025р.)

30. Firebase Realtime Database Documentation – [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs (дата звернення 10.10.2025р.)

Додаток А

Текст main блоку обробки з мікроконтролера «Raspberry pi Pico W»

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ МІКРОКОНТРОЛЛЕРА КІБЕРФІЗИЧНОЇ СИСТЕМИ

Текст програми контролера Raspbery Pi pico W

804.02070743.25018-01 12 01

Листів 9

2025

АНОТАЦІЯ

Дана програма розроблена для автоматизації системи контролю процесів живлення в гідропонній теплиці за допомогою різних сенсорів і компонентів.

Вона підключається до Wi-Fi мережі, зчитує дані з датчиків температури, вологості, рівня води, pH, електропровідності та рівня освітленості, і відображає ці дані на OLED дисплеї. Крім того, програма відправляє зібрані дані на сервер через сокетне з'єднання для подальшого аналізу і зберігання.

Система також керує реле для включення та виключення різних пристроїв в залежності від отриманих значень з датчиків, забезпечуючи автоматизацію і оптимізацію умов для росту рослин у гідропонній теплиці.

ЗМІСТ

1. Функція main

```
from machine import Pin, I2C, ADC, RTC
import utime
from dht import DHT11
import time
import ssd1306
from ssd1306 import SSD1306_I2C
import framebuf
import urequests
import network
import rp2
import sys
import socket
import struct
import onewire
import ds18x20
import usocket


server_ip = '192.168.31.33'
server_port = 12345

dat_pin = machine.Pin(15)
ds_sensor = ds18x20.DS18X20(onewire.OneWire(dat_pin))

roms = ds_sensor.scan()
print('Found DS18B20 sensors:', roms)
multi = ADC(Pin(28))
control_pins = [Pin(i, Pin.OUT) for i in (10, 11, 12, 13)]

i2c_time = I2C(1, scl = Pin(3), sda = Pin(2))
class DS1307:
    def __init__(self, i2c_time, addr=0x68):
        self.i2c = i2c_time
        self.addr = addr

    def set_time(self, year, month, day, weekday, hour, minute, second):
        bcd = lambda n: n + 6 * (n // 10)
        self.i2c.writeto_mem(self.addr, 0x00, bytes([bcd(second), bcd(minute), bcd(hour),
bcd(weekday), bcd(day), bcd(month), bcd(year - 2000)]))

    def read_time(self):
        data = self.i2c.readfrom_mem(self.addr, 0x00, 7)

        bcd_to_dec = lambda n: n - 6 * (n >> 4)
        return (
            bcd_to_dec(data[6]) + 2000,
            bcd_to_dec(data[5]),
            bcd_to_dec(data[4]),
            bcd_to_dec(data[3]),
            bcd_to_dec(data[2]),
            bcd_to_dec(data[1]),
```

```
            bcd_to_dec(data[0])
        )

#pin for connection DHT11 sensor
pin = Pin(14, Pin.OUT, Pin.PULL_DOWN)
sensor = DHT11(pin)

#setings sda and scl for I2C interface
#i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)

#setings for relay pins
relay_pins = [
    Pin(16, Pin.OUT), #like pin namber 1
    Pin(17, Pin.OUT),  # Like pin number 2
    Pin(18, Pin.OUT),  # Like pin number 3
    Pin(19, Pin.OUT)   # Like pin number 4
]

for i in range(0, 4):
    print("Humidity: {}".format(i))
    relay_pins[i].value(1)
rtc = DS1307(i2c_time)
#display setings
#WIDTH = 128
#HEIGHT = 32
i2c = I2C(0, scl=Pin(9), sda=Pin(8), freq=400000)
#display = SSD1306_I2C(WIDTH, HEIGHT, i2c)
#Create an object for work with Oled display
oled_width = 128
oled_height = 64
oled = ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

#water sensor setings like water constant
water_level_sensor = ADC(Pin(26))
water=0
compA_level_sensor = ADC(Pin(27))
compA=0
compB_level_sensor = ADC(Pin(28))
compB=0
# array for animations
images = []
def connect_to_wifi(ssid, password): #network connecting
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    if not wlan.isconnected():
        print('Connecting to network...')
        wlan.connect(ssid, password)
        while not wlan.isconnected():
            pass
    print('Network config:', wlan.ifconfig())

# change ssid and password, after
```

```
connect_to_wifi('Kak_ugodnoGG', '0636381045')

for n in range(1, 28):
    with open('/folder_anima/image%s.pbm' % n, 'rb') as f:  #open folder and image
        f.readline() # Magic number
        f.readline() # Creator comment
        f.readline() # Dimensions
        data = bytearray(f.read())
    fbuf = framebuf.FrameBuffer(data, 30, 30, framebuf.MONO_HLSB) #adjust
accordingly the width and height
    images.append(fbuf)



count=0
while count<6:
    for i in images:
        oled.blit(i, 0, 0)
        oled.blit(i, 32, 0)
        oled.blit(i, 64, 0)
        oled.blit(i, 98, 0)
        oled.show()
        time.sleep(0.01)
    count=count+1
    print(count)


# Настройка пинов как выходы
#for pin_num in relay_pins:
#    pin = Pin(pin_num, Pin.OUT)

# Функция для вывода текста на дисплей
def display_text(text):
    oled.fill(0)  # Очистка дисплея
    oled.text(text, 0, 0)  # Вывод текста в левом верхнем углу
    oled.show()  # Отображение на дисплее

# Функция для включения реле по номеру канала
def turn_relay_on(channel):
    relay_pins[channel].value(0)

# Функция для выключения реле по номеру канала
def turn_relay_off(channel):
    relay_pins[channel].value(1)


def check_water_presence(threshold=30000):
     sensor_value = water_level_sensor.read_u16()
    if sensor_value > threshold:
        return False  # Вода обнаружена
    else:
        return True
```

```
        def measure_temperature():
            try:
                temperature = sensor.temperature
                humidity = sensor.humidity
                print("Temperature: {}".format(temperature))
                print("Humidity: {}".format(humidity))
                #display_text("Temperature: {:.1f}C\nHumidity: {:.1f}%".format(temperature,
humidity))
                return temperature, humidity
            except Exception as e:
                print("Error:", e)
                return None, None

