**НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**
**Факультет інформаційних технологій**

| | |
|---|---|
| **ПОГОДЖЕНО** | **ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** |
| **Декан факультету** | **Завідувач кафедри комп'ютерних наук** |
| _____ **Ігор БОЛБОТ** | _____ **Белла ГОЛУБ** |
| (підпис) | (підпис) |
| «___» _____ 2026 р. | «___» _____ 2026 р. |

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему: «Програмне забезпечення системи моніторингу параметрів мікроклімату теплиць»**

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

| | | |
|---|---|---|
| **Гарант освітньої програми**<br>к.т.н., доцент | _____<br>(підпис) | **Ганна ВАЙГАНГ** |
| **Керівник бакалаврської кваліфікаційної роботи** | _____<br>(підпис) | **Юрій НАУРИНСЬКИЙ** |
| **Науковий консультант**<br>к.т.н., доцент | _____<br>(підпис) | **Ганна ВАЙГАНГ** |
| **Виконав** | _____<br>(підпис) | **Сергій СПЄЛОВ** |

**Київ-2026**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри комп'ютерних наук**

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2025 р.

**ЗАВДАННЯ**
**ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ**
**ЗДОБУВАЧУ**

Спєлову Сергію Сергійовичу
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»
Тема бакалаврської кваліфікаційної роботи
«Програмне забезпечення системи моніторингу параметрів мікроклімату теплиць»
затверджена наказом від «19» грудня 2025 р. № 3204 «С»

Термін подання завершеної роботи на кафедру «_22_» __травня__ 2026 р.

Вихідні дані до роботи: опис веборієнтованої системи моніторингу параметрів мікроклімату теплиць; дані сенсорів температури, вологості, освітленості та тиску; інформація про теплиці, користувачів і полив; технології React, FastAPI, PostgreSQL, Docker та ESP32; нормативні й наукові джерела з розробки систем моніторингу та автоматизованого керування мікрокліматом.

Перелік питань що розглядаються: система моніторингу параметрів мікроклімату теплиць як об'єкт дослідження; проєктування інформаційного забезпечення; розробка програмного забезпечення; рекомендації щодо впровадження та експлуатації системи

Перелік графічних документів (за необхідності).

Дата видачі завдання «__22__» ___грудня_____ 2025 р.

| | | |
|---|---|---|
| **Керівник бакалаврської кваліфікаційної роботи** | _____ (підпис) | **Юрій НАУРИНСЬКИЙ** |
| **Науковий консультант** | _____ (підпис) | **Ганна ВАЙГАНГ** |
| **Завдання взяв до виконання** | _____ (підпис) | **Сергій СПЄЛОВ** |

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

ER – Entity-Relationship

ESP32 – Espressif Systems 32-bit microcontroller

GPIO – General-Purpose Input/Output

HTTP – HyperText Transfer Protocol

HTTPS – HyperText Transfer Protocol Secure

REST – Representational State Transfer

SQL – Structured Query Language

UI – User Interface

UML – Unified Modeling Language

Wi-Fi – Wireless Fidelity

СУБД – система управління базами даних

# ВСТУП

Тепличне господарство належить до тих напрямів аграрного виробництва, у яких результат значною мірою залежить від умов внутрішнього середовища. Для нормального росту рослин у теплиці потрібно підтримувати певний температурний режим, контролювати вологість повітря та ґрунту, стежити за освітленістю і своєчасно виконувати полив. Якщо ці показники не контролюються належним чином, це ускладнює процес вирощування, впливає на якість продукції та збільшує витрати ресурсів. У державній статистиці України овочі закритого ґрунту розглядаються як окрема складова рослинницької продукції, що ще раз підтверджує практичну значущість цього напряму [1].

**Актуальність** цієї роботи полягає в потребі створення програмного застосунку, який дозволить впорядкувати спостереження за параметрами мікроклімату теплиці, накопичувати отримані дані, переглядати їх у зручному вигляді та підтримувати прийняття рішень, пов’язаних із поливом. У реальних умовах користувачеві недостатньо мати лише окремі значення з датчиків. Потрібно бачити поточний стан теплиці, мати доступ до попередніх показників, фіксувати події та швидко реагувати на відхилення від нормального режиму. Не менш важливим залишається і питання ефективного використання води, оскільки керування поливом прямо пов’язане з раціональним веденням сільськогосподарського виробництва [2].

**Метою бакалаврської кваліфікаційної роботи** є розробка вебзастосунку інтелектуальної системи моніторингу параметрів мікроклімату теплиць, який забезпечує збір, збереження, відображення та аналіз сенсорних даних, а також підтримує керування окремими процесами, що пов’язані з експлуатацією теплиці.

Для досягнення поставленої мети необхідно виконати аналіз предметної області, розглянути існуючі аналогічні рішення, спроєктувати інформаційне

забезпечення, розробити структуру програмного забезпечення, реалізувати основні функціональні модулі та перевірити їхню роботу під час тестування.

**Об’єкт дослідження** – інформаційні процеси збору, обробки та аналізу даних мікроклімату теплиць.

**Предмет дослідження** – архітектура, алгоритми та програмне забезпечення інтелектуальної системи моніторингу параметрів мікроклімату теплиць.

У роботі використовуються методи аналізу предметної області, моделювання даних і програмних компонентів, а також сучасні засоби веброзробки. Розроблюваний застосунок будується за клієнт-серверним принципом, що дозволяє відокремити інтерфейс користувача, серверну частину та засоби збереження інформації. Такий підхід дає можливість організувати впорядковану роботу з сенсорними даними, журналами подій, переглядом історії показників та подальшим аналізом стану теплиці.

**Практична цінність** роботи полягає в тому, що створений вебзастосунок може бути використаний як основа для подальшого розвитку систем моніторингу тепличного середовища. Його впровадження дозволяє спростити контроль параметрів мікроклімату, скоротити час на спостереження за станом теплиці та покращити своєчасність реагування на зміну вологості ґрунту. Окремі результати роботи можуть бути використані під час підготовки тез, доповідей або демонстраційного матеріалу до захисту.

**Структура записки.** Записка кваліфікаційної роботи складається зі вступу, чотирьох основних розділів та висновків. Також до роботи додається список використаних джерел та додатки. Загальний обсяг записки – 80 сторінок основного тексту, кількість використаних джерел – 22.

У першому розділі «Система моніторингу параметрів мікроклімату теплиць як об’єкт дослідження» розглянуто особливості функціонування теплиці як предметної області, змодельовано основні процеси за допомогою діаграм, проаналізовано існуючі рішення, а також чітко сформульовано завдання, яке покликана вирішити розроблена система.

Другий розділ «Проєктування інформаційного забезпечення» присвячено побудові логічної моделі даних у вигляді ER-діаграми, вибору СУБД, розробці структури інформаційної бази та визначенню фізичної організації бази даних.

Третій розділ «Розробка програмного забезпечення» описує процес створення програмної частини системи: від вибору інструментів і середовищ розробки до реалізації програмних модулів, алгоритмів, а також структури взаємодії компонентів та архітектури застосунку.

У четвертому розділі «Рекомендації щодо впровадження та експлуатації системи» подано результати тестування розробленого програмного забезпечення, визначено вимоги до технічного середовища для його функціонування, описано склад інсталяційного пакета, а також рекомендації щодо впровадження системи.

**Апробація результатів** кваліфікаційної роботи здійснювалася шляхом представлення основних положень і результатів дослідження на VII Всеукраїнській науково-практичній конференції студентів та аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем – 2026», що проводилася на базі Національного університету біоресурсів і природокористування України. У межах конференції було висвітлено питання розробки інтелектуальної системи моніторингу параметрів мікроклімату теплиць, особливості побудови інформаційного забезпечення та програмної реалізації системи. Матеріали дослідження опубліковано у збірнику тез конференції («Інтелектуальна система моніторингу параметрів мікроклімату теплиць»).

# 1 СИСТЕМА МОНІТОРИНГУ ПАРАМЕТРІВ МІКРОКЛІМАТУ ТЕПЛИЦЬ ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ

## 1.1 Аналіз роботи системи моніторингу мікроклімату теплиць як предметної області

Предметною областю цієї роботи є процес спостереження за станом мікроклімату теплиці, оцінювання основних параметрів середовища та прийняття рішень, пов'язаних із підтриманням нормальних умов вирощування рослин. Теплиця належить до об'єктів, у яких навіть незначне відхилення окремих показників може поступово впливати на ріст культури, швидкість вегетації, водоспоживання та загальний результат вирощування. Через це контроль внутрішнього середовища тут має не епізодичний, а постійний характер.

У межах цієї предметної області головну роль відіграють параметри, що безпосередньо характеризують стан теплиці. До них належать температура повітря, вологість повітря, вологість ґрунту, освітленість, а також інші показники, які можуть враховуватися залежно від умов вирощування. На практиці саме ці параметри дають змогу оцінити, чи перебуває середовище в допустимих межах, чи потребує воно коригування. У тепличних комплексах постійний контроль таких величин розглядається як необхідна умова ефективного керування процесом вирощування [3].

Основні параметри мікроклімату теплиці, їх функціональне призначення та вплив на процес вирощування рослин наведено в таблиці 1.1. Наведені параметри формують основу інформаційного потоку системи моніторингу та визначають набір даних, які повинні збиратися, зберігатися й аналізуватися програмним забезпеченням.

Таблиця 1.1

Основні параметри моніторингу мікроклімату теплиці

| Параметр | Призначення контролю | Можливі наслідки відхилення |
|---|---|---|
| Температура повітря | Забезпечення оптимальних умов росту рослин | Уповільнення росту, перегрів або пошкодження культур |
| Вологість повітря | Підтримання стабільного мікроклімату | Пересихання рослин або розвиток грибкових захворювань |
| Вологість ґрунту | Контроль ефективності поливу | Недостатнє або надмірне зволоження кореневої системи |
| Освітленість | Оцінювання достатності світлового режиму | Порушення фотосинтезу та зниження врожайності |
| Стан поливу | Контроль роботи системи зрошення | Нерівномірне зволоження або перевитрати води |
| Сповіщення про критичні стани | Оперативне реагування на зміни середовища | Погіршення умов вирощування та ризик втрати врожаю |

Їх взаємозв’язок безпосередньо впливає на ефективність підтримання стабільного мікроклімату теплиці, своєчасність реагування на критичні ситуації та якість прийняття рішень щодо керування середовищем вирощування рослин.

Особливе значення в системах моніторингу мікроклімату теплиць має контроль вологості ґрунту, оскільки саме цей параметр безпосередньо впливає на режим поливу та стан рослин. Недостатній рівень вологості призводить до погіршення росту культур, тоді як надмірний полив спричиняє нераціональне використання водних ресурсів і порушення оптимального режиму зволоження. Через це показники вологості ґрунту використовуються як основа для прийняття рішень щодо керування тепличним середовищем.

Функціонування системи моніторингу базується на безперервному отриманні сенсорних даних про стан мікроклімату теплиці. Інформація, отримана від сенсорних пристроїв, аналізується користувачем або програмними засобами, після чого визначається необхідність подальших дій. У разі виявлення відхилень від встановлених параметрів виконується запуск системи поливу з подальшим повторним контролем стану ґрунту. Така послідовність формує

циклічний процес спостереження, аналізу, реагування та повторної перевірки середовища [4].

Основними учасниками процесу є сенсорні пристрої збору даних, користувач, система поливу та сервіс повідомлень. Їх взаємодія забезпечує не лише контроль параметрів мікроклімату, а й фіксацію виконаних дій та оперативне інформування про зміни стану теплиці. У результаті моніторинг розглядається як засіб підтримки прийняття рішень, що поєднує збір даних, аналітичну обробку та керування технологічними процесами вирощування рослин.

Отже, предметна область охоплює процеси вимірювання параметрів мікроклімату теплиці, передавання інформації про стан середовища, її аналіз користувачем, реагування на зниження вологості ґрунту та повторний контроль після виконання поливу. Саме така логіка буде покладена в основу подальшого моделювання предметної області, де вже більш наочно будуть показані учасники процесу та зв’язки між ними.

## 1.2 Моделювання предметної області

Моделювання предметної області є важливим етапом проєктування програмного забезпечення, оскільки дає змогу формалізувати основні процеси функціонування системи, визначити взаємодію між її складовими та відобразити логіку обробки даних. Для опису структури та поведінки системи доцільно використовувати UML-моделі, які забезпечують наочне подання учасників процесу, функціональних сценаріїв і взаємозв’язків між окремими компонентами [5].

Предметна область дослідження охоплює процеси моніторингу параметрів мікроклімату теплиці, збору сенсорних даних, оцінювання стану середовища та керування поливом відповідно до отриманих показників. Особлива увага приділяється контролю вологості ґрунту, оскільки цей параметр безпосередньо впливає на прийняття рішень щодо підтримання оптимальних умов вирощування

рослин. Функціонування системи передбачає циклічну послідовність дій, що включає отримання даних, аналіз показників, виконання необхідних керуючих впливів і повторний контроль стану тепличного середовища [3].

Використання UML-діаграм дає змогу відобразити предметну область з різних точок зору: через взаємодію користувача із системою, структуру основних об’єктів, послідовність виконання процесів та логіку реагування на зміну параметрів мікроклімату. Такий підхід забезпечує більш чітке уявлення про архітектуру майбутнього програмного забезпечення та створює основу для подальшого проєктування інформаційного забезпечення і програмної реалізації системи.

Першою моделлю предметної області є діаграма прецедентів, яка використовується для визначення основних учасників системи та функцій, доступних кожному з них. Така модель дає змогу сформувати загальне уявлення про взаємодію користувачів і зовнішніх компонентів із програмним забезпеченням без деталізації внутрішньої реалізації процесів.

Основними учасниками системи виступають користувач, адміністратор і зовнішній пристрій збору даних. Користувач є основною дійовою особою системи, оскільки саме він переглядає поточні показники, аналізує графіки, працює зі сповіщеннями, використовує аналітичні дані, виконує експорт і, за потреби, ініціює полив.

Адміністратор виконує більш службові функції, пов’язані з підтримкою самої системи: керуванням користувачами, теплицями та пристроями. Зовнішній пристрій збору даних бере участь у процесі як джерело інформації про стан середовища та як елемент, що приймає відповідні команди.

Діаграма прецедентів відображає основні ролі, функціональні можливості та взаємозв’язки між учасниками процесу моніторингу параметрів мікроклімату теплиць, що забезпечує цілісне уявлення про логіку функціонування системи (рис. 1).

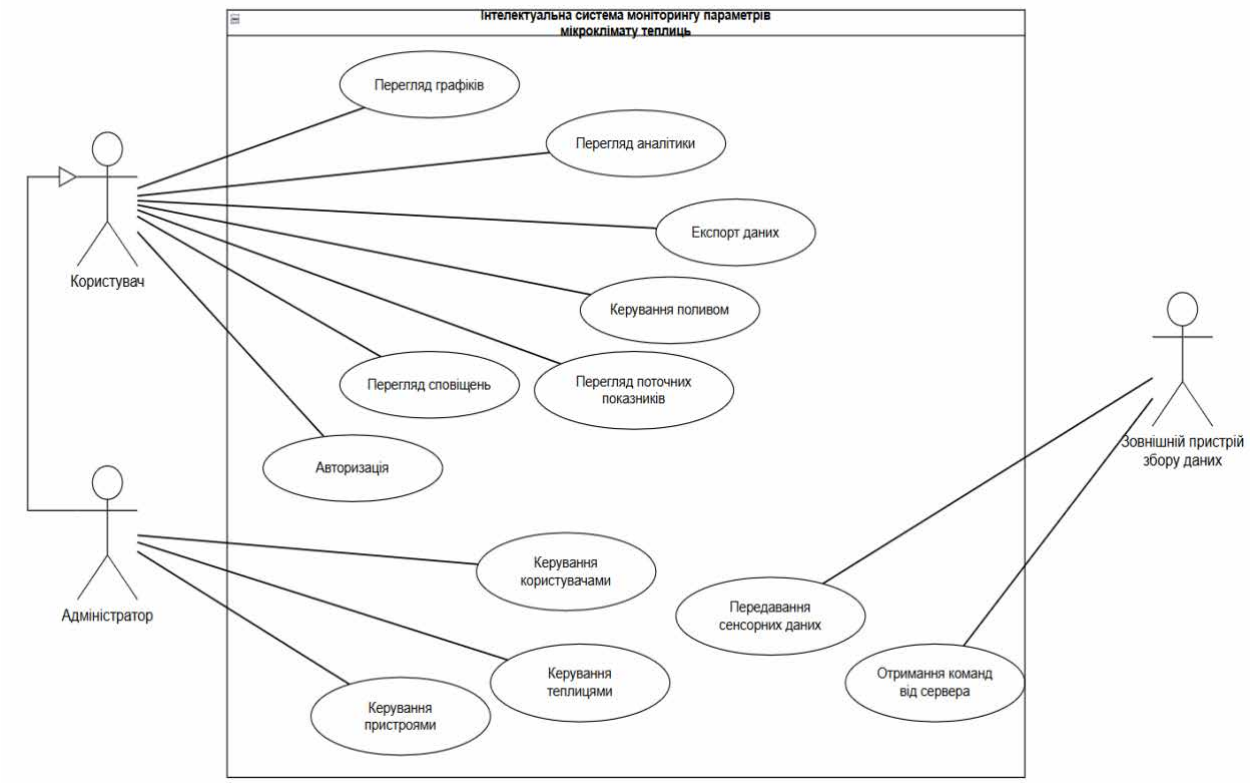


Рис. 1 Діаграма прецедентів

Діаграма прецедентів відображає основні функціональні можливості системи та взаємодію учасників із програмним забезпеченням у процесі моніторингу параметрів мікроклімату теплиці.

Діаграма послідовності використовується для відображення часової послідовності взаємодії між основними учасниками системи під час контролю вологості ґрунту. У межах моделі показано процес отримання сенсорних даних, їх аналізу та формування відповідної реакції на зміну стану середовища.

Зовнішній пристрій збору даних передає параметри мікроклімату користувачу або програмній системі для подальшого оцінювання. У разі виявлення недостатнього рівня вологості ґрунту формується команда на запуск системи поливу. Після виконання поливу модуль сповіщень інформує користувача про зміну стану середовища та виконану дію. Якщо показники перебувають у межах допустимих значень, система продовжує режим спостереження без додаткового втручання.

Діаграма послідовності дає змогу наочно відобразити взаємозв’язок між отриманням даних, аналізом параметрів середовища та реалізацією керуючих дій у системі моніторингу мікроклімату теплиць (рис. 2).

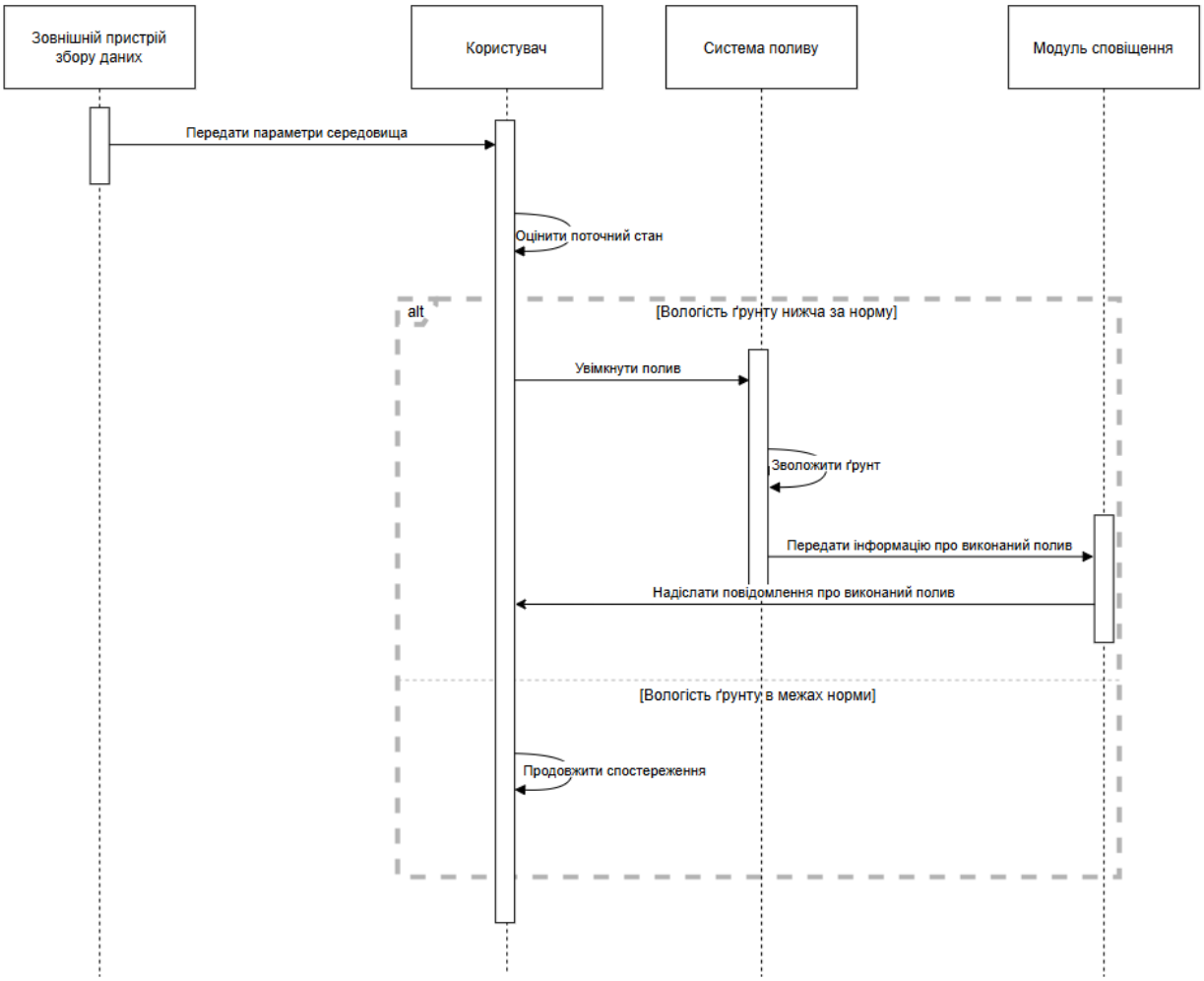


Рис. 2 Діаграма послідовності

Більш детальне уявлення про логіку функціонування системи надає діаграма активності, яка відображає послідовність виконання основних процесів моніторингу та керування мікрокліматом теплиці. На відміну від діаграми послідовності, що акцентує увагу на взаємодії учасників системи, діаграма активності зосереджується на перебігу процесу та умовах переходу між окремими етапами.

