НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри інформаційних систем і технологій**
_____ **Ігор БОЛБОТ**	_____ **Михайло ШВИДЕНКО**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: Інформаційна система аналізу стану ґрунтів

Спеціальність 126 “Інформаційні системи та технології”

Гарант освітньої програми

к.е.н., доцент _____ **Мокрієв Максим Володимирович**

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Керівник кваліфікаційної роботи

д.т.н., професор _____ **Смолій Вікторія Миколаївна**

(науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Кондратюк Олександр Максимович

(підпис) (ПІБ)

Київ 2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Завідувач кафедри інформаційних систем і технологій

Швиденко М.З, к.е.н., проф.

Підпис ініціали та прізвище_____2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

студенту(ці) Кондратюку Олександру Максимовичу

Спеціальності 126 “Інформаційні системи та технології”

1. Тема роботи: “ Інформаційна система аналізу стану ґрунтів”
Затверджена наказом ректора від 19.12.2025. №3205"С"
2. Термін подання завершеної роботи на кафедру - *(25.05.2026)*
3. Вихідні данні: масиви агрохімічних показників ґрунтів (NPK-комплекс, рівень pH, вологість), геопросторові координати земельних ділянок, API метеорологічних сервісів (Open-Meteo), математичні моделі лінійної регресії для екстраполяції часових рядів.
4. Перелік питань, що розглядаються:
5. Аналіз предметної області та недоліків існуючих програмних рішень для моніторингу сільськогосподарських угідь.
6. Проектування архітектури інформаційної системи та логічної структури реляційної бази даних SQLite.
7. Програмна реалізація графічного інтерфейсу користувача та підсистеми картографування ділянок.
8. Розробка модуля математичного прогнозування та імплементація експертної системи для генерації агротехнічних рекомендацій.

Дата видачі завдання «19» грудня 2025 р.

Керівник бакалаврської кваліфікаційної роботи _____ **Вікторія СМОЛІЙ**

(підпис)

Завдання взяв до виконання _____ **Олександр КОНДРАТЮК**

(підпис)

РЕФЕРАТ

Тема бакалаврської кваліфікаційної роботи: “Інформаційна система аналізу станів ґрунтів”.

Автор роботи: Кондратюк Олександр Максимович.

Керівник роботи: Смолій Вікторія Миколаївна.

Пояснювальна записка: 65 с., 15 рис., 1 табл., 4 дод., 32 джерел.

Графічна частина: 15 презентаційних слайдів.

Об’єкт дослідження - процеси інформаційного супроводу, збору, аналізу та прогнозування показників стану ґрунтів на сільськогосподарських угіддях.

Предмет дослідження - моделі, алгоритми, архітектурні патерни та інструментальні засоби розробки інформаційної системи аналізу стану ґрунтів.

Мета дослідження полягає у підвищенні ефективності процесів обліку, аналізу та прогнозування агрохімічних показників земельних ділянок шляхом проектування та програмної реалізації спеціалізованої десктопної інформаційної системи з елементами геоінформаційних технологій (ГІС) та машинного навчання.

Методи дослідження: системний аналіз, методи теорії реляційних баз даних, об’єктно-орієнтований підхід (ООП), методологія моделювання UML, методи математичної статистики та регресійного аналізу.

Наукова новизна полягає у вдосконаленні підходу до локального моніторингу сільськогосподарських угідь шляхом інтеграції в єдиному програмному середовищі реляційної моделі даних, алгоритмів екстраполяції часових рядів та продукційних правил експертної системи.

Практичне значення: спроектовано та розроблено готовий до впровадження програмний продукт, що дозволяє агрономічним службам автономно здійснювати моніторинг NPK-показників, генерувати математичні прогнози та формувати офіційну звітність без використання хмарних серверів.

Ключові слова**:** ІНФОРМАЦІЙНА СИСТЕМА, ТОЧНЕ ЗЕМЛЕРОБСТВО, БАЗА ДАНИХ, PYTHON, SQLITE, МАШИННЕ НАВЧАННЯ, ЛІНІЙНА РЕГРЕСІЯ, ГЕОІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ, ЕКСПЕРТНА СИСТЕМА.

ABSTRACT

Master's thesis: 64 pages, 15 figures, 1 tables, 4 appendices, 32 references.

Theme: Soil condition analysis information system.

.

Object of study: processes of information support, collection, analysis, and forecasting of soil conditions on agricultural lands.

Subject of study: models, algorithms, architectural patterns, and development tools for the soil status analysis information system.

Purpose of the study: to improve the efficiency of accounting, analysis, and forecasting of agrochemical indicators of land plots by designing and implementing a specialized desktop information system with elements of Geographic Information Systems (GIS) and Machine Learning.

Practical value: a ready-to-implement software product has been developed. It allows agronomic services to autonomously monitor NPK indicators, generate mathematical forecasts, and create official reports without using cloud servers.

Keywords: INFORMATION SYSTEM, PRECISION AGRICULTURE, DATABASE, PYTHON, SQLITE, MACHINE LEARNING, LINEAR REGRESSION, GIS, EXPERT SYSTEM.

Зміст

ВСТУП

Актуальність теми. Сьогодні розвиток агропромислового сектору вже складно уявити без технологій точного землеробства та загальної цифровізації. Щоб ефективно використовувати земельні ресурси, потрібно постійно відслідковувати стан ґрунтів: їхню кислотність (pH), вологість, а також наявність головних поживних елементів — азоту, фосфору та калію (NPK-комплекс). Оскільки клімат змінюється, ґрунти поступово виснажуються, а ціни на мінеральні добрива ростуть, керувати господарством "наосліп" чи покладатися лише на інтуїцію стає просто економічно невигідно.

Звісно, на ринку зараз є багато потужних хмарних систем для управління агробізнесом (так звані Farm Management Systems). Але проблема в тому, що більшість із них розраховані на великі транснаціональні корпорації. Для середніх або дрібних фермерів вони часто виявляються занадто дорогими, перевантаженими зайвим функціоналом і, що найголовніше, вимагають стабільного інтернету, якого в полі зазвичай немає.

У той же час, вести облік аналізів по-старому - у звичайних таблицях Excel - вкрай незручно. Такий підхід не дає змоги нормально візуалізувати дані на карті, швидко будувати математичні прогнози чи аналізувати історію поля за минулі роки. Тому розробка окремої, автономної програми, яка б працювала локально і допомагала аналізувати стан ґрунтів, є дійсно нагальною і практичною задачею.

Мета дослідження полягає у підвищенні ефективності процесів обліку, аналізу та прогнозування агрохімічних показників земельних ділянок шляхом проектування та програмної реалізації спеціалізованої десктопної інформаційної системи з елементами геоінформаційних технологій (ГІС) та машинного навчання.

Для досягнення поставленої мети було сформульовано та вирішено такі **завдання дослідження**:

1. Провести системний аналіз предметної області, дослідити існуючі програмні рішення для моніторингу ґрунтів та виявити їхні функціональні недоліки.

2. Сформулювати вимоги до програмного забезпечення та розробити логічну і фізичну моделі реляційної бази даних для зберігання геопросторової та агрохімічної інформації.

3. Спроектувати архітектуру інформаційної системи за допомогою методів об'єктно-орієнтованого моделювання (UML) та побудувати діаграми інформаційних потоків (DFD).

4. Розробити графічний інтерфейс користувача (GUI) із застосуванням сучасних патернів проектування для забезпечення ергономічної взаємодії фахівця з масивами даних.

5. Програмно реалізувати підсистему картографування ділянок з використанням відкритих картографічних сервісів та технологій кешування просторових даних.

6. Імплементувати алгоритми математичного прогнозування (лінійної регресії) для екстраполяції динаміки макроелементів, а також розробити експертну підсистему для автоматичної генерації агротехнічних рекомендацій.

7. Забезпечити можливість інтеграції системи із зовнішніми джерелами даних (метеорологічні API, файли електронних таблиць) та розробити модуль генерації офіційної звітності у форматі PDF.

Об’єкт дослідження - процеси інформаційного супроводу, збору, аналізу та прогнозування показників стану ґрунтів на сільськогосподарських угіддях.

Предмет дослідження - моделі, алгоритми, архітектурні патерни та інструментальні засоби розробки інформаційної системи аналізу стану ґрунтів.

