

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ПОГОДЖЕНО
Декан факультету

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ
Завідувач кафедри інформаційних
систем і технологій

(підпис) **Ігор БОЛБОТ**

(підпис) **Михайло ШВИДЕНКО**

«__» _____ 2026 р.

«__» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

**на тему: «Інформаційна система агрегації та інтеграції метеоданих із
використанням сервіс-орієнтованих інтерфейсів»**

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми
д.е.н., професор

(підпис) **Роман РУДЕНСЬКИЙ**

**Керівник бакалаврської
кваліфікаційної роботи**
к.е.н., доцент

(підпис) **Євгеній СТАРИЧЕНКО**

Виконав

(підпис) **Євгеній ПІСКАЛЕНКО**

Київ-2026

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

**Завідувач кафедри інформаційних систем і
технологій**

к.е.н., доцент _____ Михайло ШВИДЕНКО
(підпис)

«__» _____ 2026 р.

**ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ**

Піскаленку Євгенію Миколайовичу

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Інформаційна система агрегації та інтеграції метеоданих із використанням сервіс-орієнтованих інтерфейсів» затверджена наказом від «06» січня 2026 р. № 46 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): методології аналізу та проєктування програмних систем, UML-моделювання; показники метеорологічних параметрів; сервіси інтеграції погодних даних; засоби збереження, оброблення, та прогнозування метеоданих із використанням сервіс-орієнтованих інтерфейсів.

Перелік питань, що підлягають дослідженню:

аналіз предметної області агрометеорологічного моніторингу; проєктування інформаційної системи агрегації та інтеграції метеоданих; розробка механізмів інтеграції із зовнішніми погодними API та IoT-пристроями, програмних модулів оброблення, аналізу та візуалізації метеоданих; реалізація, тестування та впровадження програмної системи.

Дата видачі завдання «19» грудня 2026 р.

**Керівник бакалаврської
кваліфікаційної роботи**

(підпис)

Євгеній СТАРИЧЕНКО

**Завдання взяв до
виконання**

(підпис)

Євгеній ПІСКАЛЕНКО

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ	4
ВСТУП.....	5
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ	7
1.1 Опис предметної області	7
1.2 Огляд існуючих рішень предметної області	8
1.3 Аналіз вимог до інформаційної системи.....	12
1.6 Висновки до першого розділу	13
2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....	15
2.1 Логічна модель даних інформаційної системи.....	15
2.2 Проектування класової структури та коопераційних сценаріїв.....	19
2.3 Компонентна модель програмної архітектури системи	24
2.4 Пакетна структуризація програмного забезпечення системи	27
2.5 Висновки до другого розділу	30
3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ	32
3.1 Вибір технологій та інструментальних засобів реалізації.....	32
3.2 Архітектура та розгортання інформаційної системи	35
3.3 Фізична структура бази даних.....	38
3.4 Алгоритмізація програмних модулів інформаційної системи	42
3.5 Висновки до третього розділу	47
4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ.....	49
4.1 Тестування програмних модулів інформаційної системи	49
4.2 Тестування інтерфейсних модулів та основних функцій системи	51
4.3 Розгортання системи та склад інсталяційного пакету	61
4.4 Висновки до четвертого розділу	64
ВИСНОВКИ	66
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	69
Додаток А	72

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД — база даних;

ІТ — інформаційні технології;

ІШІ — інструменти штучного інтелекту;

СУБД — система управління базами даних;

ІШ — штучний інтелект;

API (Application Programming Interface) — інтерфейс прикладного програмування;

CSS (Cascading Style Sheets) — каскадні таблиці стилів;

DOM (Document Object Model) — об'єктна модель документа;

gRPC (Remote Procedure Call) — високопродуктивний фреймворк віддаленого виклику процедур;

HTTP (Hypertext Transfer Protocol) — протокол передачі гіпертексту;

IoT (Internet of Things) — інтернет речей;

JSON (JavaScript Object Notation) — текстовий формат обміну даними;

JWT (JSON Web Token) — токен для безпечної передачі даних між сторонами;

ML (Machine Learning) — машинне навчання;

MLP (Multilayer Perceptron) — багатошаровий перцептрон (тип штучної нейронної мережі);

ORM (Object-Relational Mapping) — об'єктно-реляційне відображення;

REST (Representational State Transfer) — архітектурний стиль взаємодії компонентів розподіленого застосунку;

RPS (Requests Per Second) — кількість запитів за секунду (метрика продуктивності сервера);

SOA (Service-Oriented Architecture) — сервіс-орієнтована архітектура;

SPA (Single Page Application) — односторінковий вебдодаток;

URL (Uniform Resource Locator) — уніфікований локатор ресурсів;

YAML (Yet Another Markup Language) — дружній до людини формат серіалізації даних (використовується для конфігурацій);

XML (Extensible Markup Language) — розширювана мова розмітки.

ВСТУП

Сучасне сільське господарство стрімко переходить до цифрових моделей управління, у яких дані про стан ґрунту, погоду та росту культур є ключовим ресурсом для прийняття рішень. Однією з найважливіших задач аграрного менеджменту є об'єднання показників метеорологічних станцій, датчиків польових полів та прогнозів штучного інтелекту у єдину інформаційну систему. Ця задача стає критичною для прогнозування ризиків, планування сівби та оперативного реагування на погодні зміни.

У бакалаврській роботі реалізується інформаційна система агрегації та інтеграції метеоданих з використанням сервіс-орієнтованих інтерфейсів. Система забезпечує збір даних з метеостанцій і польових сенсорів, інтеграцію з зовнішніми погодними сервісами [24], збереження у реляційній базі даних та генерацію прогнозів з використанням аналітичного модуля на основі нейромереж.

Метою даної бакалаврської роботи є розробка інформаційної системи для агрегації, інтеграції та аналітики метеоданих, яка підтримує автоматизований збір даних, їх нормалізацію, прогнозування ризиків та візуалізацію результатів у веб-інтерфейсі.

Для досягнення поставленої мети вирішено наступні завдання:

- дослідити наукову та технічну літературу з питань моніторингу агрокліматичних параметрів, сервіс-орієнтованої архітектури та прогнозування ризиків шкідників;
- сформулювати функціональні та нефункціональні вимоги до інформаційної системи;
- спроектувати архітектуру системи на основі мікросервісного підходу;
- розробити модель даних для зберігання полів, сенсорів, метео-показників, прогнозів та результатів аналізів;
- реалізувати бекенд API на мові Go з використанням фреймворку Gin;

- інтегрувати сторонній погодний API та модуль прогнозування на Python;
- створити Svelte-інтерфейс для візуалізації дашбордів, карт полів та звітів;
- протестувати та перевірити коректність роботи системи у сценаріях агрегації, прогнозування та оповіщення.

Об'єктом дослідження є процеси збору, обробки та аналізу агрокліматичних даних для підтримки управління агровиробництвом.

Предметом дослідження є архітектурні та технологічні рішення для побудови системи агрегації метеоданих, інтеграції сервіс-орієнтованих інтерфейсів та прогнозування ризиків в аграрній сфері.

Методологія роботи базується на системному підході, аналізі вимог та проєктуванні програмних систем, використанні UML для моделювання, а також на практичному застосуванні методів обробки даних та машинного навчання.

У роботі використовуються такі методи:

- аналіз та синтез інформації;
- структурування вимог та побудова функціональних моделей;
- об'єктно-орієнтоване моделювання предметної області;
- розробка API та тестування HTTP запитів;
- комп'ютерне моделювання на основі нейронних мереж для прогнозування ризиків.

У проєкті використано ітеративний підхід до розробки, що забезпечує поетапно уточнювати вимоги та перевіряти проміжні результати. На кожному етапі створювався прототип, який піддавався аналізу як з технічної точки зору, так і з погляду користувацької цінності. Це дозволило визначити пріоритетні функції для першої версії системи та знизити ризики за рахунок поступового впровадження.

Окрему увагу приділено методам контролю якості: описується структура даних, визначаються правила валідації, формуються критерії прийому результатів. Такий підхід зменшує ймовірність технічних помилок і підвищує ясність архітектурних рішень.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Аграрні інформаційні системи є ключовим елементом цифрової трансформації сільського господарства. Вони дозволяють зменшити ризики, оптимізувати витрати, підвищити продуктивність та забезпечити належну якість продукції. Типові підсистеми включають [16]:

- моніторинг полів та ґрунту;
- збір даних з метеостанцій та IoT-датчиків;
- модуль прогнозування погодних факторів та шкідників;
- адміністративний модуль для керування полями, культурами та сенсорами;
- звітність та аналітика.

У сучасних рішеннях дедалі більше використовуються хмарні сервіси, REST API та контейнерна інфраструктура, що забезпечує гнучко масштабувати систему й легко інтегрувати сторонні джерела даних.

Агрегація метеоданих / Інтеграція метеоданих — це процес поєднання даних різних типів і моделей в узгоджений інтерфейс.

У межах дослідження потрібно забезпечити:

- збір даних з датчиків полів (температура, вологість, тиск тощо);
- отримання погодних даних із зовнішніх API метеосервісів;
- збереження даних у єдиній реляційній базі;
- підготовку даних для прогнозного модуля.

Одним з ключових викликів є обробка різномірних форматів даних: локальні сенсори можуть передавати показники в одному порядку, тоді як зовнішні погодні API використовують інший набір атрибутів. Система повинна коректно вирівнювати часові ряди, підтримувати часові позначки та узгоджувати значення за допомогою метаданих. У результаті було прийнято рішення про

використання внутрішнього шарувального компонента трансформації, який нормалізує дані перед збереженням і передачею до прогнозного модуля.

Це дозволяє уникнути помилок, пов'язаних з некоректними одиницями вимірювання, відсутніми значеннями або зсувом часових діапазонів. Збирання метаданих у вигляді назви джерела, часу зняття та одиниць вимірювання робить систему більш прозорою та легкою для аудиту [18, 23].

Сервіс-орієнтований архітектурний стиль передбачає розбиття системи на незалежні служби, які обмінюються даними через стандартизовані API. У розробленій системі використовуються такі сервіси:

- `Backend API` на Go для бізнес-логіки та роботи з даними;
- `Forecasting Engine` на Python для аналітики та машинного навчання;
- `Frontend` на Svelte для побудови інтерфейсу користувача;
- `Weather API Client` для інтеграції з зовнішніми погодними сервісами;
- `PostgreSQL` для зберігання інформації.

Цей підхід підвищує незалежність компонентів, спрощує їх заміну та тестування, а також забезпечує розгортати систему модульно.

1.2 Огляд існуючих рішень предметної області

Серед існуючих рішення для агромоніторингу вирізняються наступні підходи:

- моніторинг полів через IoT-системи та датчики;
- інтеграція з публічними метеослужбами;
- використання аналітики для прогнозування захворювань і шкідників;
- візуалізація даних у дашбордах.

Порівнюючи ці рішення з розробленою системою, ми бачимо, що основна цінність полягає у поєднанні прогнозних моделей з реальними агрокліматичними даними у єдиній сервісно-орієнтованій архітектурі.

1. Climate FieldView (Bayer). Climate FieldView — комерційна платформа від Bayer Digital Farming, яка надає інструменти для агрономічного моніторингу,

аналізу врожайності та погодних умов. Система збирає дані з польових датчиків, супутникові знімки та погодні сервіси, щоб створити єдину хмарну панель управління.

Ключові можливості

- агрегація шарів даних зі сенсорів, карт польових зон і погодних прогнозів;
- візуалізація теплових карт, карт врожайності та вологості ґрунту;
- інтеграція з Polymath API та різними полями обладнання для отримання фактичних показників;
- модуль прогнозування погодних ризиків і рекомендації щодо обробки посівів.

FieldView збирає телеметрію обладнання та польові вимірювання через датчики, а також отримує доступ до погодних даних через зовнішні погодні сервіси, включно з локальними метеостанціями. Вона нормалізує часові ряди, об'єднує історичні та поточні показники у звіти.

Сильні та слабкі сторони:

Переваги: глибока інтеграція з агрономічними практиками, зручна візуалізація, масштабована хмарна платформа.

Недоліки: орієнтованість на великих агровиробників, висока вартість підписки, менше уваги до відкритого API для самостійної інтеграції з локальними серверами.

2. John Deere Operations Center. John Deere Operations Center — веб-платформа, що об'єднує дані з сільськогосподарської техніки, метеопрогнози та польовий аналіз. Система призначена для фермерів, які використовують техніку John Deere, та забезпечує спостерігати за попередніми та поточними польовими операціями.

Ключові можливості John Deere Operations Center:

- агрегація телеметрії машин, карт полів та систем керування посівами;
- показники витрат палива, стану обладнання та відстеження виконання операцій;
- модулі погоди з прогнозами і історичними даними для кожного поля;

- Share функції для обміну картою та аналітикою з агрономами.

Як система працює з даними:

Operations Center отримує дані безпосередньо з бортових комп'ютерів техніки та підключених датчиків, а також використовує погодні дані від сторонніх провайдерів. Платформа централізує всю інформацію в одному акаунті, де фермер може порівнювати різні поля і операції з точки зору продуктивності та кліматичних умов.

Сильні та слабкі сторони:

Переваги: хороша підтримка телеметрії обладнання, пов'язана аналітика з польовими операціями, інтеграція з існуючими машинними даними.

Недоліки: найкраще працює з технікою John Deere, менш гнучка щодо сторонніх джерел датчиків і локальних структур даних.

3. IBM Watson Decision Platform for Agriculture. IBM Watson Decision Platform for Agriculture — корпоративне рішення, яке поєднує в собі AI/ML-аналітику, метеопрогнози і управління ризиками. Система націлена на агрокомпанії, що потребують складних моделей прогнозування та інтеграції з різноманітними даними.

Ключові можливості:

- прогнозування погодних ризиків, шкідників і хвороб на основі AI-моделей;
- агрегування даних з датчиків IoT, супутникових знімків і зовнішніх метеосервісів;
- інтеграція з ERP/CRM для управління ланцюжками постачання;
- модулі аналітики земельних ділянок та оптимізації витрат на внесення добрив.

Як система працює з даними:

Watson Decision Platform використовує конекторні підходи для прийому даних з речей (IoT), зовнішніх API й існуючої агродокументації. Вся інформація обробляється в хмарі IBM, де застосовуються ML-моделі для генерації рекомендацій і візуалізації ризиків.

Сильні та слабкі сторони Watson Decision Platform:

Переваги: потужна AI-аналітика, великий набір інтеграцій, можливість масштабування для корпоративних клієнтів.

Недоліки: складність впровадження, вища вартість, менше підходить для невеликих агропідприємств або офлайн-середовищ.

Таблиця 1.1

Порівняння рішень для агромоніторингу

Критерій	Climate FieldView	John Deere Operations Center	IBM Watson Decision Platform
Тип користувача	агровиробники, що шукають комплексну агрономічну аналітику	клієнти John Deere, які використовують техніку та телеметрію	корпоративні оператори й холдинги з потребою в AI-аналітиці
Основні дані	метео, картографія, польові датчики	телеметрія техніки, поля, погодні дані	IoT, супутникові знімки, погодні сервіси, ERP
Інтеграція	API та сторонні сервіси	сильна прив'язка до екосистеми John Deere	гнучкі конектори і хмарні інтеграції
Переваги	зручні дашборди для агрономів; фокус на врожайності	детальна телеметрія та аналіз польових операцій	потужна AI-аналітика; масштабованість; ризикоаналіз
Обмеження	висока вартість; залежність від хмарної платформи	найкраще працює з технікою John Deere; менша гнучкість щодо сторонніх датчиків	складність впровадження; висока вартість; потреба в хмарній інфраструктурі
Доступність локального розгортання	обмежена	обмежена	може потребувати хмарної інфраструктури
Орієнтація на метеодані	прогноз агрономіка +	прогноз + операції техніки	прогноз + AI-аналіз ризиків

Як видно з таблиці 1.1, у кожного рішення є свій фокус:

Climate FieldView концентрується на агрономічних показниках та погодних даних для оптимізації врожайності;

John Deere Operations Center поєднує метеоаналіз з відстеженням польових операцій техніки;

IBM Watson Decision Platform пропонує потужну AI-аналітику та прогнозування ризиків.

1.3 Аналіз вимог до інформаційної системи

Під час дослідження предметної області було сформовано основні групи вимог до програмної системи, серед яких виділено функціональні, нефункціональні вимоги та вимоги до захисту інформації. Такий розподіл дає можливість більш структуровано описати особливості системи та визначити її ключові характеристики.

Функціональні вимоги. На основі технічного завдання і аналізу існуючих рішень [1, 15] в предметній області сформульовано наступні функції системи:

- збір метрик від сенсорів полів;
- збереження історії метео-спостережень та агрометричних показників;
- інтеграція з зовнішніми погодними API;
- запуск алгоритмів прогнозування на основі нейронних мереж;
- генерація рекомендацій щодо ризиків шкідників та стану культур;
- адміністрування полів, культур, сенсорів та шкідників;
- створення звітів для агрономів;
- інтерфейс оповіщень про середовищні ризики.

Нефункціональні вимоги. Створена система повинна відповідати таким нефункціональним вимогам:

- продуктивність: швидкий час відповіді API та відображення дашбордів;
- надійність: збереження даних та відновлення після збоїв;
- безпека: автентифікація, авторизація та захист конфіденційних даних або;

- масштабованість: можливість додавання нових полів, датчиків та моделей прогнозування;
- зручність використання: інтуїтивний інтерфейс;
- підтримка: структура коду повинна полегшувати розширення системи.

Вимоги до інтерфейсу користувача. Інтерфейс повинен бути реалізований як SPA з інтерактивними дашбордами та підтримкою відображення карт полів. Ключові вимоги:

- представлення поточних метеоданих та прогнозів в реальному часі;
- можливість фільтрації полів та сенсорів;
- графіки показників у часі;
- розмежування прав доступу для агрономів та адміністраторів;
- адаптивність до розмірів екрана.

Важливо також забезпечити доступність інтерфейсу для користувачів з різними рівнями підготовки. Це означає зрозумілу навігацію, чіткі підказки, структуру меню та мінімум складних налаштувань. Агроном повинен мати можливість швидко знайти інформацію про поле, побачити статус датчиків, порівняти прогнози та отримати рекомендації одним кліком.

Також передбачено підтримку різних ролей: адміністратор системи може управляти користувачами та налаштуваннями, агроном — працювати з полями та прогнозами, а технічний оператор — здійснювати моніторинг стану датчиків. Розмежування прав доступу підвищує безпеку та забезпечує зберігати відповідальність за дії у системі.