Модель охоплює цикл дій від моменту отримання сенсорних показників до завершення контролю стану середовища або виконання поливу. Після вимірювання параметрів мікроклімату дані передаються для аналізу поточного

стану теплиці. Далі виконується перевірка рівня вологості ґрунту. Якщо показник перебуває в межах норми, система продовжує режим спостереження. У разі недостатнього рівня вологості формується команда на запуск системи поливу, після чого здійснюється повторний контроль стану ґрунту.

Діаграма активності відображає циклічний характер процесу моніторингу, за якого перевірка параметрів середовища та керуючі дії повторюються до досягнення необхідного рівня вологості. Після стабілізації показників система формує повідомлення про виконану дію та повертається до режиму подальшого спостереження за станом тепличного середовища (рис. 3).

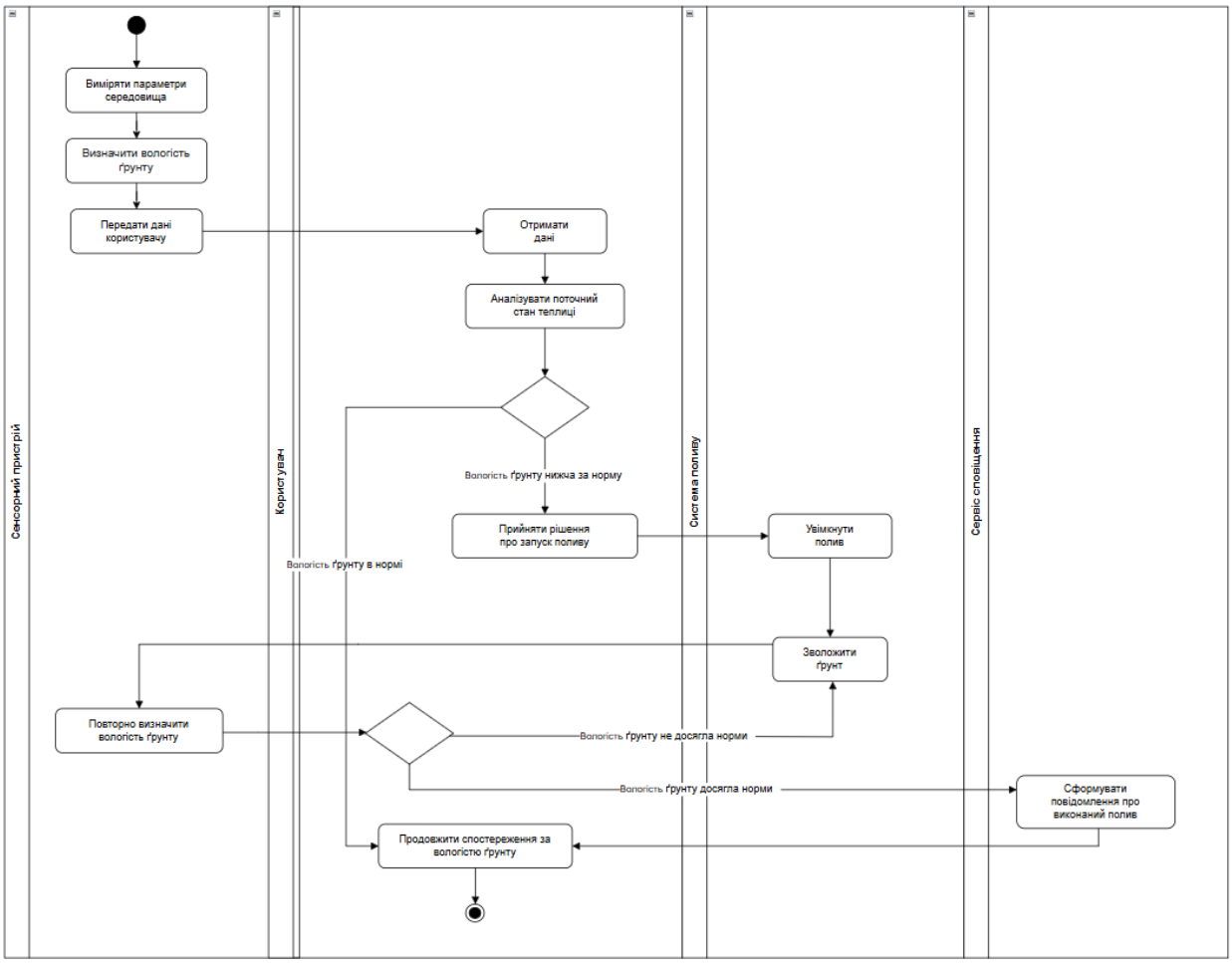


Рис. 3 Діаграма активності

Подані моделі доповнюють одна одну і дозволяють сформувати цілісне уявлення про предметну область. Діаграма прецедентів визначає коло учасників та перелік основних дій. Діаграма послідовності показує часовий порядок взаємодії між ними. Діаграма активності розкриває внутрішню логіку виконання

процесу, включаючи розгалуження та повторний контроль після поливу. У сукупності вони створюють достатню основу для переходу до наступного етапу роботи, а саме до проєктування інформаційного забезпечення та побудови логічної моделі даних.

## 1.3 Огляд існуючих аналогічних систем

Під час розробки програмного застосунку для моніторингу параметрів мікроклімату теплиць доцільно розглянути вже наявні рішення, які використовуються в цій сфері. Це дозволяє оцінити підходи, що застосовуються в готових системах, виявити їхні сильні сторони та визначити ті обмеження, які залишають простір для власної розробки. Огляд аналогів у даній роботі потрібний не лише для загального ознайомлення з ринком, а й для обґрунтування того, чому створення окремого вебзастосунку є виправданим.

Серед відомих рішень у сфері тепличної автоматизації варто насамперед виділити продукцію компанії Priva. Ця компанія спеціалізується на рішеннях для контролю клімату, енергоспоживання та керування технологічними процесами в теплицях. У її продуктах основна увага приділяється саме комплексному керуванню середовищем, коли кліматичні параметри, енергетичні ресурси й технологічні вузли працюють як єдина система. Такий підхід є сильним для великих тепличних господарств, де потрібна стабільність роботи, точне регулювання та широкий набір функцій. Разом з тим подібні рішення зазвичай орієнтовані на промисловий рівень використання, тому вони не завжди є зручними в ситуаціях, коли необхідний більш гнучкий програмний продукт із простішою адаптацією під конкретне завдання [6].

Ще одним помітним прикладом є рішення компанії Ridder, яка розвиває цифрові засоби для тепличного виробництва з акцентом на інтеграцію даних, підключення сенсорів і застосування інструментів цифрового керування. У підході Ridder добре помітна орієнтація на поєднання обладнання, сенсорних систем і програмних сервісів у межах спільного інформаційного простору.

Перевагою таких рішень є відкритість до інтеграції з іншими системами та можливість працювати з даними в більш пов’язаному середовищі. Водночас і тут акцент здебільшого робиться на комплексну виробничу інфраструктуру, а не на окремий вебзастосунок, зосереджений саме на зручному спостереженні, перегляді історії показників, журналюванні подій і підтримці локальних рішень користувача [7].

Схожий напрям демонструє компанія Hoogendoorn, яка пропонує засоби автоматизації для керування кліматом, водою, енергією та технологічними процесами в теплицях. У таких рішеннях значна увага приділяється процесному керуванню, тобто погодженій роботі різних факторів середовища в межах одного обчислювального контуру. Це дозволяє розглядати теплицю як об’єкт, де взаємопов’язані не лише окремі параметри, а й технологічні дії, що з ними пов’язані. Перевагою є високий рівень автоматизації та орієнтація на практичні виробничі задачі. Недолік для нашого випадку полягає в тому, що користувач працює з уже сформованою платформою, а не з легко модифікованим програмним середовищем, яке можна адаптувати під конкретну структуру даних і сценарій роботи [8].

Окремо варто згадати рішення компанії Argus Control Systems, яка також працює у сфері кліматичного контролю й автоматизації для теплиць та інших об’єктів вирощування. Її продукти орієнтовані на точне регулювання параметрів середовища, роботу з тривогами, моніторинг критичних показників і побудову керованого технологічного процесу. Сильна сторона подібного підходу полягає у високій надійності та виробничій спрямованості. Разом з тим такі системи зазвичай формуються навколо професійного апаратно-програмного комплексу, а тому менш придатні для розробки компактного веборієнтованого рішення, яке можна швидко змінювати, доповнювати та пристосовувати до потреб окремого користувача [9].

Для більш наочного порівняння функціональних можливостей і особливостей розглянутих систем доцільно узагальнити їх основні характеристики у вигляді порівняльної табл. 1.2.

Таблиця 1.2

Порівняльна характеристика існуючих систем моніторингу мікроклімату теплиць

| Система | Основне призначення | Переваги | Обмеження |
|---|---|---|---|
| Priva | Комплексне керування кліматом і ресурсами теплиць | Високий рівень автоматизації, точне регулювання параметрів | Орієнтація на великі промислові господарства, складність адаптації |
| Ridder | Інтеграція сенсорних систем і цифрових сервісів | Підтримка інтеграції даних та взаємодії з іншими системами | Акцент на виробничу інфраструктуру, а не на локальний вебзастосунок |
| Hoogendoorn | Автоматизація кліматичних і технологічних процесів | Узгоджене керування середовищем і технологічними вузлами | Обмежені можливості швидкої модифікації під окремі потреби |
| Argus Control Systems | Моніторинг і контроль критичних параметрів середовища | Надійність, підтримка тривог і технологічного контролю | Орієнтація на професійні апаратно-програмні комплекси |
| Розроблю-ваний веб-застосунок | Моніторинг параметрів мікроклімату та підтримка керування поливом | Простота адаптації, вебінтерфейс, накопичення історії даних, аналітика та сповіщення | Менший рівень промислової автоматизації порівняно з комплексними платформами |

Порівняльний аналіз показує, що існуючі промислові рішення орієнтовані переважно на комплексну автоматизацію великих тепличних господарств і характеризуються широким набором функцій керування технологічними процесами. Водночас розроблюваний вебзастосунок зосереджується на задачах моніторингу параметрів мікроклімату, підтримці керування поливом, аналітичному опрацюванні даних і забезпеченні зручного доступу до інформації

через вебінтерфейс, що робить його більш гнучким для локального використання та подальшого розширення функціональних можливостей.

Якщо порівняти ці системи між собою, можна побачити спільну рису: усі вони орієнтовані на глибоку автоматизацію тепличного виробництва та добре закривають задачі промислового керування середовищем. Вони підтримують роботу з датчиками, кліматичними параметрами, технологічними вузлами, механізмами регулювання та сповіщеннями. Для великих господарств це є безумовною перевагою. Проте саме в межах бакалаврської роботи важливим є інший акцент – розробка власного програмного застосунку, у якому головна увага приділяється вебінтерфейсу, логіці обробки даних, структурі інформаційної бази, аналітичному поданню показників і підтримці керування поливом. Іншими словами, у готових промислових системах на першому місці часто стоїть комплексна виробнича автоматизація, тоді як у даній роботі основою виступає програмне рішення, орієнтоване на зручну роботу з даними мікроклімату.

Проведений огляд аналогічних систем дає змогу зробити висновок, що існуючі рішення дійсно мають високий рівень функціональності, але не повністю відповідають задачі, поставленій у цій дипломній роботі. Саме тому розробка власного вебзастосунку для моніторингу параметрів мікроклімату теплиць є виправданою. Вона дозволяє зосередитися на тих можливостях, які є найбільш важливими в контексті цієї роботи: спостереженні за параметрами середовища, накопиченні історії показників, підтримці прийняття рішень щодо поливу, формуванні сповіщень та забезпеченні зручного доступу до інформації через вебінтерфейс.

## 1.4 Постановка завдання

Метою бакалаврської кваліфікаційної роботи є розробка веборієнтованого програмного забезпечення інтелектуальної системи моніторингу параметрів мікроклімату теплиць. Розроблювана система призначена для автоматизованого

збору, обробки, аналізу та візуалізації даних про стан тепличного середовища з можливістю формування рекомендацій щодо підтримання оптимальних умов вирощування рослин.

Актуальність розробки систем моніторингу мікроклімату теплиць обумовлена необхідністю підвищення ефективності тепличного господарства, автоматизації контролю параметрів навколишнього середовища та забезпечення стабільних умов для росту сільськогосподарських культур. Використання сучасних вебтехнологій, засобів візуалізації та інтелектуального аналізу даних дозволяє оперативно реагувати на зміни кліматичних показників і знижувати ризики погіршення стану рослин.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести аналіз предметної області систем моніторингу мікроклімату теплиць та дослідити сучасні підходи до автоматизації контролю кліматичних параметрів.
2. Визначити функціональні та нефункціональні вимоги до програмного забезпечення системи моніторингу.
3. Виконати моделювання предметної області та сформувати структуру взаємодії між користувачем, серверною частиною системи, базою даних і модулями збору кліматичних параметрів.
4. Спроєктувати структуру бази даних для збереження інформації про користувачів, теплиці, датчики, параметри мікроклімату та результати моніторингу.
5. Розробити архітектуру вебзастосунку та реалізувати програмні модулі отримання, обробки й візуалізації кліматичних даних.
6. Реалізувати функції авторизації користувачів, формування статистики, побудови графіків та відображення поточних параметрів мікроклімату.
7. Провести тестування програмного забезпечення та оцінити коректність функціонування системи в умовах роботи з реальними даними.

Об’єкт дослідження – інформаційні процеси збору, обробки та аналізу даних мікроклімату теплиць.

Предмет дослідження – архітектура, алгоритми та програмне забезпечення інтелектуальної системи моніторингу параметрів мікроклімату теплиць.

До вхідних даних системи належать показники температури, вологості повітря, освітленості, рівня вологості ґрунту, концентрації газів та інші параметри, що надходять від датчиків моніторингу тепличного середовища.

Вихідними даними системи є структурована інформація про стан мікроклімату теплиці, графічна візуалізація параметрів, результати аналізу кліматичних показників, статистичні звіти та рекомендації щодо підтримання оптимальних умов вирощування рослин.

# 2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

## 2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних є основою проєктування інформаційного забезпечення системи та призначена для впорядкування інформації, з якою працює програмне забезпечення. На цьому рівні визначаються основні інформаційні об’єкти предметної області, їх атрибути, взаємозв’язки та правила взаємодії між ними без прив’язки до конкретної системи управління базами даних або способів фізичної реалізації [10]. Використання логічної моделі дозволяє сформувати цілісне уявлення про структуру даних системи, забезпечити узгодженість інформаційних потоків і створити основу для подальшого проєктування бази даних.

Формування логічної моделі здійснюється відповідно до функціональних особливостей системи моніторингу параметрів мікроклімату теплиць. У межах предметної області необхідно забезпечити збереження інформації про теплиці, користувачів, сенсорні пристрої, параметри мікроклімату, результати вимірювань, події поливу, сповіщення та історію змін показників. Такий підхід дає змогу підтримувати цілісність даних, організувати накопичення історії спостережень і забезпечити подальшу аналітичну обробку інформації.

Логічна модель даних формується на основі процесів моніторингу мікроклімату, керування поливом, обробки сенсорних показників і формування сповіщень. Оскільки система орієнтована на моніторинг мікроклімату теплиці, збереження сенсорних даних, керування поливом і формування сповіщень, у моделі мають бути відображені як об’єкти середовища, так і об’єкти, пов’язані з роботою користувача та пристроїв. Узагальнена логічна модель даних системи моніторингу параметрів мікроклімату теплиць наведена на рис. 4.

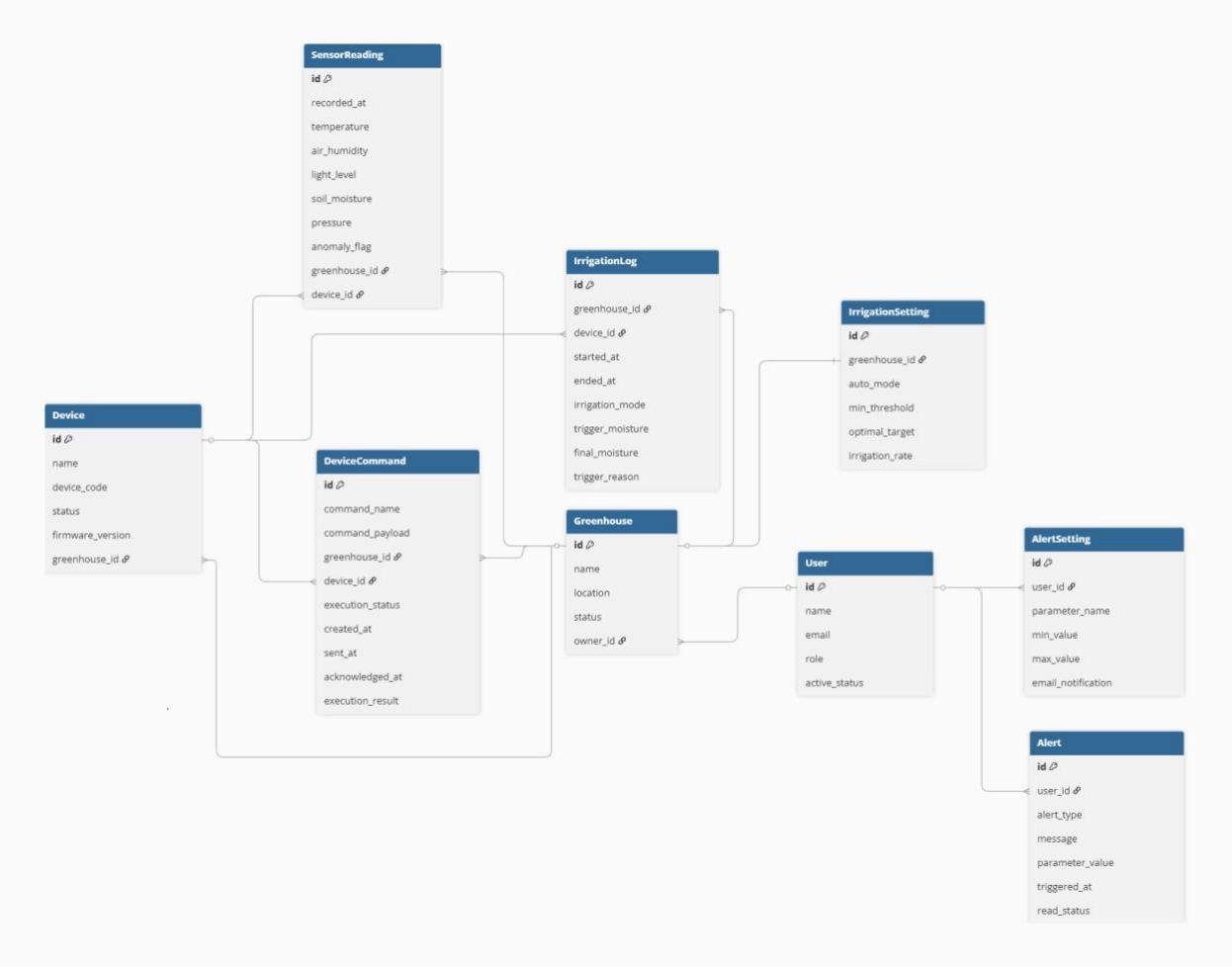


Рис. 4 ER-діаграма логічної моделі даних

Важливим компонентом логічної моделі є сутності, пов’язані з користувачами та адмініструванням системи. Вони забезпечують розмежування доступу, збереження інформації про облікові записи, налаштування користувачів і взаємодію з теплицями та підключеними пристроями. Наявність таких сутностей дозволяє реалізувати механізми авторизації, керування правами доступу та підтримки безпечної роботи програмного забезпечення.

До складу логічної моделі даних входять сутності, які забезпечують збереження інформації про користувачів, теплиці, сенсорні показники, систему поливу, сповіщення та керування пристроями. Основні сутності логічної моделі та їх призначення наведено в табл. 2.1.