Методи дослідження. Для розв’язання поставлених завдань у роботі використано комплекс наукових та інженерних методів. Системний аналіз застосовано для дослідження предметної області та формалізації вимог. Методи

теорії реляційних баз даних використано при проектуванні структури зберігання інформації. Об’єктно-орієнтований підхід та методологія UML застосовані для побудови архітектури програмного продукту. Методи математичної статистики та регресійного аналізу використані для розробки модуля прогнозування динаміки NPK-показників. Для імплементації системи застосовано методи програмної інженерії та інтеграції API-інтерфейсів.

Наукова новизна одержаних результатів полягає у вдосконаленні підходу до локального моніторингу сільськогосподарських угідь шляхом інтеграції в єдиному програмному середовищі реляційної моделі даних, алгоритмів екстраполяції часових рядів та продукційних правил експертної системи. Це дозволяє автоматизувати виявлення критичних станів ґрунту без залучення важких хмарних обчислень.

Практичне значення одержаних результатів. Спроектовано та розроблено готовий до впровадження програмний продукт у вигляді скомпільованого виконуваного файлу, що не потребує розгортання додаткового серверного середовища. Впровадження системи дозволяє агрономічним службам скоротити час на обробку лабораторних даних, мінімізувати ймовірність людської помилки при формуванні звітності та підвищити обґрунтованість рішень щодо внесення мінеральних добрив і меліорантів.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Предметна область та проблематика моніторингу агрохімічних показників ґрунту

Раціональне управління земельними ресурсами є фундаментальною проблемою сучасного агропромислового комплексу. В умовах переходу до парадигми точного землеробства критичної ваги набуває процес безперервного збору, систематизації та аналізу даних щодо стану сільськогосподарських угідь. Відсутність достовірної інформації про хімічні та фізичні властивості ґрунту на конкретних ділянках призводить до нераціонального розподілу мінеральних добрив, зниження рентабельності виробництва та екологічної деградації земель[6, 26].

Предметною областю даного дослідження є інформаційні процеси, що супроводжують моніторинг агрохімічних показників ґрунту. З інформаційної точки зору, стан ґрунту описується набором просторово-часових даних, де кожна вибірка (проба) прив’язана до конкретних географічних координат, має часову мітку та вектор кількісних характеристик. До базових індикаторів, які підлягають обов'язковому обліку та подальшому аналізу в інформаційних системах, належать:

1. **Водневий показник (pH):** визначає рівень кислотності або лужності ґрунтового розчину. Цей параметр є індикатором доступності поживних речовин для кореневої системи рослин. Зсув рівня pH за межі оптимальних значень (зазвичай 5.5-7.5) вимагає оперативного втручання шляхом внесення меліорантів (вапнування при високій кислотності або гіпсування при залуженні) [2, 4].
2. **Макроелементи (NPK-комплекс):** азот (N), фосфор (P) та калій (K). Це критичні хімічні елементи, концентрація яких (вимірюється в мг/кг) має пряму кореляцію з рівнем врожайності. Динаміка зміни NPK у часі є ключовим маркером виснаження ґрунту. [3, 29]

3. **Вологість ґрунту:** фізичний параметр, який не лише впливає на вегетацію рослин, але й визначає швидкість розчинення та абсорбції внесених хімічних добрив.

Традиційний бізнес-процес моніторингу складається з етапів фізичного відбору проб, проведення лабораторного аналізу та подальшої інтерпретації результатів агрономічною службою. Проблематика предметної області полягає в тому, що на етапі інтерпретації виникає інформаційний розрив. Лабораторні дані найчастіше генеруються у вигляді неструктурованих електронних таблиць або паперових звітів. Такий формат представлення даних унеможливлює швидкий ретроспективний аналіз. Фахівцю складно візуально оцінити тренд зниження рівня фосфору на певній ділянці за останні п'ять років, маючи перед собою лише набір ізольованих файлів.

Крім того, просторова природа даних вимагає інтеграції результатів хімічних аналізів із геоінформаційними системами (ГІС). Кожна земельна ділянка є полігоном або точкою на карті, і розуміння топології поля є необхідним для диференційованого внесення добрив. Відсутність картографічної візуалізації в стандартних табличних процесорах значно знижує ефективність прийняття рішень[27].

З позиції інженерії програмного забезпечення, вирішення описаної проблематики вимагає переходу від зберігання даних у "плоских" файлах до використання повноцінних реляційних баз даних. Це дозволить забезпечити цілісність даних, уникнути дублювання та створити міцний фундамент для надбудови аналітичних модулів. Більше того, накопичення достатнього обсягу історичних даних про рівень NPK відкриває можливості для застосування методів математичного моделювання та екстраполяції (прогнозування) [2, 4].

1.2. Системний аналіз існуючих програмних рішень для управління агрономічними даними

Стрімкий розвиток напрямку аграрних інформаційних технологій спричинив появу на ринку значної кількості програмних продуктів, орієнтованих на цифровізацію сільського господарства. Проте, незважаючи на загальну

насиченість ринку, сегмент спеціалізованого програмного забезпечення для локального моніторингу та прогнозування хімічного стану ґрунтів залишається високофрагментованим. Для обґрунтування доцільності та архітектурних рішень власної розробки було проведено критичний системний аналіз трьох основних категорій інформаційних систем, які найчастіше використовуються агрономічними службами на практиці.

1.2.1. Універсальні табличні процесори (MS Excel, Google Sheets)

Історично склалося так, що базовим інструментом для ведення реєстрів лабораторних аналізів у переважній більшості дрібних та середніх фермерських господарств залишаються табличні процесори. Перевагами такого підходу є нульовий поріг входження, відсутність необхідності розгортання додаткової IT-інфраструктури та гнучкість у налаштуванні атрибутивних полів.

Проте, з точки зору інженерії даних та теорії баз даних, використання електронних таблиць як основної інформаційної системи демонструє низку критичних архітектурних та функціональних недоліків:

Відсутність просторової прив’язки (Spatial Data): табличний процесор не дозволяє нативно візуалізувати координати забору проб на топографічній карті, що унеможливлює створення картограм розподілу поживних речовин (NPK) та оцінку їхньої просторової гетерогенності.

Порушення цілісності даних (Data Integrity): відсутність строгих обмежень на рівні схеми бази даних (Constraints) та слабка типізація призводять до високого рівня помилок, зумовлених людським фактором (наприклад, введення текстових символів у поля для числових показників кислотності).

Складність алгоритмічної обробки: побудова моделей машинного навчання (лінійної або поліноміальної регресії) для прогнозування виснаження ґрунтів на базі макросів VBA є нераціональною з точки зору продуктивності та підтримки коду. Крім того, електронні таблиці не мають вбудованих механізмів для реалізації експертних систем на базі продукційних правил[10, 12].

1.2.2. Комплексні хмарні платформи (Farm Management Systems)

До цієї категорії належать потужні комерційні продукти корпоративного рівня, такі як Cropio, OneSoil, Climate FieldView. Архітектурно це типові SaaS-рішення з тонким клієнтом (веб-браузер або мобільний додаток) та масивним бекендом на хмарних серверах (наприклад, AWS або Microsoft Azure).

Функціонал таких систем охоплює розрахунок вегетаційних індексів (NDVI) за допомогою супутникових знімків мультиспектрального аналізу, телеметрію сільськогосподарської техніки та модулі фінансового планування. Однак, при застосуванні їх для вирішення вузькопрофільних завдань локального аналізу ґрунтів, виявляються суттєві обмеження:

Економічна доцільність: модель монетизації передбачає регулярну абонентську плату, що масштабується залежно від площі земельного банку (за кожен гектар). Для фахівців-консультантів або середніх підприємств витрати на ліцензування часто перевищують економічний ефект від використання ПЗ.

Залежність від мережевої інфраструктури: оскільки архітектура базується на клієнт-серверній взаємодії, робота в "польових" умовах, де часто відсутнє або нестабільне покриття мобільних мереж, стає неможливою. Синхронізація офлайн-даних часто реалізована з помилками вирішення конфліктів.

Інформаційне перевантаження: системи містять сотні модулів, тоді як для обліку NPK-показників та генерації звітів потрібен лише мінімальний, але специфічний набір інструментів.