        sensor_ph_pin = ADC(Pin(26))

        def read_voltage_ph():
            analog_value = sensor_ph_pin.read_u16()
            voltage_ph = analog_value * 3.3 / 65535
            return voltage_ph

        def calibrate(voltage_at_ph7, voltage_at_ph4):
            slope = (7 - 4) / (voltage_at_ph7 - voltage_at_ph4)
            intercept = 7 - slope * voltage_at_ph7
            return slope, intercept

        def read_ph(slope, intercept):
            voltage_ph = read_voltage_ph()
            ph_value = slope * voltage_ph + intercept
            return ph_value

        voltage_at_ph7 = 1
        voltage_at_ph4 = 1.25

        slope, intercept = calibrate(voltage_at_ph7, voltage_at_ph4)

        sensor_ec_pin = ADC(Pin(27))

        def read_voltage_ec():
            val = sensor_ec_pin.read_u16()  # Чтение сырого значения
            voltage_ec = (val / 65535) * 3.3  # Преобразование в напряжение
            return voltage_ec

        def voltage_to_mS_cm(voltage_ec):
            conversion_factor = 8292  # µS/cm per volt, полученный ранее
            conductivity_uS_cm = voltage_ec * conversion_factor
            conductivity_mS_cm = conductivity_uS_cm / 1000  # Перевод из µS/cm в mS/cm
            return conductivity_mS_cm
        def read_wtemperature():
            ds_sensor.convert_temp()
            time.sleep_ms(750)  # DS18B20 требует некоторое время для преобразования
температуры
            for rom in roms:
```

```
            wtemp = ds_sensor.read_temp(rom)
            print('Temperature:', ds_sensor.read_temp(rom), '°C')
        return wtemp



    def select_channel(channel):
        for i in range(4):
            control_pins[i].value((channel >> i) & 1)

    def read_sensor(channel):
        select_channel(channel)
        time.sleep(0.1)  # Пауза для стабилизации сигнала
        return multi.read_u16()

    def convert_to_lumens(adc_value):
    калибровки для точности
        max_adc = 65535
        max_lumens = 1000
        lumens = (adc_value / max_adc) * max_lumens
        return lumens

    def check_water_presence1(threshold=15000):
        sensor_value = read_sensor(4)
        if sensor_value < threshold:
            return False
        else:
            return True




    wtemp = read_wtemperature()
    measure_temperature()

    while True:
        temp,hum = measure_temperature()

        current_time = rtc.read_time()
        print(": {}-{}-{} {}, {}:{}:{}".format(current_time[0], current_time[1],
current_time[2], current_time[3], current_time[4], current_time[5], current_time[6]))
        utime.sleep(1)


        if 6 <= current_time[4] < 18:
            turn_relay_on(0)
        else:
            turn_relay_off(0)
        #if check_water_presence():
        #    print("Water detected!")
```

```
#    water=1
#else:
#    print("No water detected.")
#    water=0
voltage_ec = read_voltage_ec()
conductivity = voltage_to_mS_cm(voltage_ec)
print("Conductivity:", conductivity, "mS/cm")
if conductivity < 1:
    turn_relay_on(2)
elif conductivity > 1.65:
    turn_relay_on(3)
else:
    turn_relay_off(2)
    turn_relay_off(3)

ph = read_ph(slope, intercept)
print("pH value:", ph)
print("Voltage:",read_voltage_ph())

if ph < 5:
    turn_relay_on(1)
elif ph > 7:
    turn_relay_on(3)
else:
    turn_relay_off(1)
    turn_relay_off(3)



adc_value = read_sensor(0)
light_level = convert_to_lumens(adc_value)
if check_water_presence1():
    print("Water detected!")
    water=1
else:
    print("No water detected.")
    water=0
water_level_1 = read_sensor(4)
#water_level_2 = read_sensor(3)
#water_level_3 = read_sensor(2)
if current_time[5] % 1 == 0 and current_time[6] % 40 == 0:
    wtemp = read_wtemperature()

oled.fill(0)
if  water_level_1 == 0 and water_level_2 == 1 and water_level_3 == 1:
    oled.text('Add water to', 0, 10)
    oled.text('the tank A ', 0, 20)
elif  water_level_1 == 1 and water_level_2 == 0 and water_level_3 == 1 :
    oled.text('Add water to', 0, 10)
    oled.text('the tank B', 0, 20)
elif  water_level_1 == 1 and water_level_2 == 1 and water_level_3 == 0 :
    oled.text('Add water to', 0, 10)
```

```
            oled.text('the tank Water', 0, 20)
        elif  water_level_1 == 0 and water_level_2 == 0 and water_level_3 == 0:
            oled.text('Add water all', 0, 10)
            oled.text('to the tanks', 0, 20)

        oled.text('Time: {}:{}:{}'.format(current_time[4], current_time[5], current_time[6]), 0,
0)
        if water_level_1 != 0:
            oled.text('Water Temp: {:.1f}C'.format(wtemp), 0, 8)
            oled.text('Air Temp: {:.1f}C'.format(temp if temp is not None else 0), 0, 16)
            oled.text('Humidity: {:.1f}%'.format(hum if hum is not None else 0), 0, 24)
            oled.text('pH: {:.2f}'.format(ph), 0, 32)
            oled.text('EC: {:.2f} mS/cm'.format(conductivity), 0, 40)
            oled.text('light: {:.2f} lm'.format(light_level), 0, 48)
        oled.show()
        if current_time[5] % 5 == 0 and current_time[6] % 10:
            try:
                if temp is not None and hum is not None:
                    socket = usocket.socket(usocket.AF_INET, usocket.SOCK_STREAM)
                    socket.connect((server_ip, server_port))


                    data = [current_time[0], current_time[1], current_time[2], current_time[4],
current_time[5], water_level_1, 1, 1, wtemp, temp, hum, ph, conductivity, light_level, 1, 1, 1, 1,
1]

                    bytes_data = struct.pack('8i6f5i', *data)

                    socket.send(bytes_data)
                    print('Данные отправлены')
                else:
                    print("send data faild")
            except Exception as e:

            finally:
                socket.close()
```