Таблиця 2.1

Основні сутності логічної моделі системи

| Сутність | Призначення | Основні атрибути |
|---|---|---|
| User | Зберігання інформації про користувачів системи та розмежування прав доступу | Id, Name, Email, Role, IsActive |
| Greenhouse | Опис теплиці як основного об’єкта моніторингу | Id, Name, Location, Status, UserId |
| Device | Збереження інформації про пристрої збору даних і керування | Id, Name, DeviceCode, Status, FirmwareVersion, GreenhouseId |
| SensorReading | Збереження сенсорних показників мікроклімату | Temperature, Humidity, LightLevel, SoilMoisture, Pressure, Timestamp, IsAnomaly |
| IrrigationSetting | Налаштування автоматичного поливу теплиці | Mode, MinMoisture, TargetMoisture, IrrigationIntensity |
| IrrigationLog | Фіксація історії виконання поливу | StartTime, EndTime, Mode, SoilMoistureBefore, SoilMoistureAfter, TriggerReason |
| Alert | Збереження системних сповіщень | AlertType, Message, ParameterValue, CreatedAt, IsViewed |
| AlertSetting | Правила формування сповіщень | ParameterName, MinValue, MaxValue, AdditionalNotification |
| DeviceCommand | Збереження команд, що передаються пристроям | CommandName, Payload, ExecutionStatus, SentAt, ConfirmedAt, Result |

Сутність User використовується для збереження даних користувачів системи, їх ролей та параметрів доступу. Центральним елементом моделі є сутність Greenhouse, оскільки саме з нею пов’язана більшість інших інформаційних об’єктів системи. Для кожної теплиці можуть бути підключені пристрої збору даних, описані сутністю Device, а також сенсорні записи, які накопичуються у сутності SensorReading.

Для автоматизації процесу зрошення використовуються сутності IrrigationSetting та IrrigationLog. Перша містить параметри автоматичного поливу, тоді як друга забезпечує накопичення історії виконаних подій поливу. Формування повідомлень про критичні відхилення реалізується через сутності Alert та AlertSetting, де окремо зберігаються події сповіщень і правила їх створення. Для взаємодії з IoT-пристроями передбачена сутність DeviceCommand, що забезпечує збереження інформації про передані команди та результати їх виконання.

Після визначення складу сутностей було сформовано зв’язки між ними, що забезпечують цілісність та узгодженість інформаційної моделі. Основні типи зв’язків наведено в табл. 2.2.

Таблиця 2.2

Основні зв’язки між сутностями логічної моделі

| Пов’язані сутності | Тип зв’язку | Характеристика |
|---|---|---|
| User – Greenhouse | Один-до-багатьох | Один користувач може мати декілька теплиць |
| User – Alert | Один-до-багатьох | Один користувач може отримувати багато сповіщень |
| User – AlertSetting | Один-до-багатьох | Один користувач може створювати декілька правил сповіщень |
| Greenhouse – Device | Один-до-багатьох | До однієї теплиці може бути підключено декілька пристроїв |
| Greenhouse – SensorReading | Один-до-багатьох | Для теплиці накопичується історія сенсорних вимірювань |
| Device – SensorReading | Один-до-багатьох | Один пристрій може формувати багато записів |
| Greenhouse – IrrigationSetting | Один-до-одного | Для кожної теплиці задається один набір налаштувань поливу |
| Greenhouse – IrrigationLog | Один-до-багатьох | Теплиця може мати багато подій поливу |
| Device – IrrigationLog | Один-до-багатьох | Один пристрій може брати участь у багатьох подіях поливу |
| Greenhouse – DeviceCommand | Один-до-багатьох | Для теплиці може бути сформовано багато команд |
| Device – DeviceCommand | Один-до-багатьох | Один пристрій може отримувати багато команд |

Побудована логічна модель забезпечує підтримку основних процесів функціонування системи моніторингу мікроклімату теплиці та відповідає принципам реляційного підходу до організації даних. Кожна сутність моделі відображає окремий інформаційний об’єкт предметної області, а встановлені між ними зв’язки забезпечують цілісність даних, усувають надлишковість і впорядковують процес їх збереження та обробки.

У межах сформованої структури сенсорні показники, події поливу, сповіщення, налаштування та команди пристроям зберігаються як окремі взаємопов’язані компоненти єдиної інформаційної системи. Такий підхід створює надійну основу для подальшого проєктування фізичної структури бази даних, спрощує масштабування програмного забезпечення та забезпечує можливість розширення функціоналу без порушення загальної архітектури системи.

## 2.2 Вибір системи управління базами даних

Після побудови логічної моделі даних потрібно вибрати систему управління базами даних, у середовищі якої буде реалізовано інформаційне забезпечення розроблюваного вебзастосунку. Для даної роботи це рішення має принципове значення, оскільки система повинна працювати не з окремими ізольованими записами, а з кількома групами взаємопов’язаних даних. У ній одночасно зберігаються відомості про користувачів, теплиці, пристрої, сенсорні показники, налаштування поливу, журнали поливу, сповіщення та команди пристроїв. Усі ці дані утворюють єдину структуру, тому середовище їх збереження повинно підтримувати чіткі зв’язки між сутностями, забезпечувати цілісність інформації та дозволяти зручно працювати з накопиченими записами.

У межах розроблюваної системи до бази даних висувається кілька важливих вимог. Вона повинна забезпечувати надійне збереження історії сенсорних спостережень, підтримувати зв’язки між теплицями та пристроями, дозволяти фіксувати події поливу і формувати журнали роботи системи. Крім

цього, важливим є зручне виконання вибірок за часовими проміжками, окремими теплицями, пристроями та типами подій. Оскільки частина даних накопичується постійно, система управління базами даних повинна бути придатною до роботи з великим обсягом записів і водночас не ускладнювати побудову реляційної структури.

Для реалізації інформаційного забезпечення в даній роботі обрано PostgreSQL. Ця система належить до класу об’єктно-реляційних систем управління базами даних, що поєднують можливості реляційного подання інформації з розширеними засобами її обробки [11]. Для розроблюваного вебзастосунку це є доцільним, оскільки побудована в роботі модель даних має реляційний характер і містить значну кількість взаємопов’язаних сутностей. PostgreSQL дозволяє реалізувати таку модель у зручному й надійному вигляді, підтримуючи первинні та зовнішні ключі, обмеження цілісності та повноцінну роботу з SQL-запитами.

Окремою перевагою PostgreSQL є її придатність до роботи з даними, що мають часовий характер. У системі моніторингу параметрів мікроклімату теплиць це має практичне значення, оскільки основна частина інформації надходить у вигляді послідовності сенсорних записів, що накопичуються в часі. До цього додаються журнали поливу, історія сповіщень і записи про передані команди пристроям. У такій ситуації важливо не лише зберігати великі обсяги інформації, а й мати змогу швидко працювати з вибірками за певними інтервалами часу, аналізувати дані в розрізі окремих теплиць або пристроїв і не втрачати зв’язок між пов’язаними об’єктами системи.

При виборі СУБД можна було б розглядати і MySQL, яка також є поширеним рішенням у сфері веброзробки та підтримує реляційний підхід до організації даних [12]. Проте для цієї роботи більш доцільним є використання PostgreSQL. Це пов’язано з тим, що в розроблюваній системі важливе не лише саме збереження записів, а й чітке підтримання зв’язків між ними, можливість використання обмежень, організація впорядкованої структури журналів і робота

зі складнішими вибірками. Для задач моніторингу та аналізу накопичених даних така властивість є особливо важливою.

Ще одним аргументом на користь PostgreSQL є її добра інтеграція з сучасними серверними застосунками. Розроблювана система має клієнт-серверну архітектуру, а тому база даних повинна бути придатною до постійної взаємодії із серверною частиною застосунку, що відповідає за приймання сенсорної інформації, формування журналів, роботу зі сповіщеннями та обробку команд для пристроїв. PostgreSQL добре підходить для таких умов, оскільки забезпечує стабільність роботи, підтримує розвиток структури даних і не створює обмежень для подальшого розширення системи.

З урахуванням характеру інформації, побудованої логічної моделі даних і вимог до роботи розроблюваного вебзастосунку вибір PostgreSQL є обґрунтованим. Використання цієї об’єктно-реляційної СУБД дозволяє реалізувати реляційну модель даних, підтримати цілісність інформації, зберігати історію сенсорних показників і забезпечити надійну основу для побудови фізичної структури бази даних.

## 2.3 Розробка структури інформаційної бази

Після вибору системи управління базами даних наступним етапом є розробка структури інформаційної бази. На цьому рівні вже недостатньо лише загального опису сутностей і зв’язків між ними. Потрібно визначити, які саме інформаційні об’єкти будуть зберігатися в системі, як вони будуть розподілені між таблицями та яку роль кожна таблиця виконуватиме в межах вебзастосунку. Для системи моніторингу параметрів мікроклімату теплиць це має особливе значення, оскільки вона одночасно працює з довідковою, поточною, подієвою та керуючою інформацією.

Структура інформаційної бази розроблюваної системи побудована так, щоб відокремити різні за призначенням групи даних. Одна частина таблиць описує постійні або відносно стабільні об’єкти системи, такі як користувачі,

теплиці та пристрої. Інша частина призначена для накопичення динамічних даних, що надходять у процесі роботи системи. До неї належать сенсорні записи, журнали поливу, сповіщення та команди для пристроїв. Окремо виділені таблиці налаштувань, які визначають правила поливу та формування сповіщень. Такий поділ дозволяє зробити структуру бази даних більш впорядкованою та уникнути змішування довідкових і подієвих даних.

Побудова структури інформаційної бази виконувалася з урахуванням функціонального призначення даних, характеру їх використання та інтенсивності оновлення інформації. У процесі проєктування було важливо забезпечити не лише логічний поділ інформаційних об’єктів, а й створити структуру, придатну до подальшого масштабування, накопичення історичних записів і підтримки аналітичної обробки сенсорних даних.

З огляду на особливості функціонування системи моніторингу параметрів мікроклімату теплиць, інформаційну базу доцільно поділити на декілька функціональних груп: довідкові дані, оперативні сенсорні записи, керуючі параметри, журнали подій та службову інформацію. Такий підхід дозволяє впорядкувати роботу з даними, зменшити дублювання інформації та спростити підтримку цілісності бази даних.

У процесі проєктування структури інформаційної бази важливо враховувати не лише склад інформаційних об’єктів, а й характер їх використання у програмному забезпеченні. Частина даних у системі має постійний або довготривалий характер, наприклад відомості про користувачів, теплиці чи пристрої, тоді як інша частина формується безперервно в процесі роботи системи у вигляді сенсорних вимірювань, подій поливу, сповіщень і команд. Через це структура інформаційної бази повинна забезпечувати впорядковане розмежування різних типів інформації, підтримувати цілісність зв’язків між таблицями та створювати умови для ефективного накопичення й подальшої аналітичної обробки даних мікроклімату теплиць. Саме з цією метою таблиці бази даних були згруповані відповідно до їх функціонального призначення.

Для узагальнення структури інформаційної бази основні таблиці системи та їх функціональне призначення наведено в табл. 2.3.

Таблиця 2.3

Функціональний поділ таблиць інформаційної бази

| Група таблиць | Таблиці | Призначення |
|---|---|---|
| Довідкові дані | users, greenhouses, managed_devices | Збереження інформації про користувачів, теплиці та підключені пристрої |
| Сенсорні дані | sensor_readings | Накопичення та збереження показників мікроклімату |
| Керування поливом | irrigation_settings, irrigation_logs | Налаштування режимів поливу та журналювання виконаних дій |
| Система сповіщень | alerts, alert_settings | Формування повідомлень та збереження правил їх створення |
| Керування пристроями | device_commands | Передавання та контроль виконання команд пристроям |
| Службова інформація | audit_logs | Фіксація службових подій та дій у системі |

Наведений функціональний поділ дозволяє розмежувати постійні, динамічні та службові дані системи. Довідкові таблиці містять відносно стабільну інформацію про об’єкти предметної області, тоді як сенсорні таблиці забезпечують накопичення великої кількості часових записів. Окреме виділення таблиць керування поливом, сповіщень та команд пристроям дозволяє підтримувати незалежність окремих функціональних підсистем і спрощує подальший розвиток програмного забезпечення.

Особливістю структури інформаційної бази є те, що центральне місце в ній займає таблиця greenhouses, оскільки саме через неї реалізується зв’язок між сенсорними вимірюваннями, налаштуваннями поливу, журналами подій та підключеними пристроями. Такий підхід забезпечує цілісність інформаційної моделі та дозволяє організувати зручний доступ до даних у межах окремої теплиці.

Отже, сформована структура інформаційної бази забезпечує впорядковане збереження довідкових, сенсорних, подієвих і керуючих даних системи моніторингу мікроклімату теплиць. Реалізований поділ таблиць за функціональним призначенням створює основу для підтримання цілісності інформації, подальшого накопичення історичних записів і побудови фізичної структури бази даних у середовищі PostgreSQL.

## 2.4 Схема та фізична структура бази даних

Після визначення логічної моделі даних, вибору системи управління базами даних і розробки структури інформаційної бази наступним етапом є побудова фізичної структури бази даних. На цьому рівні вже не йдеться лише про загальні сутності та їхні зв’язки. Тут визначається, яким чином ці сутності реалізуються у вигляді конкретних таблиць, які поля вони містять, як між ними підтримуються зв’язки та якими засобами забезпечується цілісність інформації. Якщо логічна модель відображала змістову сторону даних, то фізична структура показує їх практичну реалізацію в середовищі PostgreSQL.

На відміну від логічної моделі, яка відображає переважно змістову організацію інформації та взаємозв’язки між сутностями предметної області, фізична структура бази даних орієнтована на практичну реалізацію механізмів збереження й обробки інформації. Саме на цьому етапі визначаються технічні особливості організації таблиць, способи зберігання часових і сенсорних даних, використання ключів, обмежень та інших засобів підтримання цілісності інформаційної системи. Для системи моніторингу параметрів мікроклімату теплиць це має особливе значення, оскільки база даних повинна забезпечувати безперервне накопичення вимірювань, підтримку журналів подій та стабільну роботу вебзастосунку в умовах постійного оновлення даних.

Фізична модель розроблюваної бази даних побудована відповідно до інформаційних потреб вебзастосунку моніторингу параметрів мікроклімату теплиць. Її структура охоплює дані про користувачів системи, теплиці,

підключені пристрої, сенсорні записи, налаштування і журнали поливу, сповіщення та команди, що передаються пристроям. Усі ці дані утворюють єдину взаємопов’язану систему, у межах якої кожна таблиця виконує окрему функцію, але не існує ізольовано від інших. У загальному вигляді фізична структура бази даних наведена на рис. 5.

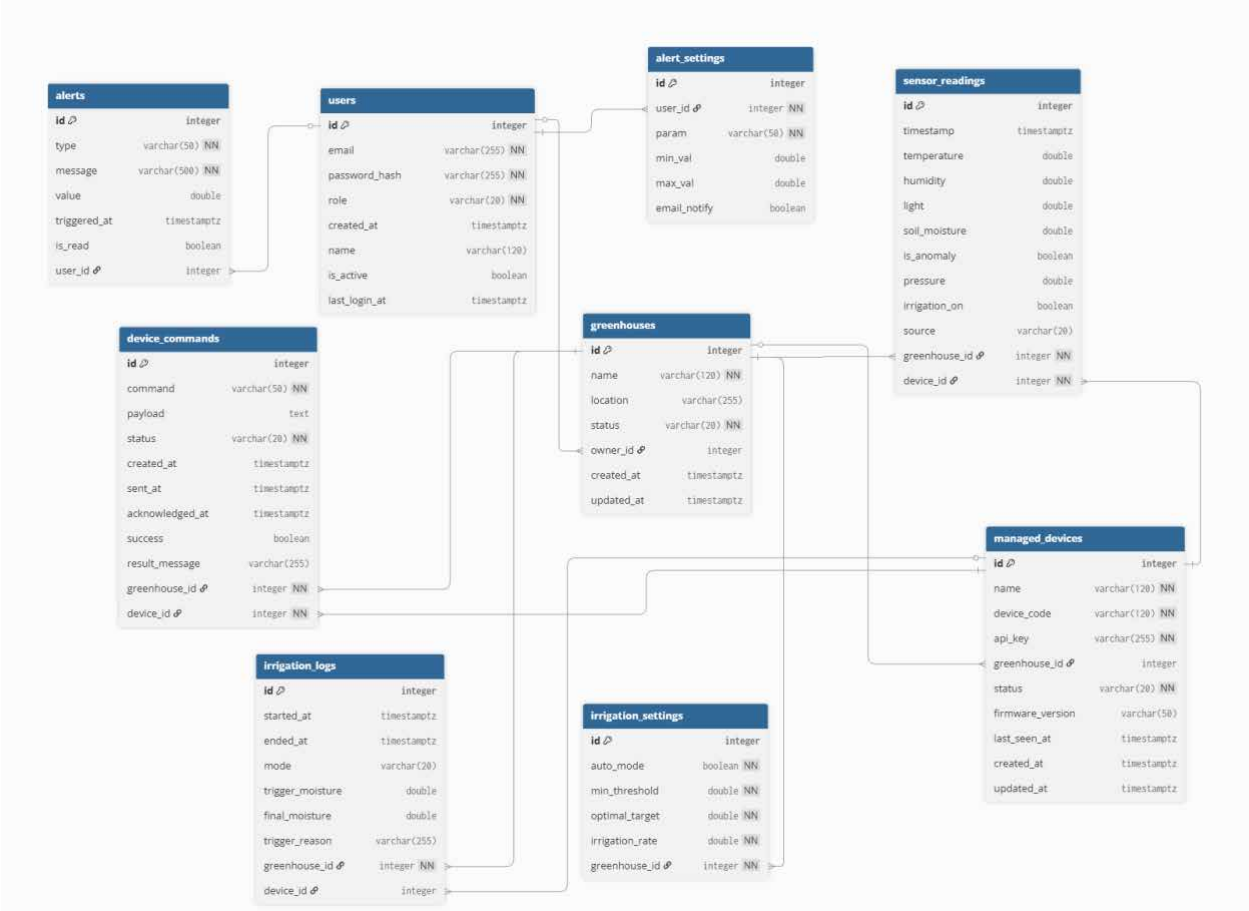


Рис. 5 ER-діаграма фізичної моделі бази даних

Після формування логічної структури інформаційної бази та визначення складу основних таблиць наступним етапом стало проєктування фізичної структури бази даних у середовищі PostgreSQL. На цьому рівні визначаються конкретні параметри реалізації таблиць, типи даних полів, способи організації зв’язків між таблицями, механізми підтримання цілісності інформації та особливості збереження сенсорних і службових записів.

Фізична структура бази даних реалізована відповідно до реляційного підходу та орієнтована на підтримку клієнт-серверної архітектури вебзастосунку. Особливістю системи є постійне накопичення сенсорних

показників мікроклімату, журналів подій поливу, команд пристроям і сповіщень, тому структура бази даних повинна забезпечувати ефективне збереження часових записів, підтримку зв’язків між сутностями та можливість виконання швидких вибірок за теплицями, пристроями та часовими інтервалами.

У фізичній моделі реалізовано окремі таблиці для довідкових даних, сенсорних вимірювань, журналів подій, налаштувань і службової інформації. Зв’язки між таблицями підтримуються за допомогою первинних і зовнішніх ключів, що забезпечує цілісність інформаційної структури та узгодженість даних під час роботи системи.

Загальну характеристику фізичної структури бази даних наведено в табл. 2.4.

Таблиця 2.4

Основні характеристики фізичної структури бази даних

| Ім’я таблиці | Основне призначення | Ключові поля | Особливості реалізації |
|---|---|---|---|
| users | Дані користувачів системи | id, email | Авторизація та ролі доступу |
| greenhouses | Дані про теплиці | id, user_id | Центральна таблиця зв’язків |
| managed_devices | Дані про пристрої | id, greenhouse_id | Ідентифікація контролерів |
| sensor_readings | Сенсорні вимірювання | id, device_id, timestamp | Великий обсяг часових даних |
| irrigation_settings | Налаштування поливу | greenhouse_id | Індивідуальні параметри поливу |
| irrigation_logs | Журнал поливу | id, greenhouse_id | Історія керуючих дій |
| alerts | Системні сповіщення | id, user_id | Фіксація критичних подій |
| alert_settings | Правила сповіщень | id, user_id | Налаштування порогових значень |
| device_commands | Команди пристроям | id, device_id | Контроль виконання команд |
| audit_logs | Службові журнали | id, created_at | Аудит дій у системі |

Під час реалізації фізичної структури бази даних особлива увага приділялася вибору типів даних. Для ідентифікаторів таблиць використовуються цілочисельні поля типу integer або bigint із автоматичною генерацією значень. Для текстових параметрів застосовуються типи varchar та text, а для сенсорних показників – numeric або double precision, що дозволяє коректно зберігати виміряні значення з плаваючою комою.

Для часових характеристик використовується тип timestamptz, який забезпечує коректне збереження часових міток із врахуванням часових поясів. Це є важливим для системи моніторингу, оскільки більшість операцій пов’язана з часовою послідовністю подій: отриманням сенсорних даних, запуском поливу, формуванням сповіщень і передаванням команд пристроям.

Підтримання цілісності даних забезпечується використанням первинних і зовнішніх ключів, обмежень NOT NULL, UNIQUE та CHECK. Зокрема, для сенсорних параметрів можуть використовуватися обмеження допустимих діапазонів значень, що дозволяє запобігати запису некоректних показників у базу даних.

Для таблиць sensor_readings, irrigation_logs та alerts доцільним є використання індексування за часовими полями та зовнішніми ключами, оскільки саме ці таблиці накопичують найбільший обсяг записів і найчастіше використовуються під час вибірок та аналітичної обробки даних.

Фізична структура бази даних узгоджується з логічною моделлю предметної області та забезпечує практичну реалізацію інформаційного забезпечення системи моніторингу параметрів мікроклімату теплиць у середовищі PostgreSQL. Реалізований підхід дозволяє підтримувати цілісність даних, накопичувати історію сенсорних вимірювань, забезпечувати роботу механізмів поливу й сповіщень та створює основу для подальшого масштабування програмного забезпечення.

# 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 3.1 Абстракції предметної області

На етапі розробки програмного забезпечення важливо виділити основні абстракції предметної області, адже саме вони дають змогу перейти від загального розуміння системи до її подальшого подання у вигляді програмних об’єктів. Такі абстракції відображають найбільш суттєві елементи предметної області, їхні характерні властивості та основні обов’язки, які необхідно врахувати під час побудови логіки програмного забезпечення [13].