Вендор-лок: закритість екосистеми та відсутність відкритого доступу до власної бази даних через SQL-запити ускладнює експорт специфічної звітності (наприклад, у локальні формати PDF) та міграцію даних.

1.2.3. Професійні геоінформаційні системи

Деякі агрохімічні лабораторії використовують класичні десктопні ГІС-додатки для накладання атрибутивних даних хімічних аналізів на векторні шари земельних ділянок. Безперечною перевагою є потужний інструментарій просторової аналітики та роботи з растровими/векторними даними. Водночас,

використання ГІС як єдиної системи обліку є нераціональним. Це програмне забезпечення загального призначення, яке має надзвичайно високий поріг входження. Для реалізації простого завдання - побудови графіка динаміки фосфору за останні п'ять років або отримання текстової рекомендації щодо вапнування - оператору необхідно писати складні запити мовою SQL (для PostGIS) або розробляти власні скрипти мовою Python безпосередньо в консолі QGIS. Це призводить до невиправданих витрат робочого часу агронома.

1.2.4. Висновки з порівняльного аналізу та обґрунтування розробки

Проведений системний аналіз ринку програмного забезпечення виявив чітко оформлену системну прогалину. Існує практичний запит на створення проміжного, гібридного рішення - локальної інформаційної системи, що поєднуватиме в собі:

Автономність та економічну доступність (на відміну від хмарних FMS платформ).

Надійність зберігання структурованих даних у реляційній базі (на відміну від табличних процесорів).

Інтегровані алгоритми математичного прогнозування та експертних рекомендацій "з коробки" (без необхідності програмування з боку користувача, як у QGIS). Для наочної демонстрації переваг обраного підходу результати аналізу існуючих рішень зведено у порівняльну таблицю 1.1.

Критерій оцінки	MS Excel	Хмарні FMS (Cropio)	ГІС (QGIS)	Розроблена ІС
Вартість впровадження	Безкоштовно	Висока (абонентська)	Безкоштовно	Безкоштовно
Автономна робота (Офлайн)	Так	Ні	Так	Так
Просторова прив'язка (Карта)	Ні	Так	Так	Так

Вбудований матем. прогноз	Ні	Так	Ні	Так
Експертні підказки (NPK)	Ні	Ні	Ні	Так

Таблиця 1.1 – Порівняльний аналіз інформаційних систем для агрономії

1.3. Визначення функціональних та нефункціональних вимог до розроблюваної інформаційної системи

Процес розробки програмного забезпечення вимагає чіткої формалізації вимог до майбутнього продукту. Специфікація вимог є основою для подальшого проектування архітектури бази даних, розробки графічного інтерфейсу та вибору інструментальних засобів (фреймворків). Вимоги до розроблюваної інформаційної системи аналізу стану ґрунтів поділяються на дві класичні категорії: функціональні (визначають поведінку системи та її бізнес-логіку) та нефункціональні (визначають атрибути якості, обмеження та середовище виконання) [30, 31].

1.3.1. Специфікація функціональних вимог

Інформаційна система повинна забезпечувати автоматизацію повного циклу обробки агрономічних даних - від первинного введення показників до генерації прогнозів та звітів. Відповідно до визначеної предметної області, реалізовувати наступний функціонал:

1. **Модуль автентифікації та адміністрування доступу:**

Система забезпечуватиме механізм безпечного входу (Login) за допомогою унікального імені користувача та пароля та надавати функціонал керування обліковими записами (додавання нових користувачів та видалення існуючих) із захистом головного профілю адміністратора від випадкового видалення.

2. **Модуль керування просторовими даними (Ділянки):**

Буде реалізовуватись повний набір базових операцій CRUD для сутності "Земельна ділянка".

Форма реєстрації ділянки має обов'язково містити поля: назва, площа (га), тип ґрунту та точні географічні координати (широта і довгота) для інтеграції з геоінформаційною підсистемою.

3. **Модуль реєстрації лабораторних досліджень (Проби ґрунту):**

Система повинна дозволяти внесення результатів хімічного аналізу із прив'язкою до конкретної ділянки та дати відбору проби, валідувати вхідні дані для показників: водневий показник (pH), вологість (%), масові частки азоту (N), фосфору (P) та калію (K), підтримувати масовий імпорт історичних даних із зовнішніх файлів формату .xlsx (MS Excel) для забезпечення зворотної сумісності з попередніми форматами обліку.

4. **Геоінформаційний модуль (GIS):**

Система має візуалізувати зареєстровані ділянки на інтерактивній глобальній карті у вигляді маркерів.

Картографічна підсистема має використовувати актуальні супутникові знімки (наприклад, Google Satellite) замість схематичних векторних карт для кращої візуальної орієнтації на місцевості.

5. **Аналітичний модуль та підсистема машинного навчання:**

Система генеруватиме інтерактивні лінійні графіки динаміки зміни макроелементів (NPK) у часі для обраної ділянки, буде містити алгоритм машинного навчання (лінійна регресія першого порядку) для розрахунку та візуалізації математичного прогнозу (екстраполяції тренду) стану ґрунту на 30 днів у майбутнє.

6. **Експертна система підтримки рішень:**

Система автоматично аналізує останню внесену пробу ґрунту та, використовуючи закладену базу знань (продукційні правила), генерувати текстові рекомендації.

Рекомендації мають стосуватися критичних відхилень: необхідності вапнування або гіпсування (на основі pH), а також цільового внесення азотних, фосфорних чи калійних добрив.

7. **Модуль інтеграції з API та звітності:**

Система повинна здійснювати автоматичні HTTP-запити до відкритих метеорологічних API (наприклад, Open-Meteo) для отримання погодних умов (температура, швидкість вітру) в реальному часі на основі координат ділянки, забезпечувати експорт зведених даних у табличний формат Excel, генерувати формалізований офіційний звіт у форматі PDF (для формату А4), який містить метадані об'єкта, висновки експертної системи та вбудоване графічне зображення побудованих трендів.

1.3.2. Специфікація нефункціональних вимог

Нефункціональні вимоги визначають технічні обмеження та стандарти якості, яким має відповідати розроблений програмний продукт:

1. **Вимоги до надійності та зберігання даних:**

Система повинна бути спроектована за архітектурою "Товстий клієнт" (Thick Client) з використанням локальної вбудованої СУБД (SQLite).

Робота основного функціоналу (крім підсистеми карт та запитів погоди) має виконуватися повністю автономно, без обов'язкового підключення до мережі Інтернет.

Картографічна підсистема повинна використовувати алгоритми локального кешування тайлів (база map_cache.db) для зниження навантаження на центральний процесор та мінімізації мережевого трафіку.

2. **Вимоги до інформаційної безпеки:**

Паролі користувачів не повинні зберігатися у базі даних у відкритому текстовому вигляді. Система зобов'язана використовувати криптографічне хешування за стандартом SHA-256.

3. **Вимоги до ергономіки та графічного інтерфейсу користувача (GUI):**

Інтерфейс має відповідати принципам проектування сучасних корпоративних дашбордів mз фіксованим боковим навігаційним меню.

Система буде використовувати за замовчуванням темну тему оформлення для зниження зорової втоми користувача при тривалій роботі з числовими

масивами даних, здійснювати валідацію користувацького вводу та виводити зрозумілі системні повідомлення у разі виникнення виняткових ситуацій.

4. **Вимоги до портативності та розгортання (Deployment):**

Додаток повинен працювати в середовищі операційної системи Windows (починаючи з версії 10).

Система повинна постачатися кінцевому споживачу у вигляді єдиного скомпільованого виконуваного файлу формату .exe. Встановлення додаткових інтерпретаторів, середовищ виконання або сторонніх бібліотек на цільовій машині не допускається.

1.4. Вибір стеку технологій та інструментальних засобів розробки

Проектування ефективної інформаційної системи нерозривно пов'язане з вибором оптимального інструментального середовища та програмних фреймворків. Відповідно до специфікованих раніше вимог (автономність, наявність графічного інтерфейсу, модулів машинного навчання та ГІС), архітектурним рішенням стало створення "товстого клієнта" для операційних систем сімейства Windows. Для реалізації такого підходу було сформовано наступний стек технологій:

1. Базова мова програмування: Основним інструментом розробки обрано високорівневу мову програмування **Python** (версії 3.x). Вибір обґрунтований її домінуючим положенням у сфері Data Science, обробки великих масивів даних та математичного моделювання. Крім того, Python пропонує парадигму об'єктно-орієнтованого програмування, що дозволяє інкапсулювати бізнес-логіку та створити модульну, легко масштабовану архітектуру додатку[13, 32].