Додаток Б

Текст програми розробленого телеграм боту

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ТЕЛЕГРАМ БОТУ

Текст програми Телеграм боту

804.02070743.25018-01 12 01

Листів 9

2025

АНОТАЦІЯ

Представлений програмний комплекс реалізує Чатбота, призначеного для інтерактивної взаємодії з системою моніторингу гідропонних теплиць. Архітектура рішення побудована за модульним принципом та включає окремі компоненти для обробки команд користувачів, взаємодії з базою даних, авторизації, логування подій та генерації аналітики на основі телеметричних даних. Такий підхід забезпечує масштабованість, розширюваність та високий рівень кібербезпеки.

ЗМІСТ

1. config.py

```
# config.py
# ------------------------------------------------------------
# Файл конфігурації бота і підключення до бази даних
# ------------------------------------------------------------

TOKEN = "YOUR_ЧАТБОТ_BOT_TOKEN"

DB = {
    "dbname": "hydroponics",
    "user": "bot_user",
    "password": "password123",
    "host": "localhost"
}
```

2. db.py

```
# db.py
# ------------------------------------------------------------
# Модуль для підключення та роботи з PostgreSQL.
# Забезпечує виконання SQL-запитів та повернення результатів.
# ------------------------------------------------------------

import psycopg2
from config import DB

# Створення глобального підключення
conn = psycopg2.connect(**DB)
cursor = conn.cursor()

def get_user_by_чатбот(чатбот_id):
    cursor.execute("SELECT user_id FROM users WHERE чатбот_id = %s", (чатбот_id,))
    return cursor.fetchone()

def get_user_by_login(login):
    cursor.execute("SELECT user_id, password_hash FROM users WHERE username = %s",
(login,))
    return cursor.fetchone()

def bind_чатбот(user_id, чатбот_id):
    cursor.execute("UPDATE users SET чатбот_id = %s WHERE user_id = %s", (чатбот_id,
user_id))
    conn.commit()

def get_greenhouse_by_user(user_id):
    cursor.execute("SELECT greenhouse_id, name FROM greenhouses WHERE user_id = %s",
(user_id,))
    return cursor.fetchone()

def get_last_telemetry(greenhouse_id):
    cursor.execute("""
        SELECT air_temperature, solution_temperature, humidity,
               reservoir1_level, reservoir2_level, reservoir3_level,
               lighting_state, timestamp
        FROM telemetry
        WHERE greenhouse_id = %s
        ORDER BY timestamp DESC
        LIMIT 1
    """, (greenhouse_id,))
    return cursor.fetchone()
```

```
def get_telemetry_range(greenhouse_id, since):
    cursor.execute("""
        SELECT air_temperature, solution_temperature, humidity, lighting_state,
               timestamp
        FROM telemetry
        WHERE greenhouse_id = %s AND timestamp >= %s
        ORDER BY timestamp ASC
    """, (greenhouse_id, since))
    return cursor.fetchall()

def insert_log(user_id, action, status):
    cursor.execute("""
        INSERT INTO audit_log(user_id, action, status, timestamp)
        VALUES (%s, %s, %s, NOW())
    """, (user_id, action, status))
    conn.commit()
```

3. utils/logging.py

```
# utils/logging.py
# ------------------------------------------------------------
# Виносить логіку аудиту в окремий модуль
# ------------------------------------------------------------

from db import insert_log

def log_action(user_id, action, status="success"):
    """
    Записує дію до таблиці audit_log.
    """
    insert_log(user_id, action, status)
```

4. auth.py

```
# auth.py
# ------------------------------------------------------------
# Модуль обробки логіну/паролю.
# Реалізує двофазну авторизацію.
# ------------------------------------------------------------

import bcrypt
from db import get_user_by_login, bind_чатбот
from utils.logging import log_action

pending_login = {}      # тимчасове сховище вводу логіну/паролю
authorized_users = {}   # авторизовані Чатбот-користувачі


def start_auth(чатбот_id):
    pending_login[чатбот_id] = {"step": "login"}


def auth_step(чатбот_id, text):
    """
    Обробляє логін та пароль користувача.
    """

    step = pending_login[чатбот_id]["step"]

    # 1. Користувач вводить логін
```

```
    if step == "login":
        record = get_user_by_login(text)
        if record is None:
            return "❌ Невірний логін. Спробуйте ще."

        user_id, password_hash = record

        pending_login[чатбот_id].update({
            "step": "password",
            "user_id": user_id,
            "password_hash": password_hash
        })

        return "Введіть пароль:"

    # 2. Користувач вводить пароль
    if step == "password":
        user_id = pending_login[чатбот_id]["user_id"]
        stored_hash = pending_login[чатбот_id]["password_hash"]

        if bcrypt.checkpw(text.encode(), stored_hash.encode()):
            bind_чатбот(user_id, чатбот_id)
            authorized_users[чатбот_id] = user_id
            pending_login.pop(чатбот_id)

            log_action(user_id, "login", "success")

            return "✅ Авторизація успішна! Використовуйте /status або /graph"
        else:
            log_action(user_id, "login", "failed")
            return "❌ Невірний пароль. Спробуйте ще."


def is_authorized(чатбот_id):
    return чатбот_id in authorized_users


def get_user_id(чатбот_id):
    return authorized_users.get(чатбот_id)
```

5. handlers/start.py

```
# ------------------------------------------------------------
# Обробка команди /start — вхід до системи
# ------------------------------------------------------------

from aiogram import types
from auth import start_auth, is_authorized
from utils.logging import log_action
from db import get_user_by_чатбот
from auth import authorized_users


async def cmd_start(msg: types.Message):
    чатбот_id = msg.from_user.id

    # Перевіряємо чи Чатбот-ID вже зареєстрований
    record = get_user_by_чатбот(чатбот_id)

    if record:
        user_id = record[0]
```

```
        authorized_users[чатбот_id] = user_id
        log_action(user_id, "login_auto", "success")
        await msg.reply("Ви вже авторизовані.\nВикористовуйте /status або /graph")
        return

    # Новий користувач — запускаємо процес логіну
    start_auth(чатбот_id)
    await msg.reply("Введіть логін:")
```