Для системи моніторингу параметрів мікроклімату теплиць це має особливе значення, оскільки вона поєднує спостереження за станом середовища, роботу з пристроями, керування поливом і своєчасне інформування користувача.


Рис. 6 Абстракції предметної області

Центральною абстракцією є теплиця, оскільки саме вона виступає основним об’єктом спостереження та керування. Для теплиці важливими властивостями є назва, розташування та поточний стан. Її обов’язки полягають у відображенні стану середовища, а також у поєднанні даних вимірювання мікроклімату з процесом поливу. Саме через теплицю користувач отримує

уявлення про поточну ситуацію та може виконувати дії, пов’язані з керуванням поливом.

Абстракція користувач відображає особу, яка працює з теплицею. До основних властивостей користувача належать ім’я, електронна адреса та роль. У межах предметної області користувач переглядає стан теплиці, керує поливом та отримує сповіщення про події, які потребують уваги. Це дозволяє розглядати користувача як активного учасника процесу моніторингу та прийняття рішень.

Абстракція контролер мікроклімату описує пристрій, який забезпечує отримання даних із датчиків. Його основними властивостями є код пристрою, стан та версія прошивки. Контролер відповідає за збирання сенсорних даних, передавання цих даних до системи та передавання команди системі поливу. У предметній області він виконує роль проміжної ланки між фізичними показниками середовища та подальшою обробкою цих показників.

Абстракція вимірювання мікроклімату відображає зафіксований набір показників середовища у певний момент часу. До її властивостей належать температура повітря, вологість повітря, вологість ґрунту, освітленість, тиск і час вимірювання. Основне призначення цієї абстракції полягає у фіксації поточного стану середовища, визначенні потреби у поливі та виявленні відхилень від допустимих меж.

Абстракція система поливу описує процес зволоження ґрунту в теплиці. Вона має такі властивості, як режим поливу, поріг вологості, цільовий рівень зволоження та стан поливу. Її обов’язками є зволоження ґрунту, підтримка необхідного рівня вологості та запуск або зупинка поливу. Завдяки цій абстракції у моделі відображено не лише спостереження за середовищем, а й можливість активного впливу на його стан.

Окремою абстракцією є сповіщення, яке використовується для інформування користувача про важливі події. Сповіщення має тип повідомлення, текст, час спрацювання та стан перегляду. Його обов’язки полягають в інформуванні користувача про відхилення параметрів, повідомленні про виконаний полив та фіксації подій, які потребують уваги.

Виділені абстракції формують змістову основу предметної області та відображають основні об’єкти системи моніторингу теплиці, їхні властивості, функції та взаємозв’язки. На їх основі визначається структура програмних об’єктів, логіка взаємодії між ними та механізми обробки даних, пов’язаних із моніторингом мікроклімату, керуванням поливом і формуванням сповіщень. Діаграми класів і кооперацій деталізують склад об’єктів системи, їхні зв’язки та сценарії взаємодії під час виконання основних функціональних процесів.

## 3.2 Моделювання структури та взаємодії об’єктів

Після визначення основних абстракцій предметної області наступним етапом є моделювання структури та взаємодії об’єктів. На цьому етапі важливо не лише перелічити складові майбутнього програмного забезпечення, а й показати, як вони пов’язані між собою та яку роль виконують у загальній логіці роботи системи. Для системи моніторингу параметрів мікроклімату теплиць таке моделювання є особливо важливим, оскільки вона поєднує декілька взаємопов’язаних процесів: отримання сенсорних даних, відображення стану теплиці, керування поливом і надсилання сповіщень користувачу.

Для подання структури об’єктів використовується мова UML, яка застосовується для моделювання програмних систем, опису їхніх складових та взаємозв’язків між ними [5]. Для моделювання структури та взаємодії об’єктів використано діаграми класів і кооперацій. Діаграма класів відображає статичну структуру системи, тобто класи, їхні атрибути, операції та зв’язки. Діаграми кооперацій, у свою чергу, показують взаємодію об’єктів у межах окремих сценаріїв роботи.

Побудова таких діаграм дозволяє перейти від загального опису предметної області до більш конкретної моделі майбутнього програмного забезпечення. Завдяки цьому можна визначити, які об’єкти будуть основними у системі, які дані вони зберігатимуть, які дії виконуватимуть та яким чином взаємодіятимуть між собою.

### 3.2.1 Діаграма класів

Діаграма класів є одним із основних засобів об’єктно-орієнтованого моделювання. Вона дає змогу представити програмну систему як сукупність класів, кожен із яких має власні атрибути, операції та зв’язки з іншими класами. На відміну від абстракцій предметної області, діаграма класів є більш деталізованою, оскільки вона вже відображає майбутню логічну структуру програмного забезпечення.

На рис. 7 зображено діаграму класів із асоціаціями.

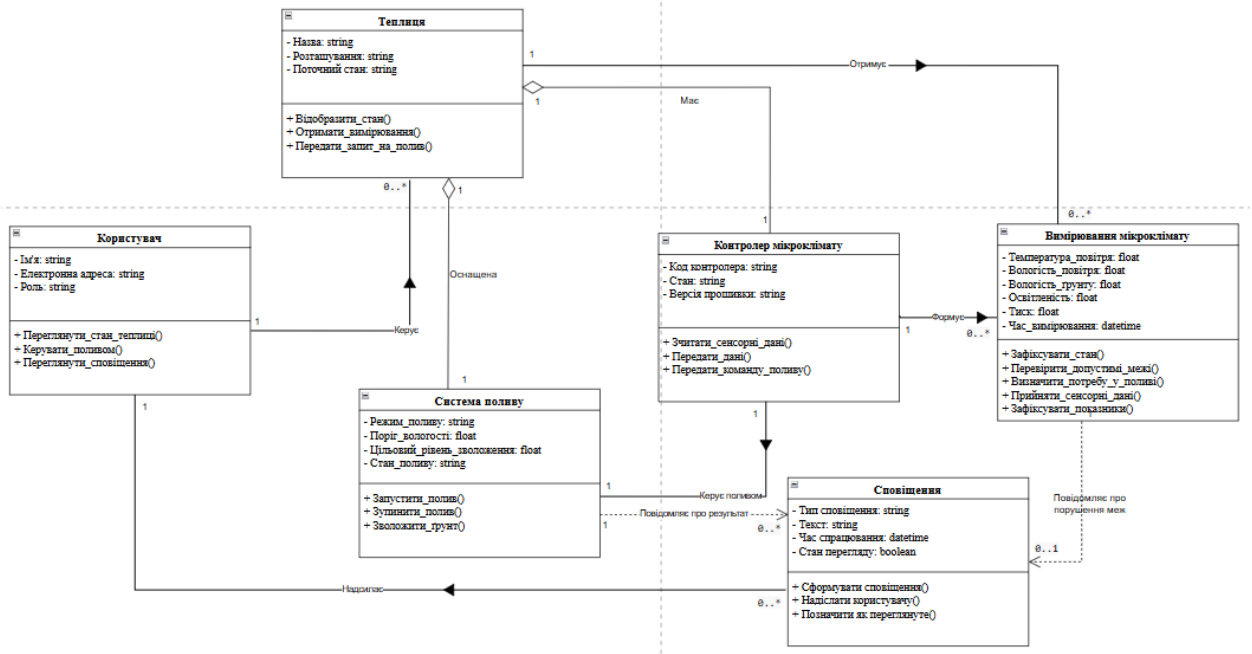


Рис. 7 Діаграма класів із асоціаціями

Діаграма класів включає шість основних класів: Користувач, Теплиця, Контролер мікроклімату, Вимірювання мікроклімату, Система поливу та Сповіщення. Ці класи були сформовані на основі абстракцій предметної області та відображають ключові об’єкти, необхідні для реалізації функціональності системи.

Клас Користувач описує особу, яка взаємодіє із системою. Його атрибутами є ім’я, електронна адреса та роль. До основних операцій класу належать перегляд стану теплиці, керування поливом і перегляд сповіщень. Такий набір операцій відповідає основним діям користувача у межах предметної області.

Клас Теплиця є центральним у моделі, оскільки саме теплиця виступає основним об’єктом моніторингу та керування. Вона має назву, розташування та поточний стан. Операції цього класу дають змогу відобразити стан теплиці, отримати вимірювання мікроклімату та передати запит на полив. Таким чином, теплиця поєднує дані моніторингу та дії, пов’язані з підтриманням належного стану середовища.

Клас Контролер мікроклімату відповідає за роботу з сенсорними даними. Його атрибутами є код контролера, стан та версія прошивки. Операції класу передбачають зчитування сенсорних даних, передавання даних та передавання команди поливу. У моделі контролер виконує роль джерела показників середовища та елемента, через який може здійснюватися керуючий вплив на систему поливу.

Клас Вимірювання мікроклімату відображає набір показників, отриманих у певний момент часу. Він містить температуру повітря, вологість повітря, вологість ґрунту, освітленість, тиск і час вимірювання. Операції цього класу пов’язані з прийняттям сенсорних даних, фіксацією поточного стану, перевіркою допустимих меж і визначенням потреби у поливі. Саме цей клас дозволяє відокремити сирі показники середовища від загального стану теплиці.

Клас Система поливу описує частину предметної області, відповідальну за зволоження ґрунту. Його атрибутами є режим поливу, поріг вологості, цільовий рівень зволоження та стан поливу. Операції класу передбачають запуск поливу, зупинку поливу та зволоження ґрунту. Завдяки цьому у моделі відображено не лише спостереження за мікрокліматом, а й можливість впливу на нього.

Клас Сповіщення призначений для інформування користувача про події, які потребують уваги. Він містить тип повідомлення, текст повідомлення, час спрацювання та стан перегляду. Основними операціями є формування сповіщення, надсилання користувачу та позначення його як переглянутого.

Між класами встановлено зв’язки, які відображають логіку їхньої взаємодії. Користувач пов’язаний із теплицею асоціацією, оскільки він переглядає її стан і керує поливом. Одна теплиця може мати вимірювання

мікроклімату, які накопичуються в процесі моніторингу. Теплиця також пов’язана з контролером мікроклімату та системою поливу, оскільки ці елементи забезпечують отримання даних і вплив на стан ґрунту.

Контролер мікроклімату пов’язаний із вимірюванням мікроклімату, оскільки саме через нього надходять сенсорні дані, на основі яких фіксуються показники середовища. Також контролер взаємодіє із системою поливу, передаючи команду на її запуск або зупинку. Окремо показано зв’язок вимірювання мікроклімату зі сповіщенням: якщо показники виходять за допустимі межі, може бути сформовано повідомлення для користувача. Система поливу також пов’язана зі сповіщенням, оскільки після виконання поливу користувач має отримати інформацію про результат.

Таким чином, діаграма класів відображає логічну структуру майбутньої системи та показує, як основні об’єкти предметної області перетворюються на програмні класи. Вона є основою для подальшого проєктування програмної архітектури та реалізації функціональних модулів.

### 3.2.2 Діаграми кооперацій

Діаграми кооперацій використовуються для моделювання взаємодії між об’єктами у межах конкретних сценаріїв. Якщо діаграма класів показує загальну структуру системи, то діаграма кооперації дає змогу простежити, як окремі об’єкти взаємодіють між собою під час виконання певної дії. Для відображення сценаріїв взаємодії між об’єктами використано дві діаграми кооперацій: перегляд стану теплиці та запуск поливу з отриманням повідомлення.

На рис. 8 зображено діаграму кооперації для сценарію перегляду стану теплиці. У цьому сценарії беруть участь чотири об’єкти: Користувач, Теплиця, Контролер мікроклімату та Вимірювання мікроклімату. Логіка взаємодії полягає в тому, що контролер мікроклімату передає сенсорні дані до об’єкта вимірювання мікроклімату. Після цього вимірювання фіксує отримані показники, перевіряє їх на відповідність допустимим межам та оновлює поточний стан теплиці. Користувач, у свою чергу, переглядає стан теплиці, отримуючи вже опрацьовану інформацію про параметри середовища.

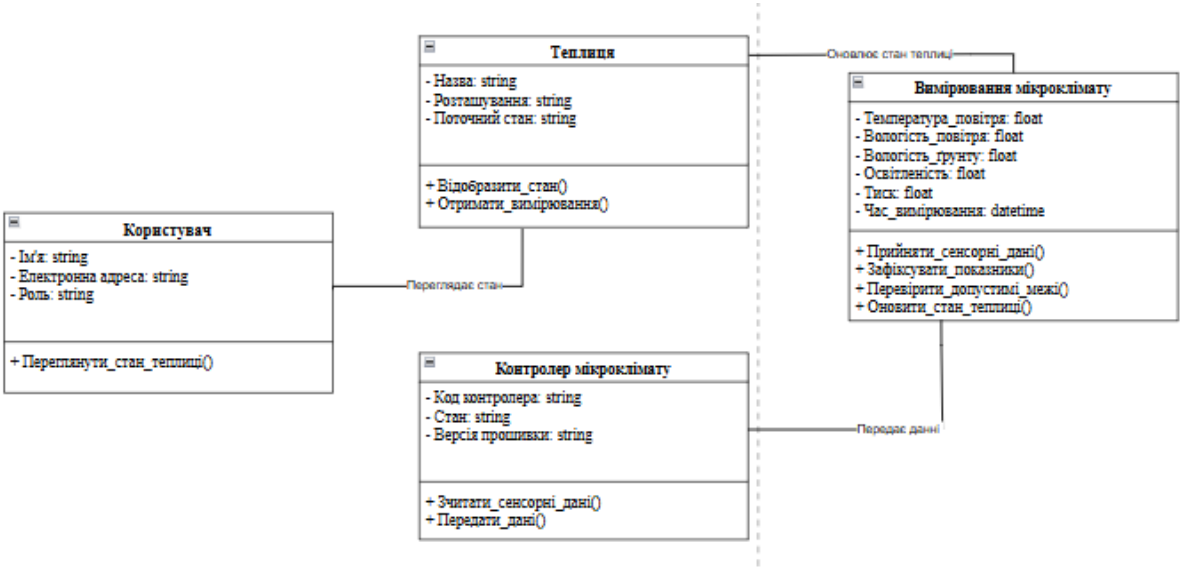


Рис. 8 Діаграма кооперації: перегляд стану теплиці

Такий сценарій дозволяє показати, що користувач не працює безпосередньо з контролером або окремими сенсорами. Для нього головним об’єктом взаємодії є теплиця, стан якої формується на основі вимірювань мікроклімату. Це робить модель зрозумілою та відповідає логіці предметної області: контролер отримує дані, вимірювання їх фіксує, теплиця відображає поточний стан, а користувач переглядає результат.

На рис. 9 зображено діаграму кооперації для сценарію запуску поливу та отримання повідомлення. У цьому сценарії взаємодіють об’єкти Користувач, Теплиця, Система поливу та Сповіщення. Користувач ініціює керування поливом через теплицю. Теплиця передає відповідний запит до системи поливу, після чого система поливу виконує запуск, зволожує ґрунт та змінює власний стан. Після виконання дії формується сповіщення, яке надсилається користувачу.

Ця кооперація відображає логіку ручного керування поливом і підтвердження результату. Вона показує, що користувач не взаємодіє напряму з усіма внутрішніми елементами, а виконує дію через теплицю як основний об’єкт керування. Сповіщення завершує сценарій, оскільки повідомляє користувача про результат виконаної дії.

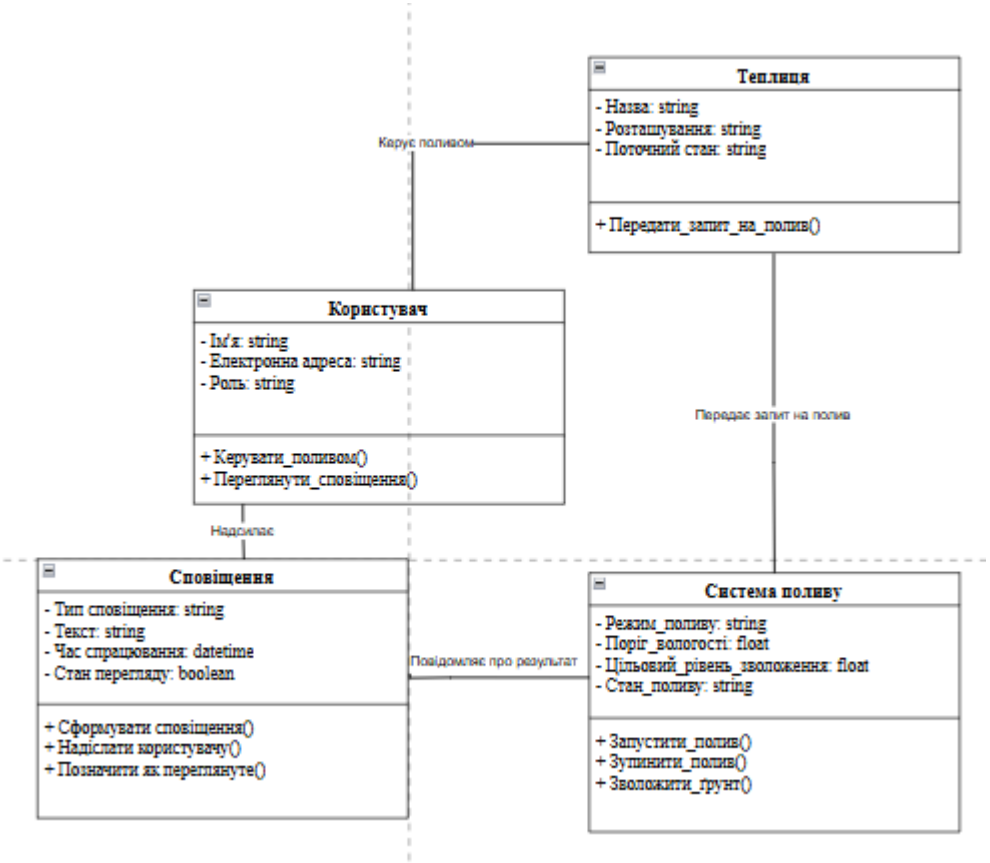


Рис. 9 Діаграма кооперації: запуск поливу та отримання повідомлення

Діаграми кооперацій доповнюють діаграму класів і показують практичне використання її об’єктів у конкретних сценаріях. Перша кооперація демонструє процес отримання та відображення актуального стану теплиці, а друга – процес запуску поливу та інформування користувача про результат. Сукупно ці діаграми підтверджують логіку взаємодії між основними класами та формують основу для подальшої реалізації програмного забезпечення.

## 3.3 Організаційна структура програмного забезпечення

Організаційна структура програмного забезпечення відображає логічний поділ розроблюваної системи на окремі функціональні частини. Такий поділ дає змогу краще зрозуміти, які модулі входять до складу системи, за які функції вони відповідають та як взаємодіють між собою під час виконання основних процесів. Для подання організаційної структури було використано діаграму пакетів UML,

яка дозволяє згрупувати елементи програмного забезпечення за їхнім призначенням і показати залежності між ними.

Діаграма пакетів відображає логічний поділ програмного забезпечення на функціональні модулі (рис. 10).

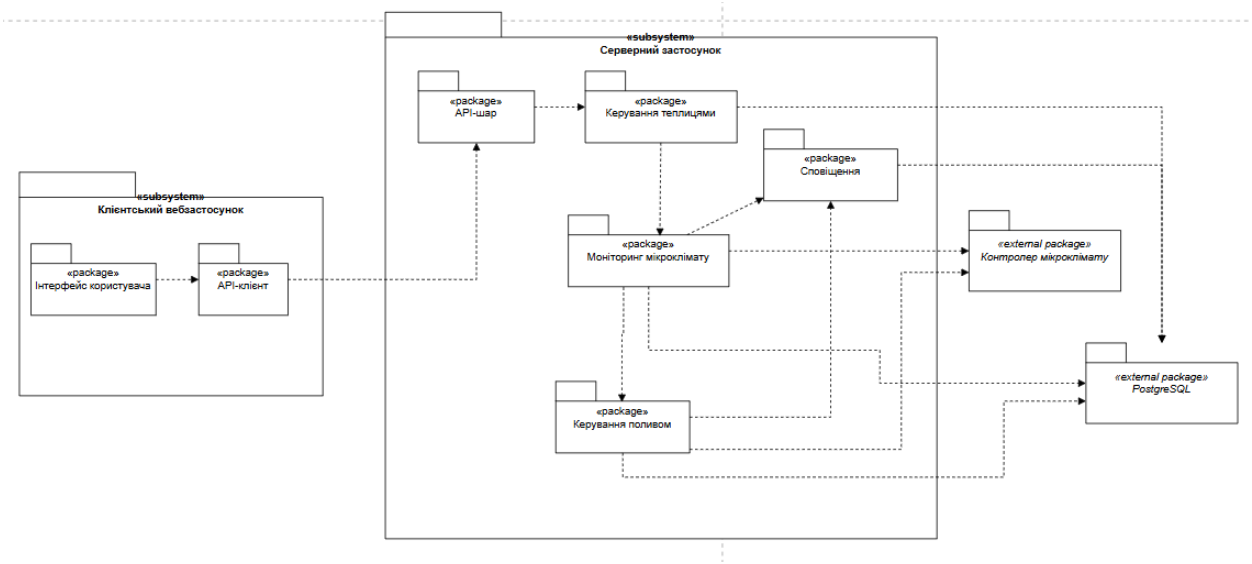


Рис. 10 Діаграма пакетів

Представлена діаграма пакетів відображає архітектурну організацію програмного забезпечення та демонструє логічний розподіл системи на окремі функціональні компоненти. Використання пакетного поділу дозволяє реалізувати модульний принцип побудови вебзастосунку, за якого окремі частини системи виконують власні функції та взаємодіють між собою через визначені механізми обміну даними. Такий підхід забезпечує зниження залежності між компонентами, спрощує супровід програмного забезпечення та створює умови для подальшого масштабування системи без порушення загальної логіки її функціонування.