2. Система управління базами даних (СУБД): Враховуючи вимогу до портативності та відмову від використання виділених серверів, в якості рушія бази даних було використано вбудовану реляційну СУБД **SQLite3**. На відміну від важких рішень, SQLite не вимагає адміністрування та конфігурації. Вся схема бази даних (таблиці, індекси, тригери) зберігається у єдиному кросплатформному файлі на диску користувача. При цьому SQLite повністю

підтримує транзакційність (ACID) та стандарти мови SQL, що гарантує надійність збереження агрохімічних показників[19].

3. Фреймворк графічного інтерфейсу користувача (GUI): Для реалізації візуальної частини додатку застосовано бібліотеку **CustomTkinter**. Це сучасна надбудова над стандартним модулем tkinter, яка дозволяє створювати апаратне прискорення та сучасні елементи управління (віджети). Використання CustomTkinter забезпечило виконання вимог щодо ергономіки: реалізацію корпоративного дашборду з підтримкою темної теми "з коробки", що суттєво перевершує застарілий дизайн класичного Tkinter та є легшим за ресурсоємні фреймворки типу PyQt[18].

4. Інструменти математичного моделювання та аналітики: Для обробки числових масивів та реалізації підсистеми прогнозування задіяно бібліотеки наукового стеку Python:

NumPy: використано для виконання високовитратних обчислень та побудови моделей машинного навчання. Зокрема, функція поліноміальної регресії застосована для екстраполяції тренду зміни макроелементів (азоту, фосфору, калію) на найближчі 30 днів[17, 24].

Pandas: інтегровано для роботи з датафреймами. Бібліотека забезпечує швидкий імпорт історичних даних з існуючих баз у форматі MS Excel та експорт зведених звітів без необхідності встановлення пакету Microsoft Office на цільовій машині[15, 21].

5. Бібліотека наукової візуалізації: Для рендерингу графіків динаміки NPK обрано галузевий стандарт - бібліотеку **Matplotlib**. Вона надає низькорівневий контроль над кожним елементом діаграми (вісями, маркерами, легендою). За допомогою модуля FigureCanvasTkAgg згенеровані графіки безшовно інтегруються безпосередньо у вікно додатку CustomTkinter, а також можуть бути експортовані як растрові зображення для формування офіційної звітності[20].

6. Геоінформаційний модуль та мережева взаємодія:

TkinterMapView: спеціалізований компонент для відображення інтерактивних карт. Бібліотека була налаштована на роботу з тайловими серверами супутникових знімків (Google Satellite) та використання локальної бази даних для кешування тайлів. Це вирішило проблему надмірного навантаження на центральний процесор (CPU) при масштабуванні карти.

Requests: використано для реалізації мережевих HTTP-запитів до відкритого метеорологічного інтерфейсу Open-Meteo API з метою отримання синоптичних даних у режимі реального часу[22,23].

7. Середовище розгортання: Для забезпечення портативності та виконання вимоги "єдиного файлу" застосовано утиліту **PyInstaller**. Цей компілятор аналізує дерево залежностей Python-скрипта, збирає інтерпретатор, усі використані бібліотеки та статичні файли, після чого пакує їх у єдиний виконуваний файл формату .exe. Це дозволяє запускати інформаційну систему на будь-якому комп'ютері під управлінням ОС Windows без попереднього встановлення середовища розробки.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ АНАЛІЗУ СТАНУ ҐРУНТІВ

2.1. Вибір та обґрунтування архітектури програмного забезпечення

Будь-яка розробка починається з вибору архітектури. Оскільки моя програма має працювати переважно "в полі" чи в невеликих лабораторіях, де часто бувають проблеми з інтернетом, я одразу відмовилися від стандартних веб-

додатків чи тонких клієнтів. Вирішив робити монолітний "товстий клієнт" - тобто програму, яка повністю і автономно виконується на комп'ютері користувача. Це дає змогу системі миттєво обробляти масиви даних і не "зависати" при втраті з'єднання із сервером. Звісно, монолітний підхід має свої недоліки, але я компенсував їх тим, що чітко розділив код на модулі: інтерфейс, робота з картами та математична аналітика працюють незалежно один від одного.

Основним архітектурним рішенням для розроблюваної системи стало використання парадигми монолітного "товстого клієнта" Ця архітектура передбачає, що вся бізнес-логіка, підсистема відображення графічного інтерфейсу та підсистема доступу до даних скомпільовані та виконуються в межах єдиного простору пам'яті на локальній обчислювальній машині користувача[10].

Такий підхід забезпечує низку інженерних переваг:

Мінімальна затримка: відсутність мережевих запитів до віддалених серверів (за винятком опціональних API-викликів для отримання погоди чи тайлів карт) дозволяє системі миттєво обробляти великі масиви історичних даних та будувати математичні прогнози в реальному часі.

Відмовостійкість: система зберігає 100% працездатності базового функціоналу аналітики та документообігу в повному офлайн-режимі.

Простота дистрибуції: кінцевий продукт не вимагає розгортання серверної інфраструктури чи налаштування контейнеризації, а постачається як єдиний виконуваний файл.

Незважаючи на монолітний характер розгортання, внутрішня структура програмного коду побудована за принципами жорсткої модульності. Розділення відповідальності дозволяє ізолювати картографічні алгоритми від математичних розрахунків, що значно спрощує тестування та подальшу підтримку коду.

Для візуалізації ієрархії підсистем та їхньої логічної взаємодії було застосовано метод структурного аналізу та розроблено діаграму функціональної

декомпозиції. Ця модель розбиває глобальну задачу моніторингу на менші, керовані підзадачі.