6. handlers/status.py

```
# handlers/status.py
# ------------------------------------------------------------
# Вивід останніх даних телеметрії
# ------------------------------------------------------------

from aiogram import types
from auth import is_authorized, get_user_id
from db import get_greenhouse_by_user, get_last_telemetry
from utils.logging import log_action


async def cmd_status(msg: types.Message):
    чатбот_id = msg.from_user.id

    if not is_authorized(чатбот_id):
        await msg.reply("Спочатку увійдіть: /start")
        return

    user_id = get_user_id(чатбот_id)

    greenhouse = get_greenhouse_by_user(user_id)
    if greenhouse is None:
        await msg.reply("У вас немає прив'язаних теплиць.")
        return

    gh_id, gh_name = greenhouse

    telemetry = get_last_telemetry(gh_id)
    if telemetry is None:
        await msg.reply("Немає даних телеметрії.")
        return

    air, sol, hum, r1, r2, r3, light, ts = telemetry

    log_action(user_id, "status", "success")

    await msg.reply(
        f"🌿 *Теплиця:* {gh_name}\n"
        f"🌡 Температура повітря: {air}°C\n"
        f"🌡 Темп. розчину: {sol}°C\n"
        f"💧 Вологість: {hum}%\n"
        f"🛢 Резервуари: {r1}, {r2}, {r3}\n"
        f"💡 Освітлення: {'ON' if light else 'OFF'}\n"
        f"⏱ Час: {ts}\n",
        parse_mode="Markdown"
    )
```

7. handlers/graphs.py

```
# handlers/graphs.py
# ------------------------------------------------------------
# Побудова графіків телеметрії за різні періоди
# ------------------------------------------------------------

from aiogram import types
from aiogram.types import InlineKeyboardMarkup, InlineKeyboardButton
import matplotlib.pyplot as plt
import io
import datetime

from db import get_greenhouse_by_user, get_telemetry_range
from auth import is_authorized, get_user_id
from utils.logging import log_action


async def cmd_graph(msg: types.Message):
    чатбот_id = msg.from_user.id

    if not is_authorized(чатбот_id):
        await msg.reply("Спочатку увійдіть: /start")
        return

    # Клавіатура вибору періоду
    kb = InlineKeyboardMarkup()
    kb.add(
        InlineKeyboardButton("1 година", callback_data="g_1h"),
        InlineKeyboardButton("24 години", callback_data="g_24h"),
    )
    kb.add(
        InlineKeyboardButton("7 днів", callback_data="g_7d"),
        InlineKeyboardButton("30 днів", callback_data="g_30d"),
    )

    await msg.reply("Оберіть період:", reply_markup=kb)


async def process_graph_callback(call: types.CallbackQuery):
    чатбот_id = call.from_user.id
    user_id = get_user_id(чатбот_id)

    greenhouse = get_greenhouse_by_user(user_id)
    gh_id, name = greenhouse

    # Визначаємо часовий інтервал
    now = datetime.datetime.now()

    if call.data == "g_1h":
        since = now - datetime.timedelta(hours=1)
    elif call.data == "g_24h":
        since = now - datetime.timedelta(hours=24)
    elif call.data == "g_7d":
        since = now - datetime.timedelta(days=7)
    else:
        since = now - datetime.timedelta(days=30)

    data = get_telemetry_range(gh_id, since)

    if not data:
```

```
        await call.message.answer("Немає даних за обраний період.")
        return

    # Побудова графіка
    timestamps = [row[4] for row in data]
    air = [row[0] for row in data]
    hum = [row[2] for row in data]

    plt.figure(figsize=(10, 6))
    plt.plot(timestamps, air, label="Температура повітря")
    plt.plot(timestamps, hum, label="Вологість")
    plt.legend()
    plt.xticks(rotation=45)

    # Зберігаємо у байтовий буфер
    buf = io.BytesIO()
    plt.savefig(buf, format='png', bbox_inches='tight')
    buf.seek(0)
    plt.close()

    log_action(user_id, "graph", "success")

    await call.message.answer_photo(buf, caption="Графік даних")
```