1.6 Висновки до першого розділу

У першому розділі виконано комплексний системний аналіз предметної області, проведено порівняльний аналіз наявних програмних аналогів та сформульовано вимоги до проектованої інформаційної системи. За результатами проведеного аналізу отримано такі висновки:

Обґрунтовано доцільність впровадження сервіс-орієнтованого архітектурного стилю (SOA) для збору, агрегації та інтеграції агрокліматичних даних. Розподіл системи на незалежні сервіси (Backend API на Go, Forecasting Engine на Python, Frontend на Svelte) забезпечує високу модульність, гнучке масштабування та ізольоване тестування компонентів. Визначено, що впровадження внутрішнього шару трансформації даних дозволяє ефективно вирішити проблему гетерогенності форматів та неузгодженості часових рядів, які надходять із локальних IoT-сенсорів та зовнішніх метеорологічних API.

Виявлено ключові архітектурні та економічні обмеження існуючих платформ агромоніторингу (Climate FieldView, John Deere Operations Center, IBM Watson Decision Platform). Встановлено, що більшість сучасних комерційних рішень орієнтовані на великі холдинги, мають високу вартість ліцензування, жорстку прив'язку до конкретних брендів сільськогосподарської техніки або закриту хмарну інфраструктуру вендора. Це підтверджує актуальність розробки автономної системи з гнучкими відкритими інтерфейсами інтеграції, здатної функціонувати в ізольованих або гібридних середовищах.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних інформаційної системи

Логічна модель даних описує структуру ключових інформаційних сутностей системи агрегації та інтеграції метеоданих, побудованої на основі сервіс-орієнтованих інтерфейсів, а також визначає взаємозв'язки між окремими компонентами даних. Розроблення такої моделі є одним із основних етапів проектування інформаційної системи, оскільки вона забезпечує формалізацію предметної області, визначення необхідного набору даних та організацію їх впорядкованого зберігання. Модель охоплює інформацію про сільськогосподарські культури, земельні ділянки, IoT-пристрої, телеметричні показники, погодні спостереження, метеорологічні параметри та результати прогнозування ризиків.

Побудова логічної моделі виконана відповідно до принципів реляційного моделювання та нормалізації даних [28]. Для кожної сутності передбачено використання первинних ключів, що забезпечують унікальну ідентифікацію записів, а також зовнішніх ключів для реалізації зв'язків між таблицями. Така структура дозволяє підтримувати цілісність даних, зменшувати надлишковість інформації та уникати її дублювання. Крім того, сформована модель створює основу для подальшої реалізації фізичної структури бази даних у середовищі СУБД PostgreSQL. На рисунку 2.1 представлено логічну модель даних інформаційної системи агрегації та інтеграції метеоданих.

Logical ER Diagram: Agro Data Management System

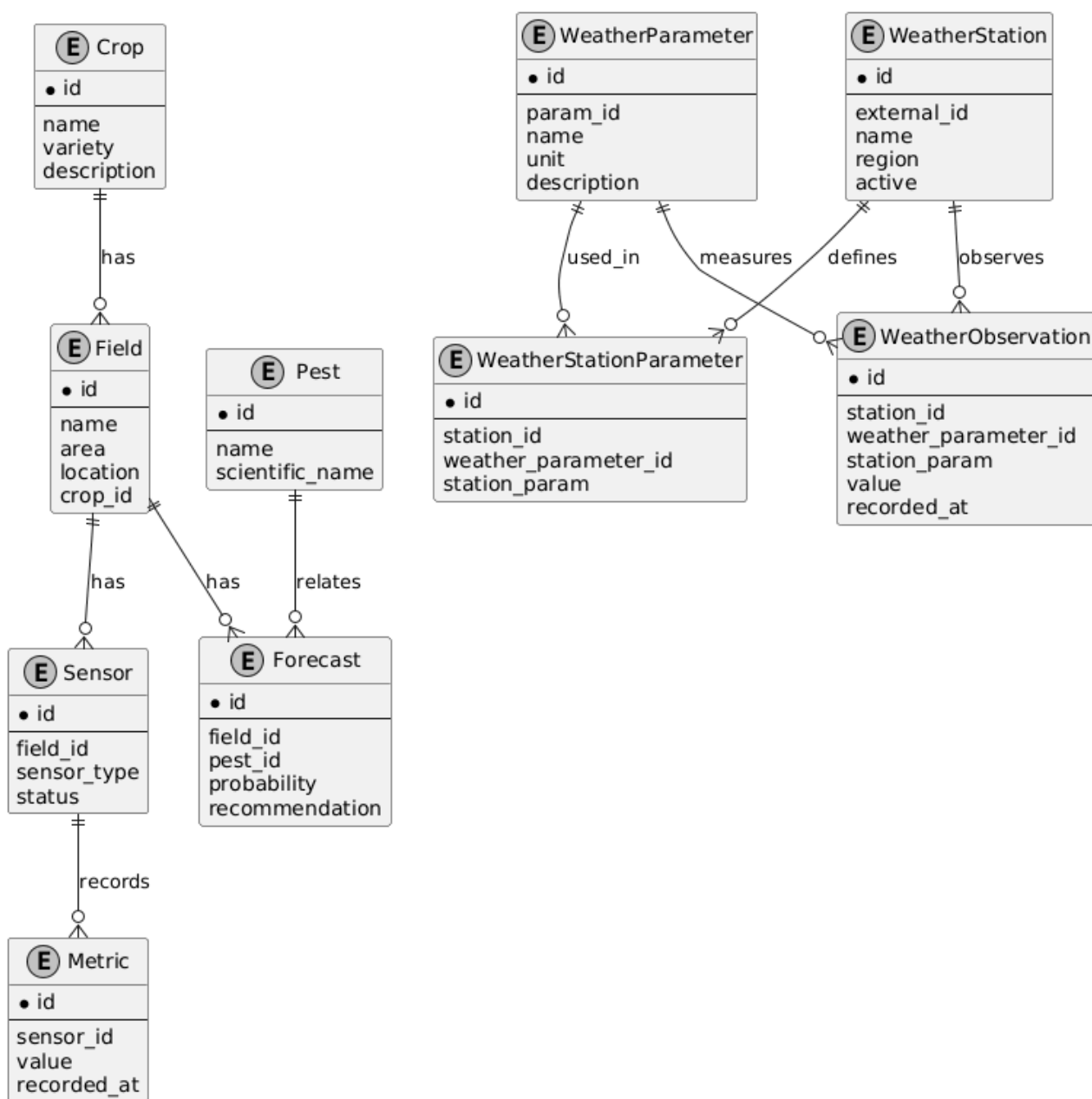


Рис. 2.1 – Логічна модель даних системи управління агродами

Центральними сутностями системи управління агродами є «Поле» (Field) та структурні одиниці метеорологічного моніторингу. Для кожної земельної ділянки (поля) у системі зберігаються ідентифікатор, назва, загальна площа, географічне розташування (координати або просторові межі) та зовнішній ключ для прив'язки до вирощуваної сільськогосподарської культури (Crop). Сутність «Культура» містить нормативно-довідкову інформацію: унікальний ідентифікатор, назву, сорт або варіацію та детальний опис

біологічних особливостей. Зв'язок між сутностями Crop та Field має тип «один-до-багатьох», оскільки одна культура може вирощуватися на багатьох полях, але кожне конкретне поле в поточний проміжок часу асоційоване з однією культурою.

До кожного поля можуть бути підключені локальні вимірювальні пристрої. Сутність «Датчик» (Sensor) описує апаратні модулі збору первинної телеметрії на полях. Вона містить атрибути ідентифікатора, посилання на поле, тип датчика (температура, вологість ґрунту, атмосферний тиск тощо) та його поточний статус. Зв'язок між Field та Sensor є зв'язком типу «один-до-багатьох».

Для накопичення часових рядів первинних показників від IoT-пристроїв використовується сутність «Метрика» (Metric), що фіксує унікальний номер запису, посилання на джерело (датчик), безпосереднє числове значення параметра та часову мітку зняття показника. Між сутностями Sensor та Metric встановлено зв'язок «один-до-багатьох».

Блок інтеграції з публічними та сторонніми метеослужбами представлений сутностями «Метеостанція» (WeatherStation), «Параметр погоди» (WeatherParameter), «Параметр метеостанції» (WeatherStationParameter) та «Спостереження погоди» (WeatherObservation). Сутність WeatherStation описує джерела інтегрованих даних (зовнішній ідентифікатор API, назва, регіон, ознака активності). Сутність WeatherParameter визначає глобальний перелік підтримуваних метрик (температура, швидкість вітру, вологість) з їхніми системними одиницями вимірювання.

Оскільки різні зовнішні сервіси використовують власні кодування та набори атрибутів, проміжна сутність WeatherStationParameter виступає внутрішнім шаром метаданих для відображення специфічних параметрів конкретної станції на уніфіковані метеопараметри системи. Історіографія фактичних агрегованих метеорологічних даних накопичується в сутності WeatherObservation, яка пов'язана з конкретною метеостанцією та уніфікованим параметром погоди, зберігаючи нормалізоване значення та часову позначку.

Аналітичний контур системи опирається на сутності «Шкідник» (Pest) та «Прогноз» (Forecast). Довідник шкідників та захворювань містить загальноприйняту назву та наукове (латинське) найменування. Сутність Forecast інтегрує дані про поле та потенційну біологічну загрозу, фіксуючи розраховану математичну ймовірність виникнення ризику та згенеровані агрономічні рекомендації щодо протидії чи обробки посівів.

Основні сутності логічної моделі даних та їх функціональне призначення наведено у таблиці 2.1.

Таблиця 2.1

Основні сутності логічної моделі даних агроінформаційної системи

Сутність	Призначення
Crop	Зберігання нормативно-довідкових даних про сільськогосподарські культури та їхні сорти
Field	Опис просторових і технологічних характеристик земельних ділянок (полів) агропідприємства
Sensor	Ідентифікація та моніторинг стану локальних IoT-датчиків, розгорнутих на полях
Metric	Накопичення первинних часових рядів телеметрії з датчиків (температура, вологість)
Pest	Реєстр фітосанітарних загроз, шкідників та захворювань сільськогосподарських культур
Forecast	Збереження результатів роботи прогностичного ML-модуля (ймовірність ризиків, рекомендації)
WeatherParameter	Загальносистемний класифікатор метеорологічних одиниць вимірювання та параметрів

Продовження таблиці 2.1

WeatherStation	Облік інтегрованих зовнішніх метеорологічних станцій та сторонніх сервісів API
WeatherStationParameter	Шар метаданих для мапування внутрішніх параметрів системи на атрибути зовнішніх API
WeatherObservation	Централізоване сховище інтегрованих та нормалізованих метеорологічних спостережень

Запропонована логічна модель забезпечує чітке розмежування первинних IoT-даних, інтегрованих метеоданих та результатів прогнозного моделювання, гарантуючи стійкість інформаційної структури системи під час інтенсивного масштабування.

2.2 Проектування класової структури та коопераційних сценаріїв

Проектування програмної архітектури інформаційної системи спрямоване на опис та структуризацію програмних компонентів, які реалізують бізнес-процеси збору, оброблення та аналізу метеоданих, а також виконання аналітичного прогнозування. Для представлення статичної структури backend-компонентів було використано UML-діаграму класів [21], яка відображає основні програмні сутності, їх атрибути, методи та взаємозв'язки між окремими об'єктами системи.

Діаграму класів інформаційної системи агрегації та інтеграції метеоданих представлено на рисунку 2.2.

UML Class Diagram: Agro Data Management System

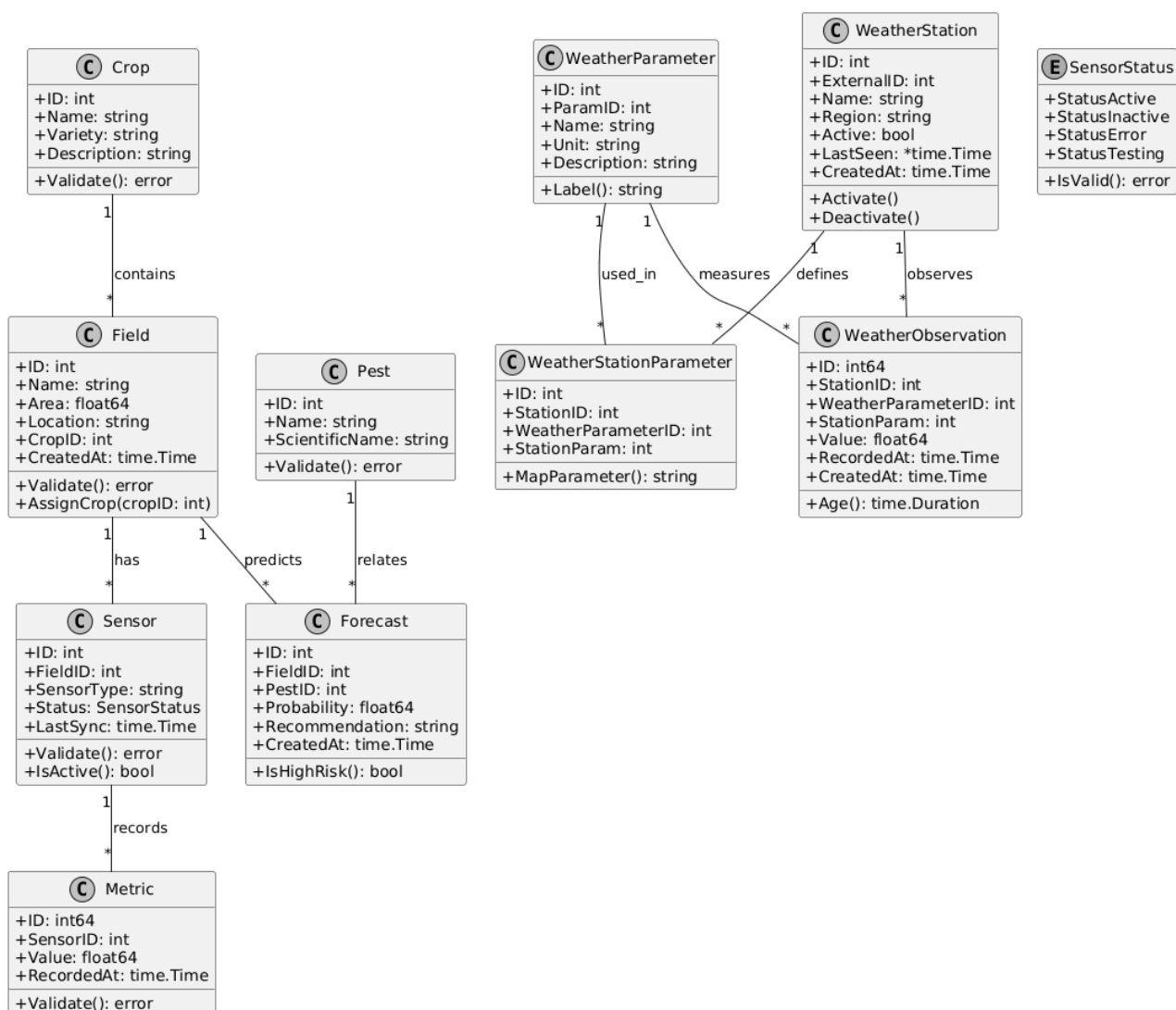


Рис. 2.2 – UML-діаграма класів інформаційної системи

У межах моделі визначено набір класів, структура яких відповідає основним сутностям бази даних, однак додатково доповнена методами перевірки даних та реалізації внутрішньої бізнес-логіки. Класи Crop, Field і Sensor містять базові атрибути та методи типу Validate(), що використовуються для контролю коректності введених даних. Для класу Field реалізовано метод AssignCrop(), який забезпечує можливість зміни культури, закріпленої за певним полем. У класі Sensor застосовано перелічуваний тип SensorStatus (StatusActive, StatusInactive, StatusError, StatusTesting) для визначення поточного стану пристрою, а перевірка його працездатності виконується за допомогою методу IsActive().

Для обробки метеоінформації спроектовано класи `WeatherParameter` (з методом `Label()` для отримання текстового представлення одиниць вимірювання), `WeatherStation` (з методами активації/деактивації `Activate()`, `Deactivate()`), а також `WeatherStationParameter` та `WeatherObservation`. Метод `MapParameter()` у класі `WeatherStationParameter` виконує ключову функцію перетворення зовнішніх кодів метрик у внутрішній уніфікований формат [20]. Клас `WeatherObservation` містить метод `Age()`, який обчислює час, що минув з моменту фіксації спостереження, для визначення актуальності даних перед передачею в аналітичний двигун.

З метою деталізації динамічних аспектів роботи сервіс-орієнтованої системи розроблено три діаграми кооперації та послідовності процесів, які демонструють взаємодію об'єктів під час виконання ключових сценаріїв.

Перший сценарій описує процес опитування зовнішніх метеосервісів через спеціалізований клієнт, нормалізацію часових рядів та їх персистенцію [2]. Взаємодія об'єктів у межах цього процесу наведена на рисунку 2.3.

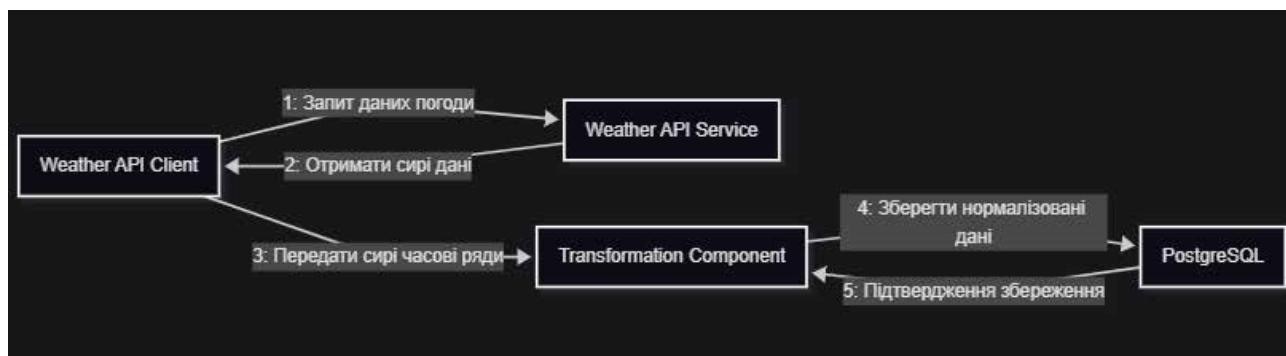


Рис. 2.3 – Діаграма кооперації процесу збору даних із зовнішніх метеосервісів

У цьому сценарії об'єкт `Weather API Client` ініціює запит даних погоди (1) до `Weather API Service`. Зовнішній сервіс повертає сирі дані (2) у форматі JSON/XML. Клієнт передає отримані сирі часові ряди (3) до внутрішнього компонента трансформації (`Transformation Component`), який здійснює вирівнювання часових позначок, перерахунок одиниць вимірювання на основі метаданих та мапування параметрів. Нормалізовані дані передаються для

збереження (4) до реляційної бази даних PostgreSQL, звідки повертається підтвердження успішного завершення операції запису (5).

