НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ 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 у теплиці»

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Гарант освітньої програми
к.т.н., доцент _____ **Ганна ВАЙГАНГ**
(підпис)

Керівник бакалаврської кваліфікаційної роботи
д.е.н., професор _____ **Ірина БОРОДКІНА**
(підпис)

Виконав _____ **Ярослав ПЕЛЕХАНЬ**
(підпис)

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

Пелеханю Ярославу Ігоровичу
(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 у теплиці»

затверджена наказом від «19» грудня 2025 р. № 1026 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
Перелік питань, що підлягають дослідженню:
Аналіз предметної області системи моніторингу параметрів мікроклімату
Розробка програмного забезпечення системи моніторингу параметрів мікроклімату
Перелік графічних документів (за необхідності).
_____

Дата видачі завдання «14» квітня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Ірина БОРОДКІНА**
(підпис)

Завдання взяв до виконання _____ **Ярослав ПЕЛЕХАНЬ**
(підпис)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ І СКОРОЧЕНЬ

- API – прикладний програмний інтерфейс (Application Programming Interface)
- CSV – формат табличних даних із розділенням значень комами (Comma-Separated Values)
- DB – база даних (Database)
- ER-діаграма – діаграма «сутність-зв’язок» (Entity-Relationship Diagram)
- GUI – графічний інтерфейс користувача (Graphical User Interface)
- IoT – інтернет речей (Internet of Things)
- JSON – текстовий формат обміну структурованими даними (JavaScript Object Notation)
- PyQt6 – бібліотека Python для створення графічних інтерфейсів на основі Qt 6
- Qt – кросплатформний фреймворк для розроблення графічних інтерфейсів
- SQL – мова структурованих запитів до баз даних (Structured Query Language)
- SQLite – вбудована файлова система керування базами даних
- UML – уніфікована мова моделювання (Unified Modeling Language)
- USB – універсальна послідовна шина для підключення пристроїв (Universal Serial Bus)
- БД – база даних
- Датчик – технічний засіб для вимірювання параметрів мікроклімату
- ІС – інформаційна система
- ПЗ – програмне забезпечення

– СКБД – система керування базами даних
– CSV-звіт – файл звіту з табличними даними, сформований системою

ВСТУП

Розвиток захищеного ґрунту, тепличного вирощування овочевих і декоративних культур, а також поширення елементів точного землеробства зумовлюють потребу у постійному контролі параметрів мікроклімату. Температура повітря, відносна вологість, вологість ґрунту та освітленість безпосередньо впливають на інтенсивність фотосинтезу, водний режим рослин, розвиток кореневої системи та ризик виникнення стресових умов. У невеликих теплицях такі параметри часто контролюються вручну, що призводить до запізнілого реагування на перегрів, переохолодження, пересушування ґрунту або надмірну вологість.

Автоматизація моніторингу параметрів мікроклімату дозволяє перейти від періодичного візуального огляду до систематичного збору даних, їх збереження у базі даних, порівняння з допустимими межами та формування сповіщень. Таке програмне забезпечення є корисним як для навчально-дослідних теплиць, так і для малих агропідприємств, оскільки забезпечує простий контроль стану середовища без складної промислової SCADA-інфраструктури.

Актуальність теми полягає у необхідності створення доступного настільного програмного забезпечення, яке забезпечує приймання даних від датчиків мікроклімату, візуалізацію поточних показників, збереження історії вимірювань, налаштування порогових значень, формування сповіщень та експорт звітної інформації. Особливістю розроблюваної системи є орієнтація на локальне використання із застосуванням Python, PyQt6 та SQLite, що спрощує встановлення, супровід і демонстрацію роботи системи у навчальних умовах.

Метою кваліфікаційної роботи є проєктування та розроблення програмного забезпечення системи моніторингу параметрів мікроклімату у теплиці, яке забезпечує автоматизований збір, збереження, аналіз і

відображення показників температури повітря, вологості повітря, вологості ґрунту та освітленості з можливістю формування попереджень і звітів.

Для досягнення поставленої мети необхідно розв’язати такі **завдання:**

– проаналізувати предметну область моніторингу мікроклімату у тепличних приміщеннях;

– розглянути існуючі програмні та апаратно-програмні рішення для контролю тепличного середовища;

– побудувати UML-моделі предметної області, зокрема діаграми прецедентів, послідовності та активності;

– сформувати функціональні, нефункціональні, технічні та безпекові вимоги до системи;

– розробити логічну та фізичну моделі даних для збереження вимірювань, датчиків, порогів, сповіщень і звітів;

– обґрунтувати вибір технологічних засобів реалізації програмного забезпечення;

– розробити програмну систему з графічним інтерфейсом, базою даних SQLite та модулем імітації датчиків;

– провести тестування основних модулів, інтерфейсу користувача та механізму формування сповіщень.

Об’єктом дослідження є процес моніторингу параметрів мікроклімату у теплиці.

Предметом дослідження є методи, моделі та програмні засоби збору, збереження, аналізу й візуалізації даних мікроклімату у локальній інформаційній системі.

Методологічною основою роботи є методи системного аналізу, об’єктно-орієнтованого проєктування, UML-моделювання, ER-моделювання, реляційного проєктування баз даних, модульного програмування, функціонального та інтерфейсного тестування.

Практична значущість роботи полягає у створенні настільного програмного застосунку, який може використовуватися для демонстрації

принципів автоматизованого моніторингу теплиці, ведення журналу вимірювань, оперативного контролю порогів та формування CSV-звітів. Розроблена програмна модель може бути розширена шляхом підключення реальних датчиків через послідовний порт, USB-інтерфейс або мікроконтролерний модуль.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Опис предметної області

Предметна область системи моніторингу параметрів мікроклімату у теплиці охоплює процеси збирання, первинної перевірки, збереження, аналізу та відображення даних про стан внутрішнього середовища тепличного приміщення. До основних параметрів, які контролюються у межах роботи, належать температура повітря, відносна вологість повітря, вологість ґрунту та освітленість. Саме ці показники є базовими для оцінювання умов росту рослин і можуть бути виміряні типовими цифровими або аналоговими датчиками.

Теплиця розглядається як локальний об’єкт моніторингу, у якому встановлюються датчики мікроклімату та, за потреби, виконавчі пристрої. Датчики формують вимірювання у вигляді числових значень з одиницями виміру, часовою міткою та службовою ознакою якості. Програмна система приймає ці дані, нормалізує їх до єдиного формату, порівнює з пороговими значеннями та відображає оператору у вигляді карток, таблиць і графіків.

До виконавчих пристроїв у такій предметній області можуть належати вентилятори, нагрівальні елементи, система поливу, приводи кватирок або модулі додаткового освітлення. У межах розроблюваної програмної системи основний акцент зроблено на моніторингу, сповіщеннях і фіксації подій, а керування обладнанням подано як логічний сценарій, який може бути реалізований у подальших версіях системи.

Типовий цикл роботи системи містить такі операції: зчитування значень із датчиків, перевірка формату і допустимості даних, збереження результатів у базі даних, оновлення інформаційної панелі, перевірка порогів, створення сповіщення у разі відхилення та формування історичного звіту. На відміну від ручного контролю, програмний підхід забезпечує фіксацію кожного вимірювання та дозволяє аналізувати динаміку зміни параметрів за обраний період.

Особливістю мікроклімату теплиці є взаємозв’язок контрольованих параметрів. Підвищення температури може призводити до зменшення відносної вологості, надмірна вологість ґрунту може свідчити про перевищення режиму поливу, а нестача освітленості впливає на продуктивність рослин у періоди короткого світлового дня. Тому програмна система має не лише зберігати окремі значення, а й надавати користувачу узагальнену картину стану теплиці.

Таблиця 1.1

Параметри тапличного середовища

Параметр	Одиниця	Призначення контролю	Типова реакція системи
Температура повітря	°C	Оцінювання теплового режиму в теплиці	Попередження про перегрів або переохолодження
Вологість повітря	%	Контроль ризику пересушування або надмірної вологості	Попередження про відхилення від допустимих меж
Вологість ґрунту	%	Оцінювання потреби у поливі та стану кореневої зони	Фіксація події недостатнього або надмірного зволоження
Освітленість	лк	Контроль світлового режиму	Попередження про недостатню або надмірну освітленість

Отже, предметна область розроблюваної системи поєднує елементи сенсорного моніторингу, локального збереження даних, порогового аналізу, інтерфейсної візуалізації та формування звітів. Для її реалізації доцільно застосувати модульну архітектуру, у якій окремо виділяються модулі графічного інтерфейсу, збору даних, моніторингу, сповіщень, доступу до бази даних і роботи зі звітами.

1.2. Аналіз існуючих рішень у сфері моніторингу мікроклімату теплиць

Існуючі програмно-апаратні рішення у сфері моніторингу мікроклімату теплиць орієнтовані на збір, відображення та аналіз показників температури,

вологості повітря, вологості ґрунту, освітленості, концентрації CO_2, стану вентиляції, поливу, обігріву та інших технологічних параметрів. Такі системи використовуються для підтримання оптимальних умов вирощування рослин, зменшення ручного контролю та своєчасного реагування на відхилення параметрів від установлених норм.

Існуючі рішення можна умовно поділити на кілька груп: промислові системи керування тепличним кліматом, професійні системи автоматизації з підтримкою виконавчих механізмів, хмарні платформи моніторингу, локальні IoT-рішення та відкриті системи на базі мікроконтролерів. Для аналізу було обрано п’ять рішень: Priva Compass Operator, Argus Controls, MySirius, Ridder HortiMaX-Go! та Home Assistant / ESPHome.

Одним із відомих професійних рішень є Priva Compass Operator. Система призначена для контролю параметрів теплиці, перегляду поточних значень, налаштування кліматичних стратегій та дистанційного керування обладнанням. Програмний інтерфейс Priva Operator дає змогу переглядати вимірювання та налаштування теплиці в режимі реального часу з браузера або мобільного пристрою. На рисунку 1.1 наведено приклад інтерфейсу Priva Compass Operator, де відображено температуру, вологість, режими охолодження, вентиляції, освітлення та поливу. Перевагами цієї системи є професійна реалізація, зручне подання даних, підтримка віддаленого доступу та можливість інтеграції з промисловими контролерами. Недоліками є залежність від фірмового обладнання, висока вартість упровадження та надлишкова функціональність для невеликої навчальної або локальної теплиці.

Рис. 1.1. Інтерфейс системи Priva Compass Operator

Система Argus Controls належить до промислових рішень автоматизації тепличних комплексів. Вона забезпечує контроль клімату, живлення рослин, зрошення, вентиляції, обігріву та інших процесів, які впливають на стан рослин. На рисунку 1.2 показано інтерфейс Argus Controls із відображенням показників температури та вологості, цільових значень, запитів на обігрів або охолодження та графіка зміни кліматичних параметрів. Основною перевагою Argus Controls є гнучкість налаштування, масштабованість і придатність для промислових тепличних господарств, дослідних теплиць та закритих агровиробничих приміщень. Недоліком є складність конфігурування, потреба у спеціалізованому обладнанні та орієнтація на професійне використання, що робить таке рішення менш доцільним для простої настільної системи моніторингу.

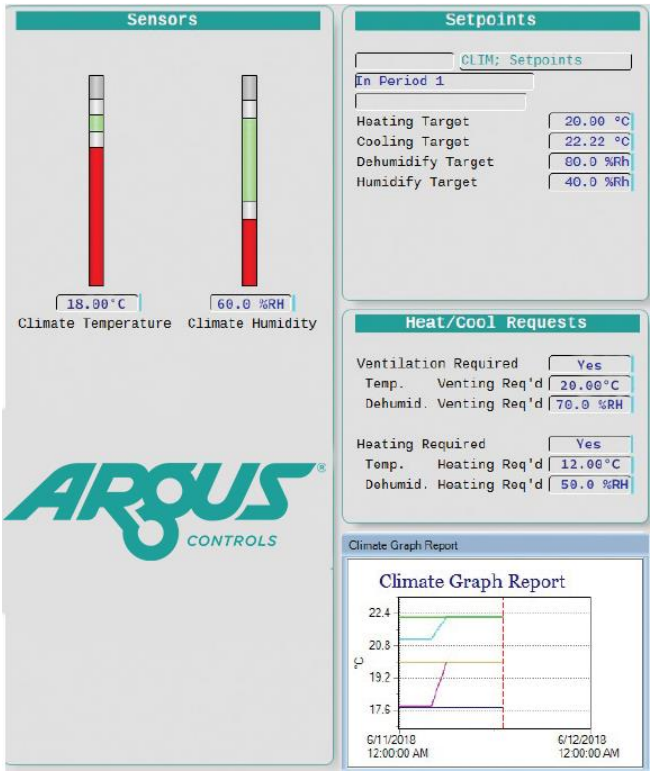


Рис. 1.2. Інтерфейс системи Argus Controls

Окрему групу становлять хмарні системи моніторингу середовища, зокрема MySirius. Це рішення використовується для контролю температури та інших фізичних параметрів, включаючи вологість, CO_2, тиск і стан контрольованих зон. Система підтримує роботу з підключеними датчиками, ведення журналу подій, формування звітів і надсилання сповіщень у разі відхилень. На рисунку 1.3 наведено приклад інтерфейсу MySirius із таблицями параметрів, графіками та вікнами налаштувань. Перевагою такого підходу є зручний централізований моніторинг, можливість роботи з різними об’єктами та наявність механізмів звітності. Водночас система залежить від хмарної інфраструктури, зовнішнього сервісу та підключення до мережі, тому для автономної локальної теплиці вона є менш придатною.

Рис. 1.3. Інтерфейс хмарної системи моніторингу MySirius

Ridder HortiMaX-Go! є прикладом спеціалізованого рішення для керування кліматом і зрошенням у теплиці. Система орієнтована на забезпечення контролю кліматичних параметрів, поливу та технологічних режимів вирощування рослин. На рисунку 1.4 показано інтерфейс Ridder HortiMaX-Go!, у якому відображаються температура, освітленість, вологість, стан обладнання та службові індикатори. Перевагою рішення є поєднання функцій моніторингу та керування, відносно зрозумілий інтерфейс, підтримка кліматичних і зрошувальних процесів. Недоліками є прив’язка до конкретної апаратної екосистеми та орієнтація на комерційне тепличне господарство, що може бути надлишковим для навчальної або демонстраційної системи.

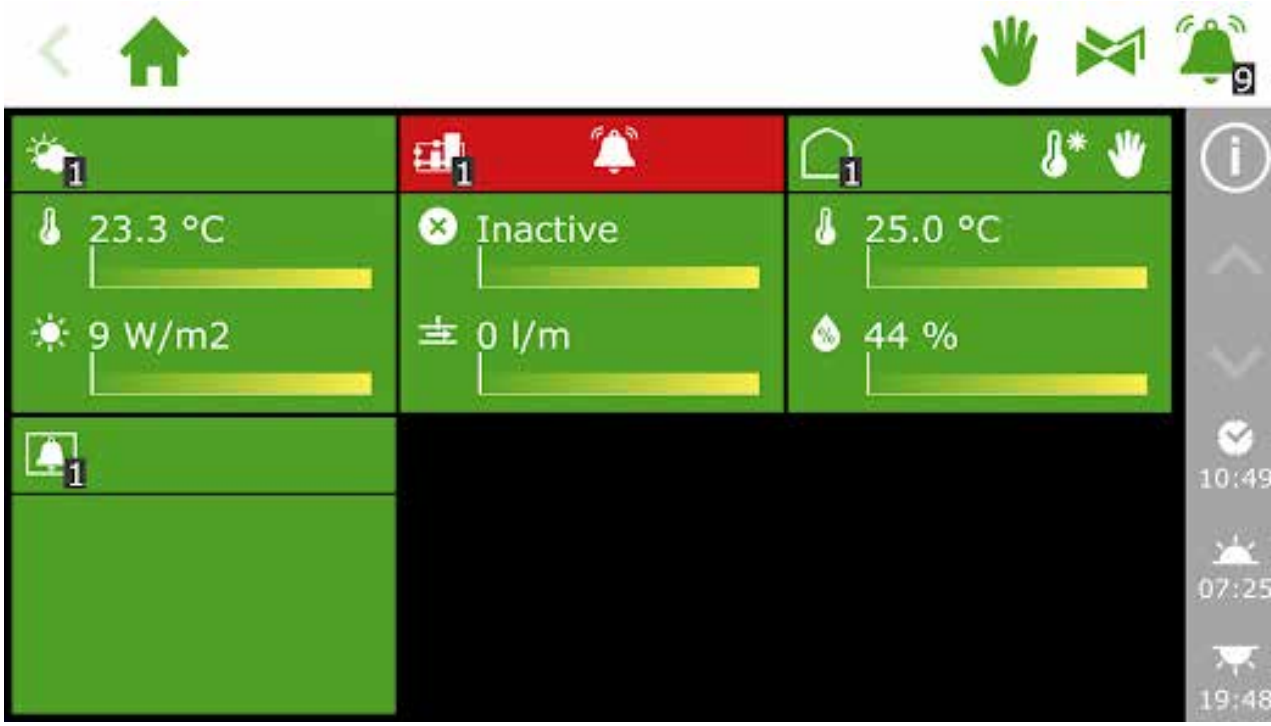


Рис. 1.4 Інтерфейс системи Ridder HortiMaX-Go!

Поширеним варіантом побудови локальних IoT-рішень є використання Home Assistant у поєднанні з ESPHome. Такий підхід дозволяє підключати датчики на базі ESP8266 або ESP32, створювати сенсорні вузли, передавати показники до локальної системи автоматизації та відображати їх у веб-інтерфейсі. ESPHome підтримує інтеграцію з Home Assistant через нативний API, що дає змогу використовувати створені сенсорні пристрої як джерела даних для автоматизацій. На рисунку 1.5 наведено приклад інтерфейсу ESPHome з переліком підключених пристроїв. Перевагами цього рішення є відкритість, гнучкість, низька вартість апаратної частини та можливість локального розгортання. Основними недоліками є потреба у технічних навичках, налаштуванні YAML-конфігурацій, прошиванні мікроконтролерів і додатковій інтеграції з графічним інтерфейсом.