8. bot.py

```
# bot.py
# ------------------------------------------------------------
# Головний модуль запуску Чатбота.
# Реєстрація хендлерів, запуск циклу.
# ------------------------------------------------------------

from aiogram import Bot, Dispatcher, executor, types
from config import TOKEN

from handlers.start import cmd_start
from handlers.status import cmd_status
from handlers.graphs import cmd_graph, process_graph_callback
from auth import pending_login, auth_step


bot = Bot(token=TOKEN)
dp = Dispatcher(bot)

# ---------------- REGISTER HANDLERS ----------------------

@dp.message_handler(commands=['start'])
async def handle_start(msg: types.Message):
    await cmd_start(msg)

@dp.message_handler(commands=['status'])
async def handle_status(msg: types.Message):
    await cmd_status(msg)

@dp.message_handler(commands=['graph'])
async def handle_graph(msg: types.Message):
    await cmd_graph(msg)

@dp.callback_query_handler(lambda c: c.data.startswith("g_"))
```

```
async def handle_graph_buttons(call: types.CallbackQuery):
    await process_graph_callback(call)

# Авторизація текстовими повідомленнями
@dp.message_handler()
async def handle_text(msg: types.Message):
    if msg.from_user.id in pending_login:
        reply = auth_step(msg.from_user.id, msg.text)
        await msg.reply(reply)

# ---------------- START BOT ----------------------

if __name__ == "__main__":
    print("Bot is running...")
    executor.start_polling(dp, skip_updates=True)
```