Другий сценарій відображає приймання, валідацію формату й діапазонів значень, а також логування первинної телеметрії від локальних польових IoT-датчиків. Схема взаємодії компонентів представлена на рисунку 2.4.

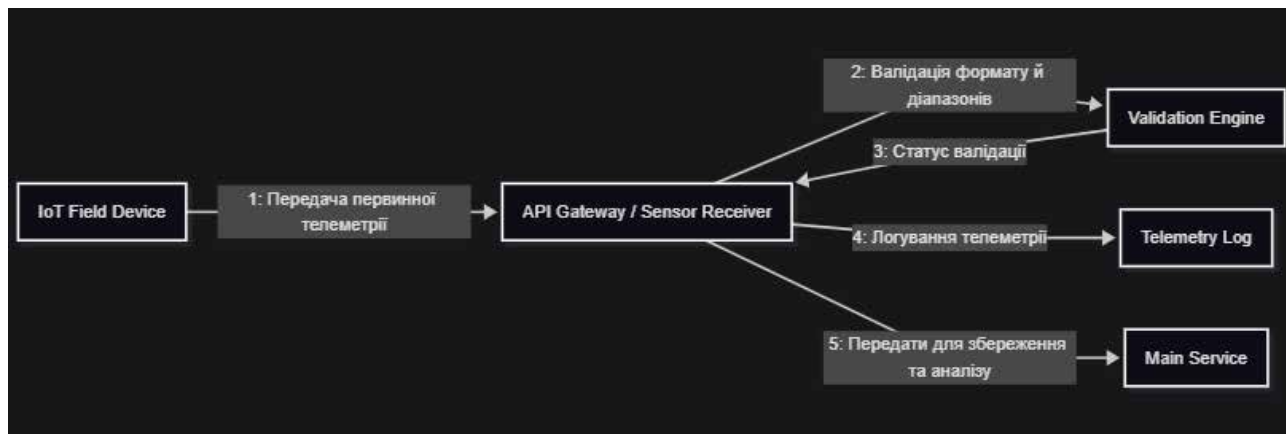


Рис. 2.4 – Діаграма кооперації валідації та приймання IoT-телеметрії

Локальний польовий пристрій IoT Field Device здійснює передачу пакета первинної телеметрії (1) на API Gateway / Sensor Receiver. Шлюз перенаправляє отримані структури даних до Validation Engine для перевірки відповідності форматів та допустимості числових діапазонів (2). Компонент валідації повертає статус перевірки (3). У разі успішного проходження фільтрів шлюз здійснює логування транзакції у Telemetry Log (4) та передає очищений масив даних для збереження та подальшого аналізу (5) до головного сервісу системи Main Service.

Третій сценарій відображає взаємодію під час передачі інтегрованого масиву метеоданих до аналітичного двигуна, розрахунку прогнозів ймовірності появи шкідників та збереження результатів. Схема кооперації наведена на рисунку 2.5.

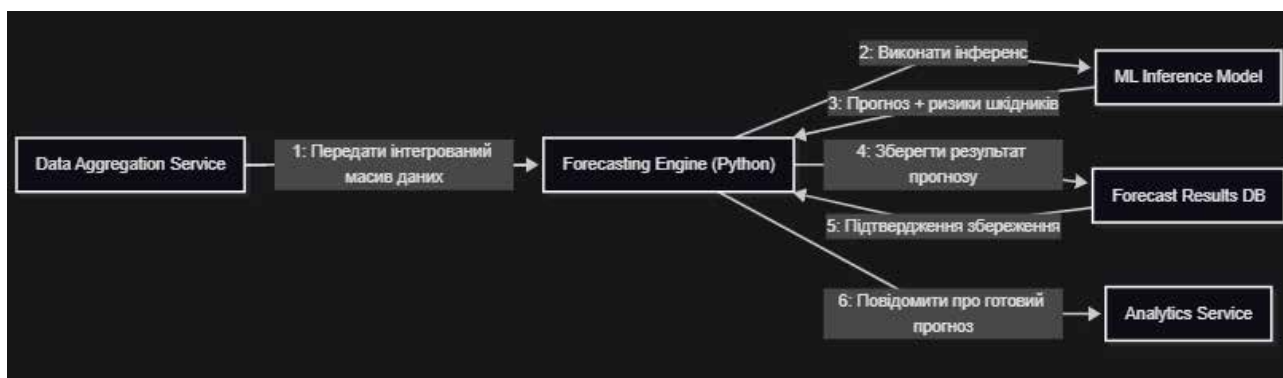


Рис. 2.5 – Діаграма кооперації процесу аналітичного прогнозування

Сервіс агрегації даних Data Aggregation Service формує консолідований payload і передає інтегрований масив агрокліматичних даних (1) до Forecasting Engine, реалізованого на Python. Двигун викликає ML Inference Model для виконання інференсу (2) на базі навчених нейромережевих моделей. Модель повертає згенерований прогноз та розраховані коефіцієнти ризиків появи шкідників (3). Forecasting Engine ініціює збереження результатів прогнозу (4) у спеціалізовану базу даних Forecast Results DB, отримує підтвердження транзакції (5) та надсилає повідомлення про готовність свіжого аналітичного звіту (6) до Analytics Service для подальшої візуалізації на дашбордах користувачів.

Основні класи програмної моделі системи та їхнє призначення узагальнено у таблиці 2.2.

Таблиця 2.2

Основні класи програмної моделі системи

Клас	Призначення
Crop	Модель опису культури із вбудованими механізмами валідації полів
Field	Бізнес-модель поля, що відповідає за зв'язок між датчиками, прогнозами та посівами
Sensor	Програмний об'єкт керування станом та конфігурацією польового датчика
Metric	Структура фіксації та первинної перевірки точкового виміру IoT-пристрою

Продовження таблиці 2.2

Pest	Об'єкт класифікації та систематизації фітосанітарних загроз
Forecast	Модель формування рекомендацій та оцінки ступеня агрокліматичного ризику
WeatherParameter	Клас конфігурування системних метеорологічних одиниць вимірювання
WeatherStation	Керування параметрами підключення та опитування сторонніх метеостанцій
WeatherStationParameter	Компонент мапування та трансформації різномірних атрибутів зовнішніх API
WeatherObservation	Програмна модель інтегрованого метеорологічного спостереження

2.3 Компонентна модель програмної архітектури системи

Проектування архітектури системи базується на сервіс-орієнтованому стилі, де кожен функціональний вузол є ізольованим компонентом із чітко визначеним інтерфейсом взаємодії. Декомпозиція системи на компоненти дозволяє забезпечити паралельну розробку, незалежне розгортання та ізольоване масштабування підсистем. Побудована UML-діаграма компонентів відображає розподіл системи на логічні шари: API, збору даних, обробки, зберігання, зовнішніх сервісів та інфраструктури.

Компонентну діаграму архітектури системи представлено на рисунку 2.6.

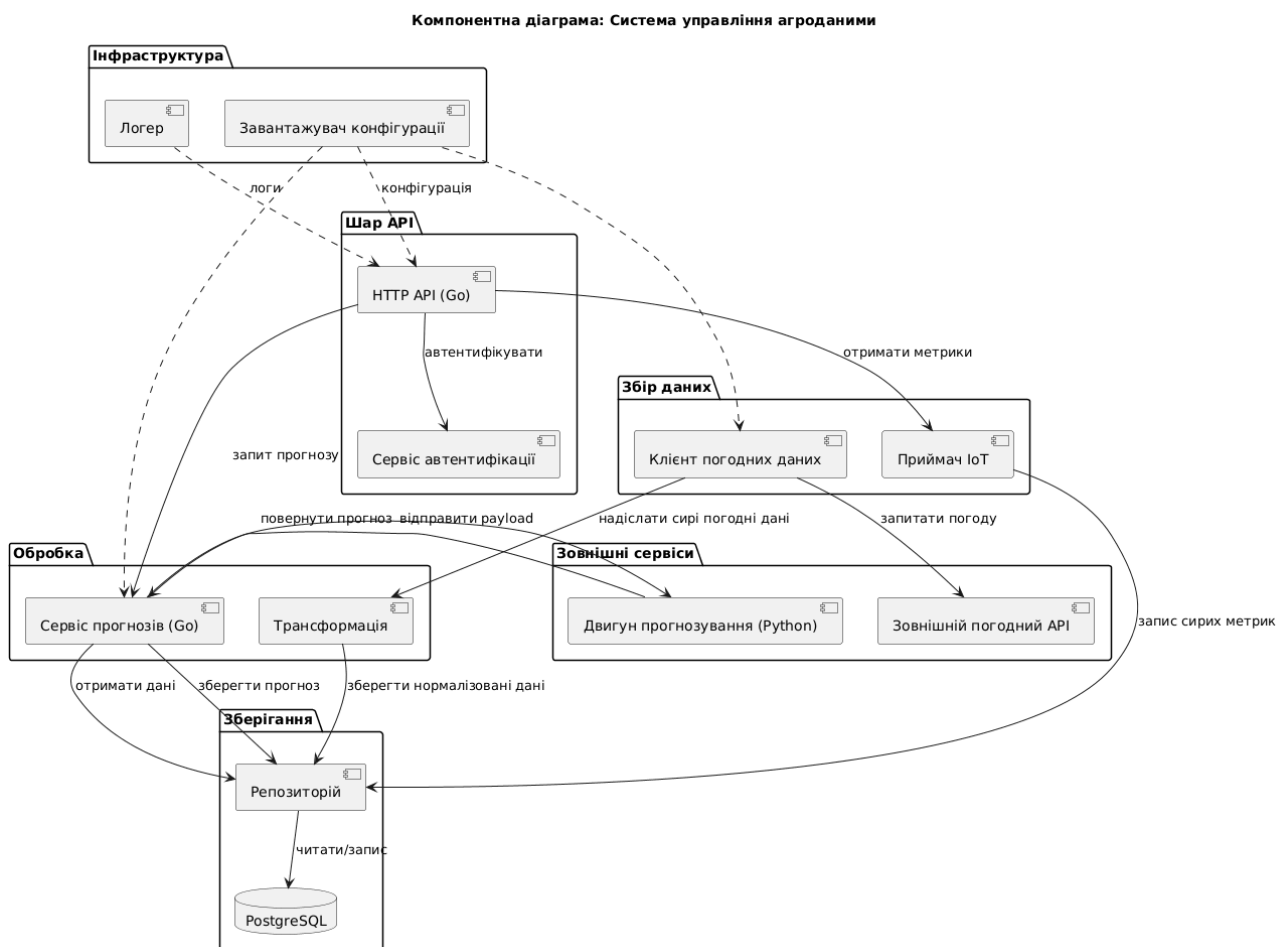


Рис. 2.6 – UML-діаграма компонентів інформаційної системи

Архітектура системи організована за модульним принципом і містить такі ключові пакети компонентів:

- **Шар API:** містить компонент HTTP API (Go), який є єдиною точкою входу для клієнтських SPA-додатків та координує виконання запитів користувачів, взаємодіючи із Сервісом автентифікації для перевірки JWT-токенів та прав доступу ролей (агроном, адміністратор, технік) [4].
- **Збір даних:** охоплює Приймач IoT (відповідає за постійне прослуховування портів та приймання телеметрії від польових сенсорів) та Клієнт погодних даних (виконує регламентні запити до зовнішніх сервісів за розкладом).
- **Обробка:** складається з компонента Трансформація (виконує очищення, валідацію та нормалізацію форматів даних) та Сервісу прогнозів (Go), який керує життєвим циклом аналітичних запитів.

- Зберігання: включає уніфікований шар Репозиторій, який ізолює логіку доступу до даних від бізнес-моделей, та реляційну базу даних PostgreSQL.
- Зовнішні сервіси: представляє Двигун прогнозування (Python), який містить математичні моделі та алгоритми машинного навчання для оцінки ризиків шкідників, та Зовнішній погодні API (сторонні метеорологічні провайдери).
- Інфраструктурний шар: забезпечує наскрізне логування за допомогою компонента Логер та централізоване керування оточенням через Завантажувач конфігурації.

Взаємозв'язки між компонентами відображають спрямованість потоків даних. HTTP API (Go) викликає Сервіс автентифікації для валідації користувача та звертається до Приймача IoT для отримання поточних метрик. Приймач IoT здійснює прямий запис сирих метрик через Репозиторій у базу даних. Клієнт погодніх даних опитує Зовнішній погодні API, передає сирі відповіді компоненту Трансформація, який після нормалізації зберігає їх через Репозиторій у PostgreSQL. Сервіс прогнозів (Go) забирає агреговані історичні дані з репозиторію, формує payload для Двигуна прогнозування (Python), отримує результат ML-інференсу та записує фінальний прогноз у сховище.

Призначення основних компонентів архітектури наведено у таблиці 2.3.

Таблиця 2.3

Компоненти архітектури інформаційної системи

Компонент	Функціональне призначення
HTTP API (Go)	Обробка вхідних клієнтських запитів, маршрутизація та формування відповідей
Сервіс автентифікації	Перевірка облікових даних користувачів, генерація та валідація сесійних токенів
Приймач IoT	Асинхронне приймання та первинне декодування пакетів телеметрії від датчиків
Клієнт погодних даних	Інтеграційний модуль для взаємодії із зовнішніми REST API метеослужб
Трансформація	Модуль нормалізації, вирівнювання часових рядів та узгодження одиниць виміру
Сервіс прогнозів (Go)	Координатор аналітичних задач, взаємодії з ML-модулем та збереження результатів
Репозиторій	Абстракція доступу до СУБД, реалізація шаблону Repository для роботи з PostgreSQL
Двигун прогнозування (Python)	Сервіс виконання ML-інференсу для розрахунку ймовірності фітосанітарних загроз
Завантажувач конфігурації	Зчитування, валідація та розповсюдження системних констант та секретів із файлів
Логер	Централізоване структурне логування подій, помилок та бізнес-метрик системи

2.4 Пакетна структуризація програмного забезпечення системи

З метою забезпечення високого рівня супроводжуваності, модульності та відповідності принципам чистої архітектури (Clean Architecture), програмний код backend-частини системи структуровано за допомогою пакетів. UML-діаграма пакетів відображає логічний поділ вихідного коду на директорії,

визначає межі інкапсуляції бізнес-логіки та ієрархію залежностей між модулями, що унеможливорює появу циклічних залежностей.

Пакетну структуру програмного забезпечення представлено на рисунку 2.7.

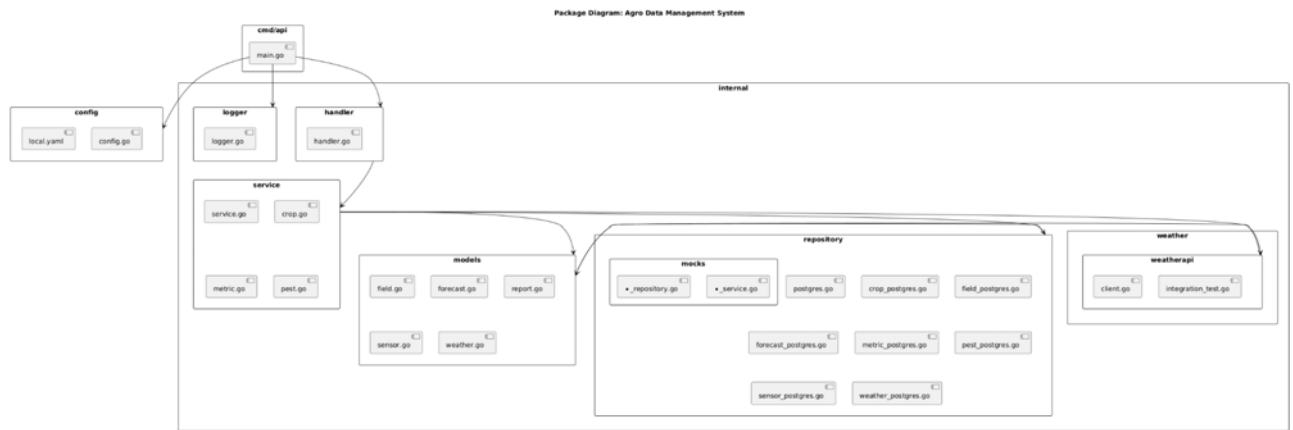


Рис. 2.7 – UML-діаграма пакетів програмного забезпечення

Вся кодова база проєкту спроектована в межах стандартної структури Go-додатків і містить такі ключові пакети:

- `cmd/api`: містить головний файл `main.go`, який є точкою запуску програми, де виконується ініціалізація логера, зчитування конфігурацій, підключення до бази даних та запуск HTTP-сервера.
- `config`: відповідає за збереження параметрів у файлі `local.yaml` та їх розбір за допомогою коду в `config.go`.
- `internal/models`: пакет містить чисті структури даних (доменні моделі), такі як `field.go`, `forecast.go`, `sensor.go`, `weather.go`, які позбавлені зовнішніх залежностей від фреймворків чи баз даних.
- `internal/repository`: пакет реалізує шар доступу до бази даних PostgreSQL (`postgres.go`, `field_postgres.go`, `weather_postgres.go` тощо). Також містить підпакет `mocks` із автоматично згенерованими інтерфейсними заглушками для проведення ізольованого модульного тестування.
- `internal/service`: пакет містить ядро бізнес-логіки системи (`crop.go`, `forecast.go`, `weather.go`). Саме тут реалізовано алгоритми перевірки умов, координації збору даних та виклику аналітичних моделей.

- `internal/handler`: рівень представлення веб-інтерфейсу API. Реалізує HTTP-контролери для обробки маршрутів, валідації вхідних JSON-payload та формування REST-відповідей.
- `internal/weather/weatherapi`: відокремлений інтеграційний пакет, що містить `client.go` для виконання HTTP-запитів до зовнішніх метеорологічних сервісів та покритий інтеграційними тестами (`integration_test.go`).

Напрямки залежностей чітко слідують правилу залежностей чистої архітектури: `main.go` ініціалізує та викликає `handler`, `config` та `logger`. Пакет `handler` залежить виключно від пакету `service`. Пакет `service` впроваджує залежності інтерфейсів `repository` та `weatherapi`. Пакети `repository`, `service`, `handler` та `weatherapi` використовують спільні описи структур даних із пакету `models`, що забезпечує наскрізну типізацію та логічну цілісність системи.

Опис призначень базових пакетів архітектури наведено у таблиці 2.4.