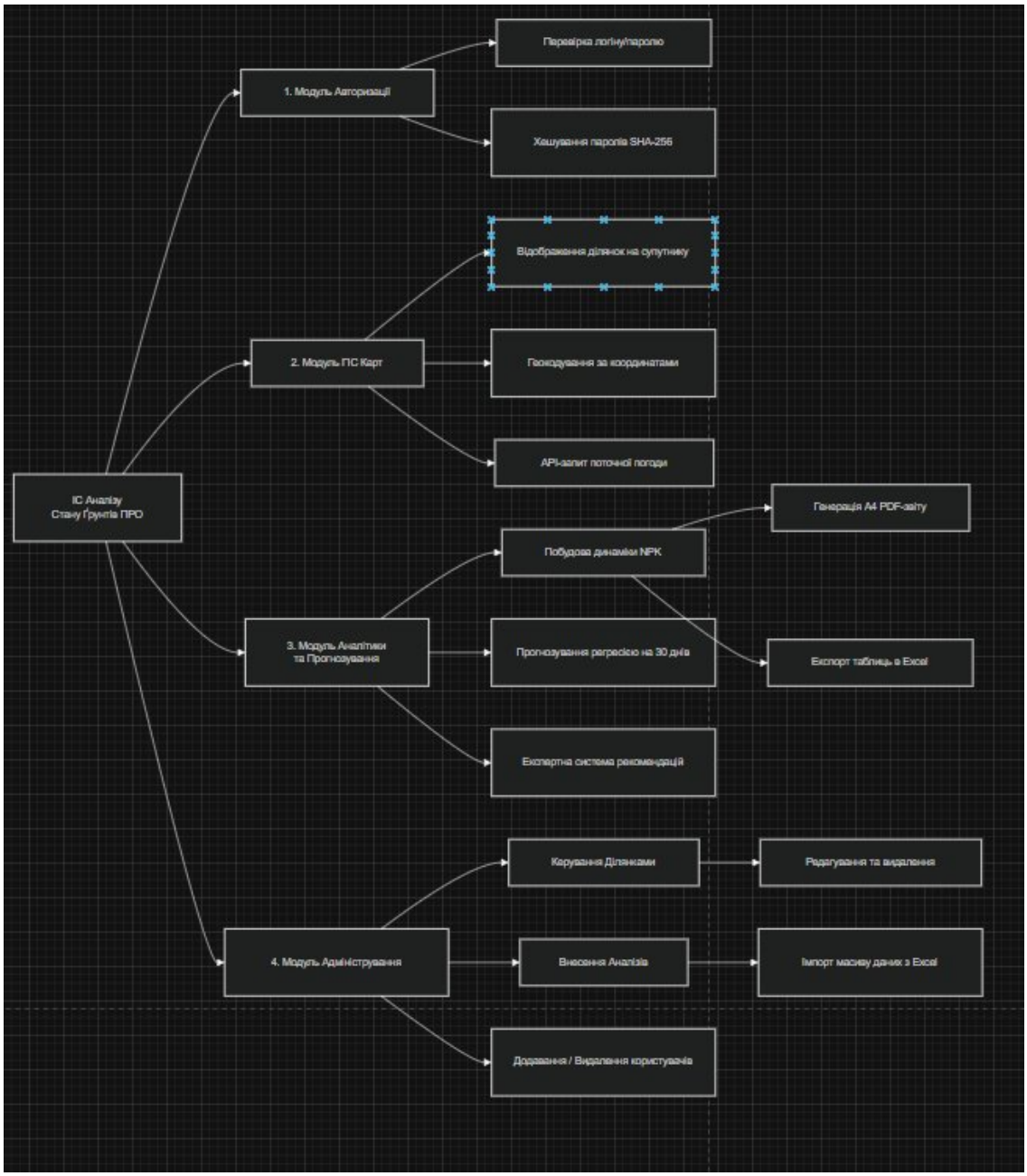


Рисунок 2.1 - Дерево функціональної декомпозиції (FDD) інформаційної системи

Як видно з Рисунка 2.1, загальна архітектура системи декомпозується на чотири фундаментальні модулі:

1. **Модуль автентифікації та безпеки:** відповідає за ідентифікацію користувачів на етапі запуску програми. Інкапсулює в собі алгоритми хешування

паролів та перевірки прав доступу, захищаючи комерційну інформацію від несанкціонованого перегляду.

2. **Модуль просторових даних (ГІС-компонент):** обробляє географічні координати земельних ділянок. Виконує роль прошарку між локальною базою даних та зовнішніми картографічними сервісами. Саме цей модуль відповідає за кешування растрових зображень супутникових знімків на жорсткий диск користувача для мінімізації мережевого трафіку.

3. **Модуль аналітики та прогнозування (Ядро системи):** найбільш ресурсоємна підсистема. Отримує "сирі" дані про вміст макроелементів (N, P, K) та рівень кислотності (pH), перетворюючи їх на візуальні графіки динаміки. У цей модуль інтегровано математичний апарат машинного навчання (лінійна регресія) для екстраполяції трендів на майбутні періоди. Крім того, модуль містить логічний блок Експертної системи, який на основі продукційних правил формує текстові агротехнічні висновки.

4. **Модуль адміністрування та документообігу:** забезпечує інтерфейс для виконання CRUD-операцій (створення, читання, оновлення, видалення) із записами в локальній базі даних. У межах цього модуля реалізовано алгоритми конвертації структур даних у бінарні формати для генерації офіційних звітів (PDF-документів для друку та таблиць MS Excel).

2.2. Інформаційне забезпечення та проектування реляційної бази даних

Що стосується збереження даних, тягнути за собою важкі сервери на кшталт MySQL чи PostgreSQL не було жодного сенсу. Це тільки ускладнило б установку програми для звичайного агронома. Тому найкращим вибором стала локальна СУБД SQLite3. Вона взагалі не потребує налаштувань, а вся база з ділянками та історією аналізів надійно лежить в одному файлі soils_data.db. При цьому вона повноцінно підтримує SQL-запити та гарантує, що дані не загубляться під час раптового вимкнення комп'ютера.

Вибір SQLite3 обґрунтовується її безсерверною архітектурою та нульовим рівнем конфігурації. Уся структура бази даних, включаючи таблиці, індекси та

зв'язки, зберігається у єдиному кросплатформному локальному файлі. Незважаючи на компактність, SQLite3 забезпечує повну підтримку транзакційності за принципами ACID та реалізує стандарти мови запитів SQL, що гарантує надійність збереження комерційної та наукової агрономічної інформації.

Проектування логічної та фізичної схеми бази даних виконувалося з дотриманням принципів нормалізації (доведення до третьої нормальної форми, 3NF). Це дозволило усунути аномалії модифікації, видалення та вставки, а також мінімізувати надмірність (дублювання) інформації.

Логічна модель бази даних складається з трьох взаємопов'язаних реляційних таблиць (відношень):

1. Таблиця Users (Облікові записи користувачів) Дана таблиця призначена для забезпечення роботи підсистеми автентифікації та розмежування прав доступу. Вона містить наступні атрибути:

id (INTEGER) - первинний ключ (Primary Key), що автоматично інкрементується (AUTOINCREMENT) та є унікальним ідентифікатором запису.

username (TEXT) - логін користувача. На рівні схеми БД для цього поля встановлено обмеження UNIQUE, що унеможливлює реєстрацію кількох фахівців під однаковим ім'ям.

password (TEXT) - криптографічний хеш пароля (алгоритм SHA-256). Система не зберігає паролі у відкритому вигляді (Plain-text) з міркувань інформаційної безпеки.

2. Таблиця Fields (Земельні ділянки) Відношення зберігає статичну та просторову інформацію про сільськогосподарські угіддя, що знаходяться під моніторингом.

id (INTEGER) - первинний ключ (Primary Key, AUTOINCREMENT).

name (TEXT) - семантична назва ділянки або її кадастровий номер. Поле має обмеження NOT NULL.

area (REAL) - площа земельної ділянки у гектарах. Використання типу даних з плаваючою комою (REAL) зумовлене необхідністю точного розрахунку норм внесення добрив.

soil_type (TEXT) - агрономічний тип ґрунту (наприклад, чорнозем типовий, сірий лісовий).

lat (REAL) та lon (REAL) - географічні координати (широта і довгота) центру ділянки. Ці атрибути є критично важливими для інтеграції записів бази даних із ГІС-підсистемою (модулем TkinterMapView) та здійснення просторових запитів до метеорологічних API.

3. Таблиця Soil_Samples (Результати лабораторних проб) Транзакційна таблиця, яка акумулює історичні часові ряди агрохімічних показників. Вона формує базу для подальшого математичного прогнозування та роботи експертної системи.

id (INTEGER) - первинний ключ.

field_id (INTEGER) - зовнішній ключ (Foreign Key), що посилається на id у таблиці Fields. Забезпечує посилальну цілісність даних (Referential Integrity).

sample_date (TEXT) - часова мітка (Timestamp) дати забору проби у форматі DD.MM.YYYY.

ph_level, moisture, nitrogen, phosphorus, potassium (всі типу REAL) - кількісні показники хімічного аналізу: водневий показник, відсоток вологості та концентрація макроелементів (азот, фосфор, калій) у мг/кг.

Для візуалізації логічної структури розробленої бази даних, атрибутивного складу сутностей та кардинальності зв'язків між ними, було побудовано діаграму Сутність-Зв'язок у нотації Чена / Crow's Foot.