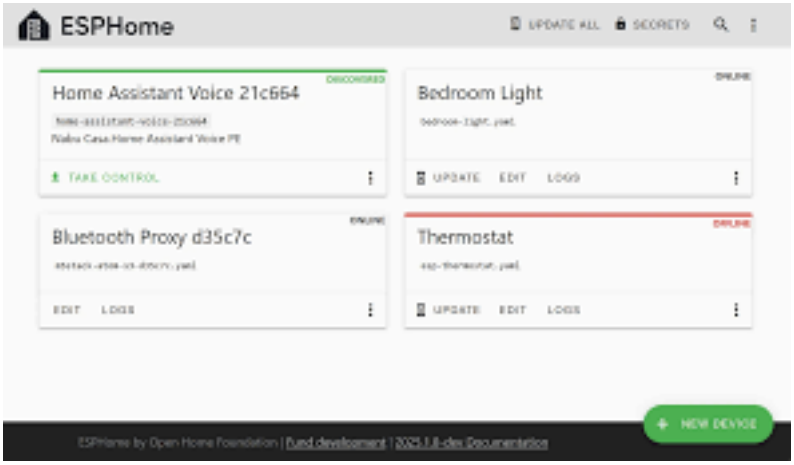


Рис. 1.5. Інтерфейс Home Assistant / ESPHome

Порівняння розглянутих рішень та розроблюваної системи наведено в таблиці 1.2.

Таблиця 1.2

Порівняння існуючих рішень у сфері моніторингу мікроклімату теплиць

Критерій	Priva Compass Operator	Argus Controls	MySirius	Ridder HortiMaX-Go!	Home Assistant / ESPHome	Розр с
Основне призначення	Професійне керування тепличним кліматом	Промислова автоматизація теплиць	Хмарний моніторинг параметрів середовища	Клімат і зрошення теплиць	Локальна IoT-автоматизація	Лока мон мікр
Контроль температури і вологості	так	так	так	так	так	так
Контроль вологості ґрунту та освітленості	так	так	частково	так	залежить від датчиків	так
Порогові значення і сповіщення	так	так	так	так	так	так
Збереження історії вимірювань	так	так	так	так	так	так

Продовження таблиці 1.2

Робота без хмарного сервісу	частково	частково	ні	частково	так	так
Простота встановлення	низька	низька	середня	середня	середня	висока

Придатність для невеликої теплиці	обмежена через надлишковість	обмежена через складність	залежить від сервісу	середня	висока, але потребує налаштування	висока
Залежність від постачальника	висока	висока	висока	висока	низька	низька
Навчальна демонстрація роботи системи	обмежена	обмежена	середня	середня	висока	висока

Аналіз існуючих рішень показує, що професійні системи Priva, Argus Controls і Ridder HortiMaX-Go! мають широкі можливості для промислового керування тепличними комплексами, проте є складними, дорогими та прив’язаними до спеціалізованого обладнання. Хмарне рішення MySirius забезпечує зручний моніторинг і звітність, але залежить від зовнішньої інфраструктури. Home Assistant / ESPHome є гнучким і доступним варіантом, однак потребує додаткового налаштування, роботи з мікроконтролерами та технічної підготовки користувача.

Отже, для невеликої або навчально-дослідної теплиці доцільним є створення локального настільного програмного забезпечення, яке забезпечує основні функції моніторингу без надмірного ускладнення. Розроблювана система орієнтована на роботу з показниками температури повітря, вологості повітря, вологості ґрунту та освітленості, збереження історії вимірювань у базі даних SQLite, налаштування порогових значень, формування сповіщень і звітів. Такий підхід дає змогу отримати автономне, просте в експлуатації та придатне для подальшого розширення програмне забезпечення системи моніторингу параметрів мікроклімату у теплиці.

1.3. Моделювання предметної області

Моделювання предметної області виконано з використанням UML-діаграм, які дозволяють формалізувати ролі користувачів, сценарії взаємодії, послідовність оброблення даних і внутрішню логіку роботи програмного

забезпечення. Для розроблюваної системи побудовано діаграму прецедентів, діаграму послідовності та діаграму активності.

На рисунку 1.6 наведено діаграму прецедентів системи. Вона відображає взаємодію оператора теплиці, адміністратора, датчиків мікроклімату та виконавчих пристроїв із програмною системою.

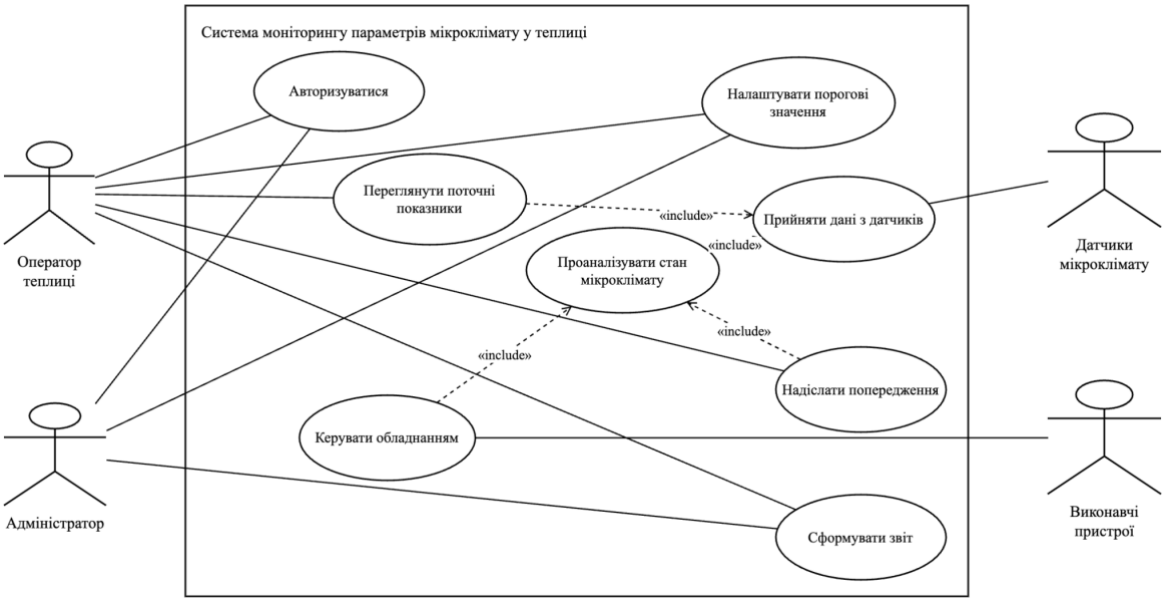


Рис. 1.6. Діаграма прецедентів системи моніторингу параметрів мікроклімату у теплиці

Основним користувачем системи є оператор теплиці, який авторизується, переглядає поточні показники, отримує попередження та формує звіти. Адміністратор має розширені можливості: налаштовує порогові значення, керує датчиками та може ініціювати логічні дії щодо обладнання. Зовнішнім джерелом даних виступають датчики мікроклімату, які передають показники температури, вологості повітря, вологості ґрунту й освітленості.

У межах діаграми прецедентів процес перегляду поточних показників пов’язаний із прийманням даних від датчиків та аналізом стану мікроклімату. Надсилання попереджень виконується тоді, коли показник виходить за межі встановленого порогу. Така структура відповідає логіці роботи програмної системи і не перевантажує модель другорядними сценаріями.

Для деталізації сценарію контролю параметрів побудовано діаграму послідовності, наведену на рисунку 1.6.

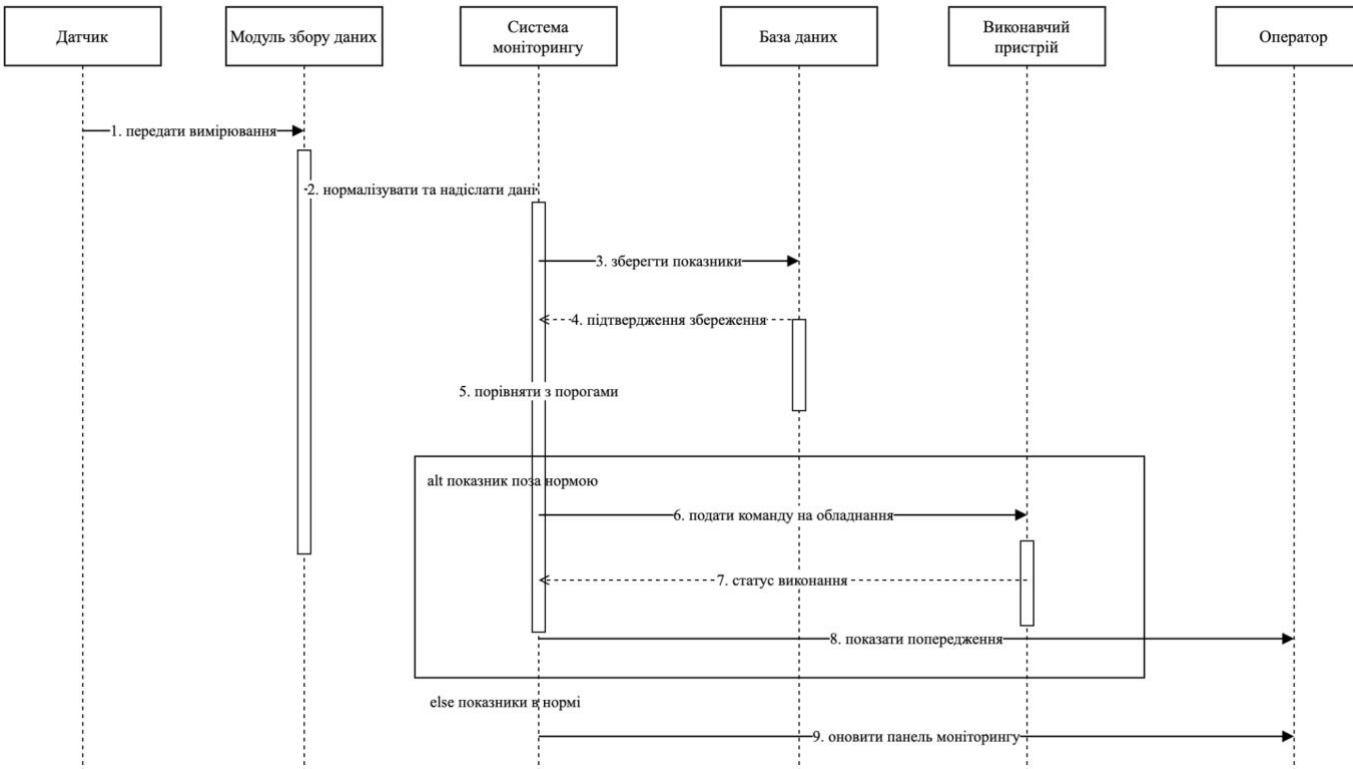


Рис. 1.6. Діаграма послідовності процесу контролю параметрів мікроклімату

Діаграма послідовності відображає типовий сценарій проходження даних від датчика до користувача. Спочатку датчик передає вимірювання до модуля збору даних, де показники нормалізуються та надсилаються до системи моніторингу. Далі система зберігає дані у базі, отримує підтвердження збереження та порівнює значення з установленими порогами.

Якщо показник перебуває поза нормою, система формує команду або попередження, отримує статус виконання від виконавчого пристрою та показує оператору повідомлення. Якщо значення перебувають у допустимих межах, система лише оновлює панель моніторингу. Отже, діаграма відображає як основний сценарій, так і альтернативну гілку при порушенні порога.

Поведінкову логіку роботи системи подано на діаграмі активності, що наведена на рисунку 1.7.

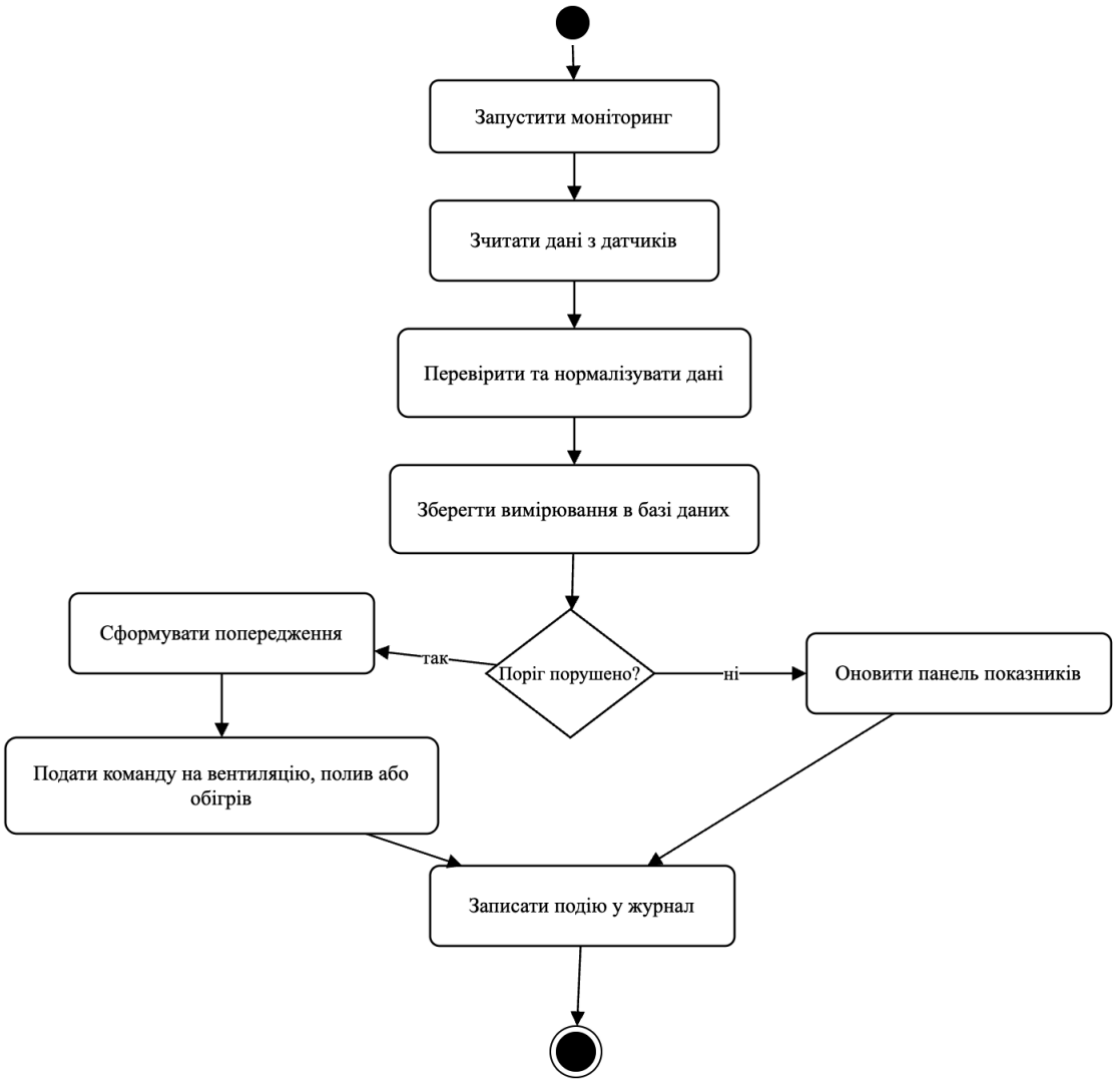


Рис. 1.7. Діаграма активності процесу моніторингу параметрів мікроклімату

Процес починається із запуску моніторингу, після чого система зчитує дані з датчиків, перевіряє та нормалізує їх, зберігає вимірювання у базі даних і порівнює показники з порогами. Якщо поріг порушено, формується попередження, а за потреби подається команда на вентиляцію, полив або обігрів. Якщо порушень немає, система оновлює панель поточних показників.

Завершальним етапом у будь-якому сценарії є запис події до журналу. Такий підхід забезпечує простежуваність дій системи, дає змогу аналізувати історію роботи та формувати звіти за обраний період. Комплекс UML-діаграм формує основу для подальшого проєктування інформаційного та програмного забезпечення.

1.4. Аналіз вимог розроблюваної системи

Розроблення програмного забезпечення системи моніторингу параметрів мікроклімату у теплиці потребує попереднього визначення вимог, які описують призначення системи, її функціональні можливості, технічні обмеження, якісні характеристики та вимоги до безпеки даних. Аналіз вимог є важливим етапом проєктування, оскільки саме на його основі формується структура програмної системи, визначаються основні модулі, інформаційні потоки, склад бази даних і логіка взаємодії користувача із застосунком.

Розроблювана система призначена для локального моніторингу основних параметрів мікроклімату теплиці: температури повітря, вологості повітря, вологості ґрунту та освітленості. Програмне забезпечення має забезпечувати приймання або імітацію даних від датчиків, відображення поточного стану теплиці, збереження історії вимірювань у базі даних, перевірку значень за встановленими порогами, формування сповіщень і створення звітів за вибраний період.

Основним користувачем системи є оператор теплиці, який здійснює перегляд поточних показників, аналіз історії вимірювань, налаштування допустимих меж параметрів і формування звітів. Адміністратор системи додатково має можливість керувати переліком датчиків, обліковими записами користувачів і базовими налаштуваннями системи. Вхідними даними є значення з датчиків або згенеровані тестові вимірювання, а вихідними даними є таблиці вимірювань, графіки, сповіщення про відхилення, CSV-звіти та записи у базі даних.

Функціональні вимоги визначають набір дій, які система повинна виконувати під час роботи. Узагальнений перелік функціональних вимог наведено у таблиці 1.3.

Таблиця 1.3

Функціональні вимоги до системи моніторингу параметрів мікроклімату у теплиці

№	Функціональна вимога	Опис
1	Авторизація користувача	Система повинна забезпечувати вхід користувача за логіном і паролем
2	Реєстрація користувача	Система повинна дозволяти створення нового облікового запису
3	Перегляд поточних показників	Користувач повинен бачити актуальні значення температури, вологості повітря, вологості ґрунту та освітленості
4	Опитування датчиків	Система повинна отримувати або імітувати нові значення параметрів мікроклімату
5	Збереження вимірювань	Усі отримані показники повинні зберігатися в базі даних SQLite
6	Перегляд історії вимірювань	Користувач повинен мати можливість переглянути вимірювання за вибраний період
7	Побудова графіків	Система повинна відображати динаміку зміни параметрів у графічному вигляді
8	Налаштування порогів	Користувач повинен мати можливість змінювати мінімальні та максимальні допустимі значення параметрів
9	Формування сповіщень	У разі виходу показника за межі норми система повинна створювати попередження
10	Перегляд сповіщень	Користувач повинен мати можливість переглядати активні та опрацьовані сповіщення
11	Формування звітів	Система повинна формувати звіт за вибраний період на основі збережених вимірювань
12	Експорт даних	Користувач повинен мати можливість експортувати вимірювання або звіти у форматі CSV
13	Керування датчиками	Система повинна дозволяти перегляд і додавання датчиків із зазначенням типу, одиниці вимірювання та статусу

Функціональні вимоги охоплюють повний цикл роботи із даними мікроклімату: отримання показників, їх збереження, перевірку, візуалізацію та подальше використання для звітності. Особливу роль у системі відіграє

механізм порогового контролю, оскільки саме він дає змогу своєчасно виявляти небажані зміни температури, вологості або освітленості.