Таблиця 2.4

Функціональне призначення пакетів програмного забезпечення

Пакет	Призначення
<code>cmd/api</code>	Точка входу в додаток, ініціалізація інфраструктури, ін'єкція залежностей (DI)
<code>config</code>	Парсинг конфігураційних файлів середовища та типізоване представлення налаштувань
<code>internal/models</code>	Опис базових сутностей предметної області (сухий домен системи без поведінки)
<code>internal/repository</code>	Шар персистенції, виконання SQL-запитів та взаємодія із СУБД PostgreSQL
<code>internal/service</code>	Реалізація бізнес-правил системи, обробка даних та координація підсервісів
<code>internal/handler</code>	Реалізація транспортного HTTP-рівня, валідація запитів, обробка кодів відповідей
<code>internal/weather/weatherapi</code>	Ізольований клієнт взаємодії із зовнішніми метеорологічними інтерфейсами

2.5 Висновки до другого розділу

У другому розділі кваліфікаційної роботи виконано повний цикл проєктування інформаційного, архітектурного та програмного забезпечення інформаційної системи агрегації та інтеграції метеоданих із використанням сервіс-орієнтованих інтерфейсів. Основну увагу приділено формалізації інформаційних потоків, забезпеченню модульності та побудові стійкої компонентної структури.

За результатами проєктування отримано такі науково-практичні результати:

1 .Розроблено логічну модель даних, яка успішно об'єднує в межах єдиної реляційної структури різноманітні інформаційні зрізи: довідники сільськогосподарських культур, часові ряди первинної телеметрії з локальних IoT-датчиків полів, а також нормалізовані метеорологічні спостереження із зовнішніх віддалених станцій. Впровадження проміжного мапування параметрів дозволило нівелювати проблему гетерогенності сторонніх API-інтерфейсів.

2 .Спроектовано об'єктно-класову структуру backend-рівня системи із чітким виділенням методів валідації, розрахунку актуальності даних та мапування атрибутів. За допомогою діаграм кооперації деталізовано та формалізовано три ключові динамічні сценарії: автоматизований збір та персистенцію погодних даних через REST-клієнт, асинхронне приймання та фільтрацію телеметрії з польових IoT-пристроїв, а також координацію міжплатформенної взаємодії з ML-двигуном на Python під час виконання прогнозного інференсу.

3 .Обґрунтовано компонентну архітектуру та пакетну структуризацію коду, які базуються на принципах сервіс-орієнтованого стилю та Clean Architecture. Поділ системи на ізольовані шари (API, збір даних, обробка, репозиторії) та використання інтерфейсних абстракцій гарантують високу масштабованість системи, спрощують супровід та дозволяють незалежно

розвивати аналітичний ML-модуль, клієнтські інтерфейси та сервіси інтеграції з новими метеорологічними провайдерами.

Розроблені проєктні специфікації, UML-моделі та схеми взаємодії є вичерпною та нормативною основою для переходу до етапу безпосередньої програмної реалізації, розгортання та тестування інформаційної системи.

3 ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Вибір технологій та інструментальних засобів реалізації

Створення інформаційної системи агрегації та інтеграції метеоданих на основі сервіс-орієнтованих інтерфейсів вимагає ретельно підбраного технологічного середовища, здатного забезпечити ефективне приймання значних потоків телеметричних даних, одночасну взаємодію з різнорідними зовнішніми API, стабільне зберігання часових рядів та оперативне виконання аналітичних алгоритмів машинного навчання. Оскільки система побудована за принципами розподіленої сервісної архітектури, використовувані технології повинні підтримувати високий рівень паралельної обробки даних, забезпечувати надійну типізацію критичних компонентів та легко інтегруватися у спільну контейнеризовану інфраструктуру.

Під час вибору технологічного стеку було враховано не лише функціональні потреби системи, зокрема асинхронний збір IoT-даних, оброблення та нормалізацію метеорологічної інформації, а також запуск моделей машинного навчання для оцінювання фітосанітарних ризиків, але й низку нефункціональних характеристик. Серед основних критеріїв розглядалися продуктивність обробки запитів, мінімізація затримок, масштабованість програмної архітектури, ізоляція окремих сервісів та швидкість розгортання програмного середовища.

Крім того, вибір технологій здійснювався з урахуванням перспектив подальшого розвитку системи та можливості її інтеграції з іншими інформаційними платформами аграрного сектору. Використання сучасних інструментів розробки та контейнеризованого підходу дозволяє забезпечити гнучкість архітектури, спростити супровід програмного забезпечення та створити умови для масштабування системи відповідно до збільшення кількості джерел метеоданих і аналітичних сервісів.

Основні інструментальні засоби та програмні компоненти, обрані для реалізації системи, наведено в таблиці 3.1.

Таблиця 3.1

Обґрунтування технологічного стеку інформаційної системи

№	Компонент системи	Технологія / Бібліотека	Функціональне призначення
1	Мова серверної частини (Core API)	Go (Golang)	Реалізація високонавантаженої бізнес-логіки, приймання телеметрії та оркестрації сервісів
2	Маршрутизація та HTTP API	github.com/gin-gonic/gin	Побудова легковагового, продуктивного REST API сервера
3	Шар доступу до даних (СУБД)	github.com/jmoiron/sqlx	Розширення стандартного пакета <code>database/sql</code> для зручного мапування структур на SQL-запити
4	Драйвер бази даних	github.com/lib/pq	Чиста реалізація Go-драйвера для взаємодії із СКБД PostgreSQL
5	Системне логування	go.uber.org/zap	Високопродуктивне структуроване логування асинхронних процесів
6	Конфігурування системи	<code>cleanenv</code> , <code>godotenv</code>	Зчитування конфігураційних файлів <code>.yaml</code> та керування змінними оточення
7	Мова аналітичного двигуна	Python	Інтеграція математичних бібліотек та виконання інференсу нейромережевих моделей
8	Сервер аналітики та ML API	FastAPI + Uvicorn	Організація швидкого асинхронного REST інтерфейсу до моделей машинного навчання

Продовження таблиці 3.1

9	Машинне навчання та аналіз	scikit-learn, NumPy, joblib	Обробка числових масивів, серіалізація та виконання моделей класифікації MLP
10	Клієнтський веб-інтерфейс	Svelte + TypeScript + Vite	Реалізація реактивного SPA-інтерфейсу з низьким обсягом бандлу
11	Віртуалізація та інфраструктура	Docker	Контейнеризація сервісів, ізоляція середовищ розробки та експлуатації
12	Управління вихідним кодом	Git	Версіонування програмних компонентів та організація командної розробки

Основною мовою розробки базового серверного компонента (Core API) обрано Go (Golang) [6]. Вибір зумовлений його архітектурними перевагами при обробці мережових потоків та низьким рівнем споживання системних ресурсів. Механізм горутин (go-routines) дозволяє ефективно обробляти тисячі одночасних з'єднань від польових IoT-датчиків без значних витрат оперативної пам'яті [22].

Фреймворк Gin обрано як базовий HTTP-маршрутизатор [7] через його високу швидкість обробки запитів (завдяки використанню алгоритму Radix tree). Пакет sqlx інтегровано для спрощення взаємодії з реляційними структурами без втрати продуктивності, яка притаманна важким ORM-системам. Для мінімізації затримок при записі логів у реальному часі задіяно бібліотеку zap від Uber, яка мінімізує виділення пам'яті в купі (heap allocations).

Аналітичний сервіс (Forecasting Engine) реалізовано на мові Python [8]. Це рішення зумовлене наявністю розвиненої інфраструктури для Data Science та Machine Learning. Бібліотека scikit-learn застосовується для виконання класифікаційних моделей (зокрема, багатошарових перцептронів MLP [25]), що оцінюють ймовірність виникнення шкідників на основі динаміки кліматичних показників. Фреймворк FastAPI забезпечує асинхронну обробку вхідних payloads

від Go-сервера [9], автоматично генеруючи інтерактивну документацію OpenAPI/Swagger, що полегшує тестування міжсервісної взаємодії.

Клієнтський рівень системи побудовано на базі фреймворку Svelte із використанням TypeScript [10] та [27] та інструмента збірки Vite [11]. На відміну від традиційних віртуальних DOM-фреймворків, Svelte компілює код на етапі збірки в чистий, оптимізований JavaScript, що забезпечує миттєве оновлення графіків метеоданих та інтерактивних карт полів на пристроях агрономів із низькою обчислювальною потужністю.

Усі програмні модулі об'єднано за допомогою інструментів контейнеризації Docker [13]. Це дозволяє нівелювати проблему розбіжності системних залежностей між Go та Python середовищами, гарантуючи ідентичність функціонування системи як на локальних машинах розробників, так і на цільових серверах розгортання.

3.2 Архітектура та розгортання інформаційної системи

Архітектура розгортання спроектована за багаторівневим принципом, що забезпечує чітку ізоляцію фізичного шару збору даних, логічного рівня обробки, аналітичного ядра та сховищ інформації. Сервіс-орієнтований характер системи дозволяє виокремити критичні вузли в автономні Docker-контейнери [26], які взаємодіють між собою через стандартизовані протоколи HTTP/REST та gRPC, знижуючи рівень зв'язності компонентів (loose coupling).

Загальну діаграму розгортання інформаційної системи агрегації агроданих представлено на рисунку 3.1.

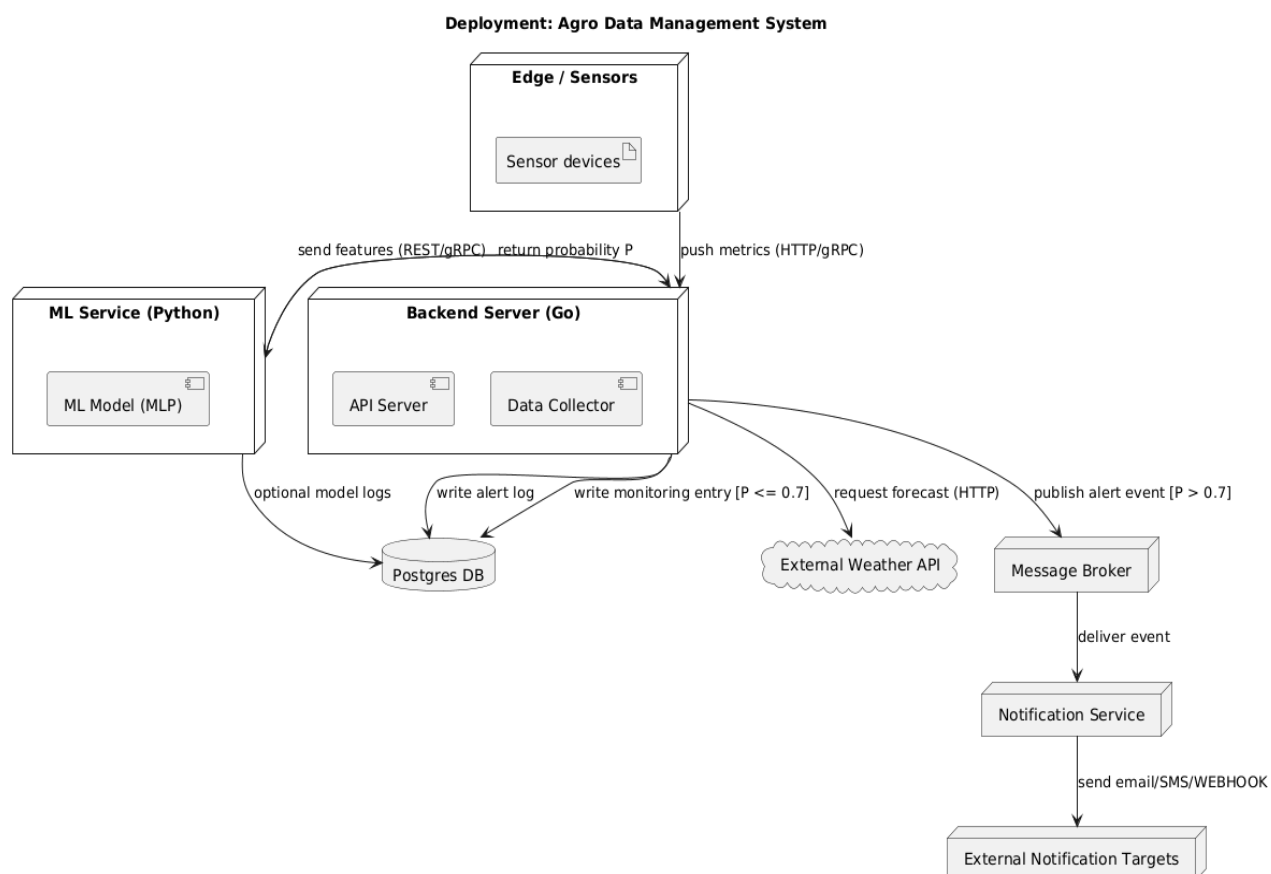


Рис. 3.1 – Діаграма розгортання інформаційної системи управління агродами
Потік інформації в системі розділено на два ключові функціональні контури:

1. Контур збору та моніторингу: Первинні показники кліматичного стану формуються на рівні Edge / Sensors польовими пристроями, які за допомогою HTTP-транспорту здійснюють надсилання пакетів телеметрії на серверний вузол Backend Server (Go). Паралельно з цим компонент Data Collector за регламентним розкладом ініціює запити до External Weather API [14] для отримання регіональних прогнозів та фактичних спостережень сторонніх метеостанцій. Усі отримані дані проходять шар очищення і записуються в базу даних PostgreSQL.

2. Аналітичний контур (Інференс): Для оцінки фітосанітарних ризиків Backend Server формує масив нормалізованих ознак і передає їх через REST інтерфейс на вузол ML Service (Python). Компонент машинного навчання на основі MLP-моделей обчислює математичну ймовірність появи загрози P.

Залежно від отриманого значення ймовірності P , система розгалужує логіку обробки подій за пороговим критерієм, де критичною межею визначено показник $P > 0.7$ (70% ймовірності розвитку патогену чи шкідника):

- При $P > 0.7$ (високий рівень загрози) подія класифікується як критичний алерт. Backend Server миттєво реєструє лог аномалії в PostgreSQL та публікує подію в шину даних Message Broker. Брокер асинхронно доставляє івент до Notification Service, який через External Notification Targets відправляє push-повідомлення, SMS або Webhook-повідомлення відповідальним агрономам.
- При $P \leq 0.7$ (нормальний фітосанітарний стан) подія обробляється у штатному режимі моніторингу: дані записуються у таблицю прогнозів PostgreSQL для відображення на планових дашбордах без ініціалізації екстрених сповіщень.

Призначення та взаємозв'язки основних рівнів архітектури розгортання наведено в таблиці 3.2.

Таблиця 3.2

Опис рівнів архітектури розгортання системи

Рівень архітектури	Компонентний склад	Тип взаємодії та протоколи	Функціональна роль
Edge / Sensors	ІоТ-модулі, польові датчики вологості та температури	HTTP POST / JSON, gRPC	Фіксація локальних фізичних параметрів агросередовища та їх відправка
External Services	Сторонні метеорологічні сервіси, API публічних станцій	HTTP GET / REST	Надання регіональних макрокліматичних даних за геокоординатами
Backend Server	API Server, Data Collector (Go-додаток в Docker)	Внутрішній транспорт, SQL-протокол, HTTP	Маршрутизація, оркестрація збору, первинна валідація та логування

Продовження таблиці 3.2

ML Service	Аналітичний двигун Python (FastAPI в Docker)	REST API / JSON payloads	Приймання ознак, запуск інференсу MLP, повернення коефіцієнтів ризику
Database Server	СУБД PostgreSQL	Спеціалізоване TCP-з'єднання (порт 5432)	Персистентне зберігання нормалізованих метаданих, профілів та прогнозів
Notification Layer	Message Broker, Notifier (Email, SMS, Webhooks)	AMQP / MQTT, SMTP, HTTP API	Асинхронна доставка екстрених сповіщень при перевищенні порогів ризику

3.3 Фізична структура бази даних

Фізична модель бази даних розроблена на основі раніше затвердженої логічної схеми сутностей та оптимізована для розгортання в середовищі реляційної СУБД PostgreSQL[12, PostgreSQL 16 Documentation. The PostgreSQL Global Development Group]. На відміну від логічного представлення, фізична структура чітко визначає типи даних для кожного атрибута, обмеження цілісності (PRIMARY KEY, FOREIGN KEY, NOT NULL), а також специфіку збереження великих числових масивів часових рядів, що генеруються вимірювальними пристроями.

Детальну схему фізичної моделі бази даних агроінформаційної системи представлено на рисунку 3.2.

ER Diagram: Agro Data Management System

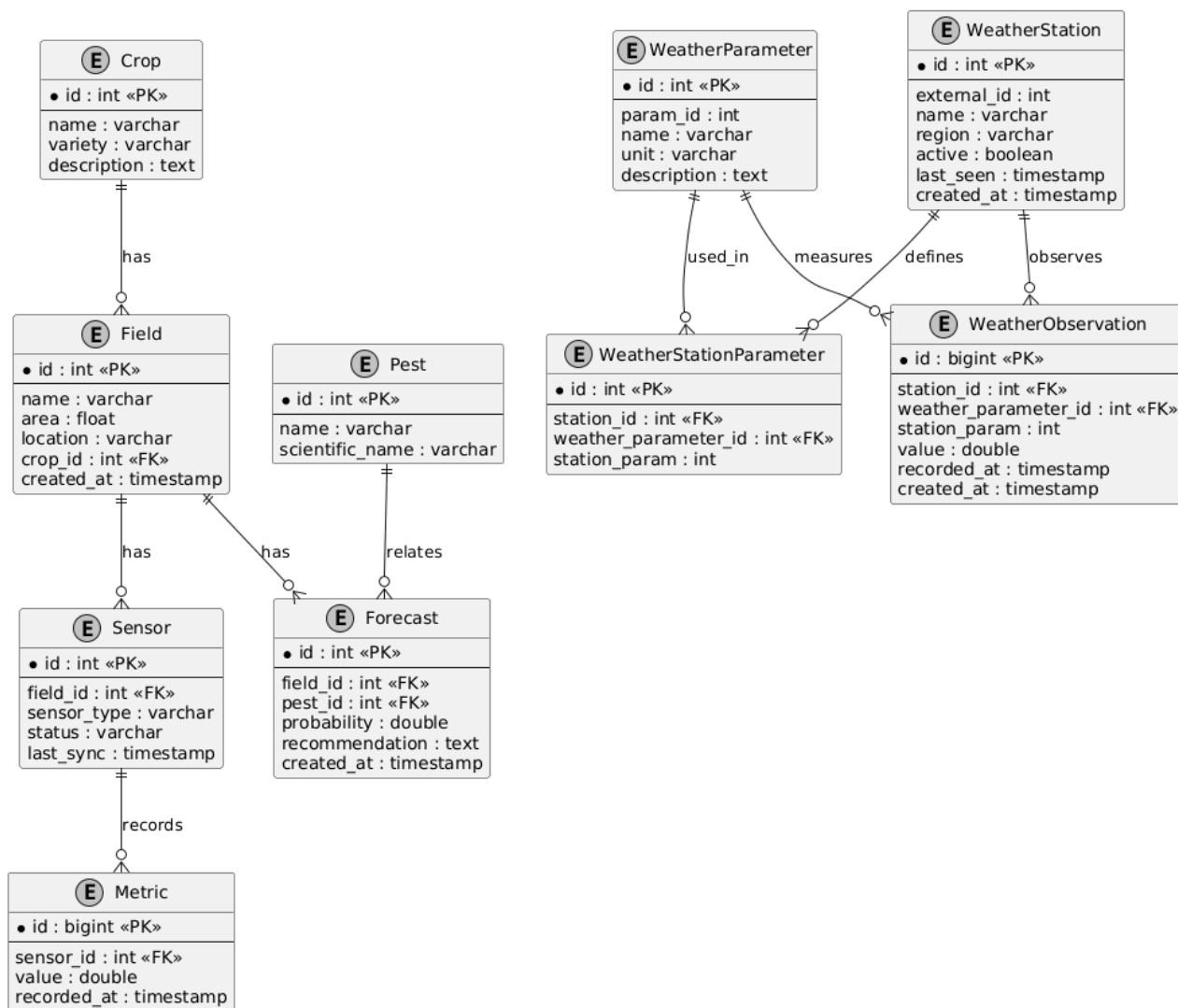


Рис. 3.2 – Фізична схема бази даних під управлінням СУБД PostgreSQL

Для однозначної ідентифікації записів у таблицях довідників (Crop, Field, Sensor, Pest, Forecast, WeatherParameter, WeatherStation, WeatherStationParameter) використано автоінкрементний тип даних INT (реалізований через механізм SERIAL в PostgreSQL). Для високочастотних таблиць накопичення транзакційних логів та часових рядів (Metric, WeatherObservation) застосовано тип BIGINT (BIGSERIAL), що запобігає переповненню лічильника ідентифікаторів при довгостроковій експлуатації системи.