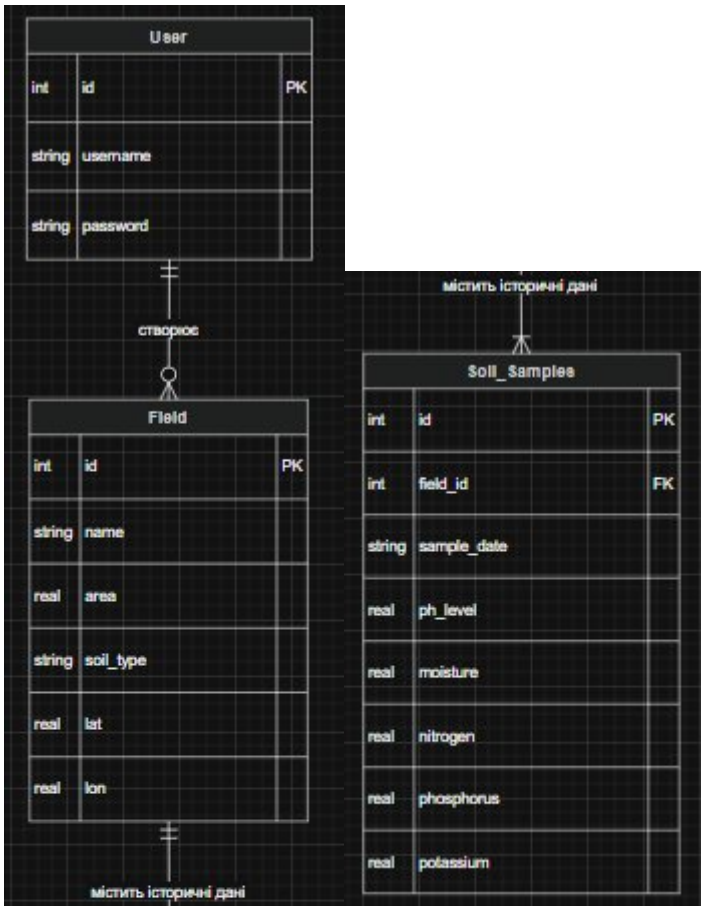


Рисунок 2.2 - Діаграма Сутність-Зв’язок (ER-модель) реляційної бази даних

На рисунку 2.2 зображено деталізовану ER-модель розробленої бази даних SQLite3. Вона складається з трьох ключових сутностей. Сутність Users містить поля username та password, де паролі зберігаються виключно у вигляді SHA-256 хешів. Базовим архітектурним патерном є зв'язок типу Один-до-Багатьох між головною таблицею Fields (Земельні ділянки) та підпорядкованою таблицею Soil_Samples. Таблиця Fields зберігає географічні координати lat та lon (тип REAL) для інтеграції з картою. Таблиця Soil_Samples за допомогою зовнішнього ключа field_id гарантує посилальну цілісність: усі результати хімічних аналізів жорстко прив'язані до конкретного поля. Такий розподіл повністю відповідає третій нормальній формі проектування баз даних.

Базовим архітектурним патерном взаємодії є зв'язок типу Один-до-Багатьох між сутностями Ділянка та Проба З точки зору бізнес-логіки це означає, що за однією земельною ділянкою в системі може бути закріплено нескінченну

множину результатів лабораторних досліджень за різні часові періоди. Такий підхід до нормалізації дозволяє ефективно агрегувати масиви даних за допомогою SQL-запитів з оператором ORDER BY sample_date для подальшої передачі їх у модуль генерації лінійних графіків та розрахунку регресійних трендів.

2.3. Об'єктно-орієнтоване моделювання програмних компонентів

Проектування внутрішньої структури програмного коду є визначальним фактором для забезпечення стабільності, масштабованості та зручності подальшого супроводу інформаційної системи. Враховуючи обрану мову програмування Python, архітектура додатку базується на парадигмі об'єктно-орієнтованого програмування (ООП). Застосування фундаментальних принципів ООП - інкапсуляції, успадкування та абстракції - дозволило реалізувати слабку зв'язність між графічним інтерфейсом користувача (GUI) та ядром обробки даних.

Весь код програми написаний з використанням об'єктно-орієнтованого програмування. Головним тут виступає клас MainApp, який бере на себе управління всіма вікнами та процесами (по суті, виконуючи роль архітектурного патерна "Фасад"). Він керує переходами між екранами: від вікна авторизації до відображення дашборду чи карти. Завдяки такому підходу, щоб перемкнути вкладку меню, нам не потрібно щоразу перемальовувати весь інтерфейс – програма просто приховує одні елементи і показує інші[14].

Внутрішня структура класу MainApp логічно поділена на кілька функціональних блоків:

1. Блок ініціалізації та маршрутизації: Для управління переходами між різними екранами системи (дашборд, аналітика, карта, керування даними) використано підхід на основі словника фреймів. Під час запуску програми в пам'яті ініціалізуються всі необхідні екрани через виклики методів типу create_analytics_frame(), create_map_frame() тощо. Перемикання між ними здійснюється за допомогою універсального методу-маршрутизатора show_frame(name), який інкапсулює логіку приховування неактивних віджетів

(grid_forget()) та відображення обраного екрану. Це забезпечує миттєвий відгук інтерфейсу без необхідності постійного перемальовування вікон.

2. Блок автентифікації: Відокремлений у методи build_login_screen() та do_login(). Цей блок створює тимчасовий контейнер для введення облікових даних, взаємодіє з функцією перевірки хешів БД та, у разі успішної авторизації, рекурсивно знищує власні віджети, викликаючи метод побудови головного дашборду build_main_dashboard().

3. Блок бізнес-логіки та інтеграції: Окремі методи класу відповідають за складні інженерні задачі. Наприклад, метод plot_analytics() не лише взаємодіє з базою даних для отримання історичних вибірок, але й ініціалізує об'єкти сторонніх класів:

Створює об'єкт фігури Figure та осей Axes з бібліотеки matplotlib для побудови графіків динаміки макроелементів.

Викликає зовнішню математичну функцію numpy.polyfit для розрахунку коефіцієнтів лінійної регресії та генерує масив майбутніх дат для відображення лінії тренду (прогнозу).

Ініціалізує об'єкт FigureCanvasTkAgg для безшовної інтеграції згенерованого наукового графіка безпосередньо в ієрархію віджетів Tkinter.

Для винесення рутинних операцій за межі головного класу графічного інтерфейсу, функції роботи з базою даних (CRUD-операції типу check_login, add_user_to_db, export_to_excel) реалізовані у вигляді незалежних модульних функцій. Це наближає структуру коду до патерну Model-View-Controller (MVC), де модульні функції відіграють роль Моделі, графічні методи - Відображення (View), а методи-обробники подій у класі MainApp - Контролера (Controller).

Структура класів додатку, їхні внутрішні атрибути, відкриті методи та залежності від сторонніх бібліотечних просторів імен (Namespaces) формалізовані за допомогою UML-діаграми класів (Static Class Diagram).