Нефункціональні вимоги характеризують якість роботи програмного забезпечення, його зручність, продуктивність, надійність і придатність до експлуатації. Для розроблюваної системи такі вимоги наведено у таблиці 1.4.

Таблиця 1.4

Нефункціональні вимоги до розроблюваної системи

№	Вимога	Характеристика
1	Зручність інтерфейсу	Інтерфейс має бути простим, зрозумілим і придатним для користувача без спеціальної підготовки
2	Швидкість роботи	Основні операції перегляду даних, оновлення показників і відкриття таблиць мають виконуватися без помітних затримок
3	Локальна автономність	Система повинна працювати на комп’ютері користувача без обов’язкового підключення до хмарного сервісу
4	Надійність збереження даних	Дані вимірювань, сповіщень, порогів і звітів повинні зберігатися у локальній базі даних
5	Масштабованість	Структура системи повинна дозволяти додавання нових типів датчиків без повної переробки програми
6	Супроводжуваність	Програмний код має бути поділений на логічні модулі: інтерфейс, сервіси, робота з базою даних, моделі даних
7	Сумісність	Застосунок має працювати на сучасних версіях Windows, macOS або Linux за наявності Python і потрібних бібліотек
8	Стабільність	Помилки введення або тимчасова відсутність даних не повинні призводити до аварійного завершення програми
9	Читабельність даних	Таблиці, картки показників, графіки та повідомлення мають бути візуально зрозумілими

З урахуванням нефункціональних вимог систему доцільно реалізувати як локальний настільний застосунок із графічним інтерфейсом. Такий підхід спрощує встановлення, не потребує обов’язкового серверного середовища та

забезпечує автономну роботу в умовах невеликої або навчально-дослідної теплиці.

Технічні вимоги визначають програмні засоби, формат збереження даних, принципи роботи з датчиками та обмеження щодо середовища виконання. Основні технічні вимоги наведено у таблиці 1.5.

Таблиця 1.5

Технічні вимоги до програмного забезпечення

№	Технічна вимога	Опис
1	Мова програмування	Для реалізації системи використовується Python 3
2	Графічний інтерфейс	Інтерфейс користувача реалізується з використанням PyQt6
3	База даних	Для локального збереження даних використовується SQLite
4	Структура даних	База даних повинна містити таблиці користувачів, теплиць, датчиків, типів датчиків, вимірювань, порогів, сповіщень і звітів
5	Формат експорту	Дані вимірювань і звітів експортуються у CSV
6	Періодичність оновлення	Система повинна підтримувати ручне та автоматичне оновлення показників
7	Діапазони параметрів	Температура, вологість і освітленість повинні перевірятися відповідно до заданих мінімальних і максимальних меж
8	Робота з датчиками	На етапі прототипу допускається програмна імітація датчиків із можливістю подальшого підключення реального контролера
9	Зв’язок із контролером	Для подальшого розвитку системи передбачається можливість обміну через USB / Serial
10	Журналювання	Основні події, сповіщення та результати перевірки повинні фіксуватися у базі даних

Технічна реалізація на основі Python, PyQt6 і SQLite відповідає поставленій задачі, оскільки забезпечує швидке створення настільного застосунку, локальне збереження даних і достатню гнучкість для навчального

або дослідного використання. SQLite є доцільним вибором для локальної системи, оскільки не потребує окремого серверного встановлення та забезпечує збереження структурованих даних у файлі бази даних.

Оскільки система передбачає роботу з обліковими записами користувачів, журналом вимірювань і налаштуваннями порогів, необхідно врахувати вимоги до безпеки. Перелік таких вимог наведено у таблиці 1.6.

Таблиця 1.6

Вимоги до безпеки програмної системи

№	Вимога	Опис
1	Автентифікація	Доступ до системи повинен надаватися лише після введення логіна та пароля
2	Захист паролів	Паролі користувачів мають зберігатися не у відкритому вигляді, а у вигляді хешу
3	Розмежування ролей	Система повинна передбачати ролі користувача та адміністратора
4	Контроль введення	Введені значення порогів, дат і параметрів повинні перевірятися на коректність
5	Захист від помилкових налаштувань	Мінімальний поріг не повинен перевищувати максимальний поріг
6	Цілісність даних	Зв’язки між датчиками, вимірюваннями, сповіщеннями і звітами повинні підтримуватися засобами бази даних
7	Збереження історії	Вимірювання та сповіщення не повинні видалятися автоматично без дії користувача
8	Стійкість до збоїв	У разі некоректного введення або відсутності даних система повинна виводити повідомлення, а не завершувати роботу

Вимоги до безпеки є важливими навіть для локальної системи, оскільки вона містить налаштування порогових значень і журнал роботи теплиці. Помилкове редагування порогів або некоректне збереження даних може призвести до неправильного формування сповіщень. Тому система повинна виконувати перевірку введених даних, контролювати права доступу та забезпечувати коректність записів у базі даних.

На основі проведеного аналізу можна зробити висновок, що розроблюване програмне забезпечення має бути локальним, модульним і зручним у використанні. Система повинна забезпечувати моніторинг основних параметрів мікроклімату теплиці, збереження історії вимірювань, побудову графіків, контроль порогових значень, формування сповіщень і звітів. Визначені функціональні, нефункціональні, технічні та безпекові вимоги створюють основу для подальшого проєктування інформаційного забезпечення, побудови UML-моделей, фізичної структури бази даних і програмної реалізації застосунку.

1.5. Постановка завдання

У межах кваліфікаційної роботи необхідно розробити програмне забезпечення системи моніторингу параметрів мікроклімату у теплиці. Система повинна забезпечувати роботу з користувачами, приймання або імітацію даних датчиків, збереження вимірювань, перевірку порогових значень, формування сповіщень, візуалізацію показників та створення звітів.

Вхідними даними системи є відомості про користувачів, перелік теплиць, довідник типів датчиків, значення вимірювань, порогові налаштування та запити користувача щодо перегляду історії або формування звіту. До вимірювань належать температура повітря у градусах Цельсія, вологість повітря у відсотках, вологість ґрунту у відсотках та освітленість у люксах.

Вихідними даними є поточний стан теплиці, історія вимірювань, записи про сповіщення, CSV-звіти, графіки динаміки показників та службові повідомлення про результати виконання операцій. Усі основні інформаційні об’єкти мають зберігатися у структурованій базі даних SQLite.

Програмна система повинна мати модульну структуру. До складу програмного забезпечення мають входити модуль графічного інтерфейсу, модуль моніторингу, модуль збору даних, модуль роботи з датчиками, модуль

сповіщень, модуль звітності, шар доступу до даних і модуль налаштувань. Така структура забезпечує зрозумілість реалізації та можливість подальшого підключення реального апаратного контролера.

Поставлене завдання полягає у створенні повноцінної локальної програмної моделі, яка відтворює основні процеси автоматизованого моніторингу мікроклімату теплиці та може бути використана для навчальної демонстрації, тестування алгоритмів контролю порогів і підготовки до інтеграції з реальними сенсорними вузлами.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Логічна модель даних визначає склад інформаційних сутностей, які використовуються у програмному забезпеченні, та зв’язки між ними. Для системи моніторингу параметрів мікроклімату у теплиці логічна модель має охоплювати користувачів, теплиці, типи датчиків, датчики, вимірювання, порогові налаштування, сповіщення та звіти.

На рисунку 2.1 наведено ER-діаграму логічної моделі даних. Вона побудована за принципом мінімально необхідної структури, без зайвого ускладнення, але з урахуванням усіх основних функцій системи.

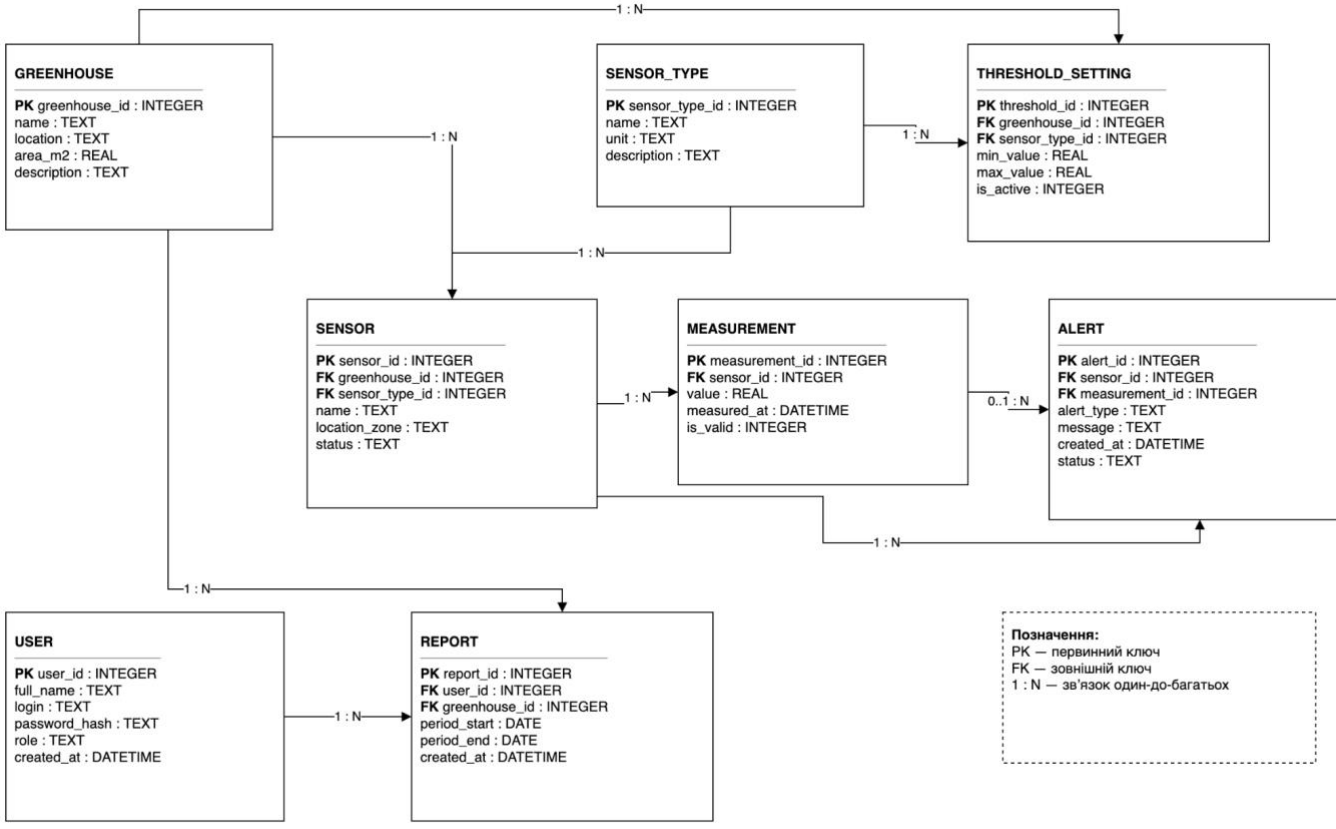


Рис. 2.1. ER-діаграма логічної моделі даних системи моніторингу мікроклімату теплиці

Сутність GREENHOUSE описує теплицю як об’єкт моніторингу. Вона містить назву, місце розташування, площу та опис. Одна теплиця може мати багато датчиків, багато порогових налаштувань і багато звітів.

Сутність SENSOR_TYPE є довідником типів контрольованих показників. Для кожного типу визначається назва, одиниця виміру та опис. Такий підхід унеможливлює дублювання однакових назв показників у таблицях датчиків, вимірювань і порогів.

Сутність SENSOR описує конкретний датчик. Вона пов’язана з теплицею і типом датчика, містить назву, зону розміщення та статус. Один датчик може формувати багато записів вимірювань і багато сповіщень.

Сутність MEASUREMENT зберігає фактичні значення, отримані від датчиків. Для кожного запису фіксується ідентифікатор датчика, значення, дата і час вимірювання та ознака коректності. Саме ця сутність є основою для побудови графіків і звітів.

Сутність THRESHOLD_SETTING визначає допустимі мінімальні та максимальні межі для кожного типу показника у конкретній теплиці. Сутність ALERT зберігає повідомлення про відхилення, а REPORT фіксує сформовані звіти за періоди.

Таблиця 2.1

Сутності логічної моделі даних

Сутність	Призначення
USER	Збереження даних користувачів і ролей доступу.
GREENHOUSE	Опис теплиці як об’єкта моніторингу.
SENSOR_TYPE	Довідник типів показників і одиниць виміру.
SENSOR	Опис фізичних або імітованих датчиків.
MEASUREMENT	Збереження історії вимірювань.
THRESHOLD_SETTING	Збереження допустимих меж показників.
ALERT	Фіксація сповіщень про відхилення параметрів.
REPORT	Збереження інформації про сформовані звіти.

Логічна модель даних відповідає вимогам предметної області та забезпечує цілісне збереження даних про теплицю, датчики, вимірювання, пороги, сповіщення і звіти. Наявність зв’язків один-до-багатьох дозволяє

розширювати систему шляхом додавання нових датчиків, теплиць і користувачів.

2.2 Діаграма класів і кооперації

Для опису програмної структури системи побудовано діаграму класів. Вона відображає ключові об’єкти програмної реалізації, їх атрибути, операції та зв’язки. Діаграма класів наведена на рисунку 2.2.

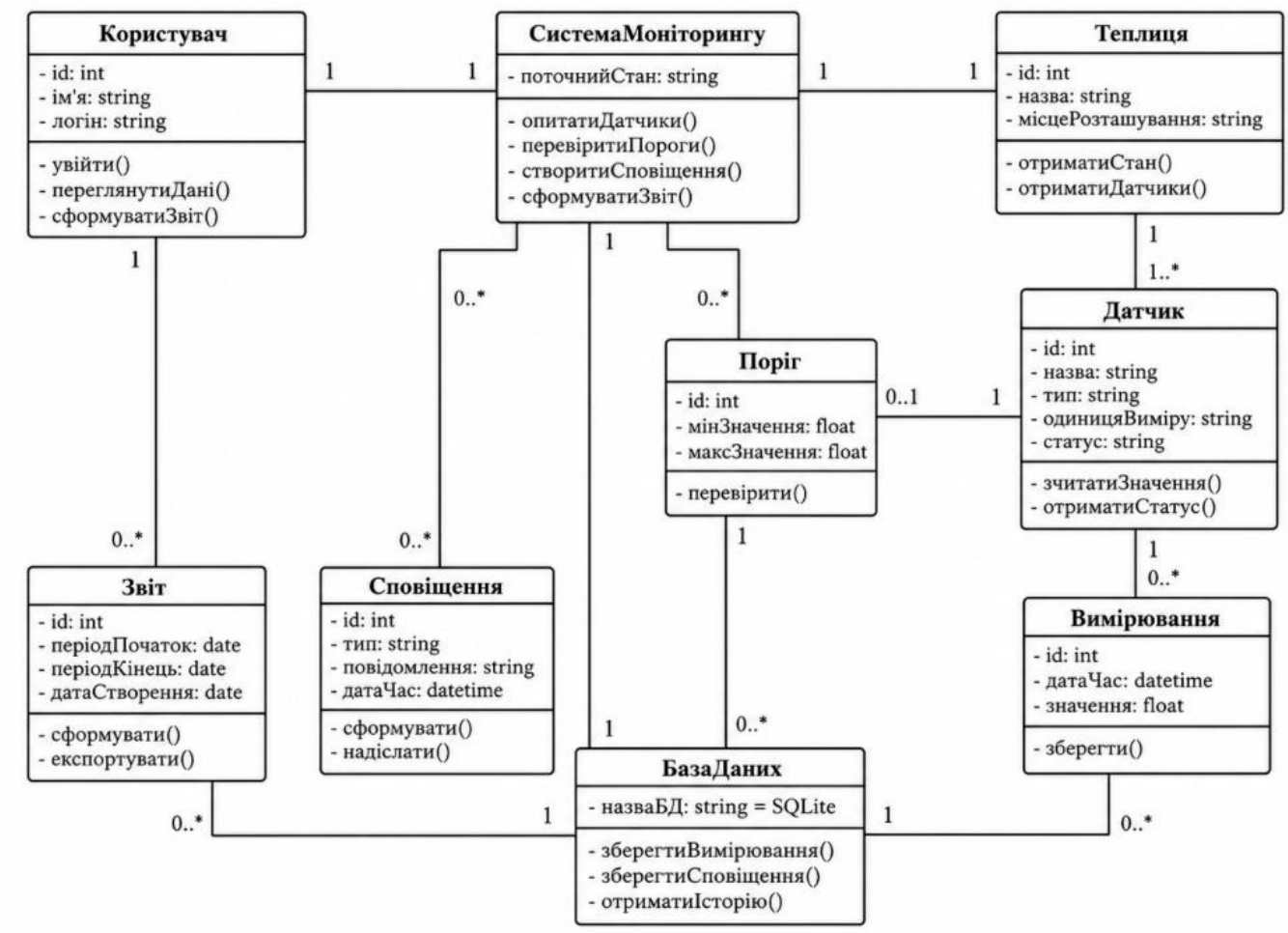


Рис. 2.2. Діаграма класів системи моніторингу параметрів мікроклімату у теплиці

Центральним класом моделі є СистемаМоніторингу. Він координує процес опитування датчиків, перевірки порогів, створення сповіщень та формування звітів. Клас Користувач описує особу, яка працює з програмою та виконує перегляд даних або формування звіту.

Клас Теплиця визначає об’єкт моніторингу і пов’язаний із класом Датчик. Кожний датчик має тип, одиницю виміру, статус і метод зчитування значення. Клас Вимірювання описує окремий запис даних, який зберігається у базі. Клас Поріг використовується для перевірки допустимих меж, а клас Сповіщення - для формування повідомлення про відхилення.

Клас БазаДаних відповідає за збереження вимірювань, сповіщень і отримання історичних даних. Такий розподіл класів відповідає модульному підходу та спрощує подальше програмування системи.

Діаграма кооперації деталізує взаємодію об’єктів у сценарії контролю параметрів мікроклімату. Вона наведена на рисунку 2.3.