Для узагальнення функціонального призначення основних пакетів програмного забезпечення та їх ролі у структурі системи доцільно подати характеристику основних компонентів у вигляді табл. 3.1.

Таблиця 3.1

Функціональне призначення пакетів програмного забезпечення

| Пакет | Призначення | Основні функції |
|---|---|---|
| Інтерфейс користувача | Взаємодія користувача із системою | Відображення сторінок, графіків, форм керування |
| API-клієнт | Обмін даними між frontend та backend | Формування та передавання HTTP-запитів |
| API-шар | Маршрутизація серверних запитів | Передавання запитів до функціональних модулів |
| Керування теплицями | Робота з інформацією про теплиці | Отримання та оновлення параметрів теплиць |
| Моніторинг мікроклімату | Аналіз сенсорних показників | Обробка температури, вологості, освітленості |
| Керування поливом | Реалізація логіки поливу | Запуск, контроль та завершення поливу |
| Сповіщення | Інформування користувача | Формування повідомлень і попереджень |
| PostgreSQL | Збереження інформації | Зберігання даних та журналів системи |
| Контролер мікроклімату | Взаємодія з обладнанням | Передавання сенсорних даних та команд |

Поданий розподіл пакетів демонструє, що програмне забезпечення реалізоване за принципом функціональної декомпозиції, коли кожен модуль відповідає за окрему частину логіки системи. Такий підхід дозволяє зменшити складність програмної структури, підвищити незалежність окремих компонентів і забезпечити більш ефективний супровід програмного забезпечення. Крім цього, використання клієнт-серверної архітектури створює можливість розмежування процесів відображення інформації, обробки даних та взаємодії з фізичним обладнанням системи моніторингу.

На діаграмі програмне забезпечення поділено на дві основні підсистеми: клієнтський вебзастосунок та серверний застосунок. Такий поділ відповідає архітектурі сучасної веборієнтованої системи, у якій користувацький інтерфейс працює у браузері, а основна обробка даних і реалізація бізнес-логіки виконуються на серверній стороні.

Клієнтська частина забезпечує взаємодію користувача із системою, відображення параметрів мікроклімату, керування поливом та отримання сповіщень. Серверна частина реалізує обробку сенсорних даних, взаємодію з базою даних PostgreSQL, формування логіки керування теплицею та передавання команд зовнішньому контролеру мікроклімату.

Взаємодія між пакетами реалізується через API-шар, який виступає єдиною точкою доступу до серверної логіки. Такий підхід забезпечує централізовану обробку запитів, зменшує залежність між компонентами та спрощує інтеграцію нових функціональних модулів.

Реалізована модульна структура програмного забезпечення створює основу для подальшого розвитку системи, зокрема додавання нових режимів поливу, розширення засобів аналітики та інтеграції додаткових сенсорних пристроїв без необхідності суттєвого перепроєктування архітектури застосунку.

## 3.4 Компонентна структура системи

Компонентна структура програмного забезпечення відображає склад системи з погляду її основних виконуваних компонентів, їх функціонального призначення та способів взаємодії між ними. На відміну від діаграми пакетів, яка демонструє логічний поділ програмного забезпечення на функціональні модулі, діаграма компонентів дозволяє деталізувати програмні, апаратно-програмні та інфраструктурні складові системи, що безпосередньо беруть участь у процесі її функціонування.

Для веборієнтованої системи моніторингу параметрів мікроклімату теплиць компонентний підхід має важливе значення, оскільки програмне забезпечення реалізує взаємодію між клієнтським вебінтерфейсом, серверною частиною, модулем аналітичної обробки даних, базою даних PostgreSQL та зовнішнім контролером ESP32, який забезпечує отримання сенсорних показників і виконання керуючих команд.

Використання компонентної моделі дозволяє формалізувати структуру програмного забезпечення, визначити межі відповідальності окремих компонентів та описати механізми їх інформаційної взаємодії. Такий підхід спрощує супровід системи, підвищує масштабованість архітектури та створює основу для подальшого розширення функціональних можливостей застосунку.

На рис. 11 наведено діаграму компонентів програмного забезпечення системи моніторингу параметрів мікроклімату теплиць.

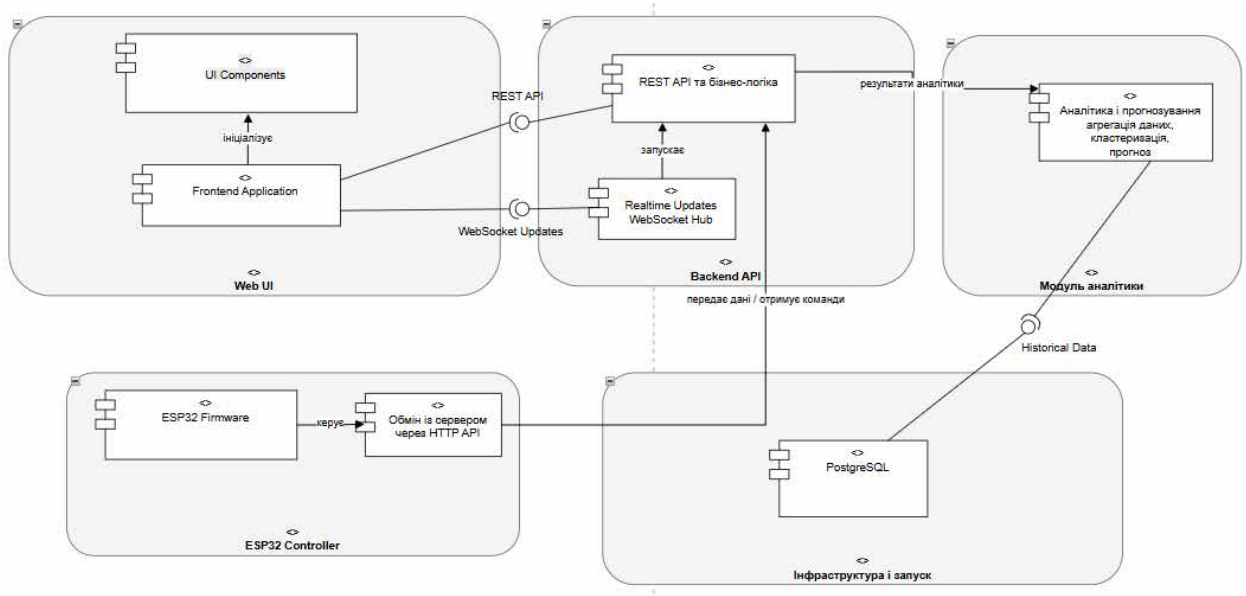


Рис. 11 Діаграма компонентів

Компонентна структура системи побудована відповідно до принципів клієнт-серверної архітектури та модульної організації програмного забезпечення. У межах розробленої системи окремі компоненти відповідають за реалізацію інтерфейсу користувача, серверної бізнес-логіки, аналітичної обробки даних, взаємодії з апаратною частиною та збереження інформації. Такий підхід забезпечує розмежування функціональних обов’язків між компонентами, зменшує залежність між окремими частинами системи та спрощує подальший розвиток програмного забезпечення.

Для узагальнення функціонального призначення основних компонентів системи та їх ролі у структурі програмного забезпечення доцільно подати характеристику компонентів у вигляді табл. 3.2.

Таблиця 3.2

Основні компоненти програмного забезпечення системи

| Компонент | Призначення | Основні функції |
|---|---|---|
| Web UI | Клієнтська частина системи | Відображення інтерфейсу та взаємодія з користувачем |
| Backend API | Серверна частина системи | Обробка запитів та реалізація бізнес-логіки |
| REST API | Інтерфейс взаємодії | Передавання даних між frontend та backend |
| WebSocket Hub | Оновлення даних у реальному часі | Передавання актуальних сенсорних показників |
| Модуль аналітики | Аналітична обробка даних | Аналіз, прогнозування та кластеризація |
| PostgreSQL | Зберігання інформації | Збереження даних системи |
| ESP32 Controller | Взаємодія з обладнанням | Отримання сенсорних даних та керування поливом |
| ESP32 Firmware | Програмне забезпечення контролера | Обробка даних датчиків і виконання команд |

Подана компонентна структура демонструє, що програмне забезпечення реалізоване за принципом функціональної декомпозиції, коли кожен компонент виконує окрему частину логіки системи та взаємодіє з іншими компонентами через визначені інтерфейси обміну даними. Такий підхід підвищує масштабованість системи, спрощує тестування окремих модулів і забезпечує можливість незалежного розвитку програмних та апаратних складових застосунку.

Клієнтська частина системи реалізована у вигляді компонента Web UI, який забезпечує взаємодію користувача з вебзастосунком, відображення параметрів мікроклімату, графіків, сповіщень та елементів керування поливом. Взаємодія між клієнтською та серверною частинами здійснюється через REST API, що забезпечує передавання запитів і отримання відповідей у стандартизованому форматі.

Центральним компонентом серверної частини є Backend API, у межах якого реалізовано обробку запитів, керування теплицями, обробку сенсорних показників, логіку поливу, формування сповіщень та взаємодію з базою даних

PostgreSQL. Для підтримки оперативного оновлення інформації використовується компонент WebSocket Hub, який забезпечує передавання актуальних даних у режимі реального часу без необхідності повторного завантаження сторінки користувачем.

Окремою складовою системи є модуль аналітики, який виконує аналіз історичних даних, формування прогнозів та аналітичну обробку параметрів мікроклімату. Для цього використовуються накопичені дані PostgreSQL, що дозволяє оцінювати тенденції зміни параметрів середовища та підтримувати прийняття рішень щодо керування теплицею.

Апаратно-програмна взаємодія реалізована за допомогою контролера ESP32, який забезпечує отримання показників із датчиків, передавання сенсорних даних на сервер та виконання команд керування поливом. Завдяки цьому система поєднує програмну платформу моніторингу з фізичним середовищем теплиці та забезпечує автоматизоване керування параметрами мікроклімату. На рис. 12 наведено схему підключення датчиків та виконавчого пристрою до контролера ESP32.

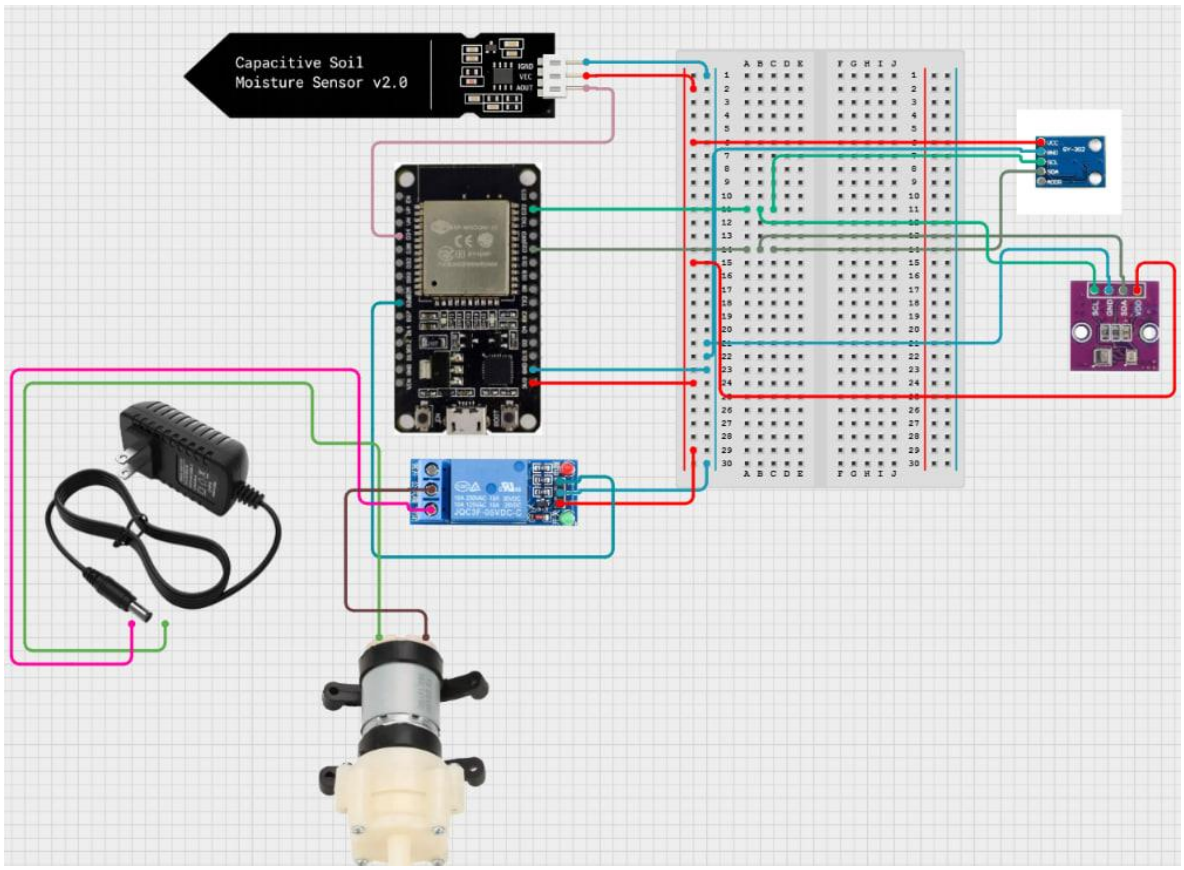


Рис. 12 Схема підключення датчиків і виконавчого пристрою до ESP32

Наведена схема демонструє практичну реалізацію апаратної частини системи моніторингу параметрів мікроклімату теплиць. У межах апаратного вузла контролер ESP32 забезпечує взаємодію між сенсорними модулями, виконавчим пристроєм поливу та серверною частиною вебзастосунку. Такий підхід дозволяє поєднати засоби збору даних, програмну обробку інформації та механізми автоматизованого керування мікрокліматом у межах єдиної інтегрованої системи.

Для узагальнення функціонального призначення основних елементів апаратної частини системи доцільно подати їх характеристику у вигляді табл. 3.3.

Таблиця 3.3

Основні елементи апаратної частини системи

| Компонент | Призначення | Основні функції |
|---|---|---|
| ESP32 Controller | Центральний керуючий елемент | Обробка сенсорних даних та керування поливом |
| Датчик вологості ґрунту | Контроль рівня вологості | Передавання аналогового сигналу вологості |
| Датчики мікроклімату | Моніторинг параметрів середовища | Вимірювання температури, вологості, освітленості, тиску |
| Релейний модуль | Керування силовим навантаженням | Комутація живлення насоса |
| Насос поливу | Зволоження ґрунту | Подача води до системи поливу |
| Джерело живлення | Енергозабезпечення системи | Живлення контролера та виконавчих пристроїв |
| HTTP API | Обмін даними із сервером | Передавання сенсорних показників та отримання команд |

Представлена структура апаратної частини демонструє реалізацію принципу розподіленої взаємодії між сенсорними пристроями, контролером та серверною частиною вебзастосунку. Контролер ESP32 виконує роль проміжного обчислювального вузла, який забезпечує первинну обробку сенсорних показників, формування пакетів даних і виконання керуючих команд, отриманих від серверної частини системи. Завдяки цьому забезпечується інтеграція

фізичного середовища теплиці з програмною платформою моніторингу та керування.

У межах апаратної частини системи ємнісний датчик вологості ґрунту використовується для визначення рівня зволоження ґрунту та передає аналоговий сигнал на контролер ESP32. Після обробки отримані значення перетворюються у цифровий формат і передаються до серверної частини системи для подальшого збереження та відображення у вебінтерфейсі.

Додаткові датчики температури, вологості повітря, освітленості та атмосферного тиску забезпечують комплексний моніторинг параметрів мікроклімату теплиці. Отримані показники використовуються для оцінювання стану середовища, формування сповіщень та підтримки механізмів автоматизованого керування поливом.

Керування насосом реалізується через релейний модуль, який забезпечує комутацію живлення виконавчого пристрою без безпосереднього навантаження на контролер ESP32. Такий підхід підвищує надійність роботи апаратної частини та дозволяє безпечно реалізувати механізми запуску і зупинки поливу.

Взаємодія між ESP32 Controller та Backend API здійснюється через HTTP API. Контролер передає на сервер сенсорні показники, а серверна частина може надсилати команди керування поливом у ручному або автоматизованому режимі. Це забезпечує двосторонній обмін даними між фізичним середовищем теплиці та програмною системою моніторингу.

PostgreSQL використовується як централізоване сховище інформації, у якому зберігаються дані сенсорних вимірювань, параметри теплиць, журнали поливу, сповіщення та інші службові записи. Завдяки цьому забезпечується накопичення історичних даних і підтримується робота аналітичних механізмів системи.

Таким чином, компонентна структура програмного забезпечення та апаратної частини реалізує інтегровану систему моніторингу й керування мікрокліматом теплиць, у межах якої програмні та апаратні компоненти взаємодіють через визначені інтерфейси обміну даними. Реалізований підхід

забезпечує масштабованість системи, спрощує її супровід та створює можливість подальшого розширення функціональних можливостей застосунку.

## 3.5 Вибір інструментарію для створення прикладного програмного забезпечення

У процесі розробки прикладного програмного забезпечення для моніторингу параметрів мікроклімату теплиць було використано інструментарій, який забезпечує створення веборієнтованої системи з клієнтською частиною, серверною бізнес-логікою, базою даних, аналітичним модулем та взаємодією з контролером мікроклімату. Вибір технологій здійснювався з урахуванням функціональних потреб системи: відображення поточних показників, побудова графіків, обробка сенсорних даних, керування поливом, формування сповіщень, збереження історії вимірювань та підтримка аналітичних розрахунків.

Як основне середовище розробки було використано Visual Studio Code. Це середовище зручне для роботи з вебпроєктами, оскільки підтримує редагування коду різними мовами програмування, роботу з розширеннями, інтегрований термінал, засоби налагодження та інтеграцію із системами контролю версій [14]. Для даного проєкту це є важливим, оскільки програмне забезпечення складається з декількох частин: frontend-застосунку, backend-застосунку, конфігурації Docker, SQL-міграцій та коду для ESP32 Controller.

Для реалізації серверної частини було обрано мову програмування Python. Її використання є доцільним через простий і зрозумілий синтаксис, широку екосистему бібліотек, підтримку веброзробки, аналітики даних та роботи з базами даних [15]. У межах цієї роботи Python застосовується не лише для створення API, а й для реалізації логіки обробки показників мікроклімату, формування сповіщень, підготовки аналітичних даних та взаємодії з базою даних.

Основним фреймворком серверної частини було обрано FastAPI. Він призначений для створення сучасних веб API на Python та підтримує роботу з типізацією, автоматичну генерацію документації API, асинхронну обробку запитів і зручну організацію маршрутів [16]. Для системи моніторингу теплиці FastAPI є вдалим вибором, оскільки серверна частина повинна швидко приймати запити від клієнтського вебзастосунку, обробляти дані від контролера, передавати поточні показники та реагувати на команди керування поливом.

Для запуску backend-застосунку використовується Uvicorn, який виконує роль ASGI-сервера. Його застосування дає змогу обслуговувати FastAPI-застосунок, зокрема HTTP-запити та WebSocket-з'єднання. Це важливо для підтримки оновлень у реальному часі, оскільки користувач повинен бачити актуальний стан теплиці без постійного ручного оновлення сторінки.

Для опису, перевірки та передачі даних у серверній частині використовується Pydantic. Завдяки йому можна задавати структуру вхідних і вихідних даних, перевіряти коректність значень та зменшувати кількість помилок під час обміну між клієнтом і сервером. Це особливо актуально для системи, яка працює з числовими показниками середовища: температурою, вологістю повітря, вологістю ґрунту, освітленістю та тиском.

Для взаємодії з базою даних застосовується SQLAlchemy разом з асинхронним драйвером asyncpg. SQLAlchemy дозволяє описувати структуру даних у вигляді моделей та виконувати запити до бази даних через об'єктно-реляційний підхід. У системі це використовується для роботи з користувачами, теплицями, пристроями, сенсорними вимірюваннями, налаштуваннями поливу, сповіщеннями та журналами подій. Для керування змінами структури бази даних використовується Alembic, який дозволяє створювати та застосовувати міграції без ручного переписування схеми.

Як систему керування базами даних було використано PostgreSQL. Вона є об'єктно-реляційною СУБД і підходить для зберігання структурованих даних, підтримки зв'язків між таблицями, часових міток, числових параметрів та історичних записів [11]. У межах розробленої системи PostgreSQL

використовується для збереження даних користувачів, теплиць, контролерів, вимірювань мікроклімату, подій поливу, сповіщень і налаштувань. Такий вибір є обґрунтованим, оскільки система повинна не лише відображати поточні значення, а й накопичувати історичні дані для подальшого аналізу.

Клієнтська частина системи реалізована з використанням React. Ця бібліотека дозволяє створювати інтерфейс у вигляді окремих компонентів, що спрощує розробку складних вебсторінок і повторне використання елементів інтерфейсу [17]. У даній системі компонентний підхід використовується для побудови панелі моніторингу, графіків, сторінок керування поливом, сповіщень, налаштувань та адміністративних сторінок. Це дозволяє підтримувати зрозумілу структуру клієнтського застосунку та розділяти інтерфейс на логічні частини.

Для розробки frontend-застосунку також використано TypeScript. Його перевагою є наявність статичної типізації, яка дозволяє точніше описувати структуру даних, що надходять від сервера, і зменшувати ризик помилок під час роботи з об’єктами вимірювань, користувачів, сповіщень та налаштувань. Для системи, яка активно обмінюється даними між frontend і backend, це є важливим, оскільки типізація допомагає підтримувати узгодженість між клієнтською і серверною частинами.