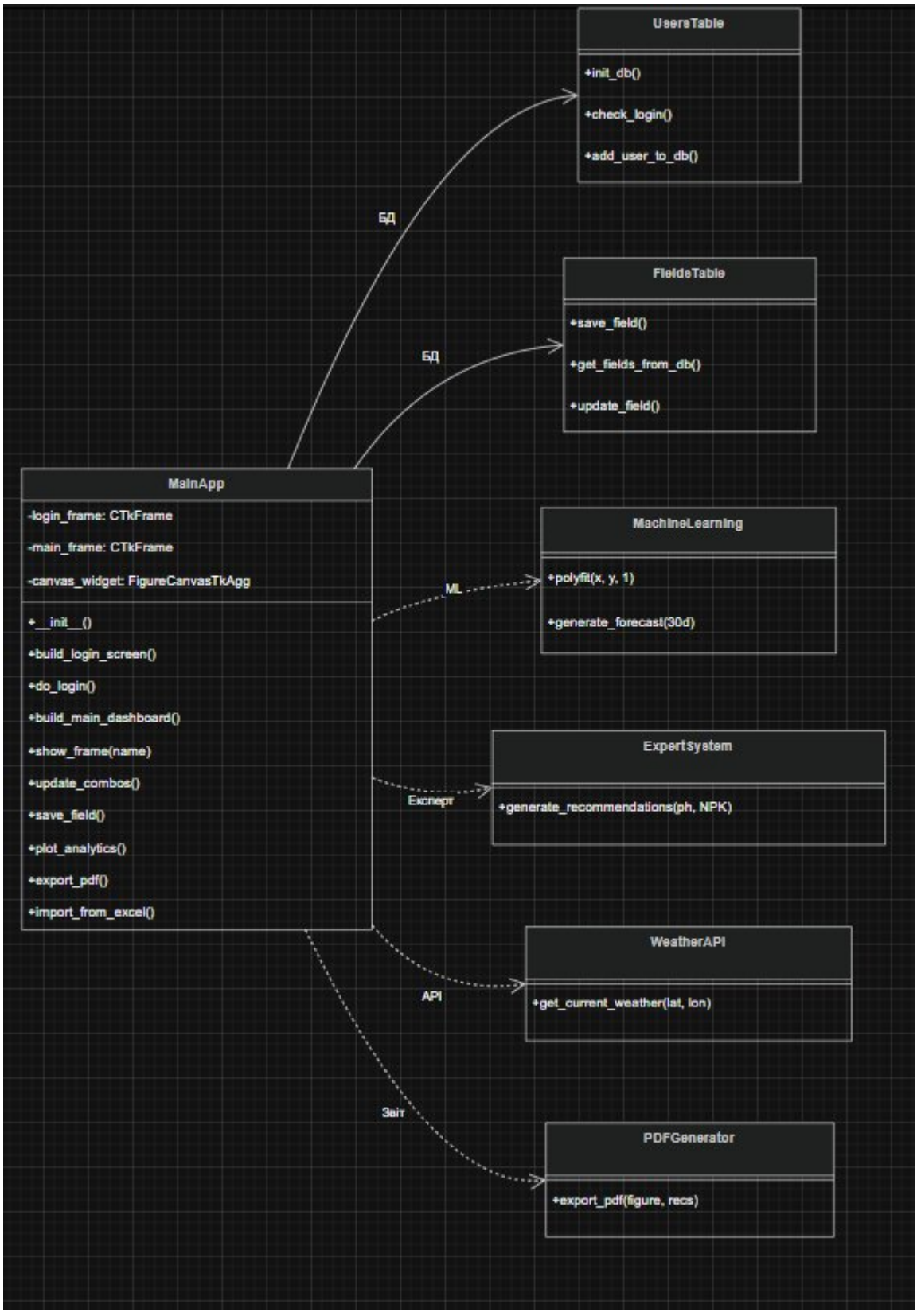


Рисунок 2.3 - UML-діаграма класів розробленого програмного додатка

Як ілюструє Рисунок 2.3, клас MainApp виступає центральним вузлом, що агрегує виклики до зовнішніх модулів: системи генерації звітів (експорт у PDF/Excel), експертної підсистеми, API-модуля отримання синоптичних даних

та підсистеми кешування геоданих. Використання об'єктно-орієнтованого підходу дозволило створити код із високим рівнем зв'язності всередині модулів, що робить систему відкритою до розширення (наприклад, додавання нових алгоритмів прогнозування) та закритою для модифікації базового ядра.

2.4. Моделювання бізнес-процесів та алгоритмів обробки даних

Динаміка функціонування інформаційної системи, тобто послідовність взаємодії між користувачем, графічним інтерфейсом, ядром обробки даних та зовнішніми веб-сервісами, є критичним аспектом, що підлягає ретельному моделюванню. Для опису алгоритмічної складової та життєвого циклу процесів було використано методологію UML, зокрема діаграми послідовності та діяльності.

Одним із найбільш ресурсоємних та технічно складних бізнес-процесів у розробленій системі є генерація комплексного аналітичного звіту (графік динаміки, прогноз та погодні умови). Алгоритм виконання цього процесу складається з послідовності синхронних та асинхронних викликів.

При ініціації процесу користувачем (натискання кнопки Побудувати графік та Прогноз), графічний інтерфейс (GUI) генерує подію, яка запускає внутрішній метод обробки plot_analytics(). Першим кроком алгоритму є звернення до бази даних (БД) із SQL-запитом SELECT lat, lon FROM Fields для отримання географічних координат обраної земельної ділянки. Отримані координати передаються у функцію мережевої взаємодії, яка формує HTTP GET-запит до відкритого метеорологічного сервера Open-Meteo API. Сервер повертає відповідь у форматі JSON, звідки парсер вилучає актуальні дані про температуру повітря та швидкість вітру для миттєвого відображення в інтерфейсі.

Паралельно з цим система виконує транзакцію читання історичних масивів даних: вилучаються всі записи аналізів для конкретного поля, відсортовані за хронологією (ORDER BY sample_date). Сформований масив передається до математичного модуля. За допомогою функції numpy.polyfit розраховується поліном першого ступеня, що дозволяє знайти коефіцієнти лінійної регресії за

методом найменших квадратів. На основі знайденого рівняння екстраполюється лінія тренду, яка візуалізує прогноз зміни макроелементів на наступні 30 днів.

Ця багатоетапна взаємодія компонентів у часі формалізована за допомогою UML-діаграми послідовності.

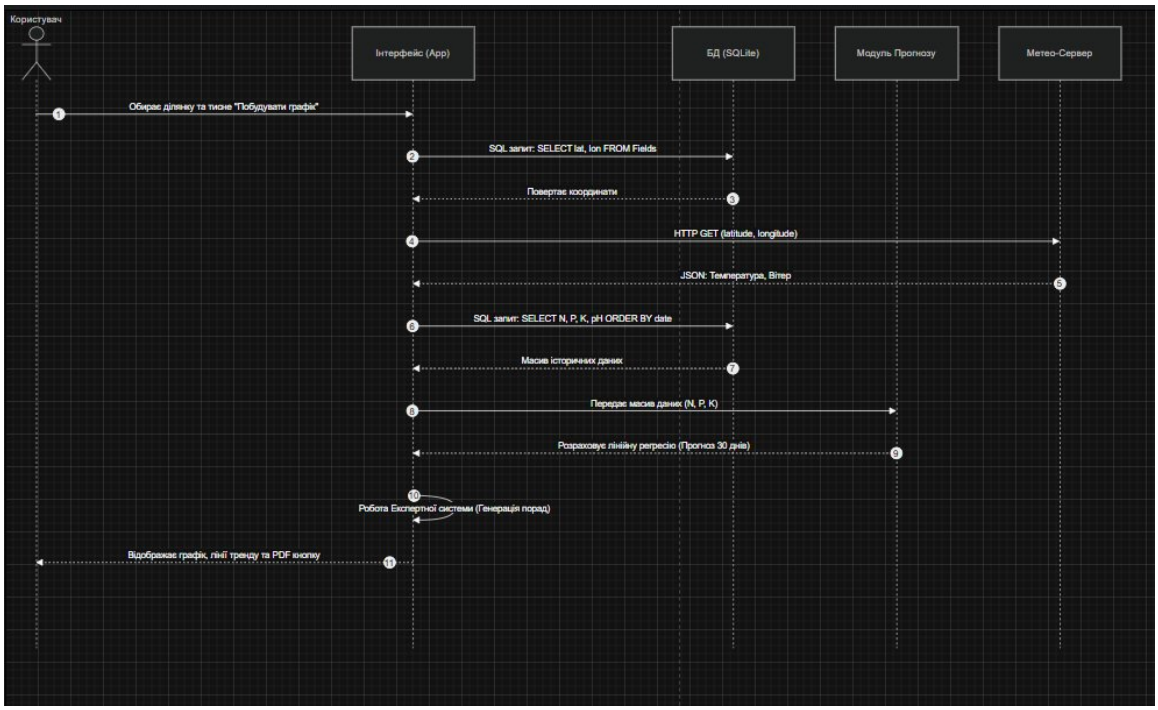


Рисунок 2.4 - UML-діаграма послідовності процесу формування аналітичного прогнозу

Не менш важливим компонентом системи є алгоритм роботи Експертної системи підтримки прийняття рішень. Її завдання - автоматизувати процес інтерпретації лабораторних даних та мінімізувати вплив людського фактору при оцінці критичних станів ґрунту.

Логіка експертної системи побудована на базі жорсткого дерева рішень, що складається з каскаду продукційних правил типу ЯКЩО - ТО. Алгоритм приймає на вхід вектор значень останньої збереженої проби (pH, N, P, K) і здійснює їх послідовну перевірку за встановленими агротехнічними порогами:

1. **Блок аналізу кислотності:** алгоритм перевіряє значення pH. Якщо показник падає нижче 5.5, до результуючого масиву додається критичне попередження з рекомендацією щодо вапнування ґрунту. Альтернативна гілка перевіряє умову pH > 7.5 (лужність), пропонуючи внесення гіпсу або кислих добрив. Якщо обидві умови хибні, фіксується нормальний стан кислотності.

2. **Блок аналізу макроелементів:** алгоритм послідовно оцінює концентрацію азоту (умова N < 40 мг/кг), фосфору (P < 20 мг/кг) та калію (K < 40 мг/кг). Виявлення дефіциту будь-якого з елементів ініціює додавання відповідної рекомендації щодо цільового підживлення (селітра, суперфосфат, калійна сіль).