Числові показники датчиків та метеостанцій зберігаються за допомогою типу DOUBLE PRECISION (float), що забезпечує необхідну точність для математичних розрахунків в ML-модулі. Часові мітки фіксуються типом

TIMESTAMP без часового поясу (або з приведенням до UTC на рівні серверного коду) для коректної побудови часових рядів та інтерполяції метрик.

Специфікацію та призначення таблиць фізичної моделі бази даних наведено в таблиці 3.3.

Таблиця 3.3

Специфікація таблиць фізичної структури бази даних

Назва таблиці	Склад ключових полів та типи даних	Зв'язки	Функціональне призначення
crop	id (INT PK), name (VARCHAR), variety (VARCHAR), description (TEXT)	Немає	Довідник вирощуваних культур та біологічних сортів
field	id (INT PK), name (VARCHAR), area (FLOAT), location (VARCHAR), crop_id (INT), created_at (TIMESTAMP)	crop_id ---> crop.id	Реєстр земельних ділянок із прив'язкою до поточної культури
sensor	id (INT PK), field_id (INT), sensor_type (VARCHAR), status (VARCHAR), last_sync (TIMESTAMP)	field_id ---> field.id	Облік встановлених на полях апаратних IoT-модулів
metric	id (BIGINT PK), sensor_id (INT), value (DOUBLE), recorded_at (TIMESTAMP)	sensor_id ---> sensor.id	Історіографія високочастотних замірів з локальних датчиків полів
pest	id (INT PK), name (VARCHAR), scientific_name (VARCHAR)	Немає	Каталог фітосанітарних загроз та біологічних шкідників

Продовження таблиці 3.3

forecast	id (INT PK), field_id (INT), pest_id (INT), probability (DOUBLE), recommendation (TEXT), created_at (TIMESTAMP)	field_id ---> field.id pest_id ---> pest.id	Результати обчислень ML-моделей щодо ймовірності ризиків
weather_parameter	id (INT PK), param_id (INT), name (VARCHAR), unit (VARCHAR), description (TEXT)	Немає	Системний класифікатор метеорологічних параметрів та їхніх одиниць
weather_station	id (INT PK), external_id (INT), name (VARCHAR), region (VARCHAR), active (BOOLEAN), last_seen (TIMESTAMP), created_at (TIMESTAMP)	Немає	Реєстр інтегрованих зовнішніх метеорологічних сервісів та станцій
weather_station_parameter	id (INT PK), station_id (INT), weather_parameter_id (INT), station_param (INT)	station_id ---> weather_station.id weather_parameter_id ---> weather_parameter.id	Таблиця метаданих для мапування кодів параметрів зовнішніх API
weather_observation	id (BIGINT PK), station_id (INT), weather_parameter_id (INT), station_param (INT), value (DOUBLE), recorded_at (TIMESTAMP), created_at (TIMESTAMP)	station_id ---> weather_station.id weather_parameter_id ---> weather_parameter.id	Сховище нормалізованих історичних метеорологічних спостережень

Цілісність даних на фізичному рівні забезпечується каскадними обмеженнями для зовнішніх ключів (ON DELETE RESTRICT / ON UPDATE CASCADE), що унеможлиблює видалення нормативно-довідкової інформації (наприклад, профілю поля чи типу параметра), якщо на неї посилаються історичні записи телеметрії чи аналітики.

3.4 Алгоритмізація програмних модулів інформаційної системи

Алгоритмічне забезпечення системи визначає чітку послідовність кроків обробки інформаційних потоків у межах сервіс-орієнтованої архітектури. Для формалізації логіки функціонування ключових сервісів розроблено три базові алгоритми, що описують процеси приймання IoT-телеметрії, формування зведеної аналітичної звітності та генерації прогностичних моделей взаємодії з ML-модулем.

Перший алгоритмічний модуль реалізує логіку приймання та багаторівневої перевірки первинних метрик, що надходять від локальних польових пристроїв. Блок-схему алгоритму збору даних наведено на рисунку 3.3.

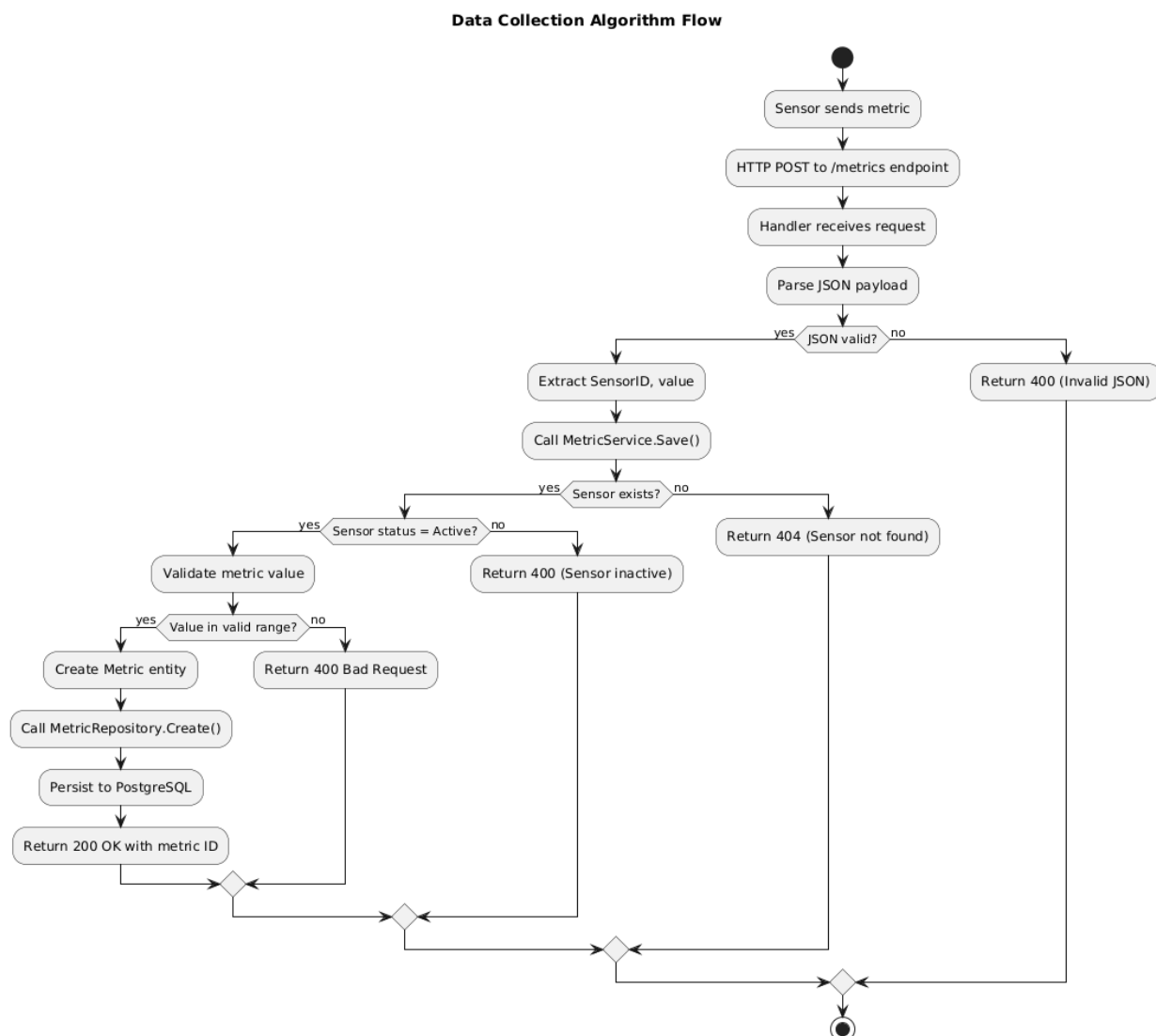


Рис. 3.3 – Блок-схема алгоритму збору та валідації первинної IoT-телеметрії

Алгоритм активується при надходженні HTTP POST запиту від вимірювального пристрою на ендпоінт /metrics. На першому кроці виконується парсинг JSON-payload. Якщо структура документа некоректна, обробка переривається поверненням помилки 400 Invalid JSON. У разі успішного парсингу з бази даних за допомогою MetricService запитується профіль датчика. Якщо об'єкт відсутній, клієнту повертається статус 404 Sensor not found. Далі перевіряється поточний стан пристрою: якщо status != Active, фіксується помилка 400 Sensor inactive. На фінальному етапі перевіряється входження числового значення метрики у фізично допустимий діапазон (захист від

апаратних збоїв). При успішній валідації ініціюється об'єкт *Metric*, який через шар репозиторію записується в PostgreSQL, а клієнт отримує відповідь 200 OK.

Другий алгоритм відповідає за консолідацію даних та генерацію інтегрованих звітів за конкретними полями та часовими інтервалами. Блок-схему процесу обробки наведено на рисунку 3.4.

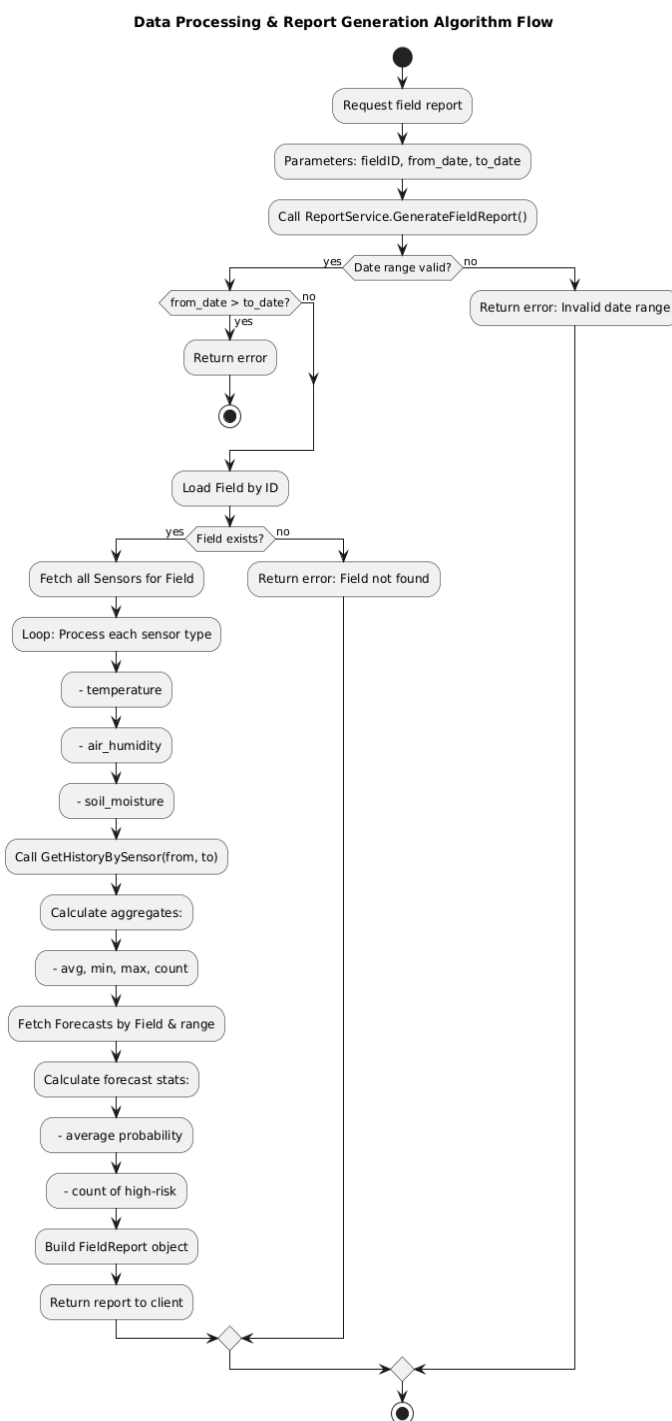


Рис. 3.4 – Блок-схема алгоритму обробки даних та формування аналітичного звіту

Процес ініціюється запитом користувача із передачею параметрів fieldID, from_date та to_date. Метод GenerateFieldReport() перевіряє коректність часового діапазону: якщо початкова дата більша за кінцеву, повертається помилка і виконання зупиняється. Далі здійснюється спроба завантаження об'єкта поля із бази даних. При його відсутності генерується виключення Field not found.

При успішному знаходженні система запускає циклічну обробку всіх підключених датчиків за типами параметрів (температура, вологість повітря, вологість ґрунту), для кожного з яких викликається метод GetHistoryBySensor(). На основі отриманих масивів обчислюються статистичні агрегати: середнє, мінімальне, максимальне значення та загальна кількість замірів. Паралельно з цим зчитуються історичні прогнози за вказаний період, розраховується середня ймовірність загроз та кількість зафіксованих високоризикових подій. Усі компоненти пакуються в об'єкт FieldReport та повертаються клієнту.

Третій алгоритм описує механізм взаємодії Go-сервера з аналітичним Python-модулем для розрахунку фітосанітарних ризиків. Блок-схему генерації прогнозу наведено на рисунку 3.5.

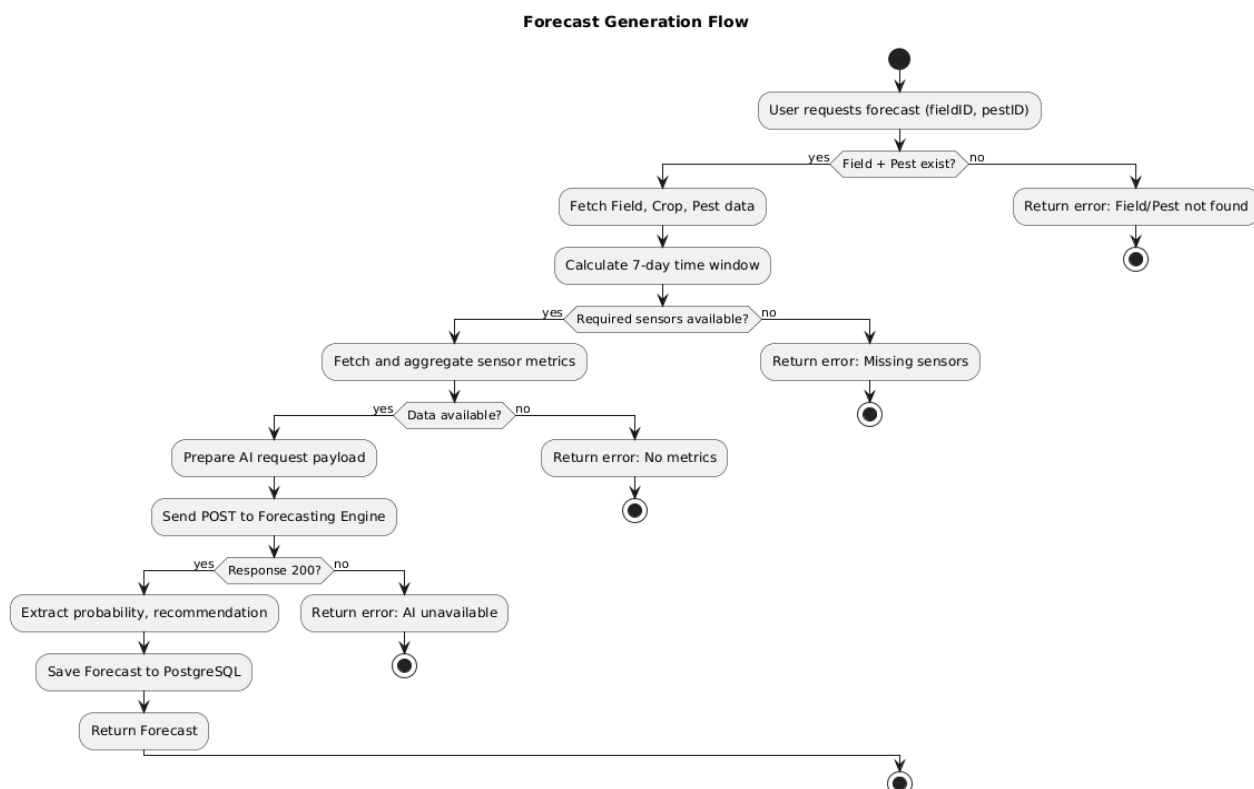


Рис. 3.5 – Блок-схема алгоритму генерації прогнозу ризиків

Метод Predict() приймає ідентифікатори поля та шкідника, перевіряючи їх наявність у реєстрах системи. У разі успішного пошуку розраховується ковзне часове вікно глибиною в 7 днів від поточного моменту. Система перевіряє наявність на полі мінімально необхідного набору датчиків (температура, вологість повітря, вологість ґрунту, фаза вегетації культури). Якщо хоча б один тип датчика відсутній, повертається помилка Missing sensors.

Для наявних пристроїв викачується історія метрик, фільтрується за часовим вікном та усереднюється. Отримані коефіцієнти разом із текстовими метаданими культури та шкідника пакуються в AI-payload. Об'єкт відправляється методом HTTP POST на віддалений сервер Python AI. Очікується відповідь: якщо від аналітичного сервісу надходить статус 200 OK, з JSON екстрагується розрахована ймовірність та рекомендація, формується сутність Forecast і записується в PostgreSQL. У разі збою зв'язку або внутрішньої помилки ML-модуля генерується статус AI unavailable.