Для збирання та запуску клієнтської частини було обрано Vite. Він забезпечує швидкий запуск середовища розробки, підтримку сучасних модулів JavaScript/TypeScript і формування оптимізованої збірки для подальшого розгортання [18]. У проєкті Vite використовується для локальної розробки frontend-застосунку, перевірки змін у реальному часі та формування готової версії клієнтської частини.

Для обміну даними між клієнтською та серверною частинами використовується Axios. Ця бібліотека забезпечує виконання HTTP-запитів до REST API, обробку відповідей сервера та передачу даних між інтерфейсом і backend-застосунком. Через такі запити користувач отримує інформацію про теплиці, переглядає сенсорні показники, запускає або зупиняє полив, а також отримує список сповіщень.

Для візуалізації даних застосовується Recharts. За його допомогою можна будувати графіки зміни температури, вологості повітря, вологості ґрунту, освітленості та інших параметрів у часі. Візуальне подання є важливою частиною системи, оскільки користувачеві потрібно не лише бачити окремі числові значення, а й оцінювати динаміку зміни мікроклімату в теплиці.

Для керування станом клієнтської частини використовується Zustand. Він дозволяє зберігати та оновлювати дані, які використовуються різними компонентами інтерфейсу. У межах системи це може бути інформація про користувача, активну теплицю, поточні показники, стан поливу або отримані сповіщення. Такий підхід спрощує обмін даними між сторінками та компонентами frontend-застосунку.

Для оформлення інтерфейсу було використано Tailwind CSS, а також допоміжні бібліотеки Headless UI та Heroicons. Tailwind CSS дає змогу швидко створювати адаптивний інтерфейс за допомогою готових утилітарних класів, а Headless UI та Heroicons використовуються для побудови окремих інтерактивних елементів та іконок. Завдяки цьому інтерфейс системи може бути одночасно простим, зрозумілим і зручним для користувача.

Аналітична частина системи реалізується засобами Python-бібліотек pandas, NumPy та scikit-learn. Бібліотека pandas використовується для роботи з табличними даними та підготовки наборів показників для подальшого аналізу [19]. NumPy забезпечує ефективну роботу з числовими масивами, а scikit-learn надає інструменти для задач прогнозування, кластеризації та іншої аналітичної обробки даних [20]. У контексті цієї роботи такі засоби застосовуються для аналізу історичних показників мікроклімату, виявлення закономірностей та формування прогнозних значень.

Для експорту даних використовується openpyxl, що дає змогу формувати файли у форматі Excel. Це корисно для збереження результатів моніторингу, подальшого аналізу показників або підготовки звітних матеріалів. Експорт даних є додатковою функцією, яка розширює можливості системи та дозволяє працювати з накопиченою інформацією поза межами вебінтерфейсу.

Для апаратно-програмної частини використовується ESP32 Controller. ESP32 є доцільним вибором для системи моніторингу теплиці, оскільки підтримує роботу з датчиками, має можливості бездротового з’єднання та може взаємодіяти з виконавчими пристроями. У межах проєкту контролер відповідає за зчитування сенсорних даних і керування насосом або іншим елементом системи поливу. Для розробки програмного забезпечення під ESP32 можуть використовуватися офіційні засоби Espressif, зокрема ESP-IDF, який є офіційним фреймворком для мікроконтролерів ESP32 [21].

Для контейнеризації та запуску системи використовується Docker Compose. Він дозволяє описати декілька сервісів в одному конфігураційному файлі та запускати їх як єдине середовище [22]. У межах цього проєкту Docker Compose використовується для організації роботи backend-застосунку, frontend-застосунку та бази даних PostgreSQL. Це спрощує розгортання системи, оскільки всі основні компоненти можна запускати узгоджено, без ручного налаштування кожного сервісу окремо.

Обраний інструментарій відповідає архітектурі та функціональним вимогам розроблюваної системи. React, TypeScript і Vite забезпечують створення зручного клієнтського вебінтерфейсу. Python і FastAPI використовуються для реалізації серверної логіки та API. PostgreSQL забезпечує надійне збереження даних, а SQLAlchemy та Alembic спрощують роботу зі структурою бази даних. ESP32 Controller відповідає за взаємодію з фізичним середовищем теплиці, а бібліотеки pandas, NumPy і scikit-learn дозволяють реалізувати аналітичну складову. Docker Compose забезпечує зручне розгортання системи як цілісного програмного комплексу.

## 3.6 Алгоритмізація та програмування програмних модулів

Алгоритмізація програмних модулів є одним із ключових етапів розробки програмного забезпечення, оскільки дозволяє формалізувати послідовність виконання операцій, пов’язаних із отриманням, обробкою, аналізом та

передаванням даних у межах інформаційної системи. Для веборієнтованої системи моніторингу параметрів мікроклімату теплиць алгоритмічна організація має особливе значення, оскільки забезпечує узгоджену взаємодію між сенсорними пристроями, серверною частиною, базою даних і клієнтським вебінтерфейсом.

У межах розробленої системи одним із центральних процесів є алгоритм обробки сенсорних даних мікроклімату. Саме він забезпечує приймання показників від контролера ESP32, перевірку коректності отриманих значень, збереження вимірювань у базі даних PostgreSQL, аналіз параметрів на наявність критичних відхилень та подальше передавання актуальної інформації до клієнтської частини системи. Крім цього, результати обробки можуть використовуватися для формування сповіщень і підтримки механізмів автоматизованого керування поливом.

Особливістю алгоритму є поєднання апаратної та програмної складових системи, оскільки процес обробки даних починається на рівні фізичних датчиків мікроклімату, а завершується відображенням інформації у вебінтерфейсі користувача та формуванням керуючих дій. Такий підхід забезпечує підтримку режиму моніторингу в реальному часі та створює основу для подальшого розширення аналітичних функцій системи.

У процесі роботи алгоритму забезпечується послідовна обробка потоку сенсорних даних, що надходять від контролера ESP32 через HTTP API. Після отримання значень система виконує їх первинну перевірку, аналіз допустимих меж та підготовку до подальшого збереження у базі даних PostgreSQL. У випадку виявлення критичних або нестандартних значень можуть формуватися повідомлення для користувача, а також ініціюватися механізми автоматизованого реагування. Така організація алгоритму дозволяє забезпечити безперервний цикл моніторингу параметрів мікроклімату та підтримувати актуальність інформації у вебінтерфейсі системи в режимі реального часу.

На рис. 13 наведено блок-схему алгоритму обробки сенсорних даних мікроклімату.

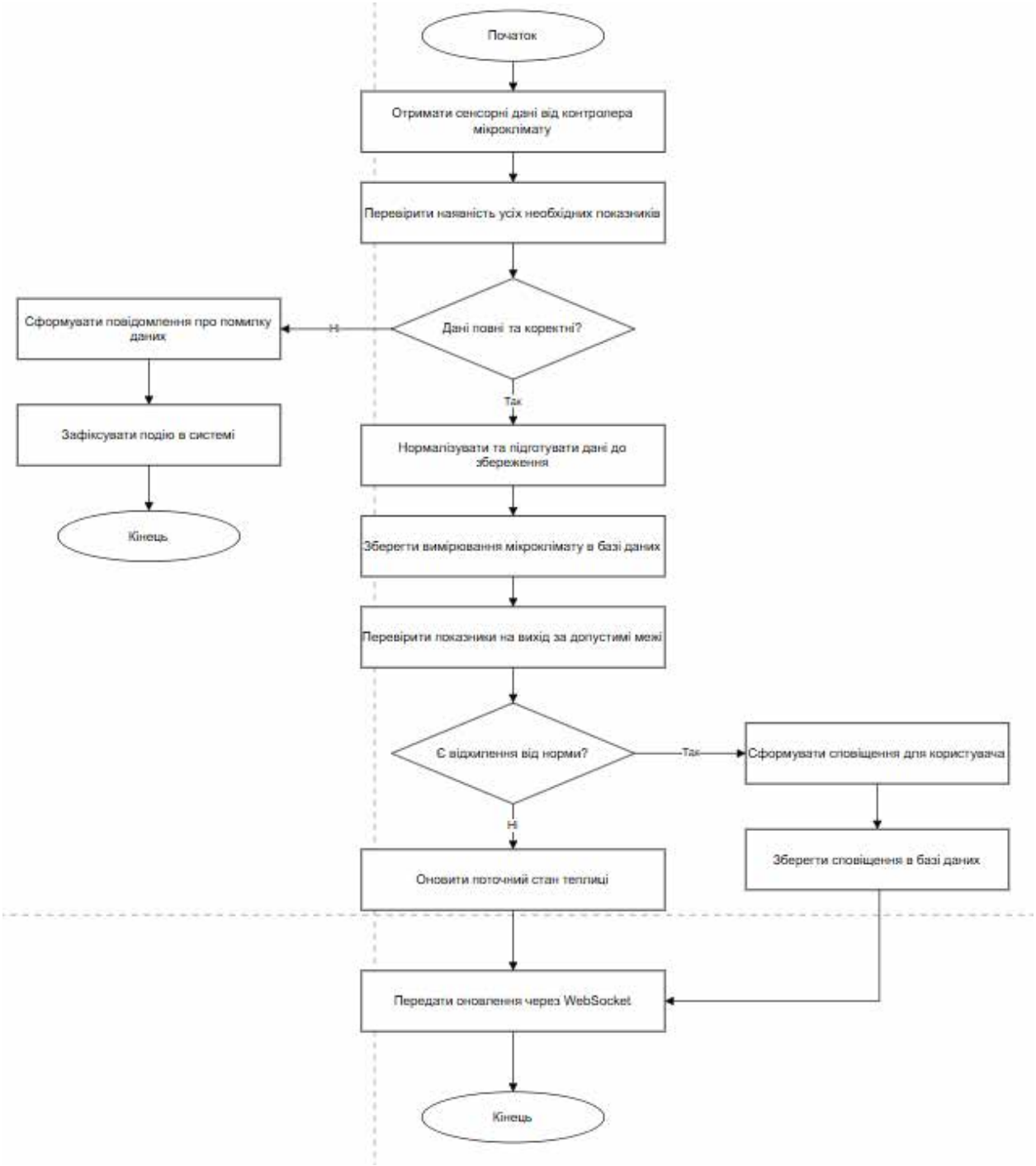


Рис. 13 Блок-схема алгоритму обробки сенсоних даних мікроклімаут

Алгоритм починається з отримання сенсорних даних від контролера мікроклімату. До таких даних належать температура повітря, вологість повітря, рівень освітленості, вологість ґрунту, тиск, стан поливу та час вимірювання. Після надходження даних система перевіряє їхню повноту і коректність. Якщо дані неповні або містять некоректні значення, формується повідомлення про помилку, а подія фіксується в системі.

Якщо дані є коректними, вони приводяться до потрібного формату та зберігаються як нове вимірювання мікроклімату. Після цього система перевіряє, чи не виходять показники за встановлені допустимі межі. У разі виявлення відхилення формується сповіщення для користувача, яке зберігається в базі даних і може бути передане через WebSocket. Якщо відхилень немає, система оновлює поточний стан теплиці та передає актуальні дані клієнтському інтерфейсу.

Далі розглянемо фрагменти коду, які реалізують основні етапи цього алгоритму.

Фрагмент коду, наведений на рис. 14, представляє модель вхідних даних від контролера мікроклімату. Вона описує набір показників, які контролер передає на сервер.

```python
class DeviceReadingIn(BaseModel):
    temperature: float
    humidity: float
    light: float
    soil_moisture: float = Field(ge=0, le=100)
    pressure: Optional[float] = None
    irrigation_on: bool = False
    timestamp: Optional[datetime] = None
```

Рис. 14 Модель вхідних даних від контролера мікроклімату

Він відповідає етапу перевірки структури вхідних даних. Значення вологості ґрунту обмежується діапазоном від 0 до 100, що дозволяє одразу відсікати некоректні значення ще на рівні схеми даних. Завдяки цьому серверна частина отримує структурований набір параметрів, з яким надалі може працювати бізнес-логіка.

На рис. 15 наведено фрагмент маршруту, який приймає показники від контролера мікроклімату. Саме цей метод є точкою входу для сенсорних даних у backend-застосунок.

```python
@router.post("/readings")
async def push_reading(
    payload: DeviceReadingIn,
    db: AsyncSession = Depends(get_db),
    current_device: ManagedDevice = Depends(get_current_managed_device),
):
    try:
        reading = await DeviceService.ingest_reading(db, payload, current_device)
    except ValueError as exc:
        raise HTTPException(status_code=409, detail=str(exc)) from exc
    await ws_hub.broadcast_sensor(
        {
            "id": reading.id,
            "temperature": reading.temperature,
            "humidity": reading.humidity,
            "light": reading.light,
            "soil_moisture": reading.soil_moisture,
            "pressure": reading.pressure,
            "irrigation_on": reading.irrigation_on,
            "is_irrigating": reading.irrigation_on,
            "timestamp": reading.timestamp.isoformat(),
            "source": reading.source,
        }
    )
    return {"status": "ok", "reading_id": reading.id}
```

Рис. 15 Метод приймання сенсорних даних від контролера

Метод push_reading отримує дані від контролера, передає їх до сервісного шару для обробки, а після успішного збереження надсилає оновлення через WebSocket. Таким чином, фрагмент реалізує одразу декілька етапів блок-схеми: отримання даних, передавання їх на обробку та оновлення клієнтської частини в реальному часі.

Реалізація такого підходу дозволяє забезпечити централізовану обробку сенсорних показників та підтримати узгоджену взаємодію між серверною логікою, базою даних і клієнтським вебінтерфейсом. Після отримання нового вимірювання система не лише виконує його збереження, а й ініціює подальші етапи аналізу даних, перевірку граничних значень та формування механізмів оперативного оновлення інформації для користувача. Це забезпечує підтримку режиму моніторингу в реальному часі та створює основу для подальшої автоматизації процесів керування мікрокліматом теплиці.

На рис. 16 показано метод ingest_reading, який створює запис сенсорного вимірювання, зберігає його в базі даних і запускає подальші перевірки.

```python
@staticmethod
async def ingest_reading(
    db: AsyncSession,
    payload: DeviceReadingIn,
    managed_device: ManagedDevice,
) -> SensorReading:
    if not managed_device or not managed_device.greenhouse_id:
        raise ValueError("No managed device linked to a greenhouse")
    reading = SensorReading(
        temperature=payload.temperature,
        humidity=payload.humidity,
        light=payload.light,
        soil_moisture=payload.soil_moisture,
        pressure=payload.pressure,
        irrigation_on=payload.irrigation_on,
        timestamp=payload.timestamp or datetime.now(timezone.utc),
        source="device",
        greenhouse_id=managed_device.greenhouse_id if managed_device else None,
        device_id=managed_device.id if managed_device else None,
        is_anomaly=False,
    )
    db.add(reading)
    if managed_device:
        managed_device.last_seen_at = reading.timestamp
        managed_device.status = "online"
    await db.commit()
    await db.refresh(reading)

    await IrrigationService.sync_from_device(db, reading)
    await IrrigationService.check_and_trigger(db, reading)
    await AlertService.check(db, reading)

    return reading
```

Рис. 16 Метод збереження сенсорних вимірювань

Метод перевіряє, чи прив’язаний контролер до теплиці, створює об’єкт SensorReading, зберігає його в базі даних, оновлює стан пристрою та запускає додаткові перевірки. Виклики sync_from_device, check_and_trigger та AlertService.check відповідають наступним крокам алгоритму: синхронізації стану поливу, перевірці потреби в автоматичному поливі та формуванню сповіщень у разі відхилень.

На рис. 17 зображено модель таблиці SensorReading, яка використовується для збереження вимірювань мікроклімату в базі даних.

```python
class SensorReading(Base):
    __tablename__ = "sensor_readings"
    __table_args__ = (
        CheckConstraint("humidity >= 0 AND humidity <= 100", name="ck_sensor_readings_humidity"),
        CheckConstraint("soil_moisture >= 0 AND soil_moisture <= 100", name="ck_sensor_readings_soil"),
        CheckConstraint("pressure IS NULL OR pressure > 0", name="ck_sensor_readings_pressure"),
        CheckConstraint(
            "source IN ('device', 'simulator', 'import', 'manual')",
            name="ck_sensor_readings_source",
        ),
        Index("ix_sensor_readings_greenhouse_timestamp", "greenhouse_id", "timestamp"),
        Index("ix_sensor_readings_device_timestamp", "device_id", "timestamp"),
    )

    id = Column(Integer, primary_key=True, index=True)
    timestamp = Column(DateTime(timezone=True), server_default=func.now(), index=True)
    temperature = Column(Float, nullable=False)
    humidity = Column(Float, nullable=False)
    light = Column(Float, nullable=False)
    soil_moisture = Column(Float, nullable=False)
    pressure = Column(Float, nullable=True)
    irrigation_on = Column(Boolean, default=False)
    source = Column(String(20), default="device")
    greenhouse_id = Column(Integer, ForeignKey("greenhouses.id"), nullable=False, index=True)
    device_id = Column(Integer, ForeignKey("managed_devices.id"), nullable=False, index=True)
    is_anomaly = Column(Boolean, default=False)
```

Рис. 17 Метод збереження вимірювання мікроклімату

У моделі визначено основні поля вимірювання: температуру, вологість, освітленість, вологість ґрунту, тиск, стан поливу, джерело даних, теплицю та пристрій. Також задано обмеження для вологості та тиску, що додатково захищає базу даних від некоректних значень.

Реалізація перевірки граничних значень є важливою складовою алгоритму обробки сенсорних даних, оскільки дозволяє своєчасно виявляти критичні зміни параметрів мікроклімату. У випадку перевищення встановлених меж система може автоматично ініціювати формування повідомлень для користувача, що забезпечує оперативне реагування на потенційно небезпечні ситуації. Такий підхід підвищує надійність функціонування системи моніторингу та підтримує механізми автоматизованого контролю стану теплиці.

Фрагмент програмного коду на рис. 18 реалізує механізм перевірки показників на вихід за допустимі межі та автоматичне формування сповіщень для користувача.

```python
@staticmethod
async def check(db: AsyncSession, reading: SensorReading) -> None:
    recipient_ids = await AlertService._resolve_recipient_ids(db, reading.greenhouse_id)
    if not recipient_ids:
        return
    result = await db.execute(select(AlertSetting))
    settings = result.scalars().all()
    for s in settings:
        if s.user_id not in recipient_ids:
            continue
        val = getattr(reading, s.param, None)
        if val is None:
            continue
        alert_type = None
        msg = None
        if s.max_val is not None and val > s.max_val:
            alert_type = f"{s.param}_high"
            msg = f"{s.param} перевищує максимум: {val:.1f} > {s.max_val:.1f}"
        elif s.min_val is not None and val < s.min_val:
            alert_type = f"{s.param}_low"
            msg = f"{s.param} нижче мінімуму: {val:.1f} < {s.min_val:.1f}"
        if alert_type and msg:
            alert = Alert(user_id=s.user_id, type=alert_type, message=msg, value=val)
            db.add(alert)
```

Рис. 18 Модель перевірки показників та формування сповіщень

Метод отримує налаштування сповіщень, порівнює значення вимірювання з мінімальними та максимальними межами і створює об’єкт Alert, якщо показник виходить за допустимий діапазон. Це відповідає блоку “Є відхилення від норми?” на блок-схемі.

Передавання сповіщень та оновлених сенсорних показників у режимі реального часу є важливою складовою функціонування системи моніторингу мікроклімату. Після виявлення критичних відхилень сформовані повідомлення повинні оперативно надходити до клієнтської частини, щоб користувач міг своєчасно реагувати на зміни стану теплиці. Для реалізації такого механізму в системі використовується технологія WebSocket, яка забезпечує постійний двосторонній зв’язок між серверною та клієнтською частинами без необхідності повторного надсилання HTTP-запитів.

Фрагмент WebSocket Hub, представлений на рис. 19, забезпечує передавання оновлених даних клієнтській частині системи в режимі реального часу.

```python
class WebSocketHub:
    def __init__(self):
        self._sensor_connections: List[WebSocket] = []
        self._alert_connections: List[WebSocket] = []

    async def connect_sensor(self, ws: WebSocket):
        await ws.accept()
        self._sensor_connections.append(ws)

    async def connect_alert(self, ws: WebSocket):
        await ws.accept()
        self._alert_connections.append(ws)

    def disconnect(self, ws: WebSocket):
        self._sensor_connections = [c for c in self._sensor_connections if c != ws]
        self._alert_connections = [c for c in self._alert_connections if c != ws]

    async def broadcast_sensor(self, data: dict):
        await self._broadcast(self._sensor_connections, data)

    async def broadcast_alert(self, data: dict):
        await self._broadcast(self._alert_connections, data)

    async def _broadcast(self, connections: List[WebSocket], data: dict):
        dead = []
        for ws in connections:
            try:
                await ws.send_text(json.dumps(data, default=str))
            except Exception:
                dead.append(ws)
        for ws in dead:
            self.disconnect(ws)
```

Рис. 19 Компонент передавання оновлень через WebSocket

Він реалізує завершальний етап алгоритму: після збереження нового вимірювання або створення сповіщення сервер передає оновлення всім активним WebSocket-з’єднанням. Завдяки цьому користувач отримує актуальні значення параметрів мікроклімату без перезавантаження сторінки.

Блок-схема та програмна реалізація підтверджують коректність алгоритму обробки сенсорних даних і механізму формування сповіщень. Розроблений алгоритм забезпечує приймання показників від контролера ESP32, перевірку їхньої коректності, збереження вимірювань у базі даних, аналіз відхилень від допустимих меж та передавання оновлень до клієнтського вебінтерфейсу. Завдяки цьому забезпечується узгоджена взаємодія між апаратною частиною, серверною логікою, базою даних і клієнтським застосунком системи моніторингу мікроклімату теплиць.