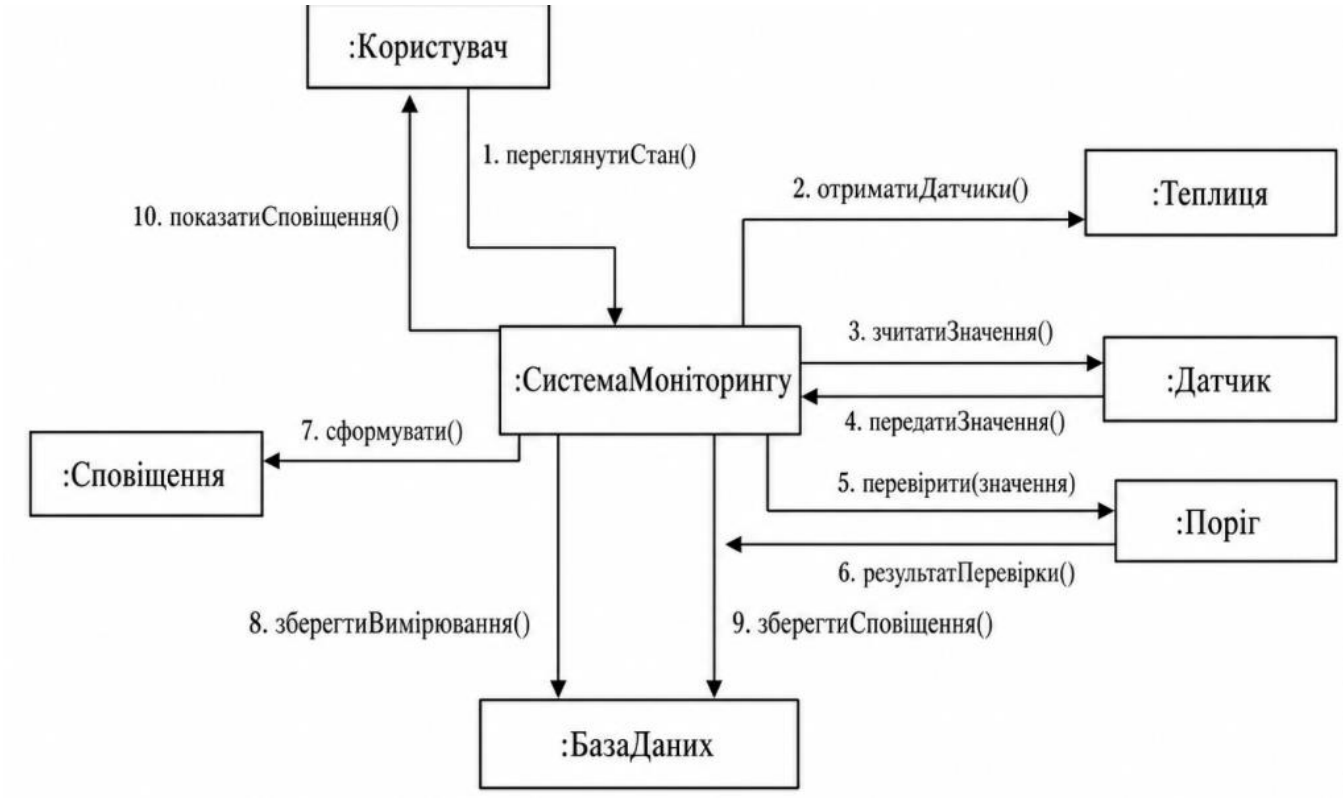


Рис. 2.3. Діаграма кооперації для сценарію контролю параметрів мікроклімату

У межах сценарію користувач ініціює перегляд стану теплиці, система отримує перелік датчиків, зчитує значення, перевіряє їх за порогами, формує сповіщення у разі відхилення та зберігає вимірювання і сповіщення у базі даних. Нумерація повідомлень на діаграмі відображає порядок виконання

операцій і дозволяє простежити логіку роботи системи без деталізації внутрішнього коду кожного класу.

2.3 Діаграма компонентів

Компонентна діаграма відображає склад програмного забезпечення з позиції великих функціональних блоків. Вона дає змогу показати, які модулі входять до системи, як вони взаємодіють між собою та які зовнішні джерела даних використовуються.

На рисунку 2.4 подано діаграму компонентів системи моніторингу параметрів мікроклімату у теплиці.

Рис. 2.4. Діаграма компонентів програмного забезпечення системи моніторингу

Графічний інтерфейс, реалізований на PyQt6, забезпечує взаємодію користувача з системою. Через нього користувач переглядає панель моніторингу, таблиці вимірювань, сповіщення, порогові значення, звіти та довідник датчиків.

Модуль моніторингу виконує основну прикладну логіку: ініціює опитування датчиків, приймає поточні значення, порівнює їх із порогами та передає інформацію у модулі візуалізації і сповіщень. Модуль збору даних відповідає за приймання показників від імітатора або потенційного апаратного контролера.

Шар доступу до даних ізолює інші модулі від прямих SQL-запитів і забезпечує роботу з базою даних SQLite. Така структура підвищує підтримуваність програмного коду та дозволяє у майбутньому замінити SQLite на іншу СКБД без повної переробки інтерфейсу.

2.4 Діаграма пакетів

Пакетна структура програмного забезпечення відображає організацію файлів і модулів проєкту. Для розроблюваної системи обрано розподіл на пакети ui, controllers, services, models, data, hardware та utils. Діаграма пакетів наведена на рисунку 2.5.

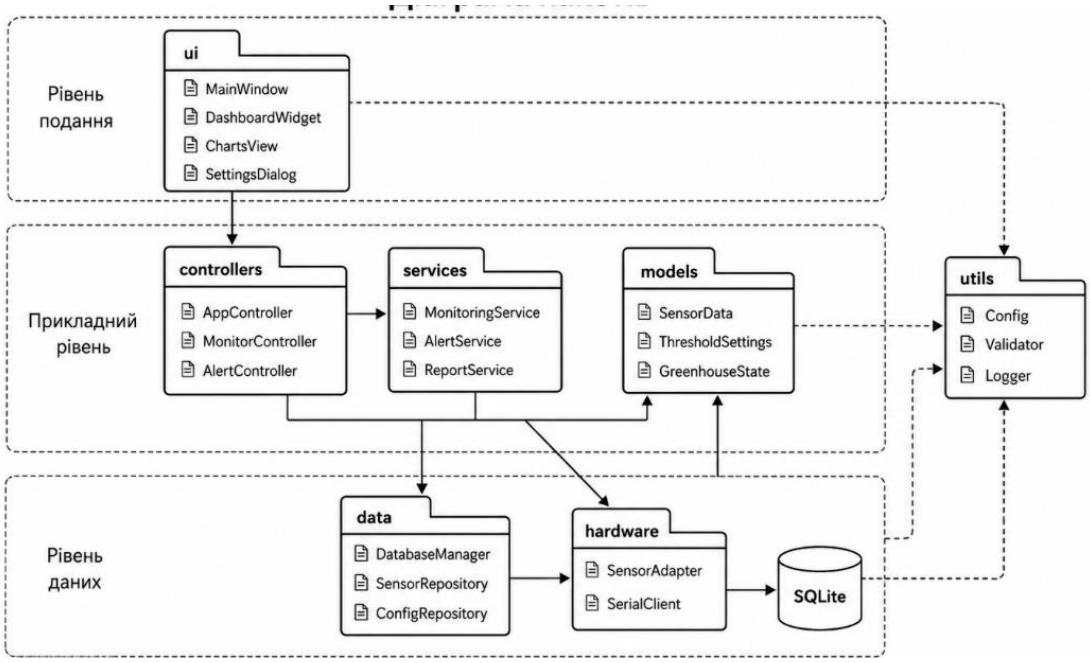


Рис. 2.5. Діаграма пакетів програмного забезпечення системи моніторингу

Пакет ui містить класи графічного інтерфейсу: головне вікно, панель моніторингу, перегляд графіків та діалог налаштувань. Пакет controllers координує взаємодію між інтерфейсом і прикладними сервісами. Пакет services містить основну бізнес-логіку: моніторинг, формування сповіщень і звітів.

Пакет models описує доменні об’єкти: дані датчика, порогові налаштування та стан теплиці. Пакет data відповідає за роботу з SQLite, репозиторіями датчиків і конфігурацій. Пакет hardware містить адаптер датчиків і клієнт послідовного порту, що дає змогу підготувати систему до підключення реального апаратного модуля. Пакет utils включає допоміжні засоби конфігурації, валідації та журналювання.

Запропонована пакетна структура є компактною і водночас достатньою для подальшого розвитку системи. Вона забезпечує відокремлення інтерфейсу, прикладної логіки, доменної моделі, доступу до даних та апаратної взаємодії.

2.5 Висновки до другого розділу

У другому розділі виконано проєктування інформаційного та програмного забезпечення системи моніторингу параметрів мікроклімату у теплиці. Розроблено логічну ER-модель даних, яка містить сутності користувачів, теплиць, типів датчиків, датчиків, вимірювань, порогових налаштувань, сповіщень і звітів.

Побудовано діаграму класів, що відображає основні програмні об’єкти системи, та діаграму кооперації для сценарію контролю параметрів мікроклімату. Визначено компонентну структуру програмного забезпечення, у якій виділено графічний інтерфейс, модуль моніторингу, модуль збору даних, модуль візуалізації, модуль сповіщень, модуль налаштувань, шар доступу до даних і базу SQLite.

Також сформовано пакетну структуру проєкту, яка відповідає реалізації на Python і PyQt6. Отримані моделі є основою для переходу до етапу вибору технологій, розроблення фізичної моделі даних, архітектури розгортання та програмної реалізації.

3 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ПРОГРАМНОЇ СИСТЕМИ ТА РЕАЛІЗАЦІЯ

3.1 Обґрунтування вибору технологічних засобів та інструментів реалізації

Для реалізації програмного забезпечення системи моніторингу параметрів мікроклімату у теплиці обрано технологічний стек Python 3, PyQt6 та SQLite. Такий вибір зумовлений необхідністю створення локального настільного застосунку з графічним інтерфейсом, простою інсталяцією, підтримкою роботи з базою даних і можливістю подальшої інтеграції з апаратними датчиками.

Python 3 обрано як основну мову програмування завдяки простому синтаксису, великій кількості бібліотек, підтримці роботи з базами даних, файлами CSV, графіками та апаратними інтерфейсами. Для навчально-дослідної системи така мова є доцільною, оскільки дозволяє швидко реалізувати прототип і водночас підтримувати достатню якість структури програмного коду.

PyQt6 використано для побудови графічного інтерфейсу користувача. Бібліотека дає змогу створювати повноцінні настільні застосунки з вкладками, таблицями, кнопками, формами, діалогами та графічними областями. Для розроблюваної системи це дозволило реалізувати панель моніторингу, журнал вимірювань, вкладку сповіщень, налаштування порогів, формування звітів і довідник датчиків.

SQLite обрано як локальну СКБД, що не потребує окремого серверного процесу. Усі дані системи зберігаються у файловій базі, яка створюється автоматично під час першого запуску програми. Такий підхід спрощує розгортання, переносимість і тестування системи.

Таблиця 3.1

Технології створення програмного забезпечення

Технологія	Призначення у системі	Причина вибору

Python 3	Реалізація основної логіки програми	Простота розроблення, підтримка модульності, наявність бібліотек

Продовження таблиці 3.1

PyQt6	Створення графічного інтерфейсу	Підтримка настільних вікон, вкладок, таблиць і форм
SQLite	Локальне збереження даних	Файлова база даних без окремого сервера
CSV	Експорт вимірювань і звітів	Зручність відкриття у табличних редакторах
draw.io	Побудова UML та ER-діаграм	Зручне створення технічних схем і діаграм

Обраний стек є достатнім для реалізації поставленого завдання та відповідає вимогам до локальної системи моніторингу. У майбутньому архітектуру можна доповнити мікроконтролером ESP32 або Arduino, бібліотекою pyserial для обміну через послідовний порт, а також мережевим модулем для віддаленого перегляду показників.

3.2 Фізична модель даних

Фізична модель даних деталізує логічну ER-модель з урахуванням конкретної СКБД SQLite. У фізичній моделі визначено назви таблиць, поля, типи даних, первинні та зовнішні ключі, а також основні зв’язки між таблицями.

На рисунку 3.1 наведено фізичну модель даних системи моніторингу параметрів мікроклімату у теплиці.

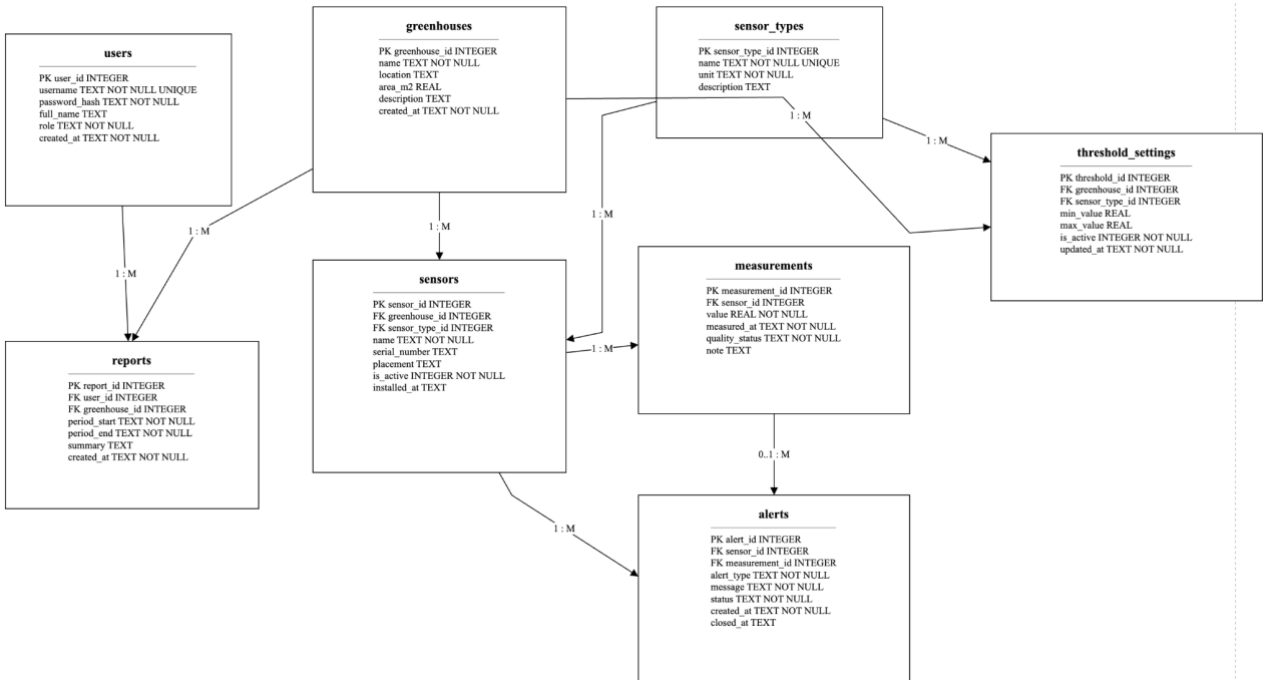


Рис. 3.1. Фізична модель даних програмного забезпечення системи моніторингу

Таблиця users використовується для збереження облікових записів користувачів. Поле username має бути унікальним, а пароль зберігається у вигляді хешу. Таблиця greenhouses містить інформацію про теплиці, а sensor_types - довідник показників з одиницями виміру.

Таблиця sensors пов’язує конкретні датчики з теплицею та типом показника. Таблиця measurements зберігає всі вимірювання і містить посилання на датчик, значення, дату та час, ознаку якості й примітку. Таблиця threshold_settings містить допустимі межі для показників, а alerts - записи про порушення порогів.

Таблиця reports фіксує сформовані звіти за обрані періоди. Зовнішні ключі забезпечують зв’язок між користувачами, теплицями, датчиками, вимірюваннями, сповіщеннями та звітами. Така структура є компактною, але повністю забезпечує потреби розроблюваної системи.

Таблиця 3.2

Сутності фізичної моделі даних

Таблиця	Основні поля	Призначення
users	user_id, username, password_hash, full_name, role	Облік користувачів системи

greenhouses	greenhouse_id, name, location, area_m2	Опис теплиць
sensor_types	sensor_type_id, name, unit	Довідник типів датчиків

Продовження таблиці 3.2

sensors	sensor_id, greenhouse_id, sensor_type_id, name, status	Перелік датчиків
measurements	measurement_id, sensor_id, value, measured_at, quality_status	Історія вимірювань
threshold_settings	threshold_id, greenhouse_id, sensor_type_id, min_value, max_value	Порогові значення
alerts	alert_id, sensor_id, measurement_id, alert_type, message, status	Сповіщення про відхилення
reports	report_id, user_id, greenhouse_id, period_start, period_end	Звіти за періоди

3.3 Архітектура та розгортання системи

Архітектура програмної системи побудована як локальний настільний застосунок з окремими логічними шарами. На рівні подання працює графічний інтерфейс PyQt6, на прикладному рівні - сервіси моніторингу, сповіщень, звітності та керування датчиками, а на рівні даних - SQLite і репозиторії доступу до таблиць.

Діаграму розгортання системи наведено на рисунку 3.2. Вона відображає взаємодію робочої станції користувача, локального застосунку, бази даних, контролера збору даних і датчиків.

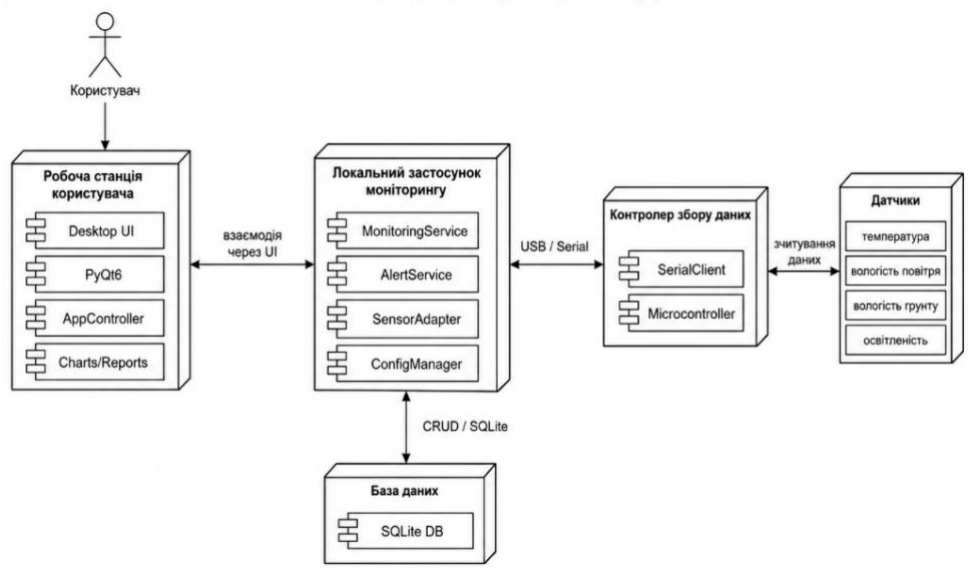


Рис. 3.2. Діаграма розгортання системи моніторингу параметрів мікроклімату у теплиці

Робоча станція користувача містить настільний інтерфейс, бібліотеку PyQt6, контролер застосунку та модулі графіків і звітів. Локальний застосунок моніторингу реалізує сервіси MonitoringService, AlertService, SensorAdapter та ConfigManager. База даних SQLite зберігається локально і взаємодіє із застосунком через CRUD-операції.