9.Test.py

```
import asyncio
import bcrypt
import datetime
import io
import matplotlib.dates as mdates

from aiogram.types import (
    Message,
    CallbackQuery,
    InlineKeyboardMarkup,
    InlineKeyboardButton,
    BufferedInputFile
)

from aiogram.types import InputFile
import matplotlib.pyplot as plt
from aiogram import Bot, Dispatcher, types, F
from aiogram.filters import Command
from aiogram.fsm.state import StatesGroup, State
from aiogram.fsm.context import FSMContext
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton, InlineKeyboardMarkup,
InlineKeyboardButton
from aiogram.types import BufferedInputFile
from db import (
    conn, cursor, get_user_by_login, get_user_by_telegram,
    bind_telegram, register_user,
    get_greenhouse_by_user, get_user_greenhouses,
    bind_greenhouse_to_user, get_telemetry_range, insert_log
)

# ------------------------------
# 🔐 TOKEN
# ------------------------------
TOKEN = "6382141506:AAFAVkg3bCRP1DeaR6i5Ng-3Kgv4f02BnD0"

# ------------------------------
# 🔄 FSM
# ------------------------------
class LoginState(StatesGroup):
    waiting_password = State()
```

```
class RegisterState(StatesGroup):
    waiting_password = State()

class GreenhouseState(StatesGroup):
    waiting_id = State()

# ----------------------------
# 📲 BOT INIT
# ----------------------------
bot = Bot(TOKEN)
dp = Dispatcher()

# ----------------------------
# 🔑 KEYBOARDS
# ----------------------------
def profile_menu():
    kb = [
        [KeyboardButton(text="➕ Додати теплицю")],
        [KeyboardButton(text="🌿 Мої теплиці")],
        [KeyboardButton(text="📊 Графіки по теплицям")],
        [KeyboardButton(text="🚪 Вийти")]
    ]
    return ReplyKeyboardMarkup(keyboard=kb, resize_keyboard=True)

def guest_menu(username):
    kb = [[KeyboardButton(text=f"📩 Надіслати логін ({username})")]]
    return ReplyKeyboardMarkup(keyboard=kb, resize_keyboard=True)

# ----------------------------
# ▶ START
# ----------------------------
@dp.message(Command("start"))
async def start_cmd(msg: types.Message):
    username = msg.from_user.username
    if not username:
        await msg.answer("⚠ Встановіть Telegram username у налаштуваннях!")
        return

    user = get_user_by_telegram(msg.from_user.id)
    if user:
        await msg.answer(f"👋 Вітаю назад, {username}!", reply_markup=profile_menu())
    else:
          await msg.answer("Привіт! Натисніть кнопку нижче, щоб надіслати логін:",
reply_markup=guest_menu(username))

# ----------------------------
# 📩 ЛОГІН (користувач надсилає username)
# ----------------------------
@dp.message(F.text.startswith("📩 Надіслати логін"))
async def receive_login(msg: types.Message, state: FSMContext):
    username = msg.from_user.username
    await state.update_data(username=username)
    user = get_user_by_login(username)

    if user:
        await msg.answer(f"Логін отримано: @{username}\nВведіть пароль:")
        await state.set_state(LoginState.waiting_password)
    else:
```

```
        await msg.answer(f"Логін @{username} не знайдено.\nСтворюємо акаунт.\nВведіть пароль:")
        await state.set_state(RegisterState.waiting_password)

# ----------------------------
# 🔑 Вхід: пароль
# ----------------------------
@dp.message(LoginState.waiting_password)
async def login_password(msg: types.Message, state: FSMContext):
    data = await state.get_data()
    username = data["username"]
    password = msg.text

    user = get_user_by_login(username)
    if not user:
        await msg.answer("❌ Користувача не знайдено.")
        await state.clear()
        return

    user_id, password_hash = user
    if bcrypt.checkpw(password.encode(), password_hash.encode()):
        bind_telegram(user_id, msg.from_user.id)
            await msg.answer(f"✅ Авторизація успішна! Вітаю, @{username}!", reply_markup=profile_menu())
    else:
        await msg.answer("❌ Невірний пароль!")

    await state.clear(-------------
@dp.message(RegisterState.waiting_password)
async def register_password(msg: types.Message, state: FSMContext):
    data = await state.get_data()
    username = data["username"]
    password = msg.text

    user_id = register_user(username, password, telegram_id=msg.from_user.id)
    if user_id:
            await msg.answer(f"🎉 Реєстрація успішна!\nВітаю, @{username}!", reply_markup=profile_menu())
    else:
        await msg.answer("❌ Помилка реєстрації.")

    await state.clear()

# ----------------------------
# 👤 ПРОФІЛЬ
# ----------------------------
@dp.message(F.text == "🌿 Мої теплиці")
async def list_greenhouses(msg: types.Message):
    user = get_user_by_telegram(msg.from_user.id)
    if not user:
        await msg.answer("❌ Ви не авторизовані!")
        return
    user_id = user[0]

    gh_list = get_user_greenhouses(user_id)
    if not gh_list:
        await msg.answer("У вас ще немає теплиць 🌱")
        return
```

```
    text = "🌿 Ваші теплиці:\n"
    for gh_id, name in gh_list:
        text += f"• {name} (ID: {gh_id})\n"
    await msg.answer(text)

# ----------------------------
# ➕ Додати теплицю
# ----------------------------
@dp.message(F.text == "➕ Додати теплицю")
async def add_greenhouse_start(msg: types.Message, state: FSMContext):
    user = get_user_by_telegram(msg.from_user.id)
    if not user:
        await msg.answer("❌ Авторизуйтесь через /start")
        return
    await msg.answer("Введіть ID теплиці, яку хочете додати:")
    await state.set_state(GreenhouseState.waiting_id)

@dp.message(GreenhouseState.waiting_id)
async def connect_greenhouse(msg: types.Message, state: FSMContext):
    user_id = get_user_by_telegram(msg.from_user.id)[0]
    try:
        gh_id = int(msg.text.strip())
    except ValueError:
        await msg.answer("❌ Некоректний ID. Введіть число:")
        return

    gh = get_greenhouse_by_user(gh_id)
    if not gh:
        await msg.answer("❌ Теплиця з таким ID не знайдена!")
        return
    gh_id_db, gh_name, gh_owner = gh
    if gh_owner is not None:
        await msg.answer("❌ Ця теплиця вже прив’язана до іншого користувача!")
        return

    bind_greenhouse_to_user(user_id, gh_id)
      await msg.answer(f"🌱 Теплиця {gh_name} (ID: {gh_id}) успішно додана!",
reply_markup=profile_menu())
    await state.clear()

# ----------------------------
# 📊 Графіки
# ----------------------------
@dp.message(F.text == "📊 Графіки по теплицям")
async def show_greenhouse_buttons(msg: types.Message):
    user_id = get_user_by_telegram(msg.from_user.id)[0]
    gh_list = get_user_greenhouses(user_id)
    if not gh_list:
        await msg.answer("У вас ще немає теплиць 🌱")
        return

          kb_buttons = [[InlineKeyboardButton(text=f"{name} (ID:{gh_id})",
callback_data=f"gh_{gh_id}")] for gh_id, name in gh_list]
    kb = InlineKeyboardMarkup(inline_keyboard=kb_buttons)
    await msg.answer("Оберіть теплицю:", reply_markup=kb)

@dp.callback_query(lambda c: c.data.startswith("gh_"))
async def select_period(call):
    await call.answer()
```

```
    gh_id = int(call.data.split("_")[1])
    kb = InlineKeyboardMarkup(inline_keyboard=[
        [InlineKeyboardButton(text="1 година", callback_data=f"g_{gh_id}_1h"),
         InlineKeyboardButton(text="24 години", callback_data=f"g_{gh_id}_24h")],
        [InlineKeyboardButton(text="7 днів", callback_data=f"g_{gh_id}_7d"),
         InlineKeyboardButton(text="30 днів", callback_data=f"g_{gh_id}_30d")]
    ])
       await call.message.answer(f"Оберіть період для теплиці ID {gh_id}:",
reply_markup=kb)

@dp.callback_query(lambda c: c.data.startswith("g_"))
async def show_graph(call: CallbackQuery):
    await call.answer()

    parts = call.data.split("_")
    gh_id = int(parts[1])
    period = parts[2]

    mapping = {
        "1h": datetime.timedelta(hours=1),
        "24h": datetime.timedelta(hours=24),
        "7d": datetime.timedelta(days=7),
        "30d": datetime.timedelta(days=30)
    }
    since = datetime.datetime.now() - mapping[period]

    data = get_telemetry_range(gh_id, since)
    if not data:
        await call.message.answer("❌ Немає даних за обраний період.")
        return

    timestamps = [row[4] for row in data]
    air = [row[0] for row in data]
    hum = [row[2] for row in data]

    # --------------------------
    # 📊 Побудова графіка
    # --------------------------
    plt.figure(figsize=(12, 6))
    plt.plot(timestamps, air, label="Температура повітря")
    plt.plot(timestamps, hum, label="Вологість")
    plt.legend()

    ax = plt.gca()

    # --------------------------
    # 🗓️ Форматування осі X
    # --------------------------
    if period == "30d":  # місяць → кожен день
        ax.xaxis.set_major_locator(mdates.DayLocator(interval=1))
        ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d"))

    elif period == "7d":  # тиждень → кожні 12 годин
        ax.xaxis.set_major_locator(mdates.HourLocator(interval=12))
        ax.xaxis.set_major_formatter(mdates.DateFormatter("%m-%d %H:%M"))

    elif period == "24h":  # доба → кожні 2 години
        ax.xaxis.set_major_locator(mdates.HourLocator(interval=2))
        ax.xaxis.set_major_formatter(mdates.DateFormatter("%H:%M"))
```

```
    elif period == "1h":  # година → кожні 5 хвилин
        ax.xaxis.set_major_locator(mdates.MinuteLocator(interval=5))
        ax.xaxis.set_major_formatter(mdates.DateFormatter("%H:%M"))

    else:  # fallback
        ax.xaxis.set_major_locator(mdates.AutoDateLocator())
        ax.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m-%d"))

    plt.xticks(rotation=45)
    plt.tight_layout()

    # --------------------------
    # 📤 Відправлення графіка
    # --------------------------
    buf = io.BytesIO()
    plt.savefig(buf, format='png', bbox_inches='tight')
    buf.seek(0)
    plt.close()

    photo = BufferedInputFile(buf.read(), filename="graph.png")

    await call.message.answer_photo(
        photo,
        caption=f"Графік теплиці ID {gh_id} за період {period}"
    )

# ----------------------------
# 🚪 ВИХІД
# ----------------------------
@dp.message(F.text == "🚪 Вийти")
async def logout(msg: types.Message):
     cursor.execute("UPDATE users SET telegram_id = NULL WHERE telegram_id = %s",
(msg.from_user.id,))
    conn.commit()
               await      msg.answer("🚪       Ви      вийшли      з      акаунту!",
reply_markup=guest_menu(msg.from_user.username))

# ----------------------------
# ▶ RUN BOT
# ----------------------------
async def main():
    print("Bot started!")
    await dp.start_polling(bot)

if __name__ == "__main__":
    asyncio.run(main())
```