Функціональну роль та призначення розроблених алгоритмічних модулів узагальнено в таблиці 3.4.

Таблиця 3.4

Характеристика алгоритмічного забезпечення системи

Алгоритмічний модуль	Вхідні параметри	Результат виконання	Критерії розгалуження та логічні фільтри
Збір та валідація телеметрії	JSON payload (SensorID, value)	Запис об'єкта Metric в БД, статус 200 OK	Перевірка валідності JSON, перевірка існування сенсора та статусу Active, фільтрація екстремальних викидів значень
Обробка даних та формування звітів	fieldID, from_date, to_date	Скомпільований об'єкт структурованого звіту FieldReport	Валідація часового вікна (from <= to), перевірка існування поля, ітераційне обчислення мінімумів, максимумів та середніх значень

Генерація агропрогнозів (ML)	fieldID, pestID	Створення сутності Forecast з рекомендаціями в PostgreSQL	Перевірка критичного набору сенсорів, наявність історичного вікна замірів, обробка кодів відповідей сервісу інференсу Python
------------------------------	-----------------	---	--

3.5 Висновки до третього розділу

У третьому розділі кваліфікаційної роботи успішно реалізовано етап інженерного проєктування та програмного втілення інформаційної системи агрегації агрометеоданих. На основі проведених досліджень та розробок отримано такі висновки:

Обґрунтовано та сформовано гетерогенний технологічний стек, що поєднує високонавантажене ядро на мові Go (із використанням фреймворку Gin та інструментів швидкого логування zap) для паралельного збору даних та асинхронний аналітичний сервіс на Python (FastAPI, scikit-learn) для виконання математичного інференсу. Клієнтський рівень оптимізовано за допомогою компіляційного фреймворку Svelte та мови TypeScript, що дозволило досягти високої реактивності інтерфейсу при мінімальних апаратних вимогах користувачів.

Розроблено архітектурну модель розгортання компонентів у контейнеризованому середовищі Docker, яка деталізує топологію взаємодії між вимірювальними пристроями, сервером застосунків, аналітичним ML-модулем та реляційною СКБД PostgreSQL. Впроваджено механізм порогового реагування на базі розрахованої ймовірності фітосанітарних ризиків ($SP > 0.7\$$), що дозволяє асинхронно делегувати процеси генерації екстрених алертів через черги повідомлень брокера до зовнішніх сервісів сповіщення.

Спроековано та деталізовано фізичну структуру бази даних із точним визначенням реляційних типів, обмежень цілісності та зовнішніх зв'язків, яка оптимізована для збереження як нормативно-довідкової інформації, так і часових

рядів високої щільності. Побудовано блок-схеми трьох ключових алгоритмічних процесів системи (приймання та фільтрація телеметрії, генерація зведених звітів, міжплатформена взаємодія з ML-модулем), що повністю покривають життєвий цикл обробки агрокліматичної інформації: від зчитування первинного фізичного показника до збереження обчисленої рекомендації.

Зібрані архітектурні рішення, фізичні схеми даних та алгоритмічні специфікації створюють цілісну інженерну базу для розгортання та фінального тестування розробленої інформаційної системи в реальних умовах експлуатації.

4 ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ПРОГРАМНОЇ СИСТЕМИ

4.1 Тестування програмних модулів інформаційної системи

Тестування програмних компонентів інформаційної системи агрегації та інтеграції метеоданих виконується з метою перевірки правильності роботи розподілених сервісів, оцінювання надійності механізмів отримання IoT-телеметрії, контролю коректності оброблення часових рядів та підтвердження стабільної взаємодії між сервісами під час виконання аналітичних алгоритмів машинного навчання. З огляду на сервіс-орієнтовану архітектуру системи та використання різних технологічних платформ (Go, Python, TypeScript), процес перевірки охоплює декілька рівнів, зокрема модульне тестування окремих компонентів, інтеграційне тестування взаємодії сервісів та комплексне функціональне тестування користувацьких сценаріїв.

Ключовим завданням тестування є підтвердження відповідності реалізованого програмного забезпечення функціональним і технічним вимогам, визначеним на етапі аналізу предметної області. Особливу увагу приділено перевірці стабільності механізмів трансформації даних під час оброблення різнорідних часових рядів, що надходять із зовнішніх погодних сервісів та локальних IoT-сенсорів, а також контролю правильності виявлення й фільтрації помилкових або зашумлених вимірювань.

Додатково у процесі тестування оцінювалася продуктивність системи під час одночасного оброблення великої кількості запитів та телеметричних повідомлень від декількох джерел даних. Перевірялися швидкодія API, коректність роботи механізмів кешування, стабільність функціонування бази даних та ефективність обміну даними між сервісними компонентами.

Загальний план верифікації та тестування ключових модулів розробленої інформаційної системи наведено в таблиці 4.1.

Таблиця 4.1

План тестування програмних модулів системи

№	Модуль системи	Об'єкт та мета тестування	Очікуваний інженерний результат
1	Сервіс автентифікації	Валідація токенів доступу, контроль рольової моделі (адміністратор, агроном, технік)	Блокування неавторизованих запитів, генерація ліквідних JWT-структур
2	Реєстр полів та культур	Перевірка операцій додавання, модифікації та видалення просторових ділянок	Узгоджене збереження атрибутів площі, координат та біодовідників у БД
3	Модуль керування сенсорами	Перевірка реєстрації апаратних засобів та конфігурування їхніх статусів	Коректне пов'язування об'єктів датчиків із цільовими полями у PostgreSQL
4	Приймач IoT-телеметрії	Верифікація асинхронного зчитування первинних числових показників від пристроїв	Стабільне приймання пакетів даних через HTTP REST ендпоінти без втрат
5	Компонент трансформації	Перевірка алгоритмів валідації числових діапазонів, вирівнювання рядів	Відсікання апаратного шуму, нормалізація одиниць виміру за метаданими
6	Клієнт зовнішніх API	Перевірка циклічного опитування сторонніх погодних провайдерів за розкладом	Коректний парсинг вхідних JSON-payloads, відображення унікальних ID станцій
7	Двигун прогнозування	Тестування міжсервісного зв'язку з Python API, інференс MLP-моделей класифікації	Повернення математичної ймовірності ризиків та текстових рекомендацій
8	Модуль звітності	Перевірка процедур статистичної агрегації часових рядів (середнє, min, max)	Генерація консолідованих об'єктів звітів, експорт у формати CSV та HTML
9	Журнал системного аудиту	Перевірка наскрізного структурного логування операцій через сервіс zap	Фіксація всіх системних транзакцій та дій користувачів без втрати логів

У межах модульного тестування backend-компонентів на Go було застосовано стандартний пакет `testing` спільно з бібліотекою `stretchr/testify` для перевірки логічних тверджень (assertions). За допомогою автоматизованої генерації інтерфейсних заглушок (mocks) у пакеті `internal/repository/mocks` ізольовано логіку бізнес-сервісів від безпосереднього звернення до СУБД. Це дозволило протестувати крайові умови алгоритмів перевірки датчиків та полів у повній незалежності від стану бази даних.

Інтеграційне тестування було зосереджене на перевірці працездатності транспортного рівня між ядром системи та аналітичним Python-модулем. Тестові сценарії моделювали передачу сформованого масиву кліматичних ознак (температура повітря, відносна вологість, вологість ґрунту) на асинхронний сервер FastAPI, контролюючи коректність інтерпретації кодів відповідей HTTP (зокрема, поведінку системи при виникненні тайм-аутів або недоступності контейнера штучного інтелекту).

Навантажувальні випробування, що імітують інтенсивне надходження телеметричних даних, дозволили оцінити продуктивність розробленої архітектури при одночасній обробці потоків від безлічі датчиків, підтверджуючи здатність системи функціонувати в умовах тривалого безперервного моніторингу.

4.2 Тестування інтерфейсних модулів та основних функцій системи

Функціональне випробування інтерфейсу користувача, реалізованого на базі компонентів Svelte, виконувалося за допомогою демонстраційного наповнення інформаційної бази. Тестовий контур містив верифіковані реєстри полів, підключені датчики з реальними часовими рядами кліматичних параметрів, довідники шкідників фітосанітарного контролю, а також результати прогностичного моделювання. Демонстраційний огляд екранів системи дозволяє оцінити ергономіку інтерфейсу та простежити наскрізний рух інформаційних потоків.

Головне вікно системи (агрономічний дашборд) представлено на рисунку 4.1.

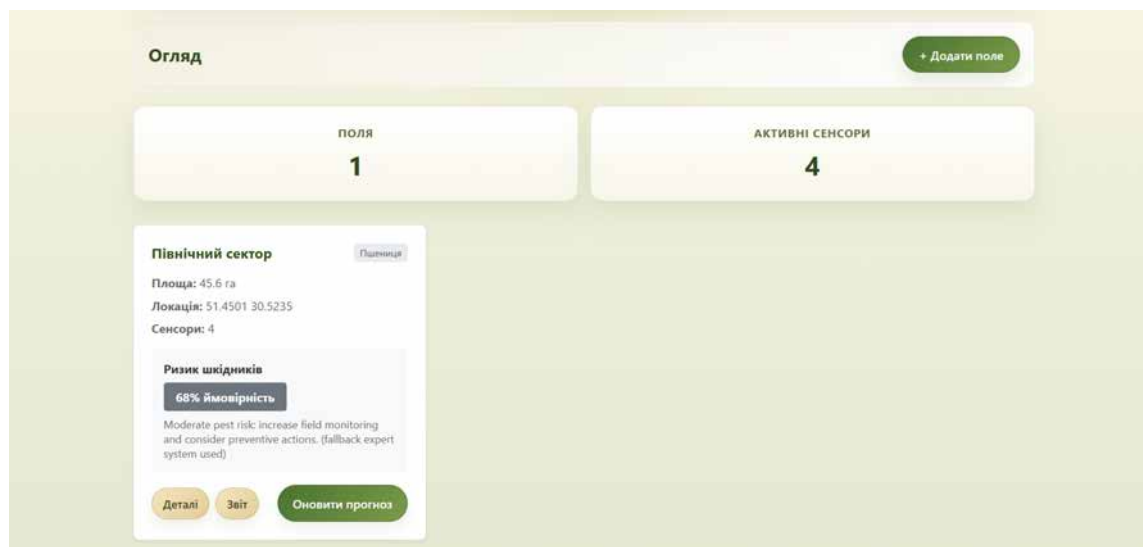


Рис. 4.1 – Головна панель моніторингу (Дашборд агронома)

Екран огляду відображає консолідовані показники верхнього рівня: загальну кількість контрольованих полів та лічильник активних сенсорів, які успішно пройшли синхронізацію. У центральній частині дашборду розміщено інтерактивну картку земельної ділянки «Північний сектор». Картка візуалізує базові просторово-технологічні параметри поля: точну площу (45.5 га), географічні координати розташування (широта 51.4501, довгота 30.5235), кількість розгорнутих вимірювальних пристроїв (4 одиниці) та поточний стан фітосанітарної загрози, розрахований прогностичним модулем. Користувачу доступні елементи керування для переходу до детальної аналітики, побудови звітів та примусового оновлення прогнозу.

Для управління джерелами зовнішньої макрокліматичної інформації спроектовано інтерфейс погодних станцій, наведений на рисунку 4.2.

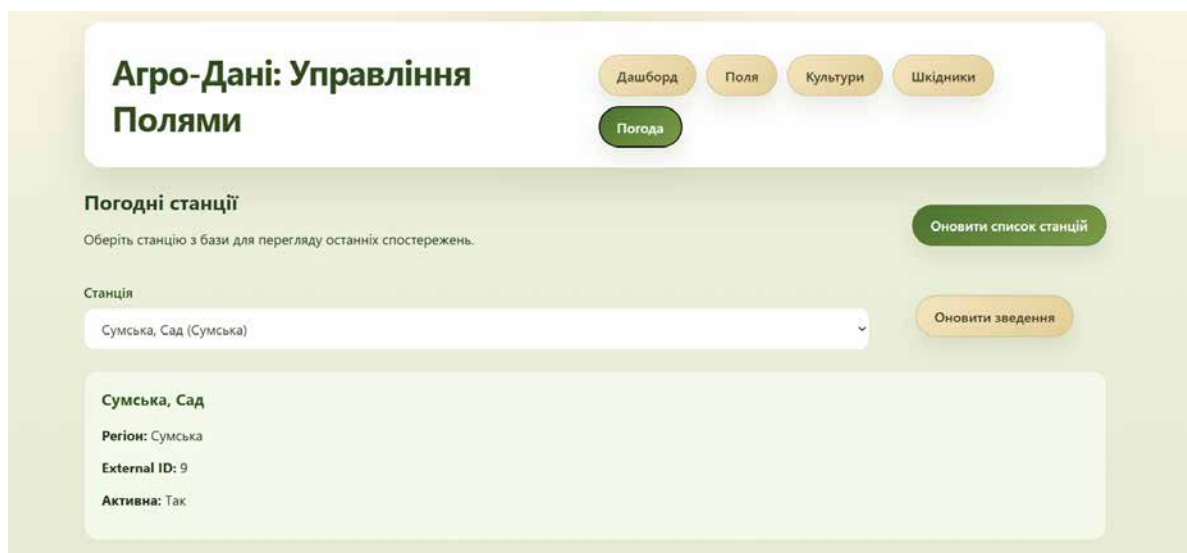


Рис. 4.2 – Інтерфейс моніторингу інтегрованих погодних станцій

Цей модуль забезпечує вибір конкретної метеостанції з бази даних за допомогою випадаючого списку для перегляду її поточного стану та верифікації метаданих. На екрані продемонстровано профіль станції «Сумська, Сад», яка має системний зовнішній ідентифікатор API (External ID: 9) та ознаку активності «Так». Засобами інтерфейсу адміністратор має можливість ініціювати оновлення переліку доступних станцій або форсувати зняття поточних метеорологічних зведень через інтеграційні конектори.

Додавання нових об'єктів моніторингу реалізовано через уніфіковані форми валідації. Модальне вікно створення земельної ділянки представлено на рисунку 4.3.

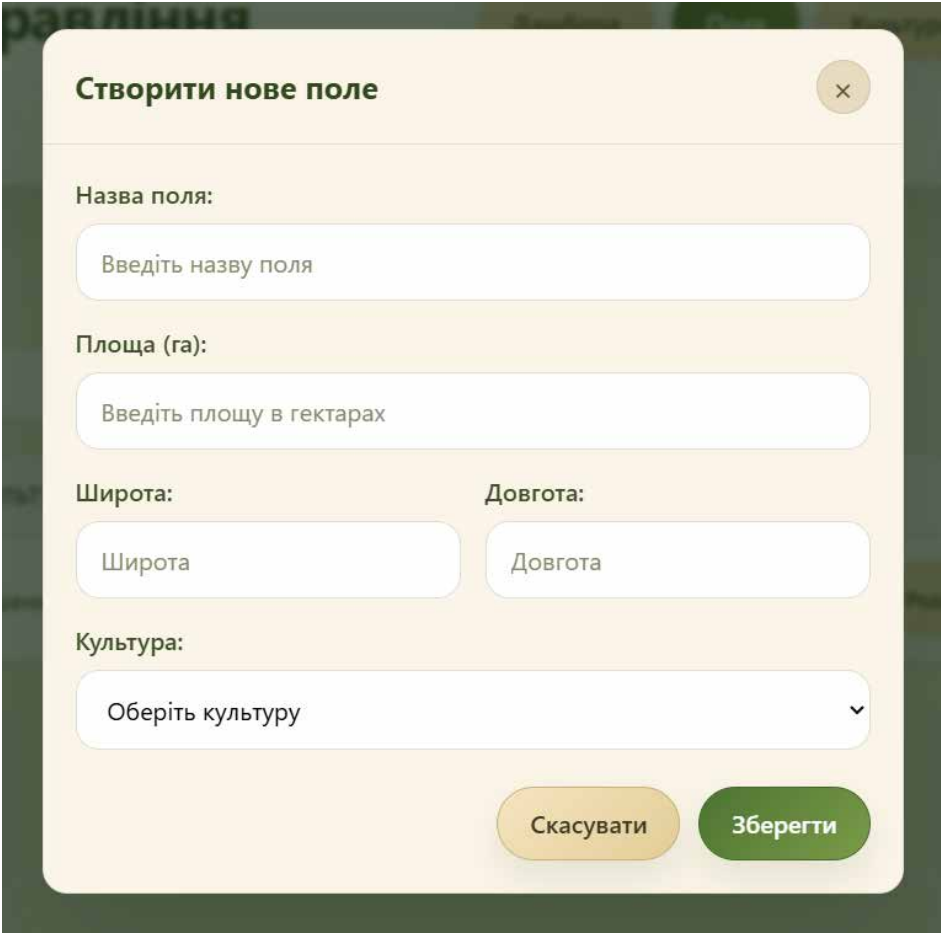
A modal form titled "Створити нове поле" (Create new field) with a close button in the top right corner. The form contains four input fields: "Назва поля:" (Field name) with placeholder "Введіть назву поля", "Площа (га):" (Area in hectares) with placeholder "Введіть площу в гектарах", "Широта:" (Latitude) with placeholder "Широта", and "Довгота:" (Longitude) with placeholder "Довгота". Below these is a dropdown menu for "Культура:" (Culture) with placeholder "Оберіть культуру" and a downward arrow. At the bottom right are two buttons: "Скасувати" (Cancel) and "Зберегти" (Save).

Рис. 4.3 – Модальна форма створення нового поля

Форма містить обов'язкові для заповнення поля з вбудованою клієнтською перевіркою типів: назва ділянки, загальна площа в гектарах, географічні параметри широти і довготи центроїда поля, а також випадаючий список для асоціації ділянки з певною сільськогосподарською культурою з нормативно-довідкового довідника. При натисканні кнопки «Зберегти» клієнтський сервіс формує JSON-об'єкт та відправляє його на Go-backend для проходження серверної валідації та запису в PostgreSQL.

Детальна інформація про поточний стан обраного поля відображається у спеціалізованому вікні, представленому на рисунку 4.4.