Контролер збору даних у межах поточної реалізації може бути представлений програмним імітатором датчиків. У подальшому він може бути замінений на реальний мікроконтролерний модуль, який передаватиме дані через USB/Serial. Така схема розгортання не потребує складної серверної інфраструктури й підходить для локального використання у навчальній або експериментальній теплиці.

3.4 Алгоритмізація програмних модулів системи

Алгоритмічне забезпечення системи охоплює процеси авторизації, опитування датчиків, збереження вимірювань, перевірки порогових значень, формування сповіщень, відображення графіка та створення звіту. Основним є алгоритм циклу моніторингу.

Після запуску програми користувач проходить авторизацію. Якщо логін і пароль коректні, відкривається головне вікно з вкладками. Система ініціалізує базу даних, завантажує довідники теплиці, датчиків і порогів, після чого виконує перше опитування датчиків. У режимі імітації генеруються значення температури, вологості повітря, вологості ґрунту та освітленості.

Кожне значення записується до таблиці measurements. Після збереження система отримує порогові межі для відповідного типу показника і виконує порівняння. Якщо значення менше мінімального або більше максимального порога, створюється запис у таблиці alerts. Поточний стан відображається на

панелі моніторингу, а у разі активних сповіщень користувач бачить відповідне повідомлення у нижній частині вікна.

Таблиця 3.3

Модулі системи з використанням алгоритмів

Модуль	Основний алгоритм	Результат роботи
AuthController	Перевірка логіна і хешу пароля	Вхід користувача або повідомлення про помилку
SensorAdapter	Отримання або імітація показників датчиків	Набір значень з часовою міткою
MonitoringService	Збереження вимірювань і перевірка порогів	Оновлений стан теплиці
AlertService	Формування записів про відхилення	Активне або опрацьоване сповіщення
ReportService	Вибір даних за період і експорт CSV	Файл звіту та запис у БД
DatabaseManager	Виконання SQL-запитів	Цілісне збереження даних

Алгоритми реалізовано так, щоб основні модулі не залежали безпосередньо від графічного інтерфейсу. Це спрощує тестування і дозволяє надалі замінити імітатор датчиків на реальний апаратний модуль без суттєвих змін у структурі програми.

3.5 Висновки до третього розділу

У третьому розділі обґрунтовано вибір технологічного стеку Python 3, PyQt6 та SQLite для реалізації локального настільного застосунку моніторингу мікроклімату теплиці. Розроблено фізичну модель даних, яка деталізує таблиці, поля та зв’язки для збереження користувачів, теплиць, датчиків, вимірювань, порогів, сповіщень і звітів.

Описано архітектуру та схему розгортання системи, де виділено робочу станцію користувача, локальний застосунок, базу даних, контролер збору даних і датчики. Сформовано алгоритми роботи основних модулів:

авторизації, збору даних, моніторингу, сповіщень, звітності та доступу до бази даних.

Реалізовано користувацький інтерфейс з основними вкладками, що забезпечують моніторинг, перегляд історії, роботу зі сповіщеннями, налаштування порогів, формування звітів і керування датчиками. Отримане програмне забезпечення є завершеною локальною системою, придатною для подальшого тестування.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ

4.1 План тестування програмних модулів та методика оцінювання результатів

Тестування програмного забезпечення системи моніторингу параметрів мікроклімату у теплиці є необхідним етапом перевірки працездатності розробленого застосунку. Основна мета тестування полягає у встановленні відповідності програмної системи сформованим функціональним, нефункціональним і технічним вимогам, а також у перевірці стабільності роботи окремих модулів під час виконання типових сценаріїв користувача.

Розроблена система реалізована як локальний настільний застосунок із графічним інтерфейсом PyQt6 та базою даних SQLite. Тому тестування повинно охоплювати не лише перевірку інтерфейсу, а й коректність роботи збереження вимірювань, формування сповіщень, налаштування порогових значень, побудову графіків та створення звітів. Особлива увага приділяється перевірці взаємодії між модулем моніторингу, адаптером датчиків, сервісом сповіщень і шаром доступу до бази даних.

План тестування передбачає виконання функціональних, інтеграційних, інтерфейсних і приймальних перевірок. Функціональне тестування спрямоване на перевірку окремих можливостей системи: авторизації, реєстрації, опитування датчиків, збереження вимірювань, роботи порогів, створення сповіщень і формування звітів. Інтеграційне тестування дає змогу оцінити правильність обміну даними між графічним інтерфейсом, сервісами оброблення, базою даних і модулем імітації датчиків. Інтерфейсне тестування перевіряє зручність роботи з вкладками, таблицями, кнопками, формами введення та графіками. Приймальне тестування підтверджує, що система виконує основні сценарії, потрібні оператору теплиці

Таблиця 4.1

План тестування програмних модулів системи

Модуль системи	Що перевіряється	Очікуваний результат
Модуль авторизації	Вхід із тестовими даними admin/admin, введення неправильного пароля, реєстрація нового користувача	Користувач входить до системи за правильними даними, а в разі помилки отримує відповідне повідомлення
Панель моніторингу	Відображення поточних значень температури, вологості повітря, вологості ґрунту та освітленості	Поточні показники відображаються у відповідних картках без помилок і затримок
Модуль опитування датчиків	Ручне та автоматичне отримання нових значень параметрів мікроклімату	Нові вимірювання створюються та передаються до модуля оброблення
Модуль вимірювань	Перегляд історії вимірювань, фільтрація за датою та типом показника	Таблиця коректно відображає записи за вибраними параметрами
Модуль графіків	Побудова графіка динаміки вибраного параметра	Графік оновлюється відповідно до збережених вимірювань
Модуль сповіщень	Формування попередження при виході показника за межі встановленого порога	У таблиці сповіщень з’являється новий запис із датою, показником, значенням і текстом повідомлення
Модуль порогових значень	Зміна мінімальних і максимальних допустимих значень параметрів	Нові пороги зберігаються у базі даних і використовуються під час наступної перевірки
Модуль звітів	Формування звіту за вибраний період та експорт у CSV	Створюється файл звіту, а інформація про нього зберігається в таблиці reports
Модуль керування датчиками	Додавання нового датчика, перегляд списку датчиків і їх статусів	Новий датчик з’являється у таблиці та може використовуватися системою
База даних SQLite	Збереження користувачів, датчиків, вимірювань, порогів, сповіщень і звітів	Дані записуються без втрат, зв’язки між таблицями залишаються коректними

Методика оцінювання результатів тестування базується на порівнянні фактичної поведінки системи з очікуваними результатами. Для кожної перевірки визначається вхідна дія, очікуваний результат і критерій успішності.

Тест вважається пройденим, якщо система виконує дію без аварійного завершення, коректно оновлює інтерфейс, правильно записує дані до бази та формує зрозуміле повідомлення у разі помилки або відхилення показника.

Таблиця 4.2

Методика оцінювання результатів тестування

Критерій оцінювання	Метод перевірки	Допустимий результат
Коректність авторизації	Виконання входу з правильними та неправильними обліковими даними	Правильні дані забезпечують вхід, неправильні дані викликають повідомлення про помилку
Коректність збереження вимірювань	Порівняння записів у таблиці вимірювань після опитування датчиків	Кожне вимірювання має дату, датчик, показник, значення, одиницю вимірювання та ознаку якості
Робота порогового контролю	Зміна порогів і повторне опитування датчиків	При виході значення за межі норми створюється сповіщення
Стабільність інтерфейсу	Багаторазове перемикання вкладок, оновлення таблиць і запуск опитування	Інтерфейс не зависає, елементи керування залишаються доступними
Коректність фільтрації даних	Вибір періоду та типу показника у вкладці вимірювань	У таблиці відображаються лише записи, що відповідають вибраному фільтру
Правильність експорту CSV	Формування файлу з вимірюваннями або звітом за період	CSV-файл створюється та містить структуровані рядки з даними
Робота модуля сповіщень	Формування активного сповіщення та позначення його як опрацьованого	Статус сповіщення змінюється, а запис залишається в історії
Цілісність бази даних	Перевірка зв’язків між таблицями sensors, measurements, alerts і reports	Дані не дублюються без потреби, записи пов’язані з відповідними датчиками та користувачами
Зручність використання	Оцінювання доступності основних функцій через вкладки інтерфейсу	Користувач може швидко знайти моніторинг, вимірювання, сповіщення, пороги, звіти та датчики

Для проведення тестування використовується тестовий обліковий запис адміністратора, набір стандартних датчиків і попередньо задані порогові значення. Перевірка виконується у звичайному режимі роботи застосунку: користувач входить у систему, запускає опитування датчиків, переглядає

оновлені показники, аналізує історію вимірювань, змінює пороги, перевіряє появу сповіщень і формує звіт за вибраний період. Такий підхід дає змогу оцінити систему не ізольовано, а в умовах реального сценарію використання.

Оцінювання результатів тестування також передбачає перевірку реакції системи на некоректні дії користувача. До таких ситуацій належать введення неправильного пароля, спроба зберегти некоректні порогові значення, вибір періоду без вимірювань, формування звіту без доступних даних або додавання датчика з неповними полями. У таких випадках система повинна не завершувати роботу аварійно, а виводити зрозуміле повідомлення і залишатися працездатною.

Отже, запропонований план тестування охоплює основні програмні модулі системи моніторингу параметрів мікроклімату у теплиці та дозволяє перевірити її відповідність поставленим вимогам. Методика оцінювання результатів спрямована на підтвердження коректності збереження даних, стабільності графічного інтерфейсу, правильності роботи порогового контролю, формування сповіщень і створення звітів. Отримані результати тестування є основою для подальшого аналізу ефективності роботи системи та підготовки її до практичного використання.

4.2 Тестування інтерфейсних модулів системи

Користувацький інтерфейс реалізовано у вигляді настільного застосунку з вкладковою структурою. Основні вкладки відповідають ключовим функціям системи: панель моніторингу, вимірювання, сповіщення, пороги, звіти та датчики. Такий підхід дає змогу швидко переходити між поточним станом теплиці, історичними даними та налаштуваннями.

На рисунку 4.1 наведено екран входу до системи. Для тестування передбачено обліковий запис admin/admin, а також можливість створення нового користувача.

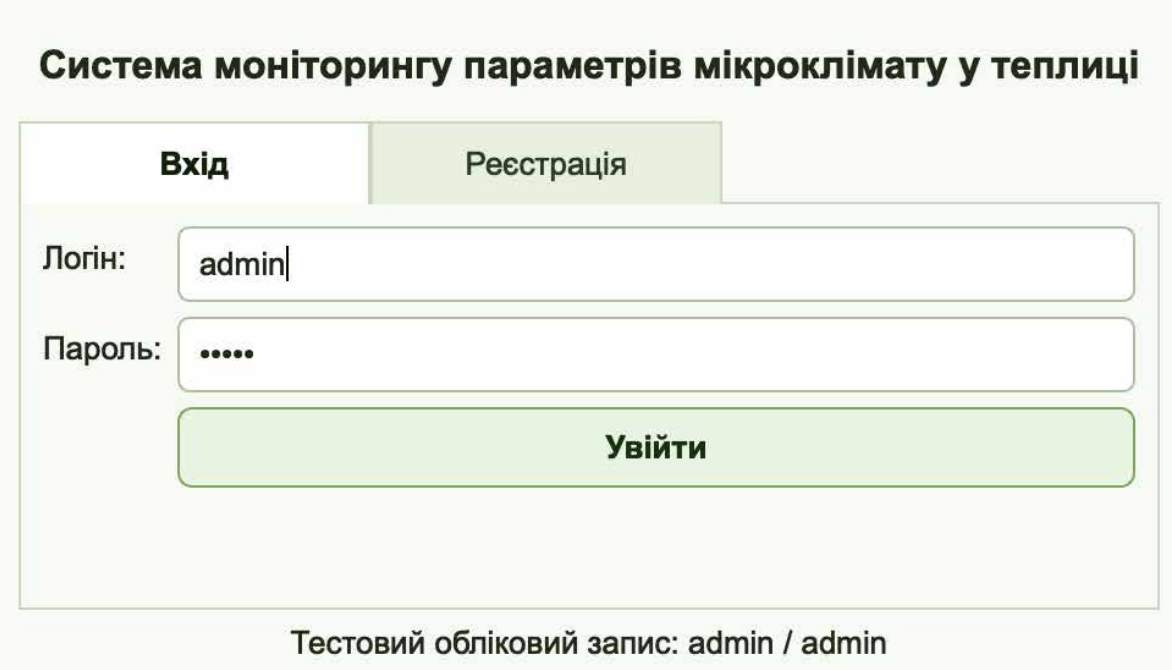


Рис. 4.1. Вікно авторизації користувача

Вкладка реєстрації нового користувача наведена на рисунку 4.2. Користувач вводить ПІБ, логін і пароль, після чого запис створюється у базі даних.

Рис. 4.3. Вікно реєстрації нового користувача

Головна панель моніторингу наведена на рисунку 4.4. Вона містить картки з поточними значеннями температури, вологості повітря, вологості ґрунту та освітленості, а також графік динаміки вибраного показника.

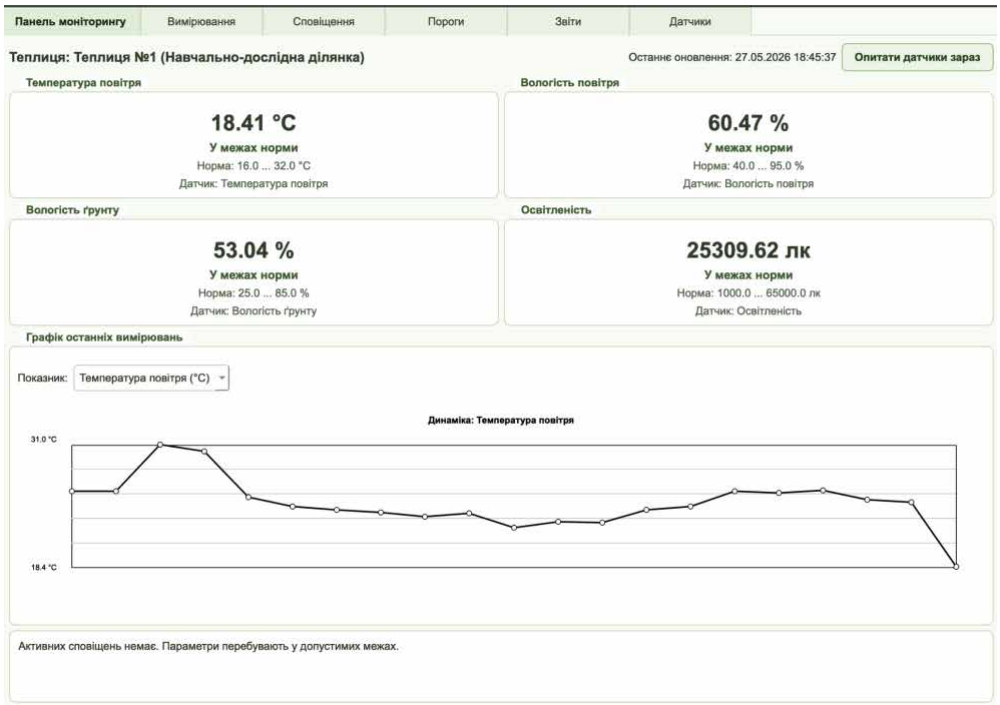


Рис. 4.4. Панель моніторингу параметрів мікроклімату у нормальному стані

Якщо один або кілька показників виходять за допустимі межі, система відображає активні сповіщення на панелі моніторингу. Приклад такого стану наведено на рисунку 4.5.