# 4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

## 4.1 Тестування системи

Тестування є важливим етапом розробки програмного забезпечення, оскільки воно дозволяє перевірити коректність реалізованої логіки, виявити можливі помилки та підтвердити працездатність основних функціональних модулів системи. Для розробленої системи моніторингу параметрів мікроклімату теплиці тестування проводилося з урахуванням ключових процесів: обробки показників датчиків, визначення стану вологості ґрунту, роботи алгоритму поливу, виконання аналітичних розрахунків та взаємодії користувача з серверною частиною через API.

У межах роботи було використано автоматизоване тестування серверної частини програмного забезпечення. Тести виконувалися у середовищі Docker-контейнера backend за допомогою фреймворку pytest. Такий підхід дозволяє запускати перевірки в ізольованому середовищі, близькому до реальних умов роботи застосунку, та отримувати повторюваний результат незалежно від локальних налаштувань операційної системи.

Для перевірки розробленої системи було застосовано два основні види тестування: модульне та інтеграційне. Модульне тестування використовувалося для перевірки окремих алгоритмів і функцій системи, тоді як інтеграційне тестування було спрямоване на перевірку взаємодії між API, сервісним шаром, базою даних та механізмами авторизації.

Модульне тестування охоплювало перевірку логіки зміни вологості ґрунту, роботи алгоритму автополиву та функцій аналітичного модуля. Зокрема, перевірялися сценарії висихання ґрунту за різних умов, збільшення вологості під час поливу, обмеження значень вологості в допустимих межах, а також реакція системи на досягнення або недосягнення цільового рівня вологості. Окремо

тестувалися алгоритми аналітики, що використовуються для кластеризації та прогнозування даних мікроклімату.

На рис. 20 наведено результат виконання модульних тестів.

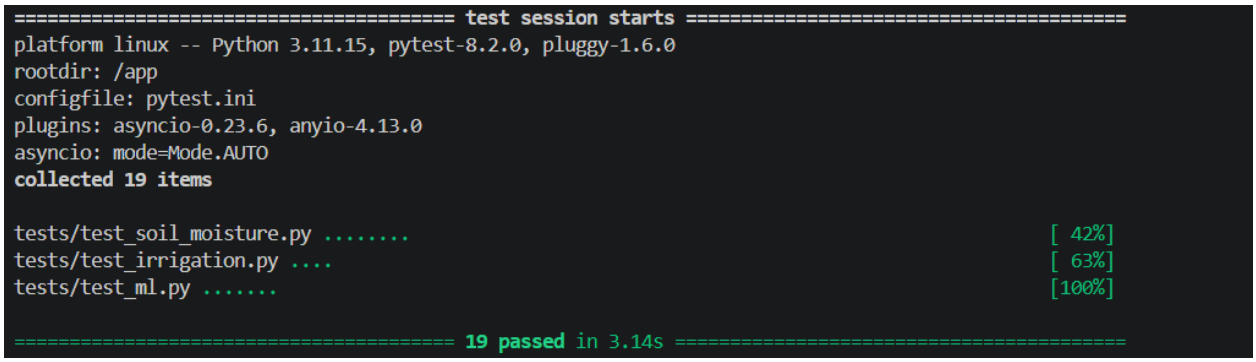


Рис. 20 Результат виконання модульних тестів

Результати модульного тестування свідчать про коректність реалізації окремих функціональних компонентів програмного забезпечення. У процесі перевірки було виконано 19 модульних тестів, кожен із яких завершився успішно. Це підтверджує правильність роботи алгоритмів, пов'язаних із моделюванням вологості ґрунту, обробкою сенсорних даних, логікою керування поливом та аналітичною обробкою інформації. Успішне проходження модульних тестів свідчить про стабільність реалізованих програмних модулів і коректність виконання окремих функцій системи.

Наступним етапом перевірки працездатності програмного забезпечення стало інтеграційне тестування, метою якого є оцінювання коректності взаємодії між основними компонентами серверної частини системи. На відміну від модульного тестування, яке перевіряє окремі функції ізольовано, інтеграційне тестування дозволяє оцінити узгодженість роботи взаємопов'язаних компонентів у межах єдиного програмного середовища.

У межах інтеграційного тестування перевірялися процеси реєстрації та авторизації користувачів, доступ до захищених маршрутів, отримання сенсорних показників, взаємодія з базою даних PostgreSQL, а також формування статистичних та аналітичних даних. Проведене тестування підтвердило коректність обміну даними між серверними модулями, стабільність API-взаємодії та правильність роботи механізмів обробки запитів.

На рис. 21 наведено результат виконання інтеграційних тестів.

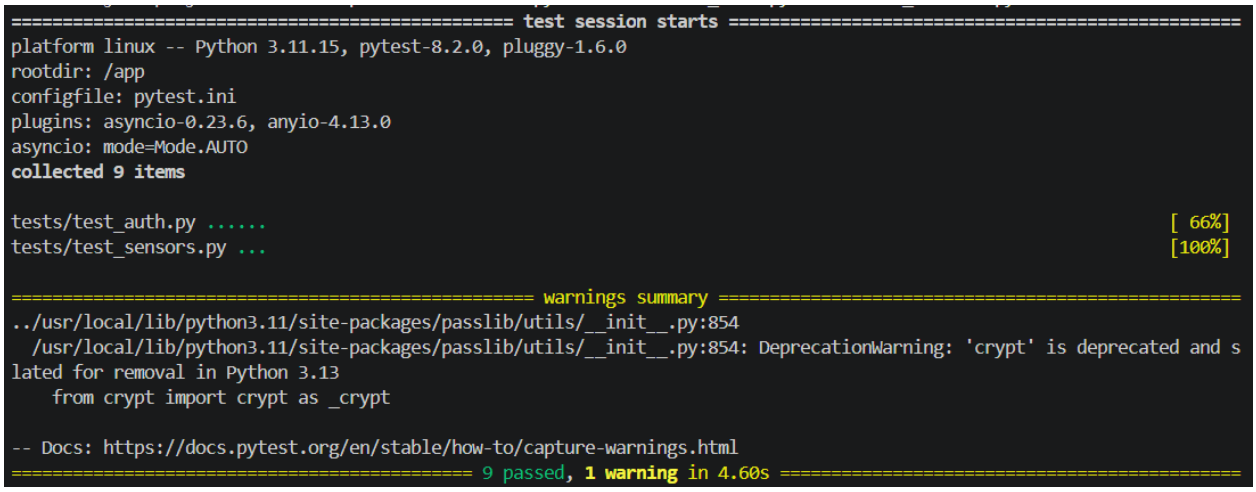


Рис. 21 Результат виконання інтеграційних тестів

Результати інтеграційного тестування підтверджують коректність взаємодії між основними компонентами програмного забезпечення. У процесі перевірки було успішно виконано 9 інтеграційних тестів, що свідчить про стабільну роботу механізмів авторизації, обробки запитів, взаємодії з базою даних та обміну даними між серверними модулями системи.

Під час виконання тестування було зафіксовано службове попередження, пов’язане з використанням сторонньої бібліотеки passlib, яка застосовується для хешування паролів користувачів. Дане повідомлення не впливає на працездатність програмного забезпечення та не призводить до порушення виконання тестових сценаріїв, оскільки всі перевірки були завершені успішно.

Проведене тестування дозволило оцінити працездатність основних функціональних можливостей системи та підтвердити коректність реалізації клієнт-серверної взаємодії. Отримані результати свідчать про стабільність роботи програмного забезпечення під час виконання типових сценаріїв використання, пов’язаних із отриманням сенсорних даних, обробкою запитів користувача, формуванням сповіщень та взаємодією з базою даних. Крім цього, результати тестування підтвердили узгоджену роботу програмних модулів у межах єдиної інформаційної системи моніторингу мікроклімату теплиць.

Для узагальнення результатів тестування у табл. 4.1 наведено основні групи тестів, виконаних під час перевірки програмного забезпечення.

Таблиця 4.1

Результати автоматизованого тестування системи

| Вид тестування | Тестові файли | Що перевірялося | Результат |
|---|---|---|---|
| Модульне | test_soil_moisture.py | Зміна вологості ґрунту, висихання, зволоження, допустимі межі значень | Успішно |
| | test_irrigation.py | Логіка запуску поливу, повний цикл поливу, вплив швидкості зволоження | Успішно |
| | test_ml.py | Кластеризація, прогнозування, обробка недостатньої кількості даних | Успішно |
| Інтеграційне | test_auth.py | Реєстрація, авторизація, перевірка доступу до захищених функцій | Успішно |
| | test_sensors.py | Отримання поточних сенсорних даних, статистика, перевірка API | Успішно |

Таким чином, результати тестування підтвердили коректність роботи основних модулів розробленої системи. Успішне проходження 19 модульних та 9 інтеграційних тестів свідчить про правильність реалізації алгоритмів обробки даних, логіки поливу, аналітичних функцій та механізмів взаємодії серверної частини з базою даних і користувацькими запитами. Загалом було виконано 28 автоматизованих тестів, що дозволяє зробити висновок про стабільність основних функціональних можливостей системи.

## 4.2 Вимоги до апаратного та програмного забезпечення

Для забезпечення стабільної роботи веборієнтованої системи моніторингу параметрів мікроклімату теплиць необхідно дотримуватися визначених вимог до апаратного та програмного забезпечення. Коректне функціонування системи залежить від узгодженої роботи серверної частини, клієнтського вебінтерфейсу, бази даних PostgreSQL, контролера ESP32 та підключених сенсорних і виконавчих пристроїв. Дотримання технічних вимог забезпечує безперервне отримання сенсорних показників, їх передавання на сервер, збереження у базі

даних, обробку програмними модулями та подальше відображення у вебінтерфейсі користувача.

Особливістю розробленої системи є поєднання програмної та апаратної складових у межах єдиної клієнт-серверної архітектури. У процесі функціонування застосунку серверна частина забезпечує обробку запитів, роботу REST API, підтримку WebSocket-з'єднань, взаємодію з базою даних та реалізацію бізнес-логіки системи. Контролер ESP32 відповідає за отримання сенсорних показників мікроклімату, передавання даних до backend-частини та виконання команд керування поливом. Такий підхід дозволяє реалізувати інтегровану систему моніторингу та автоматизованого керування параметрами тепличного середовища.

Відповідно до діаграми розгортання системи, наведеної на рис. 22, програмні та апаратні компоненти розміщуються на декількох логічних вузлах. Користувач взаємодіє із системою через клієнтський пристрій із веббраузером, серверна частина функціонує у середовищі контейнеризації, окремо розгортається сервер бази даних PostgreSQL, а контролер ESP32 забезпечує взаємодію з датчиками та виконавчими пристроями теплиці. Така структура дозволяє забезпечити розподіл функціональних обов'язків між компонентами системи, підвищити масштабованість архітектури та спростити супровід програмного забезпечення.

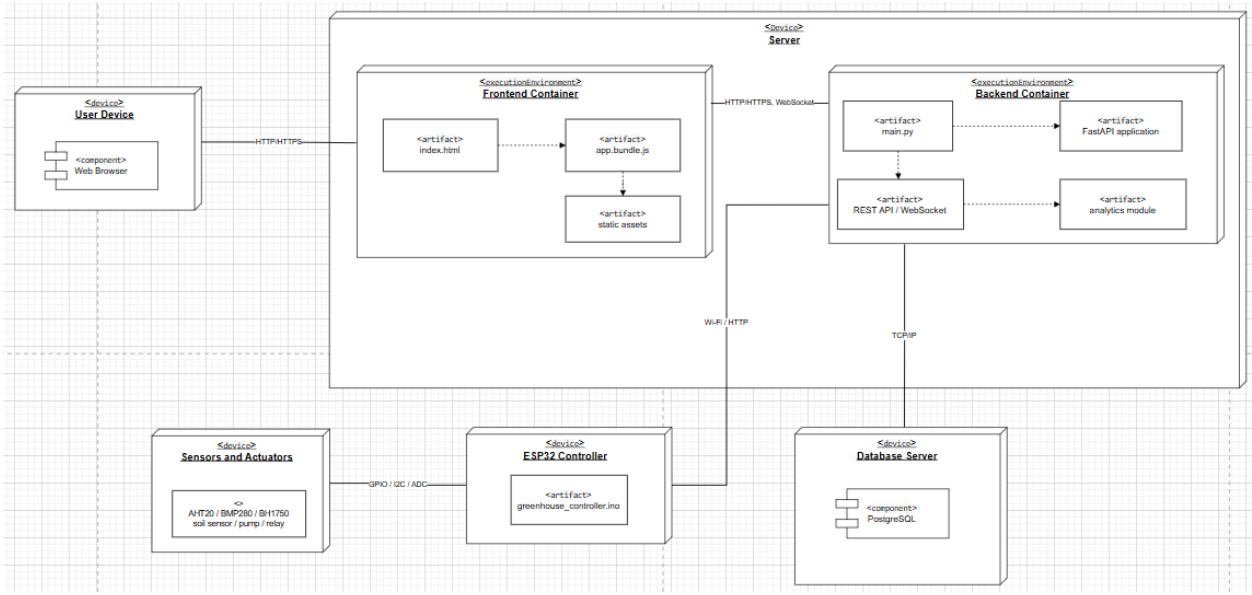


Рис. 22 Діаграма розгортання

Розроблений веборієнтований застосунок розгорнуто на серверній платформі, що забезпечує можливість віддаленого доступу до системи через мережу Інтернет без необхідності встановлення окремого клієнтського програмного забезпечення. Взаємодія користувача із системою здійснюється через веббраузер, що дозволяє переглядати параметри мікроклімату, аналізувати графіки показників, керувати поливом та отримувати сповіщення з різних типів пристроїв.

Архітектура системи реалізована за клієнт-серверним принципом. Клієнтська частина побудована із використанням React, TypeScript та Vite і забезпечує реалізацію вебінтерфейсу користувача. Серверна частина реалізована мовою Python із використанням FastAPI та Uvicorn і відповідає за REST API, WebSocket-взаємодію, обробку сенсорних даних, реалізацію бізнес-логіки, формування сповіщень та керування поливом.

Для централізованого зберігання інформації використовується система керування базами даних PostgreSQL, у якій зберігаються дані користувачів, параметри теплиць, сенсорні вимірювання, журнали подій та службова інформація. Взаємодія між програмною та апаратною частинами системи реалізована через контролер ESP32, який забезпечує отримання сенсорних показників і передавання команд виконавчим пристроям через HTTP API.

Для забезпечення стабільної роботи системи необхідно враховувати як програмні, так і апаратні характеристики середовища розгортання. Відповідність технічних параметрів визначеним вимогам дозволяє підтримувати безперервний обмін даними між компонентами системи, коректну роботу серверної частини, обробку сенсорних показників у режимі реального часу та надійне функціонування апаратного вузла теплиці.

Основні вимоги до апаратного забезпечення системи моніторингу параметрів мікроклімату теплиць наведено у табл. 4.2.

Таблиця 4.2

Вимоги до апаратного забезпечення системи

| Компонент системи | Основні вимоги |
|---|---|
| Клієнтський пристрій | Процесор рівня Intel Core i3 або аналогічний, оперативна пам’ять від 4 ГБ, мережевий адаптер або Wi-Fi, підтримка сучасного веббраузера |
| Серверна частина | Процесор рівня Intel Core i5 або аналогічний, оперативна пам’ять від 8 ГБ, підтримка Docker-контейнерів, стабільне підключення до Інтернету |
| Сервер бази даних | PostgreSQL 15, оперативна пам’ять від 4 ГБ, TCP/IP-з’єднання, підтримка резервного копіювання |
| Апаратний вузол теплиці | Контролер ESP32, датчики AHT20, BMP280, BH1750, датчик вологості ґрунту, релейний модуль, насос, джерело живлення |

Наведені апаратні вимоги забезпечують стабільне функціонування системи моніторингу мікроклімату, підтримують безперервний обмін даними між серверною, клієнтською та апаратною частинами застосунку, а також створюють умови для надійної роботи сенсорних і виконавчих пристроїв теплиці.

Основні програмні засоби та технології, використані під час реалізації системи, наведено у табл. 4.3.

Таблиця 4.3

Програмні засоби та технології системи

| Компонент | Використані технології |
|---|---|
| Frontend | React 18.3.1, TypeScript 5.4.5, Vite 5.4.21, Tailwind CSS |
| Backend | Python 3.11.15, FastAPI 0.111.0, Uvicorn 0.30.0 |
| Робота з базою даних | PostgreSQL 15, SQLAlchemy 2.0.30, Alembic 1.13.1 |
| Аналітична обробка | scikit-learn 1.4.2, pandas 2.2.2, numpy 1.26.4 |
| API-взаємодія | REST API, WebSocket |
| Контейнеризація | Docker Engine 24.0, Docker Compose v2 |
| Контролер та прошивка | ESP32, Arduino IDE 2.x |

Використання наведеного програмного стеку забезпечує підтримку сучасної вебархітектури, реалізацію механізмів обробки сенсорних даних у

режимі реального часу та інтеграцію програмної системи з апаратними компонентами теплиці. Застосування контейнеризації спрощує розгортання програмного забезпечення, його оновлення та подальше масштабування.

Таким чином, реалізована архітектура системи забезпечує стабільну роботу веборієнтованого застосунку моніторингу параметрів мікроклімату теплиць, підтримує взаємодію між клієнтською, серверною та апаратною частинами й створює основу для подальшого розвитку функціональних можливостей програмного забезпечення.

## 4.3 Склад інсталяційного пакету

Інсталяційний пакет програмного забезпечення призначений для комплексного розгортання системи моніторингу параметрів мікроклімату теплиць у серверному середовищі. Оскільки розроблена система є веборієнтованою та підтримує віддалений доступ через веббраузер, встановлення окремого клієнтського застосунку не передбачається. Взаємодія користувача із системою здійснюється через вебінтерфейс, тоді як основні програмні компоненти розгортаються на серверній платформі.

Особливістю інсталяційного пакету є поєднання програмної та апаратної складових системи. До його складу входять frontend- і backend-компоненти вебзастосунку, база даних PostgreSQL, конфігураційні файли контейнеризації, механізми ініціалізації структури бази даних, а також програмне забезпечення контролера ESP32, що забезпечує взаємодію із сенсорними та виконавчими пристроями теплиці.

Використання контейнеризації дозволяє стандартизувати процес розгортання програмного забезпечення, спростити налаштування серверного середовища та забезпечити незалежність компонентів системи. Завдяки застосуванню Docker та Docker Compose frontend-, backend- і database-компоненти можуть запускатися як окремі ізольовані сервіси із визначеними параметрами взаємодії.

Основні компоненти інсталяційного пакету наведено у табл. 4.4.

Таблиця 4.4

Основні компоненти інсталяційного пакету

| Компонент | Призначення |
|---|---|
| Frontend-застосунок | Реалізація клієнтського вебінтерфейсу системи |
| Backend-застосунок | Реалізація REST API, WebSocket та бізнес-логіки |
| Dockerfile frontend | Формування контейнера клієнтської частини |
| Dockerfile backend | Формування контейнера серверної частини |
| docker-compose.yml | Координація запуску контейнерів системи |
| Конфігураційні файли середовища | Збереження параметрів підключення та службових налаштувань |
| Міграції бази даних | Створення та оновлення структури PostgreSQL |
| Службові скрипти | Ініціалізація, тестування та адміністрування системи |
| Прошивка ESP32 | Робота з датчиками та керування поливом |
| Документація | Інструкції щодо встановлення та налаштування системи |

Поданий склад інсталяційного пакету демонструє комплексний підхід до розгортання системи, у межах якого програмні, серверні та апаратні компоненти функціонують як єдина інтегрована структура. Такий підхід забезпечує спрощення процесу встановлення, підтримує масштабованість архітектури та створює умови для подальшого супроводу програмного забезпечення.

Розгортання серверної частини системи здійснюється за допомогою Docker Engine та Docker Compose, що дозволяє автоматизувати процес запуску контейнерів і спростити налаштування програмного середовища. Перед початком роботи необхідно налаштувати параметри середовища, зокрема дані підключення до PostgreSQL, параметри авторизації, секретні ключі та адресу backend API.

Після запуску backend-контейнера автоматично виконуються міграції бази даних за допомогою Alembic, у результаті чого створюється необхідна структура таблиць системи. Frontend-контейнер забезпечує доступ користувача до вебінтерфейсу через браузер без необхідності встановлення окремого клієнтського програмного забезпечення.

Окремим етапом інсталяції є налаштування контролера ESP32. Прошивка greenhouse_controller.ino завантажується за допомогою Arduino IDE або іншого сумісного середовища розробки. У процесі налаштування необхідно визначити параметри Wi-Fi-з’єднання, адресу backend API, API-ключ пристрою та конфігурацію GPIO-пінів для підключення сенсорів і релейного модуля.

Для коректного функціонування системи необхідно забезпечити стабільне мережеве з’єднання між серверною частиною та контролером ESP32. Це дозволяє підтримувати безперервне передавання сенсорних показників, своєчасне отримання команд керування поливом та актуалізацію інформації у вебінтерфейсі користувача.

Таким чином, розроблений інсталяційний пакет забезпечує комплексне розгортання веборієнтованої системи моніторингу параметрів мікроклімату теплиць та підготовку апаратного вузла до експлуатації. Використання контейнеризації спрощує встановлення та супровід програмного забезпечення, а інтеграція контролера ESP32 забезпечує взаємодію програмної системи з реальними сенсорними та виконавчими пристроями теплиці. Після налаштування всіх компонентів система готова до експлуатації та може використовуватися для віддаленого моніторингу мікроклімату й автоматизованого керування поливом.

# ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи на тему «Програмне забезпечення системи моніторингу параметрів мікроклімату теплиці» було здійснено повний цикл розробки програмної системи, що охоплює аналіз предметної області, формування вимог, проєктування бази даних, моделювання структури програмного забезпечення, реалізацію вебзастосунку, інтеграцію з апаратним вузлом на базі ESP32 та підготовку рекомендацій щодо впровадження й експлуатації.

У ході аналізу предметної області було розглянуто особливості контролю мікроклімату теплиць, визначено роль автоматизованого моніторингу для підтримання належних умов вирощування рослин, а також обґрунтовано необхідність використання програмного забезпечення для збору, збереження, аналізу та візуалізації сенсорних даних. На основі проведеного аналізу було визначено основні функції майбутньої системи: перегляд поточних показників мікроклімату, робота з графіками, керування поливом, формування сповіщень, аналітична обробка даних та взаємодія з контролером ESP32.

У процесі проєктування було побудовано логічну та фізичну моделі бази даних. Для зберігання інформації обрано об’єктно-реляційну систему керування базами даних PostgreSQL. Структура бази даних охоплює сутності користувачів, теплиць, пристроїв, сенсорних вимірювань, команд керування, подій поливу, сповіщень і налаштувань. Такий підхід забезпечує цілісне збереження даних, можливість накопичення історії показників і подальше використання цієї інформації для аналітики.

Програмне забезпечення було реалізовано у вигляді веборієнтованої системи з клієнт-серверною архітектурою. Клієнтська частина розроблена з використанням React, TypeScript та Vite, що дозволило створити зручний інтерфейс для перегляду стану теплиці, графіків, аналітики, сповіщень і керування поливом. Серверна частина реалізована на базі FastAPI та забезпечує

роботу REST API, WebSocket-з’єднань, авторизації, обробки сенсорних даних, керування поливом і взаємодії з базою даних.

Окремою складовою роботи стала інтеграція програмної системи з апаратним вузлом на базі ESP32. Контролер виконує зчитування показників із датчиків температури, вологості повітря, освітленості, тиску та вологості ґрунту, після чого передає їх на серверну частину системи. Також ESP32 отримує команди керування та забезпечує вмикання або вимикання насоса через релейний модуль. Це дозволило поєднати вебзастосунок із реальним фізичним середовищем теплиці.

Для підвищення практичної цінності системи було реалізовано функціонал аналітики та прогнозування. Аналітичний модуль дозволяє обробляти історичні дані мікроклімату, виявляти закономірності у зміні показників, виконувати кластеризацію та формувати прогнозні значення. Це створює основу для подальшого розвитку системи в напрямі інтелектуальної підтримки прийняття рішень щодо догляду за рослинами.

Працездатність основних функціональних можливостей була перевірена за допомогою автоматизованого тестування. Було виконано модульні тести для перевірки алгоритмів зміни вологості ґрунту, логіки поливу та аналітичних функцій, а також інтеграційні тести для перевірки авторизації, роботи API та отримання сенсорних даних. Загалом було виконано 28 автоматизованих тестів, які завершилися успішно, що підтвердило коректність реалізації основних компонентів системи.

Розроблена система розгорнута на сервері та передбачає віддалений доступ користувача через веббраузер. Для спрощення встановлення і супроводу використовується контейнеризація за допомогою Docker та Docker Compose. Інсталяційний пакет містить frontend- і backend-компоненти, конфігураційні файли, міграції бази даних, службові скрипти, прошивку ESP32 та документацію щодо запуску системи.

У результаті виконання роботи всі поставлені завдання були досягнуті. Розроблене програмне забезпечення забезпечує моніторинг параметрів

мікроклімату теплиці, збереження та аналіз даних, керування поливом, формування сповіщень і взаємодію з апаратним контролером. Отримане рішення є практичним, функціональним і може бути використане як основа для подальшого розвитку автоматизованих систем контролю тепличного середовища.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Державна служба статистики України. Методологічні положення державного статистичного спостереження «Площі, валові збори та урожайність сільськогосподарських культур»: наказ від 24.09.2024 № 231 – [Електронний ресурс] – режим доступу: https://www.ukrstat.gov.ua/norm_doc/2024/231/231_2024.pdf (дата звернення: 12.05.2026).
2. Food and Agriculture Organization of the United Nations. Water management – [Електронний ресурс] – режим доступу: https://www.fao.org/land-water/water/water-management (дата звернення: 12.05.2026).
3. Kucheruk P., Yatsenko V., Strelbitskyi M. SCADA system for managing energy flows of an industrial greenhouse // Енергетика і автоматика. 2022. № 4. С. 128–136 – [Електронний ресурс] – режим доступу: https://journals.nubip.edu.ua/index.php/Energiya/en/article/view/energiya2022.04.024 (дата звернення: 12.05.2026).
4. Електротехнологічний комплекс енергоефективного опромінення рослин в теплицях // Енергетика і автоматика. 2024 – [Електронний ресурс] – режим доступу: https://journals.nubip.edu.ua/index.php/Energiya/uk/article/view/51858 (дата звернення: 13.05.2026).
5. Object Management Group. OMG Unified Modeling Language (OMG UML), Version 2.5.1 – [Електронний ресурс] – режим доступу: https://www.omg.org/spec/UML/2.5.1 (дата звернення: 13.05.2026).
6. Priva. Greenhouse control systems and climate control solutions – [Електронний ресурс] – режим доступу: https://www.priva.com/horticulture/solutions/climate-and-process-computers (дата звернення: 14.05.2026).
7. Ridder. Digital, Data & AI solutions for greenhouses and horticulture – [Електронний ресурс] – режим доступу: https://ridder.com/digital-data-ai-solutions (дата звернення: 14.05.2026).

8. Hoogendoorn Growth Management. Worldwide innovator in horticultural automation – [Електронний ресурс] – режим доступу: https://hoogendoorn.com/en (дата звернення: 16.05.2026).
9. Argus Control Systems. Precision climate controls and automation for horticulture – [Електронний ресурс] – режим доступу: https://arguscontrols.com/ (дата звернення: 16.05.2026).
10. Oracle. Oracle SQL Developer Data Modeler User's Guide – [Електронний ресурс] – режим доступу: https://docs.oracle.com/en/database/oracle/sql-developer-data-modeler/18.3/dmdug/oracle-sql-developer-data-modeler-users-guide.pdf (дата звернення: 16.05.2026).
11. PostgreSQL Global Development Group. PostgreSQL Documentation – [Електронний ресурс] – режим доступу: https://www.postgresql.org/docs/ (дата звернення: 08.05.2026).
12. Oracle. MySQL 8.4 Reference Manual – [Електронний ресурс] – режим доступу: https://dev.mysql.com/doc/refman/8.4/en/ (дата звернення: 08.05.2026).
13. Oracle. Object-Oriented Programming Concepts – [Електронний ресурс] – режим доступу: https://docs.oracle.com/javase/tutorial/java/concepts/ (дата звернення: 08.05.2026).
14. Microsoft. Visual Studio Code Documentation – [Електронний ресурс] – режим доступу: https://code.visualstudio.com/docs (дата звернення: 10.05.2026).
15. Python Software Foundation. Python 3 Documentation – [Електронний ресурс] – режим доступу: https://docs.python.org/3/ (дата звернення: 10.05.2026).
16. FastAPI. FastAPI Documentation – [Електронний ресурс] – режим доступу: https://fastapi.tiangolo.com (дата звернення: 09.05.2026).
17. Meta Open Source. React Documentation – [Електронний ресурс] – режим доступу: https://react.dev/learn (дата звернення: 11.05.2026).
18. Vite. Getting Started – [Електронний ресурс] – режим доступу: https://vite.dev/guide/ (дата звернення: 08.05.2026).

19. pandas. pandas documentation – [Електронний ресурс] – режим доступу: https://pandas.pydata.org/docs/ (дата звернення: 14.05.2026).
20. scikit-learn. scikit-learn: machine learning in Python – [Електронний ресурс] – режим доступу: https://scikit-learn.org/stable/ (дата звернення: 14.05.2026).
21. Espressif Systems. ESP-IDF Programming Guide – [Електронний ресурс] – режим доступу: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/ (дата звернення: 14.05.2026).
22. Docker. Docker Compose Documentation – [Електронний ресурс] – режим доступу: https://docs.docker.com/compose/ (дата звернення: 14.05.2026).

# ДОДАТОК А

## Лістинг клієнтської частини програмного забезпечення

### Файл App.tsx

```
import { BrowserRouter, Routes, Route, Navigate } from 'react-router-dom'
import { Toaster } from 'react-hot-toast'
import { useAuthStore } from './store/authStore'
import Layout from './components/layout/Layout'
import AdminLayout from './components/admin/AdminLayout'
import Home from './pages/Home'
import Login from './pages/Login'
import Dashboard from './pages/Dashboard'
import Charts from './pages/Charts'
import IrrigationControl from './pages/IrrigationControl'
import Analytics from './pages/Analytics'
import Alerts from './pages/Alerts'
import Settings from './pages/Settings'
import AdminDashboard from './pages/admin/AdminDashboard'
import AdminUsers from './pages/admin/AdminUsers'
import AdminGreenhouses from './pages/admin/AdminGreenhouses'
import AdminDevices from './pages/admin/AdminDevices'
import AdminIssues from './pages/admin/AdminIssues'
import AdminLogs from './pages/admin/AdminLogs'

function PrivateRoute({ children }: { children: React.ReactNode }) {
  const token = useAuthStore((s) => s.accessToken)
  return token ? <>{children}</> : <Navigate to="/login" replace />
}

function AdminRoute({ children }: { children: React.ReactNode }) {
  const token = useAuthStore((s) => s.accessToken)
  const user = useAuthStore((s) => s.user)
  if (!token) return <Navigate to="/login" replace />
  return user?.role === 'admin' ? <>{children}</> : <Navigate to="/dashboard" replace
/>
}

function PublicOnlyRoute({ children }: { children: React.ReactNode }) {
  const token = useAuthStore((s) => s.accessToken)
  const user = useAuthStore((s) => s.user)
  if (!token) return <>{children}</>
  return <Navigate to={user?.role === 'admin' ? '/admin' : '/dashboard'} replace />
}

export default function App() {
  return (
    <BrowserRouter>
      <Toaster position="top-right" toastOptions={{ style: { background: '#1f2937',
color: '#f9fafb' } }} />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route
          path="/login"
          element={
            <PublicOnlyRoute>
```

```
              <Login />
            </PublicOnlyRoute>
          }
        />
        <Route
          path="/"
          element={
            <PrivateRoute>
              <Layout />
            </PrivateRoute>
          }
        >
          <Route path="dashboard" element={<Dashboard />} />
          <Route path="charts" element={<Charts />} />
          <Route path="irrigation" element={<IrrigationControl />} />
          <Route path="analytics" element={<Analytics />} />
          <Route path="forecast" element={<Forecast />} />
          <Route path="alerts" element={<Alerts />} />
          <Route path="settings" element={<Settings />} />
        </Route>
        <Route
          path="/admin"
          element={
            <AdminRoute>
              <AdminLayout />
            </AdminRoute>
          }
        >
          <Route index element={<AdminDashboard />} />
          <Route path="users" element={<AdminUsers />} />
          <Route path="greenhouses" element={<AdminGreenhouses />} />
          <Route path="devices" element={<AdminDevices />} />
          <Route path="issues" element={<AdminIssues />} />
          <Route path="logs" element={<AdminLogs />} />
        </Route>
        <Route path="*" element={<Navigate to="/" replace />} />
      </Routes>
    </BrowserRouter>
  )
}
```

**Файл IrrigationControl.tsx**

```
import { useEffect, useState, useCallback, useRef } from 'react'
import { irrigationApi } from '../api/irrigation'
import { sensorsApi } from '../api/sensors'
import IrrigationProgress from '../components/irrigation/IrrigationProgress'
import SoilMoistureChart from '../components/irrigation/SoilMoistureChart'
import { useWebSocket } from '../hooks/useWebSocket'
import { useSensorStore, type SensorData } from '../store/sensorStore'
import toast from 'react-hot-toast'
import { formatISO, subHours } from 'date-fns'

const WS_URL = (import.meta.env.VITE_WS_URL || 'ws://localhost:8000') + '/ws/sensors'
const WS_ALERTS = (import.meta.env.VITE_WS_URL || 'ws://localhost:8000') +
'/ws/alerts'

interface Settings { auto_mode: boolean; min_threshold: number; optimal_target:
number; irrigation_rate: number }
```

```
interface Status { is_irrigating: boolean; current_moisture: number; target: number;
mode: string }
type NumericSettingKey = 'min_threshold' | 'optimal_target' | 'irrigation_rate'

const numericSettingFields: Array<{ key: NumericSettingKey; label: string; min:
number; max: number }> = [
  { key: 'min_threshold', label: 'Мін. поріг (%)', min: 0, max: 100 },
  { key: 'optimal_target', label: 'Оптимальна вологість (%)', min: 0, max: 100 },
  { key: 'irrigation_rate', label: 'Швидкість поливу (%/хв)', min: 0.1, max: 10 },
]

export default function IrrigationControl() {
  const [status, setStatus] = useState<Status | null>(null)
  const [settings, setSettings] = useState<Settings | null>(null)
  const [historyData, setHistoryData] = useState<{ timestamp: string; value: number
}[]>([])
  const [form, setForm] = useState<Settings | null>(null)
  const [saving, setSaving] = useState(false)
  const { setCurrent } = useSensorStore()
  const lastDoneToastRef = useRef<{ key: string; ts: number }>({ key: '', ts: 0 })

  const handleToggleAutoMode = async () => {
    if (!form) return
    const prev = form.auto_mode
    const next = !prev

    // Optimistic UI: reflect switch movement instantly, then persist on backend.
    setForm((f) => (f ? { ...f, auto_mode: next } : f))

    try {
      const updated = await irrigationApi.updateSettings({ auto_mode: next })
      setSettings(updated)
      setForm((f) => (f ? { ...f, auto_mode: updated.auto_mode } : f))
      toast.success(`Автополив ${updated.auto_mode ? 'увімкнено' : 'вимкнено'}`)
      irrigationApi.getStatus().then(setStatus).catch(() => {})
    } catch {
      setForm((f) => (f ? { ...f, auto_mode: prev } : f))
      toast.error('Не вдалося змінити режим автополиву')
    }
  }

  const loadAll = () => {
    irrigationApi.getStatus().then(setStatus).catch(() => {})
     irrigationApi.getSettings().then((s: Settings) => { setSettings(s); setForm(s)
}).catch(() => {})
      sensorsApi.getHistory('soil_moisture', formatISO(subHours(new Date(), 6)),
undefined, '5min')
      .then((d: { data: [] }) => setHistoryData(d.data || []))
      .catch(() => {})
  }

  useEffect(() => { loadAll() }, [])

  const applySensorReading = useCallback((d: Partial<SensorData>) => {
     const soilMoisture = typeof d.soil_moisture === 'number' ? d.soil_moisture :
undefined
    const isIrrigating = d.is_irrigating ?? d.irrigation_on
    const timestamp = d.timestamp || new Date().toISOString()
```

```
    setCurrent({ ...d, is_irrigating: Boolean(isIrrigating) } as SensorData)
    setStatus((s) => {
      if (!s) return s
      return {
        ...s,
        current_moisture: soilMoisture ?? s.current_moisture,
        is_irrigating: isIrrigating ?? s.is_irrigating,
      }
    })

    if (soilMoisture !== undefined) {
      setHistoryData((points) => {
           const nextPoint = { timestamp, value: soilMoisture, is_irrigating:
Boolean(isIrrigating) }
        const last = points[points.length - 1]
             if (last && Math.abs(new Date(last.timestamp).getTime() - new
Date(timestamp).getTime()) < 1000) {
          return [...points.slice(0, -1), nextPoint]
        }
        return [...points.slice(-119), nextPoint]
      })
    }
  }, [setCurrent])

  const handleWs = useCallback((data: unknown) => {
    applySensorReading(data as Partial<SensorData>)
  }, [applySensorReading])
  useWebSocket(WS_URL, handleWs)

  useEffect(() => {
    let cancelled = false

    const refreshCurrentReading = async () => {
      try {
        const reading = await sensorsApi.getCurrent()
        if (!cancelled) {
          applySensorReading(reading)
        }
      } catch {}
    }

    const timer = window.setInterval(refreshCurrentReading, 1000)
    return () => {
      cancelled = true
      window.clearInterval(timer)
    }
  }, [applySensorReading])

  const handleAlertWs = useCallback((data: unknown) => {
    const d = data as { type?: string; message?: string; value?: number }
     if (!d.type || (!d.type.endsWith('_done') && d.type !== 'irrigation_done'))
return

     const msg = d.message || `Полив завершено. Вологість ґрунту: ${(d.value ??
0).toFixed(1)}%`
    const key = `irrigation_done|${msg}`
    const now = Date.now()
```

```
    if (lastDoneToastRef.current.key === key && now - lastDoneToastRef.current.ts <
2000) {
      return
    }

    lastDoneToastRef.current = { key, ts: now }
    toast.success(msg)
    irrigationApi.getStatus().then(setStatus).catch(() => {})
  }, [])
  useWebSocket(WS_ALERTS, handleAlertWs)

  const handleStart = async () => {
    try {
      await irrigationApi.start()
      toast.success('Полив розпочато')
      window.dispatchEvent(new Event('alerts:updated'))
      loadAll()
    } catch { toast.error('Помилка запуску') }
  }

  const handleStop = async () => {
    try {
      await irrigationApi.stop()
      toast.success('Полив зупинено')
      window.dispatchEvent(new Event('alerts:updated'))
      loadAll()
    } catch { toast.error('Помилка зупинки') }
  }

  const handleSave = async () => {
    if (!form) return
    setSaving(true)
    try {
      await irrigationApi.updateSettings(form)
      toast.success('Налаштування збережено')
      loadAll()
    } catch { toast.error('Помилка збереження') } finally { setSaving(false) }
  }

  return (
    <div>
      <h1 className="text-2xl font-bold mb-6">Управління поливом</h1>
            {status   &&   <IrrigationProgress   isIrrigating={status.is_irrigating}
currentMoisture={status.current_moisture} target={status.target} />}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6 mb-6">
        <div className="bg-gray-900 border border-gray-800 rounded-xl p-5">
          <h2 className="text-lg font-semibold mb-4">Ручне керування</h2>
          <div className="flex gap-3">
                  <button  onClick={handleStart}  disabled={status?.is_irrigating}
className="flex-1 bg-blue-600 hover:bg-blue-700 disabled:opacity-40 text-white py-
2.5 rounded-lg font-medium transition-colors">
              Запустити полив
            </button>
                  <button  onClick={handleStop}  disabled={!status?.is_irrigating}
className="flex-1 bg-red-600 hover:bg-red-700 disabled:opacity-40 text-white py-2.5
rounded-lg font-medium transition-colors">
              Зупинити полив
            </button>
          </div>
```

```
          <div className="mt-4 text-sm text-gray-500">
                      Поточна  вологість:  <span  className="text-cyan-400  font-
medium">{status?.current_moisture.toFixed(1)}%</span>
          </div>
        </div>
        {form && (
          <div className="bg-gray-900 border border-gray-800 rounded-xl p-5">
            <h2 className="text-lg font-semibold mb-4">Налаштування автополиву</h2>
            <div className="flex items-center gap-3 mb-4">
              <button
                onClick={handleToggleAutoMode}
               className={`w-12 h-6 rounded-full transition-colors ${form.auto_mode
? 'bg-green-500' : 'bg-gray-700'}`}
              >
                 <div className={`w-5 h-5 bg-white rounded-full mx-0.5 transition-
transform ${form.auto_mode ? 'translate-x-6' : ''}`} />
              </button>
               <span className="text-sm text-gray-400">Автополив {form.auto_mode ?
'увімкнено' : 'вимкнено'}</span>
            </div>
            {numericSettingFields.map(({ key, label, min, max }) => (
              <div key={key} className="mb-3">
                <label className="block text-sm text-gray-400 mb-1">{label}</label>
                <input
                  type="number"
                  step="0.1"
                  min={min}
                  max={max}
                  value={form[key]}
                          onChange={(e)  =>  setForm((f)  =>  f  ?  {  ...f,  [key]:
parseFloat(e.target.value) } : f)}
                   className="w-full bg-gray-800 border border-gray-700 rounded-lg
px-3 py-1.5 text-gray-100 focus:outline-none focus:border-green-500 text-sm"
                />
              </div>
            ))}
              <button onClick={handleSave} disabled={saving} className="w-full bg-
green-600 hover:bg-green-700 disabled:opacity-50 text-white py-2 rounded-lg text-sm
font-medium mt-2">
              {saving ? 'Збереження...' : 'Зберегти'}
            </button>
          </div>
        )}
      </div>
      <SoilMoistureChart data={historyData} minThreshold={form?.min_threshold ?? 30}
optimalTarget={form?.optimal_target ?? 65} />
    </div>
  )
}
```

# ДОДАТОК Б

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**
**Факультет інформаційних технологій**

## Декларація про академічну доброчесність та використання ШІ

Я, Спєлов Сергій Сергійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення системи моніторингу параметрів мікроклімату теплиць» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [ ] інструменти генеративного ШІ не використовувалися.
- [ ] інструменти ШІ (вказати назву: ChatGPT, Gemini, Claude, DeepL, _____інші (вказати назву)) були використані для:
  - [ ] перекладу іншомовних джерел;
  - [ ] стилістичного редагування та перевірки граматики;
  - [ ] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«___» _____ 2026 р. _____ **Сергій СПЄЛОВ**
(підпис)