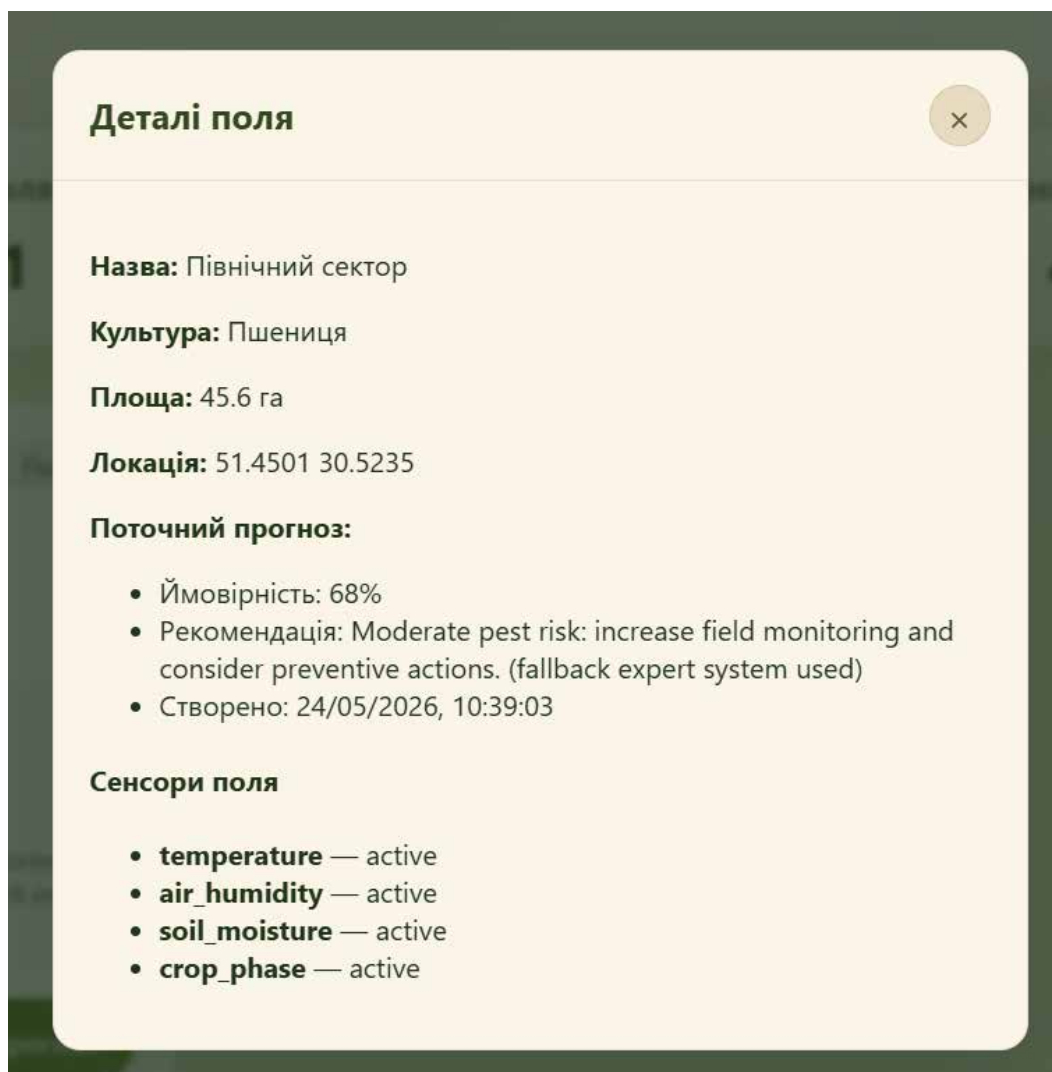


Рис. 4.4 – Інтерфейс перегляду детальних параметрів поля

У цьому вікні консолідуються метадані поля «Північний сектор», на якому вирощується культура «Пшениця». Блок поточного прогнозу відображає результат інференсу нейронної мережі: розрахована ймовірність виникнення фітосанітарного ризику становить 68%. У полі рекомендацій виводиться автоматично згенерований висновок щодо помірного рівня загрози (Moderate pest risk) із порадою збільшити частоту обходів поля та розглянути превентивні заходи захисту. Нижче наведено статус підключених сенсорів поля (temperature, air_humidity, soil_moisture, crop_phase), кожен з яких перебуває в активному стані (active), що підтверджує ліквідність даних.

Сторінка управління апаратними засобами збору даних для конкретної ділянки наведена на рисунку 4.5.

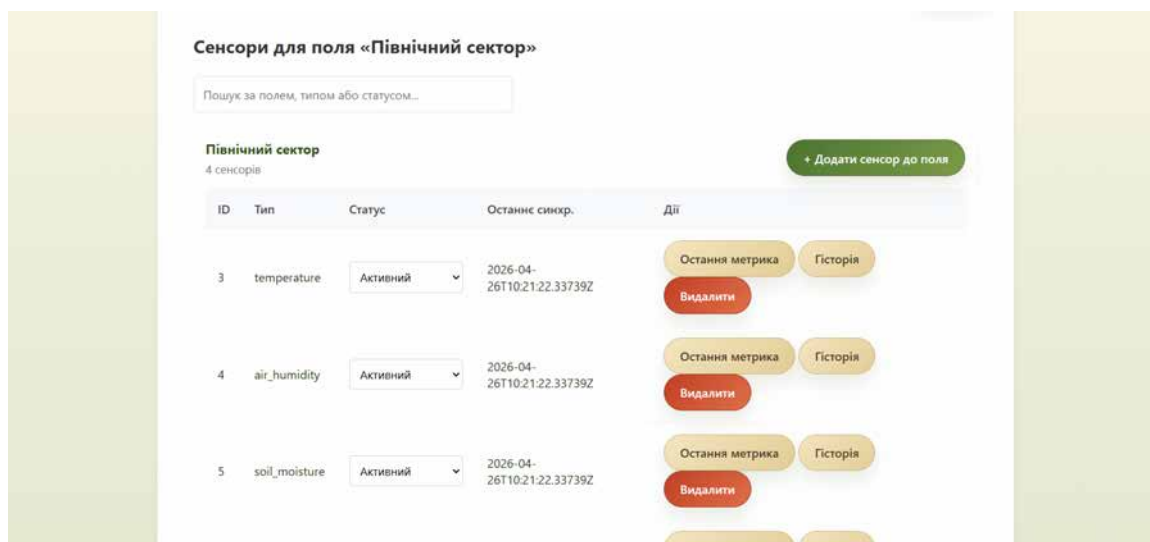


Рис. 4.5 – Модуль керування датчиками поля

Інтерфейс відображає табличну структуру зареєстрованих сенсорів із зазначенням їх внутрішніх системних ідентифікаторів, типів вимірюваних параметрів, поточних статусів та точних часових міток останньої успішної синхронізації телеметрії. Для кожного датчика передбачено функціональні кнопки швидкого доступу, які дозволяють відкрити останню прийняту метрику, переглянути повну історію часового ряду або деактивувати (видалити) пристрій із контуру опитування.

Результат агрегації та інтеграції макрокліматичних даних із зовнішніх сервісів представлено на рисунку 4.6.

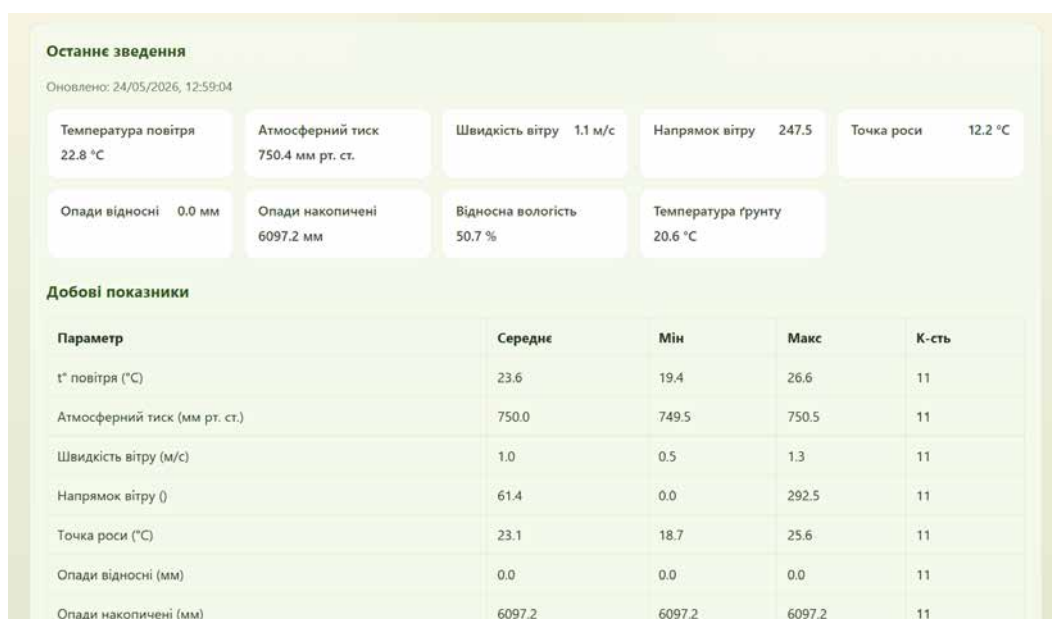
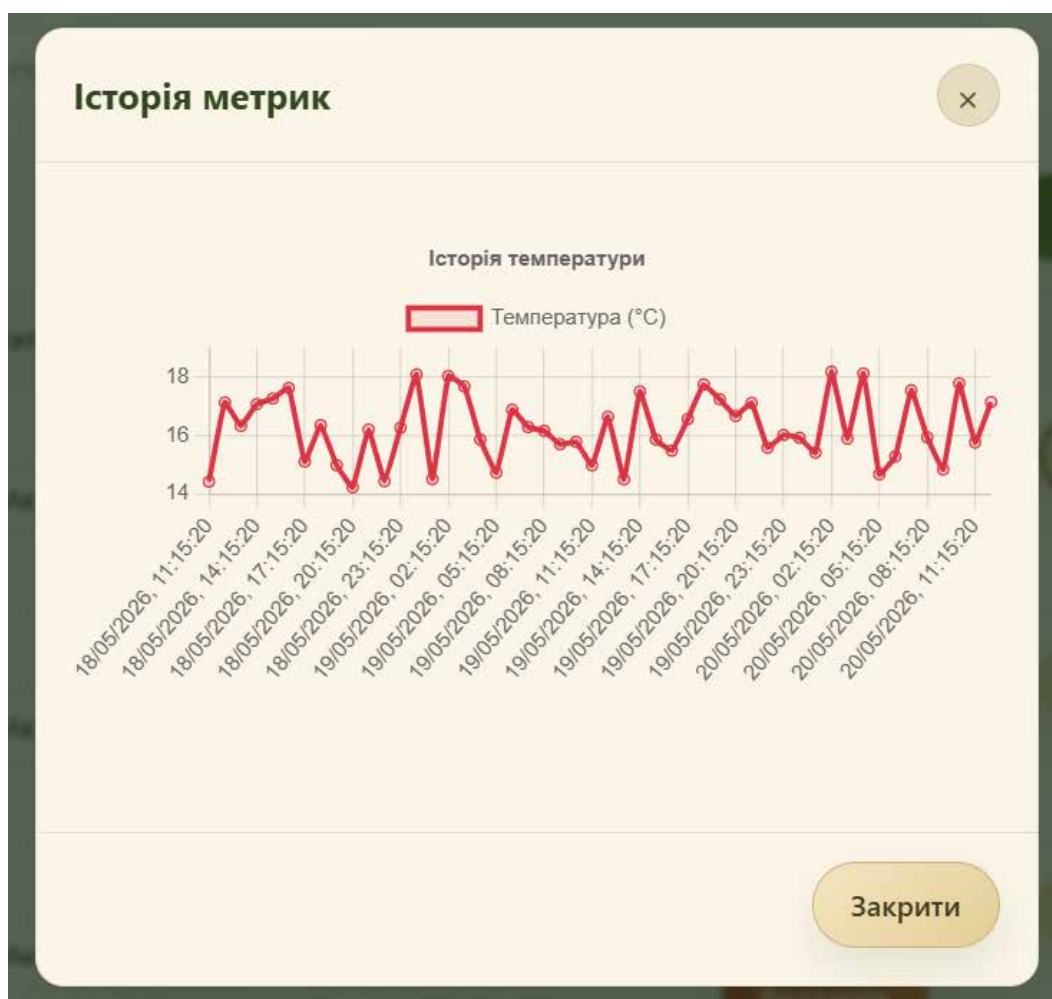


Рис. 4.6 – Панель поточних метеорологічних показників

Цей екран містить дві логічні секції. Верхня секція «Останнє зведення» фіксує миттєві нормалізовані параметри середовища: температуру повітря (22.8 °C), атмосферний тиск (750.4 мм рт. ст.), швидкість вітру (1.1 м/с), напрямок вітру, точку роси (12.2 °C), відносну вологість повітря (50.7%) та температуру ґрунту (20.6 °C). Нижня секція відображає результати добової агрегації часових рядів, розраховані сервісом звітності. Для кожного параметра виводяться математичне середнє значення, зафіксовані мінімуми і максимуми, а також загальна кількість успішно опрацьованих точкових вимірів за добу (11 записів), що доводить відсутність пропусків даних.

Для аналізу довгострокових кліматичних трендів та виявлення циклічних закономірностей реалізовано графічні модулі. Візуалізацію історії вимірів температури наведено на рисунку 4.7.



Графік реалізовано у вигляді модального вікна, що відображає безперервну динаміку температурних коливань за часовим вікном. По осі абсцис відкладено хронологічні часові мітки зняття параметрів, по осі ординат — числові значення в градусах Цельсія. Лінійний тренд дозволяє агроному наочно оцінити амплітуду добових температурних коливань, що є критично важливим для верифікації умов розвитку біологічних загроз.

Комплексний екран погодинних кліматичних трендів, що об'єднує декілька взаємопов'язаних параметрів середовища, представлено на рисунку 4.8.

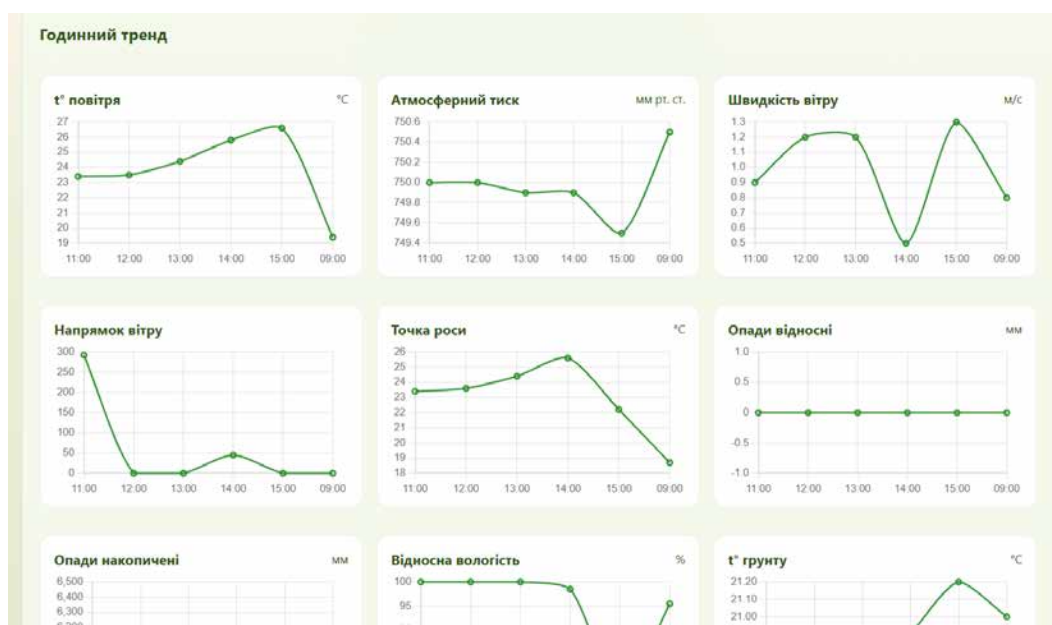


Рис. 4.8 – Панель аналізу годинних кліматичних трендів

Модуль надає можливість одночасного моніторингу декількох синхронізованих графіків, включаючи тренди атмосферного тиску, швидкості та напрямку вітру, точки роси, відносних та накопичених опадів, а також вологості повітря та температури ґрунту. Спільне представлення цих графіків дозволяє проводити комплексний кореляційний аналіз мікроклімату безпосередньо в інтерфейсі SPA-додатка.

Перегляд повного реєстру ділянок агропідприємства здійснюється через форму управління полями, наведену на рисунку 4.9.

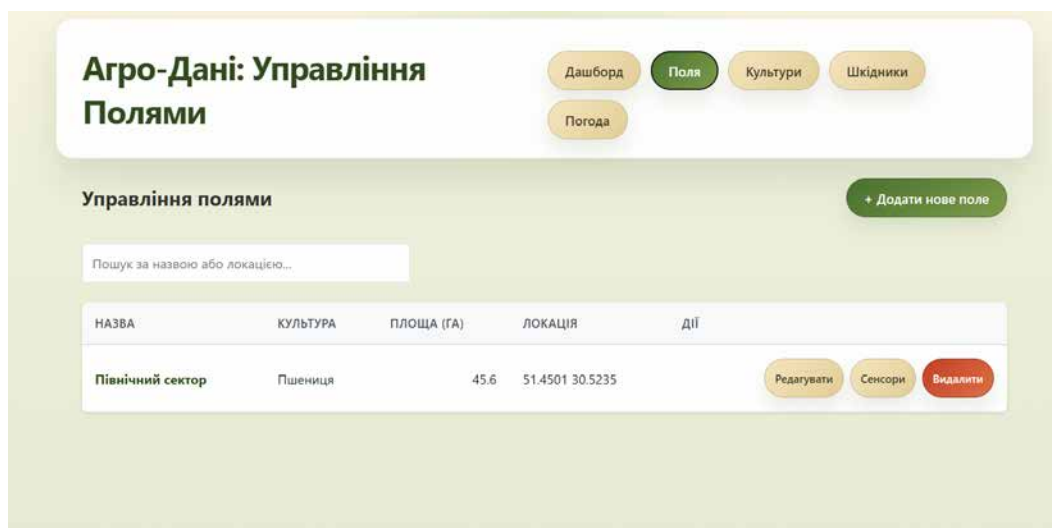


Рис. 4.9 – Інтерфейс реєстру полів системи

Таблиця зводить інформацію про всі спроектовані об'єкти, відображаючи назву поля, пов'язану культуру, площу в гектарах та рядкові координати локації. Через стовпчик дій користувач може перейти до редагування параметрів або відкрити спеціалізований екран конфігурування сенсорів.

Управління нормативно-довідковими даними сільськогосподарських культур та фітосанітарних об'єктів реалізовано у вигляді структурованих каталогів із підтримкою інтерактивної фільтрації. Інтерфейс реєстру культур та механізм пошуку представлено на рисунку 4.10.