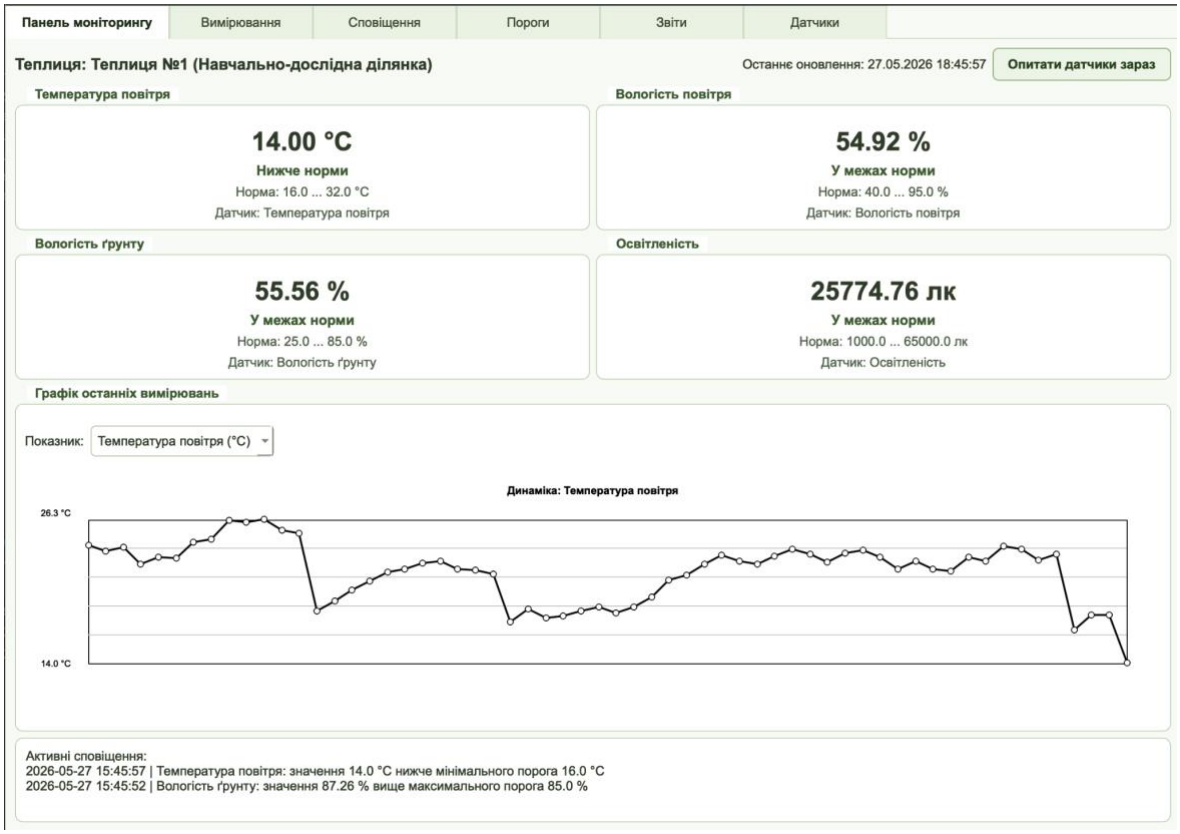
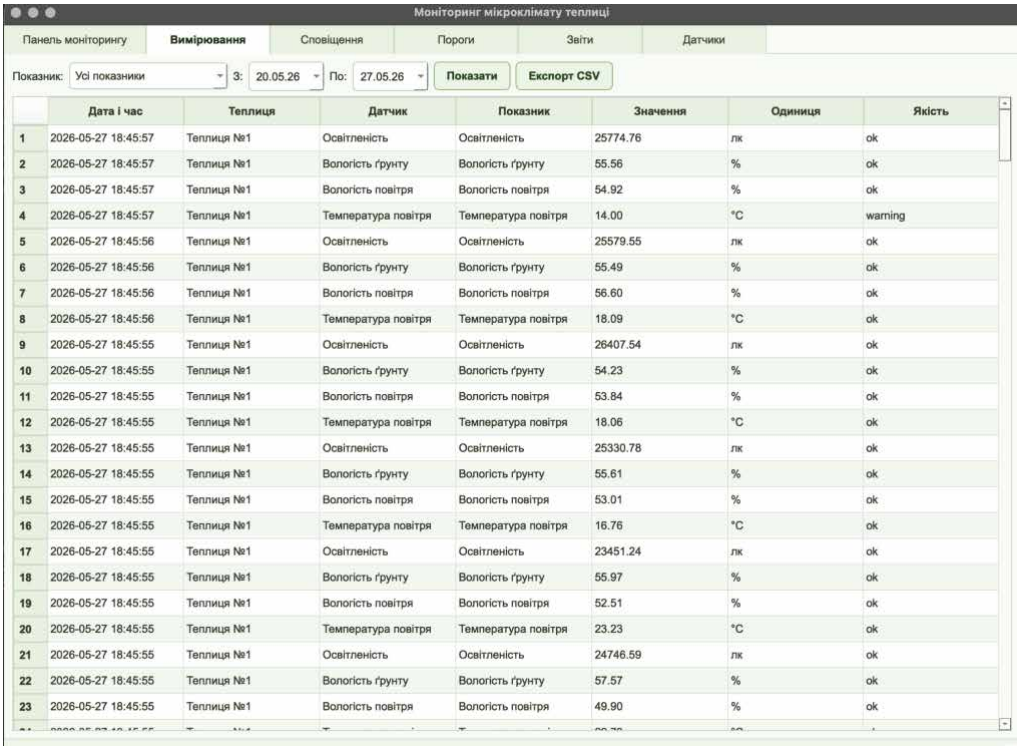


Рис. 4.5. Панель моніторингу при виявленні відхилення параметрів

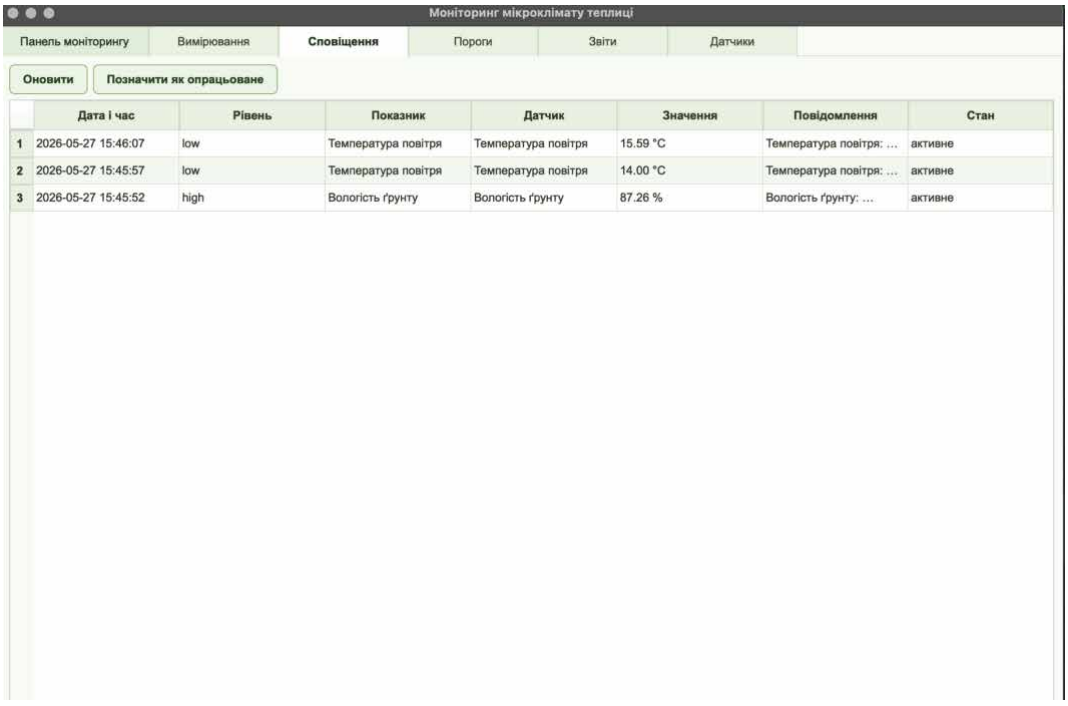
Вкладка вимірювань дає змогу переглядати історію даних за датами і типами показників. Приклад наведено на рисунку 4.6.



	Дата і час	Теплиця	Датчик	Показник	Значення	Одиниця	Якість
1	2026-05-27 18:45:57	Теплиця №1	Освітленість	Освітленість	25774.76	лк	ok
2	2026-05-27 18:45:57	Теплиця №1	Вологість ґрунту	Вологість ґрунту	55.56	%	ok
3	2026-05-27 18:45:57	Теплиця №1	Вологість повітря	Вологість повітря	54.92	%	ok
4	2026-05-27 18:45:57	Теплиця №1	Температура повітря	Температура повітря	14.00	°C	warning
5	2026-05-27 18:45:56	Теплиця №1	Освітленість	Освітленість	25579.55	лк	ok
6	2026-05-27 18:45:56	Теплиця №1	Вологість ґрунту	Вологість ґрунту	55.49	%	ok
7	2026-05-27 18:45:56	Теплиця №1	Вологість повітря	Вологість повітря	56.60	%	ok
8	2026-05-27 18:45:56	Теплиця №1	Температура повітря	Температура повітря	18.09	°C	ok
9	2026-05-27 18:45:55	Теплиця №1	Освітленість	Освітленість	26407.54	лк	ok
10	2026-05-27 18:45:55	Теплиця №1	Вологість ґрунту	Вологість ґрунту	54.23	%	ok
11	2026-05-27 18:45:55	Теплиця №1	Вологість повітря	Вологість повітря	53.84	%	ok
12	2026-05-27 18:45:55	Теплиця №1	Температура повітря	Температура повітря	18.06	°C	ok
13	2026-05-27 18:45:55	Теплиця №1	Освітленість	Освітленість	25330.78	лк	ok
14	2026-05-27 18:45:55	Теплиця №1	Вологість ґрунту	Вологість ґрунту	55.61	%	ok
15	2026-05-27 18:45:55	Теплиця №1	Вологість повітря	Вологість повітря	53.01	%	ok
16	2026-05-27 18:45:55	Теплиця №1	Температура повітря	Температура повітря	16.76	°C	ok
17	2026-05-27 18:45:55	Теплиця №1	Освітленість	Освітленість	23451.24	лк	ok
18	2026-05-27 18:45:55	Теплиця №1	Вологість ґрунту	Вологість ґрунту	55.97	%	ok
19	2026-05-27 18:45:55	Теплиця №1	Вологість повітря	Вологість повітря	52.51	%	ok
20	2026-05-27 18:45:55	Теплиця №1	Температура повітря	Температура повітря	23.23	°C	ok
21	2026-05-27 18:45:55	Теплиця №1	Освітленість	Освітленість	24746.59	лк	ok
22	2026-05-27 18:45:55	Теплиця №1	Вологість ґрунту	Вологість ґрунту	57.57	%	ok
23	2026-05-27 18:45:55	Теплиця №1	Вологість повітря	Вологість повітря	49.90	%	ok

Рис. 4.6. Вкладка перегляду історії вимірювань

Вкладка сповіщень, наведена на рисунку 4.7, містить записи про порушення порогових значень. Користувач може оновити список або позначити сповіщення як опрацьовані.



	Дата і час	Рівень	Показник	Датчик	Значення	Повідомлення	Стан
1	2026-05-27 15:46:07	low	Температура повітря	Температура повітря	15.59 °C	Температура повітря: ...	активне
2	2026-05-27 15:45:57	low	Температура повітря	Температура повітря	14.00 °C	Температура повітря: ...	активне
3	2026-05-27 15:45:52	high	Вологість ґрунту	Вологість ґрунту	87.26 %	Вологість ґрунту: ...	активне

Рис. 4.7. Вкладка активних сповіщень

На вкладці порогів користувач може переглядати та змінювати мінімальні й максимальні допустимі значення для кожного показника. Приклад наведено на рисунку 4.8.

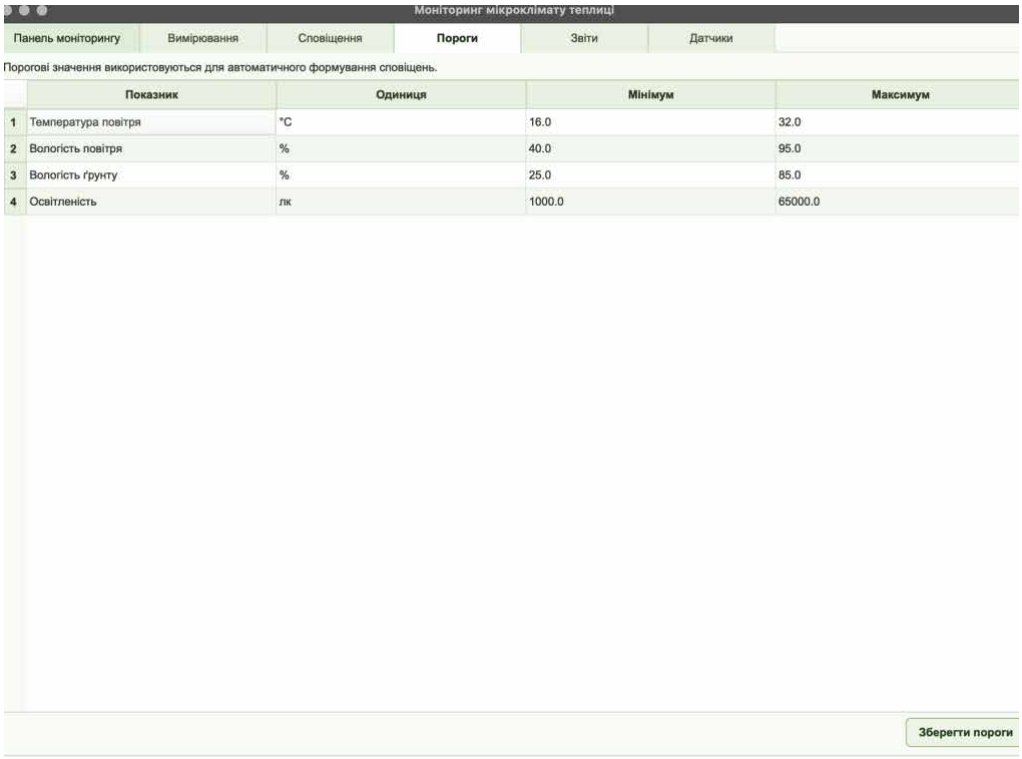


Рис. 4.8. Вкладка налаштування порогових значень

Вкладка звітів дозволяє сформувати CSV-звіт за обраний період. Інтерфейс формування звіту наведено на рисунку 4.9.

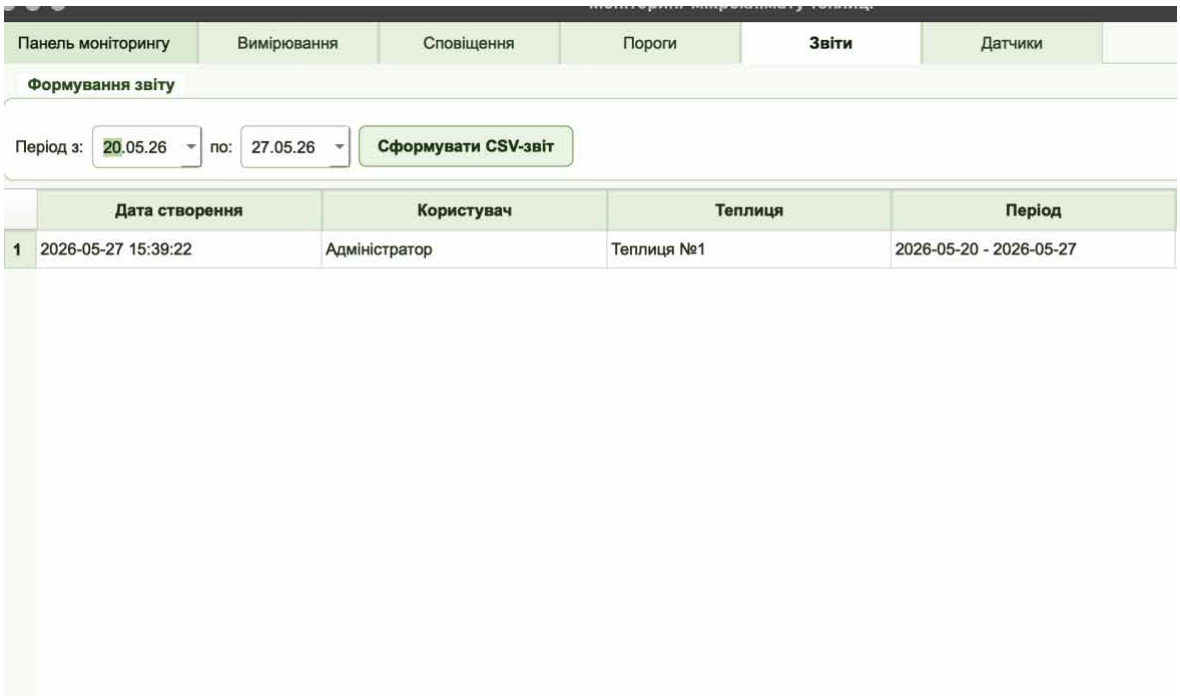


Рис. 4.9. Вкладка формування звітів

Вкладка датчиків використовується для перегляду наявних датчиків та додавання нових. Вона наведена на рисунку 3.11.

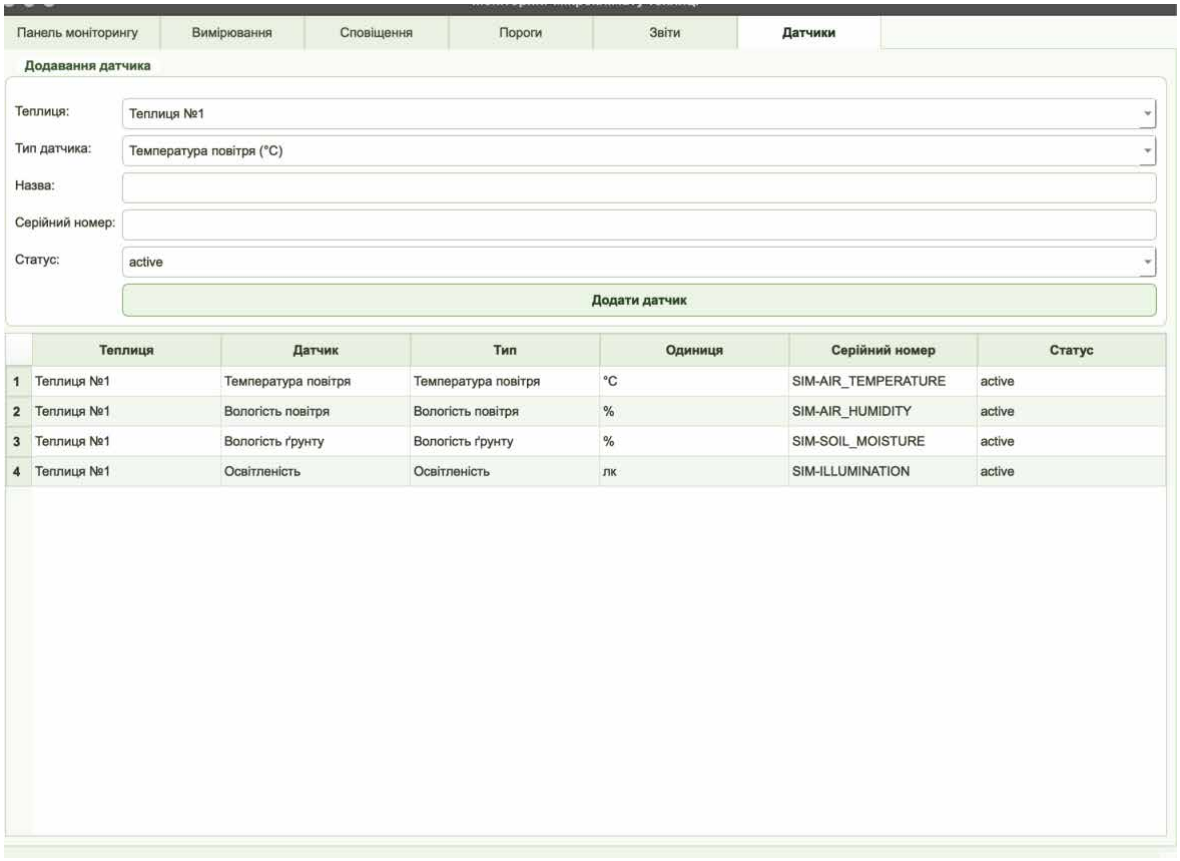


Рис. 4.10. Вкладка керування датчиками

Реалізований інтерфейс відповідає функціональним вимогам, має єдине стилістичне оформлення та забезпечує доступ до всіх основних функцій системи без додаткових вікон або складної навігації.