Додаток В

Команди створення БД

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
“ДНІПРОВСЬКА ПОЛІТЕХНІКА”

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ БАЗИ ДАНИХ

Команди створення БД

804.02070743.25018-01 12 01

Листів 9

2025

АНОТАЦІЯ

Представлений комплекс команд реалізує створення БД під отримання даних з мікроконтролера системи, призначеного для інтерактивної взаємодії з кіберфізичною системою моніторингу гідропонних теплиць.

ЗМІСТ

1. Команди створення

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    чатбот_id BIGINT UNIQUE NOT NULL,
    full_name TEXT,
    role TEXT DEFAULT 'user',
    created_at TIMESTAMP DEFAULT NOW()
);
CREATE TABLE greenhouses (
    id SERIAL PRIMARY KEY,
    owner_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
    name TEXT NOT NULL,
    access_key TEXT UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT NOW()
);
CREATE TABLE telemetry (
    id SERIAL PRIMARY KEY,
    greenhouse_id INTEGER NOT NULL REFERENCES greenhouses(id) ON DELETE
CASCADE,
    timestamp TIMESTAMP DEFAULT NOW(),

    temperature REAL,
    humidity REAL,
    water_temperature REAL,
    ph REAL,
    conductivity REAL,
    light_level REAL,
    water_level_1 REAL
);
CREATE TABLE event_logs (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) ON DELETE SET NULL,
    greenhouse_id INTEGER REFERENCES greenhouses(id) ON DELETE CASCADE,
    event_type TEXT NOT NULL,
    description TEXT,
    created_at TIMESTAMP DEFAULT NOW()
);
```