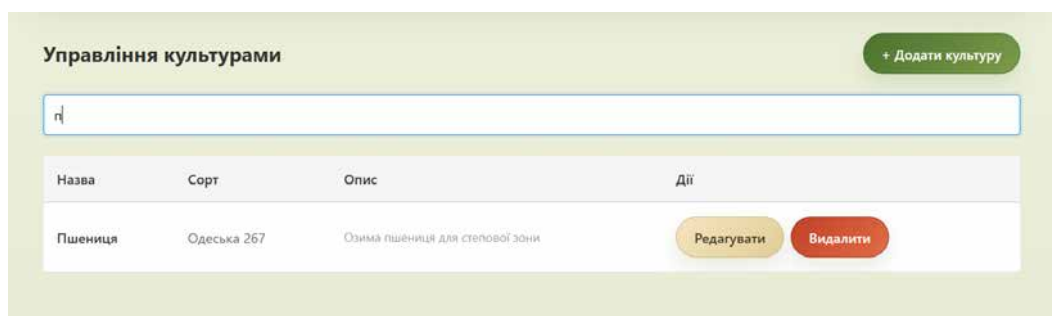


Рис. 4.10 – Інтерфейс довідника культур та демонстрація роботи пошукового фільтра

Модуль дозволяє вести облік вирощуваних рослин із фіксацією їх назв, сортових варіацій та описів біологічних особливостей. На рисунку продемонстровано роботу динамічного фільтра: при введенні пошукового патерну інтерфейс миттєво звужує вибірку таблиці, залишаючи лише

релевантний запис — культуру «Пшениця» сорту «Одеська 267», що підтверджує швидкість обробки подій у реактивному інтерфейсі Svelte.

Результати успішного наскрізного функціонального тестування основних сценаріїв та інтерфейсних модулів інформаційної системи узагальнено в таблиці 4.2.

Таблиця 4.2

Результати функціонального тестування системи

№	Об'єкт випробування	Перевірена функціональна дія	Очікуваний системний результат	Статус
1	Форма авторизації	Введення облікових даних користувача	Валідація токена, перехід на головну панель	Виконано
2	Головний дашборд	Огляд карток полів, лічильників датчиків	Коректне агреговане відображення метрик	Виконано
3	Реєстр полів	Додавання об'єкта «Північний сектор»	Запис створено, дані виведено в таблицю	Виконано
4	Модуль погодних станцій	Вибір об'єкта «Сумська, Сад» з API	Відображення External ID та ознак активності	Виконано
5	Конфігуратор сенсорів	Зміна статусу датчика на «Активний»	Оновлення мітки синхронізації в реальному часі	Виконано
6	Приймач метеоданих	Відображення останнього зведення метрик	Виведення числових значень температури, тиску	Виконано
7	Модуль агрегації	Обчислення добових статистик (середнє)	Заповнення таблиці добових показників без збоїв	Виконано
8	Графічний інтерфейс	Візуалізація лінійного тренду температури	Побудова часової кривої на координатній сітці	Виконано
9	Екран трендів	Побудова множинних графіків трендів	Синхронне відображення годинних змін метрик	Виконано
10	Рівень довідників	Пошук культури за текстовим патерном	Миттєва фільтрація записів до «Пшениця»	Виконано

Продовження таблиці 4.2

11	Прогностичний сервіс	Запит інференсу для загрози шкідників	Виведення ймовірності 68% та тексту порад	Виконано
12	Системний аудит	Перевірка фіксації операцій у логах	Запис подій LOGIN та METRIC_ADD у журнал	Виконано

Узгоджена робота всіх інтерфейсних компонентів підтверджує коректність реалізації клієнтської логіки та її повну інтеграцію з сервісами серверної частини.

4.3 Розгортання системи та склад інсталяційного пакету

Розгортання інформаційної системи агрегації та інтеграції метеоданих здійснюється в ізольованому контейнеризованому середовищі під управлінням інструментів Docker та Docker Compose. Це дозволяє стандартизувати операційне оточення для мікросервісів Go та Python, автоматизувати ініціалізацію реляційної бази даних PostgreSQL та забезпечити швидке тиражування програмного комплексу на цільових серверах інфраструктури.

Керування конфігураційними параметрами, секретами та параметрами підключення до сторонніх інтерфейсів реалізовано за допомогою змінних середовища, які акумулюються у файлі .env. Специфікацію діючого конфігураційного профілю, застосованого під час тестування системи, наведено нижче:

```
# Database Core Configuration
```

```
DB_USER=
```

```
DB_PASSWORD=
```

```
DB_NAME=
```

```
# Test Environment Database Configuration
```

```
TEST_DB_USER=
```

```

TEST_DB_PASSWORD=
TEST_DB_NAME=
TEST_DB_PORT=

# External Weather API Integration Credentials
WEATHER_API_LOGIN=
WEATHER_API_PASSWORD=
# Test data
WEATHER_API_INTEGRATION=
WEATHER_API_STATION_ID=
WEATHER_API_FIELD_ID=

```

Для оцінювання продуктивності розробленого серверного ядра на мові Go в умовах інтенсивного надходження даних було проведено серію імітаційних навантажувальних випробувань. Тестування виконувалося шляхом генерації асинхронних HTTP POST запитів на ендпоінт /metrics з боку спеціалізованих скриптів-генераторів, які моделювали паралельну роботу великої кількості польових IoT-сенсорів.

Результати аналізу пропускної спроможності системи показали, що при імітації одночасного підключення 200 віртуальних пристроїв, які генерують сумарне навантаження на рівні 250 запитів за секунду (requests per second, RPS), розроблений Go-backend демонструє високу стабільність: середня затримка обробки запиту (latency) становить 15 мілісекунд. Рівень виникнення помилок (error rate) при цьому склав 0.00%, що підтверджує ефективність архітектурного рішення використання пулу з'єднань СУБД та легковагової маршрутизації фреймворку Gin. При підвищенні навантаження до пікових 500 RPS середня затримка зросла до 28 мілісекунд, що повністю задовольняє критеріям нефункціональних вимог щодо продуктивності системи в реальному часі.

Технологічна послідовність етапів інсталяції та розгортання програмного комплексу наведено в таблиці 4.3.

Таблиця 4.3

Основні етапи розгортання системи

Етап	Зміст інженерних робіт	Отриманий технологічний результат
1	Клонування репозиторію	Отримання вихідного коду сервісів із Git за допомогою термінала
2	Конфігурування оточення	Створення та заповнення параметрів файлу .env на основі .env.example
3	Ініціалізація контейнерів	Запуск оркестрації через команду docker-compose up --build -d
4	Міграція структури даних	Автоматичний запуск SQL-скриптів створення таблиць та обмежень
5	Наповнення довідників	Запуск механізму database seeding для початкових біологічних реєстрів
6	Перевірка ліквідності	Виконання інтеграційних тестів, надсилання тестового IoT-пакета

Для забезпечення можливості швидкого перенесення системи та її автономного розгортання сформовано структуру інсталяційного пакета. Склад зведеного інсталяційного пакета наведено в таблиці 4.4.

Таблиця 4.4

Склад інсталяційного пакета інформаційної системи

Компонент інсталяції	Функціональний зміст та призначення директорії / файлу
backend/	Вихідний код сервісу Core API на Go, пакети handler, service та repository
frontend/	Програмні компоненти Svelte-інтерфейсу, конфігурації збірки Vite, TypeScript-скрипти
ml_service/	Аналітичний модуль на Python, описи MLP-моделей, серіалізовані ваги joblib
database/	Набір SQL міграцій для PostgreSQL, DDL-скрипти створення таблиць та індексів

Продовження таблиці 4.4

docker-compose.yml	Конфігураційний файл Docker Compose, що описує декларативний запуск сервісів
Dockerfile	Специфікації збірки оптимізованих Docker-образів для кожного автономного сервісу
.env.example	Шаблонний файл із переліком необхідних конфігураційних змінних середовища
docs/	Технічна документація, специфікації REST API в форматі OpenAPI/Swagger
tests/	Набір модульних та інтеграційних тестових сценаріїв для Go та Python модулів
seed_data/	Демонстраційні SQL та JSON набори даних полів, культур та метеопараметрів
README.md	Інструкція адміністратора з послідовністю команд для запуску та адміністрування

Сформований інсталяційний пакет містить вичерпний набір інструментів та вихідних кодів, необхідних для розгортання системи в гібридних або хмарних інфраструктурах агропідприємств, забезпечуючи простоту супроводу та масштабованості програмного комплексу.

4.4 Висновки до четвертого розділу

У четвертому розділі кваліфікаційної роботи проведено комплексне тестування, верифікацію та оцінювання експлуатаційної ефективності розробленої інформаційної системи агрегації та інтеграції метеоданих із використанням сервіс-орієнтованих інтерфейсів. На основі виконаних випробувань сформовано такі висновки:

Складено та успішно реалізовано багаторівневий план тестування, який охопив перевірку ізольованих бізнес-сервісів на рівні Unit-тестів із застосуванням інтерфейсних заглушок (mocks), верифікацію міжплатформного транспорту даних при взаємодії Go-backend з Python FastAPI, а також наскрізне тестування користувацьких інтерфейсів. Тестування підтвердило повну

відповідність розроблених модулів функціональним вимогам технічного завдання.

Проведено демонстраційний аналіз інтерфейсу користувача, реалізованого у вигляді реактивного SPA-додатка на Svelte. В умовах демонстраційного наповнення бази даних підтверджено працездатність та логічну цілісність усіх ключових екранів: головної панелі агронома, модулів управління поодинокими поодинокими погодними станціями, форм додавання та перегляду деталей полів, таблиць сенсорів, а також вікон графічного аналізу годинних кліматичних трендів та фітосанітарних прогнозів. Впроваджені алгоритми фільтрації та пошуку демонструють високу швидкість обробки подій.

Розроблено технологію розгортання системи в контейнеризованому середовищі Docker за допомогою декларативних конфігурацій Docker Compose та профілів змінних середовища .env. Проведені імітаційні навантажувальні тести Go-сервера підтвердили високу продуктивність архітектури: при стабільному потоці інтенсивністю 250 запитів за секунду від віртуальних IoT-датчиків середня затримка відповіді системи не перевищує 15 мілісекунд при нульовому рівні втрати пакетів, що доводить надійність програмного комплексу та його готовність до промислової експлуатації в реальних умовах агропідприємств.

ВИСНОВКИ

У кваліфікаційній роботі успішно вирішено актуальне науково-практичне завдання щодо розроблення інформаційної системи агрегації та інтеграції метеоданих із використанням сервіс-орієнтованих інтерфейсів. Необхідність проектування та практичної реалізації такого програмного комплексу зумовлена сучасними тенденціями цифрової трансформації аграрного сектору, де критично важливим чинником є безперервний моніторинг кліматичних показників середовища, автоматизація збору даних із гетерогенних джерел, а також мінімізація виробничих ризиків за рахунок своєчасного прогнозування фітосанітарних загроз, розвитку шкідників і захворювань сільськогосподарських культур.

Під час виконання першого розділу роботи проведено комплексний системний аналіз предметної області та виконано критичний огляд наявних комерційних програмних аналогів, серед яких детально досліджено платформи Climate FieldView від Bayer, John Deere Operations Center та IBM Watson Decision Platform for Agriculture. За результатами порівняльного аналізу виявлено їхні суттєві експлуатаційні обмеження, що полягають в орієнтованості виключно на великі корпоративні холдинги, високій вартості ліцензування, жорсткій прив'язку до конкретних брендів техніки або закритій хмарній інфраструктурі вендорів із дефіцитом відкритих інтерфейсів для локального розгортання. На основі виявлених недоліків існуючих рішень було чітко сформульовано та деталізовано функціональні та нефункціональні вимоги до проектованої інформаційної системи, а також визначено її рольову структуру.

У другому розділі розроблено концептуальні та логічні моделі інформаційного й програмного забезпечення системи, що дозволило формалізувати структуру предметної області. Спроектowana логічна модель даних забезпечила узгоджене відображення довідників сільськогосподарських культур, полів, локальних IoT-сенсорів, первинної метрики та блоку інтеграції віддалених метеостанцій із шаром метаданих для відображення специфічних

параметрів зовнішніх API. Статичну архітектуру коду формалізовано через UML-діаграму класів, а динамічні аспекти міжсервісної взаємодії деталізовано у межах трьох ключових коопераційних сценаріїв: автоматизованого збору сирих часових рядів через клієнт зовнішніх API, асинхронного приймання та валідації телеметрії від польових пристроїв і міжплатформеної координації з аналітичним двигуном. Побудовані компонентна і пакетна моделі підтвердили доцільність сервіс-орієнтованого архітектурного стилю та принципів чистої архітектури для забезпечення модульності кодової бази.

Третій розділ кваліфікаційної роботи присвячено практичній реалізації та інженерному обґрунтуванню інструментальних засобів програмного комплексу. Для створення високонавантаженого серверного ядра системи обрано мову Go з фреймворком Gin, sqlx та zap-логуванням, тоді як аналітичний двигун прогнозування реалізовано на Python за допомогою FastAPI, scikit-learn та joblib для виконання інференсу нейромережових моделей класифікації. Клієнтську частину оптимізовано на базі компіляційного фреймворку Svelte та TypeScript під збіркою Vite. У межах цього етапу розроблено фізичну схему бази даних під управлінням СУБД PostgreSQL із каскадними обмеженнями цілісності часових рядів, а також виконано алгоритмізацію ключових процесів за допомогою блок-схем, які зафіксували послідовність очищення апаратного шуму датчиків, добової агрегації та порогового реагування на ризики при перевищенні критерію ймовірності загрози у сімдесят відсотків.

У четвертому розділі проведено комплексне випробування, функціональне верифікування та навантажувальне оцінювання працездатності створеної системи. На основі демонстраційного наповнення бази даних підтверджено коректність функціонування інтерфейсних форм SPA-додатка, включаючи головний дашборд агронома, інструменти моніторингу інтегрованих погодних станцій, модальні вікна створення полів, панелі погодинних трендів та результати аналітичного інференсу з видачею автоматичних текстових порад захисту рослин. Контейнеризація в Docker забезпечила стабільне розгортання системи на основі профілів змінних середовища, а експериментальні

навантажувальні тести Go-сервера продемонстрували стійкість архітектури при потоці у двісті п'ятдесят запитів за секунду від віртуальних IoT-датчиків, фіксуючи середню затримку відповіді API на рівні п'ятнадцяти мілісекунд при нульовому коефіцієнті виникнення помилок.

Практичним результатом дослідження є автономне програмне рішення з відкритими інтерфейсами, яке реалізує повний, замкнений операційний цикл роботи з агрокліматичними даними: від зчитування фізичного показника польового датчика чи віддаленого сервісу до його нормалізації, аналітичної обробки за допомогою алгоритмів машинного навчання, візуалізації трендів та автоматичного сповіщення. Таким чином, поставлену мету кваліфікаційної роботи повністю досягнуто, а розроблений інструментарій створює надійне інженерне підґрунтя для оптимізації технологічних процесів у сільському господарстві, підвищення прозорості кліматичного аудиту земельних ділянок та мінімізації виробничих ризиків в умовах реальної експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ISO/IEC 27001:2022. Information security, cybersecurity and privacy protection. Information security management systems. Requirements. Geneva : International Organization for Standardization, 2022. URL: <https://www.iso.org/standard/27001>
2. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : doctoral dissertation. University of California, Irvine, 2000. URL: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
3. Masse M. REST API Design Rulebook. Sebastopol : O'Reilly Media, 2011. 116 p.
4. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT). Internet Engineering Task Force, 2015. URL: <https://datatracker.ietf.org/doc/html/rfc7519>
5. OpenAPI Specification. Version 3.1.0. OpenAPI Initiative. URL: <https://spec.openapis.org/oas/v3.1.0>
6. Go Documentation. The Go Authors. URL: <https://go.dev/doc/>
7. Gin Web Framework Documentation. URL: <https://gin-gonic.com/docs/>
8. Python Documentation. Python Software Foundation. URL: <https://docs.python.org/>
9. FastAPI Documentation. URL: <https://fastapi.tiangolo.com/>
10. Svelte Documentation. URL: <https://svelte.dev/docs>
11. Vite Reference Guide. URL: <https://vitejs.dev/guide/>
12. PostgreSQL 16 Documentation. The PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/16/>
13. Docker Documentation. Docker Inc. URL: <https://docs.docker.com/>
14. OpenWeatherMap API Specification. URL: <https://openweathermap.org/api>
15. OWASP Top Ten Web Application Security Risks. OWASP Foundation, 2021. URL: <https://owasp.org/Top10/>

- 16.Liakos K. G., Busato P., Moshou D., Pearson S., Bochtis D. Machine Learning in Agriculture: A Review. *Sensors*. 2018. Vol. 18, No. 8. P. 2674. DOI: <https://doi.org/10.3390/s18082674>
- 17.Adel M., El-Soud M. A. Internet of Things in Agriculture: Opportunities and Challenges. *International Journal of Advanced Computer Science and Applications*. 2020. Vol. 11, No. 4. P. 254–261. URL: https://thesai.org/Downloads/Volume11No4/Paper_33-Internet_of_Things_in_Agriculture.pdf
- 18.Wolfert S., Ge L., Verdouw C., Bogaardt M. J. Big Data in Smart Farming – A review. *Agricultural Systems*. 2017. Vol. 153. P. 69–80. DOI: <https://doi.org/10.1016/j.agry.2017.01.023>
- 19.Ferentinos K. P. Deep learning models for plant disease detection and diagnosis. *Computers and Electronics in Agriculture*. 2018. Vol. 145. P. 311–318. DOI: <https://doi.org/10.1016/j.compag.2017.12.011>
- 20.Bailey D. Design patterns for building tolerant Web APIs. *IEEE Software*. 2015. Vol. 32, No. 4. P. 42–47.
- 21.Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
- 22.Donovan A. A., Kernighan B. W. The Go Programming Language. Boston : Addison-Wesley, 2015. 400 p.
- 23.Erl T. Service-Oriented Architecture: Concepts, Technology, and Design. Upper Saddle River : Prentice Hall, 2005. 792 p.
- 24.Ray P. P. An Internet of Things-based architecture for smart agriculture. *Transactions on Emerging Telecommunications Technologies*. 2017. Vol. 28, No. 4. Article e2944. DOI: <https://doi.org/10.1002/ett.2944>
- 25.Pedregosa F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*. 2011. Vol. 12. P. 2825–2830.
- 26.Newman S. Building Microservices: Designing Fine-Grained Systems. Sebastopol : O'Reilly Media, 2021. 612 p.

27. Bierman G., Abadi M., Torgersen M. Understanding TypeScript. *European Conference on Object-Oriented Programming*. Springer, Cham, 2014. P. 257–281. DOI: https://doi.org/10.1007/978-3-319-08738-2_11
28. Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2016. 1272 p.

Додаток А

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І
ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій Декларація про академічну доброчесність та використання ШІ

Я, Піскаленко Євгеній Миколайович, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Інформаційна система агрегації та інтеграції метеоданих із використанням сервіс-орієнтованих інтерфейсів» 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 (ChatGPT) 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.

«22» травня 2026 р.

_____ **Євгеній ПІСКАЛЕНКО**