4.3 Результати тестування та аналіз ефективності системи

За результатами тестування встановлено, що розроблене програмне забезпечення системи моніторингу параметрів мікроклімату у теплиці виконує основні функції, визначені на етапі аналізу вимог. Система забезпечує авторизацію та реєстрацію користувачів, автоматичне створення локальної бази даних SQLite, опитування датчиків у режимі імітації, збереження вимірювань, перевірку порогових значень, формування сповіщень, перегляд історії, побудову графіків і створення CSV-звітів.

У процесі тестування не було виявлено критичних помилок, які блокують роботу основних модулів. Головне вікно системи стабільно реагує на перемикання вкладок, повторне опитування датчиків, оновлення таблиць і формування звітів. Дані, створені під час роботи програми, зберігаються у локальній базі SQLite та залишаються доступними після повторного запуску застосунку. Це підтверджує коректність роботи шару доступу до даних і зв’язків між таблицями користувачів, датчиків, вимірювань, порогових значень, сповіщень і звітів.

Ефективність розробленої системи проявляється у скороченні часу контролю стану теплиці. Оператор отримує всю основну інформацію на одній панелі: поточні значення параметрів, допустимі межі, статус кожного показника, графік динаміки та список активних сповіщень. Завдяки цьому користувачеві не потрібно вручну порівнювати значення температури, вологості або освітленості з нормативними межами. Система автоматично виконує цю перевірку та формує повідомлення у разі відхилення.

Важливою перевагою є локальний характер роботи програми. Застосунок не потребує хмарного сервера для виконання базових функцій, а всі дані зберігаються на робочій станції користувача. Це спрощує встановлення, зменшує залежність від зовнішніх сервісів і робить систему придатною для навчально-дослідної або невеликої виробничої теплиці. За потреби система може бути розширена підключенням реального контролера збору даних через USB / Serial або інший інтерфейс обміну.

У таблиці 4.3 наведено узагальнені результати перевірки працездатності основних функцій системи.

Таблиця 4.3

Узагальнені результати тестування програмної системи

Функція системи	Результат перевірки	Оцінка
Авторизація користувача	Вхід за тестовим обліковим записом виконується коректно	Працює
Реєстрація користувача	Новий користувач створюється після заповнення форми	Працює

Автоматичне створення бази даних	База SQLite створюється під час першого запуску	Працює

Продовження таблиці 4.3

Опитування датчиків	Система генерує вимірювання для основних параметрів мікроклімату	Працює
Збереження вимірювань	Записи зберігаються у таблиці measurements	Працює
Перегляд історії	Дані відображаються у таблиці з можливістю фільтрації	Працює
Побудова графіка	Графік оновлюється відповідно до вибраного показника	Працює
Перевірка порогів	Значення порівнюються з мінімальними та максимальними межами	Працює
Формування сповіщень	При порушенні порога створюється активне повідомлення	Працює
Опрацювання сповіщень	Статус події змінюється після дії користувача	Працює
Формування CSV-звіту	Файл звіту створюється за вибраний період	Працює
Керування датчиками	Підтримується перегляд і додавання датчиків	Працює

Аналіз результатів тестування показав, що розроблена система відповідає поставленій меті та може використовуватися для демонстрації принципів автоматизованого моніторингу параметрів мікроклімату у теплиці. Система забезпечує зручне подання даних, фіксацію історії вимірювань, контроль допустимих меж і підготовку звітної інформації. У порівнянні з ручним контролем параметрів застосунок підвищує оперативність отримання інформації та зменшує ризик пропуску критичних відхилень.

Разом із тим система має потенціал для подальшого розвитку. На наступних етапах її можна доповнити підтримкою реальних датчиків, модулем керування виконавчими пристроями, віддаленим доступом через локальну мережу, розширеною аналітикою та автоматичним резервним копіюванням бази даних. Однак у межах поставленого завдання реалізованого функціоналу достатньо для повноцінного локального моніторингу основних параметрів мікроклімату.

Склад інсталяційного пакету наведено у таблиці 4.4.

Таблиця 4.4

Склад інсталяційного пакету програмної системи

Елемент пакету	Призначення
main.py	Головний файл запуску програми
requirements.txt	Перелік бібліотек Python, необхідних для роботи системи
run_macos_linux.sh	Скрипт запуску застосунку в середовищі macOS або Linux
run_windows.bat	Скрипт запуску застосунку в середовищі Windows
greenhouse_microclimate.db	Локальна база даних SQLite, яка створюється автоматично під час роботи
exports/	Каталог для збереження сформованих CSV-звітів
README.md	Інструкція із запуску програми, встановлення залежностей і використання тестового входу
assets/	Каталог допоміжних ресурсів інтерфейсу, якщо вони використовуються у програмі

Наведена структура інсталяційного пакету забезпечує просте розгортання системи на робочій станції користувача. Для запуску достатньо встановити залежності з файлу requirements.txt і виконати відповідний стартовий скрипт або запустити main.py безпосередньо через інтерпретатор Python. Наявність локальної бази SQLite спрощує перенесення системи між комп’ютерами та не потребує встановлення окремого серверного програмного забезпечення.

Отже, результати тестування підтвердили працездатність розробленої системи, правильність взаємодії основних модулів і відповідність реалізованого функціоналу вимогам до локального моніторингу мікроклімату теплиці. Програмне забезпечення може бути використане як навчально-дослідна система, прототип для подальшого підключення реальних датчиків або основа для розширення функцій автоматизованого керування тепличним середовищем.

4.4 Висновки до четвертого розділу

У четвертому розділі сформовано план тестування програмних модулів системи моніторингу параметрів мікроклімату у теплиці та визначено методику оцінювання результатів. Проведено перевірку авторизації, панелі моніторингу, історії вимірювань, сповіщень, порогових значень, звітів і довідника датчиків.

Результати тестування підтвердили працездатність основних функцій програмного забезпечення. Система коректно зберігає дані у SQLite, формує сповіщення при відхиленні показників, відображає історію вимірювань і забезпечує експорт звітної інформації.

Запропонований склад інсталяційного пакету дозволяє запускати систему на локальній робочій станції без складної інфраструктури. Розроблене програмне забезпечення може бути використане як основа для подальшого підключення реальних датчиків і виконавчих пристроїв теплиці.

ВИСНОВКИ

У кваліфікаційній роботі розроблено програмне забезпечення системи моніторингу параметрів мікроклімату у теплиці. Система орієнтована на локальне використання та забезпечує контроль температури повітря, вологості повітря, вологості ґрунту й освітленості з можливістю збереження історії, перевірки порогів, формування сповіщень і звітів.

У першому розділі проаналізовано предметну область моніторингу мікроклімату теплиць, визначено основні параметри контролю та розглянуто існуючі рішення. Встановлено, що промислові системи мають розвинену функціональність, але часто є надлишковими для невеликих і навчально-дослідних теплиць, тоді як локальне настільне програмне забезпечення забезпечує простішу експлуатацію і незалежність від хмарної інфраструктури.

Побудовано UML-моделі предметної області: діаграму прецедентів, діаграму послідовності та діаграму активності. Вони відображають ролі оператора, адміністратора, датчиків і виконавчих пристроїв, а також основну логіку збирання, оброблення, збереження та аналізу показників.

У другому розділі розроблено логічну модель даних, діаграму класів, діаграму кооперації, діаграму компонентів і діаграму пакетів. Сформована структура охоплює користувачів, теплиці, датчики, вимірювання, пороги, сповіщення та звіти, а також визначає модулі графічного інтерфейсу, моніторингу, збору даних, візуалізації, сповіщень, налаштувань і доступу до бази даних.

У третьому розділі обґрунтовано вибір Python 3, PyQt6 та SQLite як технологічної основи системи. Розроблено фізичну модель бази даних, схему розгортання, алгоритми основних модулів і реалізовано графічний інтерфейс із вкладками авторизації, моніторингу, вимірювань, сповіщень, порогів, звітів і датчиків.

У четвертому розділі проведено тестування програмного забезпечення. Перевірено авторизацію, реєстрацію, опитування датчиків, збереження

вимірювань, роботу порогів, формування сповіщень, перегляд історії, створення CSV-звітів і керування датчиками. Результати підтвердили працездатність системи та відповідність основним вимогам.

Практична цінність розробленої системи полягає у можливості використання її як навчально-дослідного програмного засобу для демонстрації принципів автоматизованого моніторингу теплиці. Подальший розвиток може передбачати підключення реальних датчиків через USB/Serial, реалізацію керування вентиляцією, поливом та обігрівом, а також створення мережевого або мобільного інтерфейсу для віддаленого контролю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bhujel A., Basak J. K., Khan F., Arulmozhi E., Jaihuni M., Sihalath T., Lee D., Park J., Kim H. T. Sensor Systems for Greenhouse Microclimate Monitoring and Control: a Review. Journal of Biosystems Engineering. 2020. Vol. 45, No. 4. P. 341-361. DOI: 10.1007/s42853-020-00075-6. URL: https://scholarworks.gnu.ac.kr/handle/sw.gnu/8278.

2. Design and implementation of intelligent monitoring system for agricultural environment in IoT. Internet of Things. 2023. DOI: 10.1016/j.iot.2023.101029. URL: https://www.sciencedirect.com/science/article/abs/pii/S2542660523003529.

3. Austria A. C. H., Fabros J. S., Sumilang K. R. G., Bernardino J., Doctor A. C. Development of IoT Smart Greenhouse System for Hydroponic Gardens. arXiv, 2023. URL: https://arxiv.org/abs/2305.01189.

4. Lopez-Martinez J., Blanco-Claraco J. L., Perez-Alonso J., Callejon-Ferre A. J. Distributed network for measuring climatic parameters in heterogeneous environments: Application in a greenhouse. arXiv, 2024. URL: https://arxiv.org/abs/2401.13420.

5. Object Management Group. OMG Unified Modeling Language. Version 2.5.1. 2017. URL: https://www.omg.org/spec/UML/2.5.1.

6. IEEE/ISO/IEC 12207-2026. Systems and software engineering - Software life cycle processes. IEEE Standards Association, 2026. URL: https://standards.ieee.org/ieee/12207/11416/.

7. Python Software Foundation. Python 3 Documentation. URL: https://docs.python.org/3/.

8. Riverbank Computing. PyQt Download and Documentation. URL: https://www.riverbankcomputing.com/software/pyqt/download.

9. The Qt Company. Qt for Python Documentation. URL: https://doc.qt.io/qtforpython-6/.

10. SQLite Consortium. SQLite Documentation. URL: https://www.sqlite.org/.

11. SQLite Consortium. About SQLite. URL: https://www.sqlite.org/about.html.

12. Python Enhancement Proposals. PEP 249 - Python Database API Specification v2.0. URL: https://peps.python.org/pep-0249/.

13. Matplotlib Development Team. Matplotlib Documentation. URL: https://matplotlib.org/stable/users/index.html.

14. diagrams.net. Diagram editor documentation and application. URL: https://www.diagrams.net/.

15. OWASP Foundation. Application Security Verification Standard. URL: https://owasp.org/www-project-application-security-verification-standard/.

ДОДАТОК А

Фрагмент програмного коду створення та ініціалізації бази даних SQLite

```
import sqlite3
import hashlib
from datetime import datetime


DB_NAME = "greenhouse_microclimate.db"


def hash_password(password: str) -> str:
    return hashlib.sha256(password.encode("utf-8")).hexdigest()


class DatabaseManager:
    def __init__(self, db_path: str = DB_NAME):
        self.db_path = db_path
        self.create_tables()
        self.seed_data()

    def connect(self):
        connection = sqlite3.connect(self.db_path)
        connection.row_factory = sqlite3.Row
        connection.execute("PRAGMA foreign_keys = ON")
        return connection

    def create_tables(self):
        with self.connect() as conn:
            cursor = conn.cursor()

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS users (
                    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
                    username TEXT NOT NULL UNIQUE,
                    password_hash TEXT NOT NULL,
                    full_name TEXT NOT NULL,
                    role TEXT NOT NULL DEFAULT 'operator',
                    created_at TEXT NOT NULL
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS greenhouses (
                    greenhouse_id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL,
                    location TEXT,
                    area_m2 REAL,
                    description TEXT,
                    created_at TEXT NOT NULL
                )
            """)

            cursor.execute("""
                CREATE TABLE IF NOT EXISTS sensor_types (
                    sensor_type_id INTEGER PRIMARY KEY AUTOINCREMENT,
                    name TEXT NOT NULL UNIQUE,
                    unit TEXT NOT NULL,
                    description TEXT
                )
            """)
```

```
cursor.execute("""
   CREATE TABLE IF NOT EXISTS sensors (
      sensor_id INTEGER PRIMARY KEY AUTOINCREMENT,
      greenhouse_id INTEGER NOT NULL,
      sensor_type_id INTEGER NOT NULL,
      name TEXT NOT NULL,
      serial_number TEXT,
      placement TEXT,
      is_active INTEGER NOT NULL DEFAULT 1,
      installed_at TEXT NOT NULL,
      FOREIGN KEY (greenhouse_id) REFERENCES greenhouses(greenhouse_id),
      FOREIGN KEY (sensor_type_id) REFERENCES sensor_types(sensor_type_id)
   )
""")

cursor.execute("""
   CREATE TABLE IF NOT EXISTS measurements (
      measurement_id INTEGER PRIMARY KEY AUTOINCREMENT,
      sensor_id INTEGER NOT NULL,
      value REAL NOT NULL,
      measured_at TEXT NOT NULL,
      quality_status TEXT NOT NULL DEFAULT 'ok',
      note TEXT,
      FOREIGN KEY (sensor_id) REFERENCES sensors(sensor_id)
   )
""")

cursor.execute("""
   CREATE TABLE IF NOT EXISTS threshold_settings (
      threshold_id INTEGER PRIMARY KEY AUTOINCREMENT,
      greenhouse_id INTEGER NOT NULL,
      sensor_type_id INTEGER NOT NULL,
      min_value REAL,
      max_value REAL,
      is_active INTEGER NOT NULL DEFAULT 1,
      updated_at TEXT NOT NULL,
      FOREIGN KEY (greenhouse_id) REFERENCES greenhouses(greenhouse_id),
      FOREIGN KEY (sensor_type_id) REFERENCES sensor_types(sensor_type_id)
   )
""")

cursor.execute("""
   CREATE TABLE IF NOT EXISTS alerts (
      alert_id INTEGER PRIMARY KEY AUTOINCREMENT,
      sensor_id INTEGER NOT NULL,
      measurement_id INTEGER NOT NULL,
      alert_type TEXT NOT NULL,
      message TEXT NOT NULL,
      status TEXT NOT NULL DEFAULT 'active',
      created_at TEXT NOT NULL,
      closed_at TEXT,
      FOREIGN KEY (sensor_id) REFERENCES sensors(sensor_id),
      FOREIGN KEY (measurement_id) REFERENCES measurements(measurement_id)
   )
""")

cursor.execute("""
   CREATE TABLE IF NOT EXISTS reports (
      report_id INTEGER PRIMARY KEY AUTOINCREMENT,
      user_id INTEGER NOT NULL,
      greenhouse_id INTEGER NOT NULL,
```

```
            period_start TEXT NOT NULL,
            period_end TEXT NOT NULL,
            summary TEXT,
            file_path TEXT,
            created_at TEXT NOT NULL,
            FOREIGN KEY (user_id) REFERENCES users(user_id),
            FOREIGN KEY (greenhouse_id) REFERENCES greenhouses(greenhouse_id)
        )
    """)

    conn.commit()

def seed_data(self):
    with self.connect() as conn:
        cursor = conn.cursor()

        cursor.execute("SELECT COUNT(*) FROM users")
        if cursor.fetchone()[0] == 0:
            cursor.execute("""
                INSERT INTO users (username, password_hash, full_name, role, created_at)
                VALUES (?, ?, ?, ?, ?)
            """, (
                "admin",
                hash_password("admin"),
                "Адміністратор",
                "admin",
                datetime.now().isoformat(timespec="seconds")
            ))

        cursor.execute("SELECT COUNT(*) FROM greenhouses")
        if cursor.fetchone()[0] == 0:
            cursor.execute("""
                INSERT INTO greenhouses (name, location, area_m2, description, created_at)
                VALUES (?, ?, ?, ?, ?)
            """, (
                "Теплиця №1",
                "Навчально-дослідна ділянка",
                48.0,
                "Теплиця для контролю параметрів мікроклімату",
                datetime.now().isoformat(timespec="seconds")
            ))

        sensor_types = [
            ("Температура повітря", "°C", "Контроль температури у теплиці"),
            ("Вологість повітря", "%", "Контроль відносної вологості повітря"),
            ("Вологість ґрунту", "%", "Контроль вологості ґрунту"),
            ("Освітленість", "лк", "Контроль рівня освітленості")
        ]

        for name, unit, description in sensor_types:
            cursor.execute("""
                INSERT OR IGNORE INTO sensor_types (name, unit, description)
                VALUES (?, ?, ?)
            """, (name, unit, description))

        cursor.execute("SELECT COUNT(*) FROM sensors")
        if cursor.fetchone()[0] == 0:
            cursor.execute("SELECT greenhouse_id FROM greenhouses LIMIT 1")
            greenhouse_id = cursor.fetchone()[0]

            cursor.execute("SELECT sensor_type_id, name FROM sensor_types")
            types = cursor.fetchall()
```

```
    for sensor_type in types:
        cursor.execute("""
            INSERT INTO sensors (
                greenhouse_id, sensor_type_id, name,
                serial_number, placement, is_active, installed_at
            )
            VALUES (?, ?, ?, ?, ?, ?, ?)
        """, (
            greenhouse_id,
            sensor_type["sensor_type_id"],
            sensor_type["name"],
            f"SIM-{sensor_type['sensor_type_id']}",
            "Основна зона теплиці",
            1,
            datetime.now().isoformat(timespec="seconds")
        ))

cursor.execute("SELECT COUNT(*) FROM threshold_settings")
if cursor.fetchone()[0] == 0:
    cursor.execute("SELECT greenhouse_id FROM greenhouses LIMIT 1")
    greenhouse_id = cursor.fetchone()[0]

    thresholds = {
        "Температура повітря": (16.0, 32.0),
        "Вологість повітря": (40.0, 95.0),
        "Вологість ґрунту": (25.0, 85.0),
        "Освітленість": (1000.0, 65000.0)
    }

    cursor.execute("SELECT sensor_type_id, name FROM sensor_types")
    for row in cursor.fetchall():
        min_value, max_value = thresholds[row["name"]]
        cursor.execute("""
            INSERT INTO threshold_settings (
                greenhouse_id, sensor_type_id,
                min_value, max_value, is_active, updated_at
            )
            VALUES (?, ?, ?, ?, ?, ?)
        """, (
            greenhouse_id,
            row["sensor_type_id"],
            min_value,
            max_value,
            1,
            datetime.now().isoformat(timespec="seconds")
        ))

conn.commit()
```

ДОДАТОК Б

Фрагмент програмного коду модуля моніторингу, імітації датчиків і формування сповіщень

```
import random
from datetime import datetime


class SensorSimulator:
    def generate_value(self, sensor_type: str) -> float:
        if sensor_type == "Температура повітря":
            return round(random.uniform(14.0, 34.0), 2)

        if sensor_type == "Вологість повітря":
            return round(random.uniform(35.0, 98.0), 2)

        if sensor_type == "Вологість ґрунту":
            return round(random.uniform(20.0, 90.0), 2)

        if sensor_type == "Освітленість":
            return round(random.uniform(800.0, 70000.0), 2)

        return 0.0


class MonitoringService:
    def __init__(self, database):
        self.database = database
        self.simulator = SensorSimulator()

    def poll_sensors(self):
        sensors = self.get_active_sensors()
        created_measurements = []

        for sensor in sensors:
            value = self.simulator.generate_value(sensor["sensor_type"])
            measurement_id = self.save_measurement(sensor["sensor_id"],
value)
            quality_status = self.check_thresholds(sensor, measurement_id,
value)

            created_measurements.append({
                "sensor_id": sensor["sensor_id"],
                "sensor_name": sensor["name"],
                "sensor_type": sensor["sensor_type"],
                "unit": sensor["unit"],
                "value": value,
                "quality_status": quality_status
            })

        return created_measurements

    def get_active_sensors(self):
        with self.database.connect() as conn:
            cursor = conn.cursor()
            cursor.execute("""
                SELECT
                    s.sensor_id,
                    s.name,
```

```
                    s.greenhouse_id,
                    st.sensor_type_id,
                    st.name AS sensor_type,
                    st.unit
                FROM sensors s
                JOIN sensor_types st ON s.sensor_type_id = st.sensor_type_id
                WHERE s.is_active = 1
                ORDER BY st.sensor_type_id
            """)
            return cursor.fetchall()

    def save_measurement(self, sensor_id: int, value: float) -> int:
        with self.database.connect() as conn:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO measurements (sensor_id, value, measured_at,
quality_status)
                VALUES (?, ?, ?, ?)
            """, (
                sensor_id,
                value,
                datetime.now().isoformat(timespec="seconds"),
                "ok"
            ))
            conn.commit()
            return cursor.lastrowid

    def check_thresholds(self, sensor, measurement_id: int, value: float) ->
str:
        with self.database.connect() as conn:
            cursor = conn.cursor()
            cursor.execute("""
                SELECT min_value, max_value
                FROM threshold_settings
                WHERE greenhouse_id = ?
                  AND sensor_type_id = ?
                  AND is_active = 1
                LIMIT 1
            """, (
                sensor["greenhouse_id"],
                sensor["sensor_type_id"]
            ))

            threshold = cursor.fetchone()

            if threshold is None:
                return "ok"

            min_value = threshold["min_value"]
            max_value = threshold["max_value"]

            if min_value is not None and value < min_value:
                message = (
                    f"{sensor['sensor_type']}: значення {value}
{sensor['unit']} "
                    f"нижче мінімального порога {min_value} {sensor['unit']}"
                )
                self.create_alert(sensor["sensor_id"], measurement_id, "low",
message)
                self.update_measurement_status(measurement_id, "warning")
                return "warning"

            if max_value is not None and value > max_value:
```

```
                message = (
                    f"{sensor['sensor_type']}: значення {value}
{sensor['unit']} "
                    f"вище максимального порога {max_value} {sensor['unit']}"
                )
                self.create_alert(sensor["sensor_id"], measurement_id,
"high", message)
                self.update_measurement_status(measurement_id, "warning")
                return "warning"

            return "ok"

    def create_alert(self, sensor_id: int, measurement_id: int, alert_type:
str, message: str):
        with self.database.connect() as conn:
            cursor = conn.cursor()
            cursor.execute("""
                INSERT INTO alerts (
                    sensor_id, measurement_id, alert_type,
                    message, status, created_at
                )
                VALUES (?, ?, ?, ?, ?, ?)
            """, (
                sensor_id,
                measurement_id,
                alert_type,
                message,
                "active",
                datetime.now().isoformat(timespec="seconds")
            ))
            conn.commit()

    def update_measurement_status(self, measurement_id: int, status: str):
        with self.database.connect() as conn:
            cursor = conn.cursor()
            cursor.execute("""
                UPDATE measurements
                SET quality_status = ?
                WHERE measurement_id = ?
            """, (status, measurement_id))
            conn.commit()
```

ДОДАТОК В

Фрагмент програмного коду головного вікна, графічного інтерфейсу та формування CSV-звіту

```
import csv
import os
from datetime import datetime

from PyQt6.QtWidgets import (
    QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
    QLabel, QPushButton, QTableWidget, QTableWidgetItem,
    QTabWidget, QComboBox, QFileDialog, QMessageBox
)
from PyQt6.QtCore import Qt


class MainWindow(QMainWindow):
    def __init__(self, database, monitoring_service, user):
        super().__init__()

        self.database = database
        self.monitoring_service = monitoring_service
        self.user = user

        self.setWindowTitle("Моніторинг мікроклімату теплиці")
        self.resize(1300, 820)

        self.tabs = QTabWidget()
        self.setCentralWidget(self.tabs)

        self.monitoring_tab = QWidget()
        self.measurements_tab = QWidget()
        self.alerts_tab = QWidget()
        self.reports_tab = QWidget()

        self.tabs.addTab(self.monitoring_tab, "Панель моніторингу")
        self.tabs.addTab(self.measurements_tab, "Вимірювання")
        self.tabs.addTab(self.alerts_tab, "Сповіщення")
        self.tabs.addTab(self.reports_tab, "Звіти")

        self.create_monitoring_tab()
        self.create_measurements_tab()
        self.create_alerts_tab()
        self.create_reports_tab()

        self.refresh_dashboard()
        self.load_measurements()
        self.load_alerts()

    def create_monitoring_tab(self):
        layout = QVBoxLayout(self.monitoring_tab)

        header_layout = QHBoxLayout()
        self.greenhouse_label = QLabel("Теплиця: Теплиця №1")
        self.greenhouse_label.setObjectName("titleLabel")

        self.refresh_button = QPushButton("Опитати датчики зараз")
        self.refresh_button.clicked.connect(self.poll_sensors)
```

```
        header_layout.addWidget(self.greenhouse_label)
        header_layout.addStretch()
        header_layout.addWidget(self.refresh_button)

        layout.addLayout(header_layout)

        self.cards_layout = QHBoxLayout()

        self.temperature_card = QLabel("Температура повітря\n-- °C")
        self.air_humidity_card = QLabel("Вологість повітря\n-- %")
        self.soil_moisture_card = QLabel("Вологість ґрунту\n-- %")
        self.light_card = QLabel("Освітленість\n-- лк")

        for card in [
            self.temperature_card,
            self.air_humidity_card,
            self.soil_moisture_card,
            self.light_card
        ]:
            card.setAlignment(Qt.AlignmentFlag.AlignCenter)
            card.setMinimumHeight(130)
            card.setObjectName("metricCard")
            self.cards_layout.addWidget(card)

        layout.addLayout(self.cards_layout)

        self.active_alerts_label = QLabel("Активних сповіщень немає.")
        self.active_alerts_label.setObjectName("alertPanel")
        layout.addWidget(self.active_alerts_label)

        layout.addStretch()

    def create_measurements_tab(self):
        layout = QVBoxLayout(self.measurements_tab)

        filter_layout = QHBoxLayout()

        self.measurement_filter = QComboBox()
        self.measurement_filter.addItems([
            "Усі показники",
            "Температура повітря",
            "Вологість повітря",
            "Вологість ґрунту",
            "Освітленість"
        ])

        show_button = QPushButton("Показати")
        show_button.clicked.connect(self.load_measurements)

        export_button = QPushButton("Експорт CSV")
        export_button.clicked.connect(self.export_measurements_csv)

        filter_layout.addWidget(QLabel("Показник:"))
        filter_layout.addWidget(self.measurement_filter)
        filter_layout.addWidget(show_button)
        filter_layout.addWidget(export_button)
        filter_layout.addStretch()

        layout.addLayout(filter_layout)

        self.measurements_table = QTableWidget()
        self.measurements_table.setColumnCount(7)
```

```
        self.measurements_table.setHorizontalHeaderLabels([
            "Дата і час",
            "Теплиця",
            "Датчик",
            "Показник",
            "Значення",
            "Одиниця",
            "Якість"
        ])

        layout.addWidget(self.measurements_table)

    def create_alerts_tab(self):
        layout = QVBoxLayout(self.alerts_tab)

        button_layout = QHBoxLayout()

        refresh_button = QPushButton("Оновити")
        refresh_button.clicked.connect(self.load_alerts)

        close_button = QPushButton("Позначити як опрацьоване")
        close_button.clicked.connect(self.close_selected_alert)

        button_layout.addWidget(refresh_button)
        button_layout.addWidget(close_button)
        button_layout.addStretch()

        layout.addLayout(button_layout)

        self.alerts_table = QTableWidget()
        self.alerts_table.setColumnCount(6)
        self.alerts_table.setHorizontalHeaderLabels([
            "Дата і час",
            "Рівень",
            "Показник",
            "Значення",
            "Повідомлення",
            "Стан"
        ])

        layout.addWidget(self.alerts_table)

    def create_reports_tab(self):
        layout = QVBoxLayout(self.reports_tab)

        button_layout = QHBoxLayout()

        self.report_filter = QComboBox()
        self.report_filter.addItems([
            "Останні 7 днів",
            "Останні 30 днів",
            "Усі дані"
        ])

        create_button = QPushButton("Сформувати CSV-звіт")
        create_button.clicked.connect(self.create_csv_report)

        button_layout.addWidget(QLabel("Період:"))
        button_layout.addWidget(self.report_filter)
        button_layout.addWidget(create_button)
        button_layout.addStretch()
```

```
    layout.addLayout(button_layout)

    self.reports_table = QTableWidget()
    self.reports_table.setColumnCount(4)
    self.reports_table.setHorizontalHeaderLabels([
        "Дата створення",
        "Користувач",
        "Теплиця",
        "Файл"
    ])

    layout.addWidget(self.reports_table)

def poll_sensors(self):
    self.monitoring_service.poll_sensors()
    self.refresh_dashboard()
    self.load_measurements()
    self.load_alerts()

def refresh_dashboard(self):
    latest = self.get_latest_measurements()

    cards = {
        "Температура повітря": self.temperature_card,
        "Вологість повітря": self.air_humidity_card,
        "Вологість ґрунту": self.soil_moisture_card,
        "Освітленість": self.light_card
    }

    for sensor_type, card in cards.items():
        item = latest.get(sensor_type)
        if item:
            card.setText(
                f"{sensor_type}\n"
                f"{item['value']} {item['unit']}\n"
                f"{'У межах норми' if item['quality_status'] == 'ok' else 'Поза нормою'}"
            )

    active_alerts = self.get_active_alerts()
    if active_alerts:
        text = "Активні сповіщення:\n"
        for alert in active_alerts[:3]:
            text += f"{alert['created_at']} | {alert['message']}\n"
        self.active_alerts_label.setText(text)
    else:
        self.active_alerts_label.setText(
            "Активних сповіщень немає. "
            "Параметри перебувають у допустимих межах."
        )

def get_latest_measurements(self):
    with self.database.connect() as conn:
        cursor = conn.cursor()
        cursor.execute("""
            SELECT
                st.name AS sensor_type,
                st.unit,
                m.value,
                m.quality_status,
                m.measured_at
            FROM measurements m
            JOIN sensors s ON m.sensor_id = s.sensor_id
```

```
            JOIN sensor_types st ON s.sensor_type_id = st.sensor_type_id
            WHERE m.measurement_id IN (
                SELECT MAX(m2.measurement_id)
                FROM measurements m2
                JOIN sensors s2 ON m2.sensor_id = s2.sensor_id
                GROUP BY s2.sensor_type_id
            )
        """)
        rows = cursor.fetchall()

    return {row["sensor_type"]: row for row in rows}

def get_active_alerts(self):
    with self.database.connect() as conn:
        cursor = conn.cursor()
        cursor.execute("""
            SELECT a.created_at, a.message, a.status
            FROM alerts a
            WHERE a.status = 'active'
            ORDER BY a.created_at DESC
        """)
        return cursor.fetchall()

def load_measurements(self):
    selected_type = self.measurement_filter.currentText()

    query = """
        SELECT
            m.measured_at,
            g.name AS greenhouse,
            s.name AS sensor,
            st.name AS parameter,
            m.value,
            st.unit,
            m.quality_status
        FROM measurements m
        JOIN sensors s ON m.sensor_id = s.sensor_id
        JOIN sensor_types st ON s.sensor_type_id = st.sensor_type_id
        JOIN greenhouses g ON s.greenhouse_id = g.greenhouse_id
    """

    params = []

    if selected_type != "Усі показники":
        query += " WHERE st.name = ?"
        params.append(selected_type)

    query += " ORDER BY m.measured_at DESC LIMIT 200"

    with self.database.connect() as conn:
        cursor = conn.cursor()
        cursor.execute(query, params)
        rows = cursor.fetchall()

    self.measurements_table.setRowCount(len(rows))

    for row_index, row in enumerate(rows):
        values = [
            row["measured_at"],
            row["greenhouse"],
            row["sensor"],
            row["parameter"],
```

```
            str(row["value"]),
            row["unit"],
            row["quality_status"]
        ]

        for column_index, value in enumerate(values):
            self.measurements_table.setItem(
                row_index,
                column_index,
                QTableWidgetItem(value)
            )

def load_alerts(self):
    with self.database.connect() as conn:
        cursor = conn.cursor()
        cursor.execute("""
            SELECT
                a.created_at,
                a.alert_type,
                st.name AS parameter,
                m.value,
                a.message,
                a.status
            FROM alerts a
            JOIN measurements m ON a.measurement_id = m.measurement_id
            JOIN sensors s ON a.sensor_id = s.sensor_id
            JOIN sensor_types st ON s.sensor_type_id = st.sensor_type_id
            ORDER BY a.created_at DESC
        """)
        rows = cursor.fetchall()

    self.alerts_table.setRowCount(len(rows))

    for row_index, row in enumerate(rows):
        values = [
            row["created_at"],
            row["alert_type"],
            row["parameter"],
            str(row["value"]),
            row["message"],
            row["status"]
        ]

        for column_index, value in enumerate(values):
            self.alerts_table.setItem(
                row_index,
                column_index,
                QTableWidgetItem(value)
            )

def close_selected_alert(self):
    row = self.alerts_table.currentRow()

    if row < 0:
        QMessageBox.warning(
            self,
            "Сповіщення",
            "Оберіть сповіщення для позначення як опрацьованого."
        )
        return

    created_at = self.alerts_table.item(row, 0).text()
```

```
    with self.database.connect() as conn:
        cursor = conn.cursor()
        cursor.execute("""
            UPDATE alerts
            SET status = 'closed',
                closed_at = ?
            WHERE created_at = ?
        """, (
            datetime.now().isoformat(timespec="seconds"),
            created_at
        ))
        conn.commit()

    self.load_alerts()
    self.refresh_dashboard()

def export_measurements_csv(self):
    os.makedirs("exports", exist_ok=True)
    file_path = os.path.join(
        "exports",
        f"measurements_{datetime.now().strftime('%Y%m%d_%H%M%S')}.csv"
    )

    with open(file_path, "w", newline="", encoding="utf-8-sig") as csv_file:
        writer = csv.writer(csv_file, delimiter=";")
        writer.writerow([
            "Дата і час",
            "Теплиця",
            "Датчик",
            "Показник",
            "Значення",
            "Одиниця",
            "Якість"
        ])

        for row in range(self.measurements_table.rowCount()):
            writer.writerow([
                self.measurements_table.item(row, column).text()
                if self.measurements_table.item(row, column) else ""
                for column in range(self.measurements_table.columnCount())
            ])

    QMessageBox.information(
        self,
        "Експорт завершено",
        f"CSV-файл збережено:\n{file_path}"
    )

def create_csv_report(self):
    os.makedirs("exports", exist_ok=True)
    file_path = os.path.join(
        "exports",
        f"greenhouse_report_{datetime.now().strftime('%Y%m%d_%H%M%S')}.csv"
    )

    with self.database.connect() as conn:
        cursor = conn.cursor()
        cursor.execute("""
            SELECT
                m.measured_at,
                g.name AS greenhouse,
```

```
            st.name AS parameter,
            m.value,
            st.unit,
            m.quality_status
        FROM measurements m
        JOIN sensors s ON m.sensor_id = s.sensor_id
        JOIN sensor_types st ON s.sensor_type_id = st.sensor_type_id
        JOIN greenhouses g ON s.greenhouse_id = g.greenhouse_id
        ORDER BY m.measured_at DESC
    """)
    rows = cursor.fetchall()

with open(file_path, "w", newline="", encoding="utf-8-sig") as csv_file:
    writer = csv.writer(csv_file, delimiter=";")
    writer.writerow([
        "Дата і час",
        "Теплиця",
        "Показник",
        "Значення",
        "Одиниця",
        "Стан"
    ])

    for row in rows:
        writer.writerow([
            row["measured_at"],
            row["greenhouse"],
            row["parameter"],
            row["value"],
            row["unit"],
            row["quality_status"]
        ])

with self.database.connect() as conn:
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO reports (
            user_id, greenhouse_id, period_start, period_end,
            summary, file_path, created_at
        )
        VALUES (?, ?, ?, ?, ?, ?, ?)
    """, (
        self.user["user_id"],
        1,
        "all",
        "all",
        "CSV-звіт за даними моніторингу мікроклімату теплиці",
        file_path,
        datetime.now().isoformat(timespec="seconds")
    ))
    conn.commit()

QMessageBox.information(
    self,
    "Звіт сформовано",
    f"Звіт успішно збережено:\n{file_path}"
)
```

ДОДАТОК Г

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ ЗДОБУВАЧА

Я, Пелехань Ярослав Ігорович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Програмне забезпечення системи моніторингу параметрів мікроклімату у теплиці» 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. _____**Ярослав ПЕЛЕХАНЬ**
(підпис)