3. **Блок агрегації результатів:** сформований масив текстових рекомендацій об'єднується в єдиний структурований висновок. Цей текст виводиться у спеціальний текстовий віджет на дашборді та автоматично інтегрується в офіційний звіт (модуль експорту PDF) для подальшого документообігу.

Логіка маршрутизації умов та послідовність виконання операцій алгоритму експертної системи візуалізована за допомогою UML-діаграми діяльності**(Додаток А)**

Таким чином, розроблена архітектура, спроектована модель бази даних та формалізовані алгоритми обробки інформації повністю задовольняють вимоги технічного завдання. Проведене на етапі проектування моделювання гарантує стабільність бізнес-процесів та створює надійне інженерне підґрунтя для переходу до етапу безпосередньої програмної реалізації (імплементації) інформаційної системи.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ІНСТРУКЦІЯ КОРИСТУВАЧА

3.1. Опис інтерфейсу користувача та процесу авторизації

Коли бази даних та архітектура були готові, я перейшов до написання самого коду та створення інтерфейсу. Замість стандартного і візуально застарілого модуля tkinter, я використав сучасну бібліотеку CustomTkinter. Вона дозволила зробити дизайн набагато приємнішим і одразу додала підтримку темної теми. Враховуючи, що фахівцям доводиться годинами вдивлятися в цифри аналізів на моніторі, темний фон значно знижує навантаження на очі.

Відповідно до вимог інформаційної безпеки, доступ до комерційної та наукової інформації, що зберігається в базі даних, є обмеженим. При запуску виконуваного файлу програми ініціалізується стартовий фрейм авторизації (build_login_screen). Інтерфейс містить текстові поля для введення логіна та пароля, причому поле пароля використовує маскування вводу (символ *) для запобігання візуальній компрометації даних.

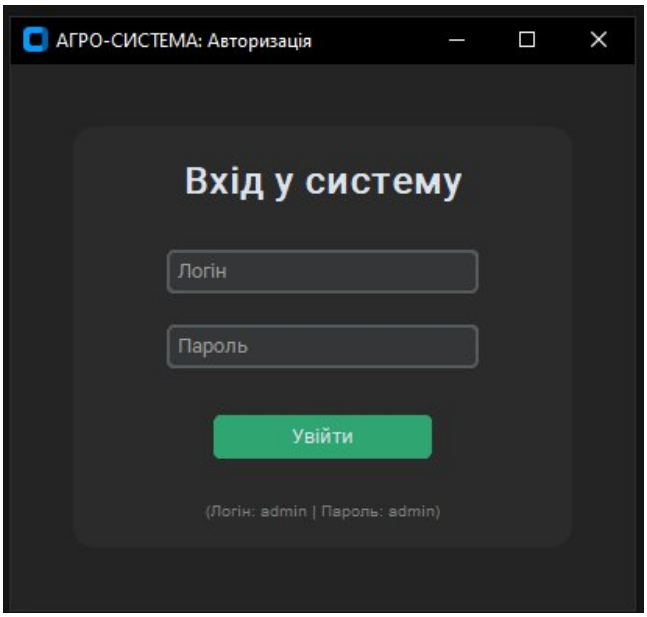


Рисунок 3.1 - Графічний інтерфейс підсистеми автентифікації користувача

При натисканні кнопки Увійти система зчитує введені дані та передає їх до функції check_login(). Алгоритм застосовує криптографічну хеш-функцію SHA-256 до введеного пароля і порівнює отриманий хеш із записом у таблиці

Users локальної бази даних SQLite. У разі невідповідності генерується системне модальне вікно з повідомленням про помилку. При успішній автентифікації фрейм входу знищується, і програма ініціалізує головне вікно системи (Main Dashboard).

Головне вікно спроектовано за принципом адаптивної сітки. Робочий простір розділено на дві основні логічні зони: фіксовану бокову навігаційну панель зліва та динамічну область відображення контенту справа. Розмір вікна за замовчуванням становить 1200x800 пікселів із встановленими обмеженнями на мінімальний розмір для запобігання деформації віджетів.

Бокова панель містить набір навігаційних кнопок, які відповідають за виклик методу-маршрутизатора show_frame(). Цей метод керує видимістю попередньо ініціалізованих фреймів, що забезпечує миттєве перемикання між модулями без повторного рендерингу (перемальовування) всього вікна.

Першим екраном, який зустрічає користувача після успішного входу, є Головна панель управління. На цьому екрані реалізовано агрегацію базових статистичних метрик бази даних. Під час ініціалізації фрейму система виконує SQL-запити до таблиць Fields та Soil_Samples із використанням агрегатної функції COUNT(), після чого виводить на екран загальну кількість зареєстрованих земельних ділянок та сумарну кількість проведених лабораторних аналізів.

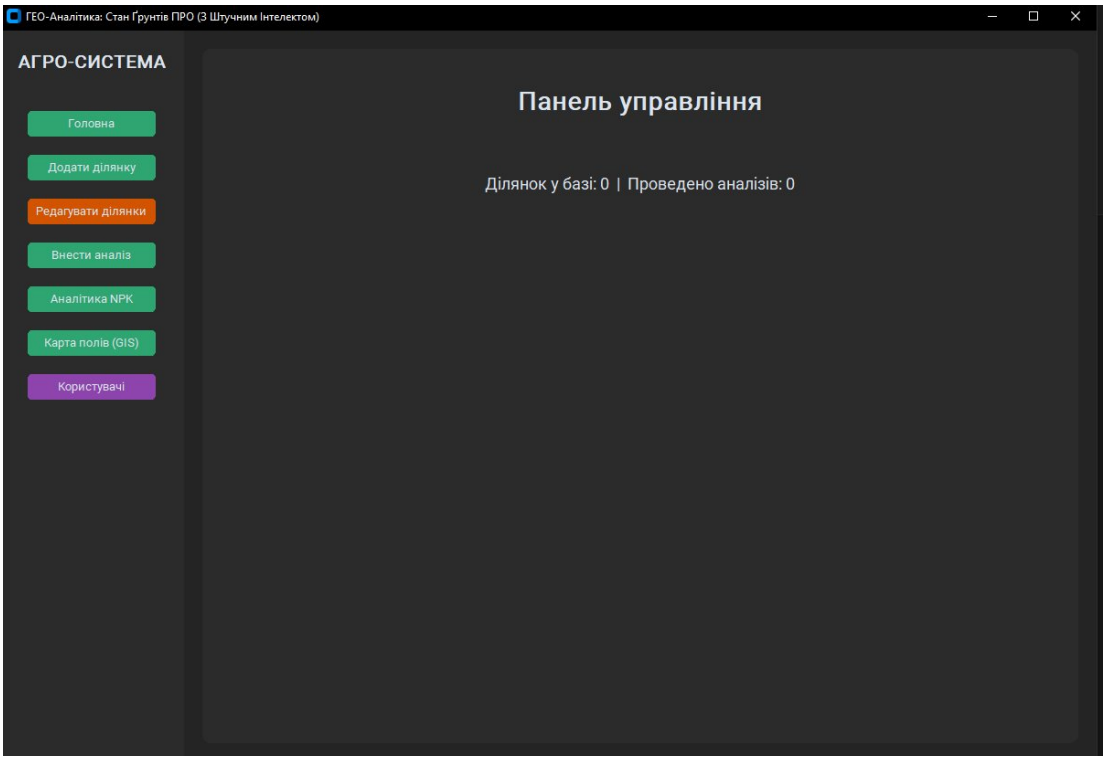


Рисунок 3.2 - Головна панель управління (Дашборд) інформаційної системи

Такий підхід до побудови інтерфейсу забезпечує високий рівень ергономіки та дозволяє користувачу швидко оцінити поточний обсяг інформації в базі даних одразу після запуску програми.

3.2. Робота з геопросторовими даними та підсистема картографування

Наступним критичним елементом взаємодії є модуль керування просторовими даними. Для забезпечення коректної роботи аналітичних та метеорологічних функцій, кожна нова земельна ділянка повинна бути зареєстрована в системі.

Процес реєстрації здійснюється у вкладці Додати ділянку. Екран містить форму введення, яка складається з текстових полів для фіксації назви поля, його площі у гектарах, типу ґрунту, а також точних географічних координат: широти та довготи. Застосовано базову валідацію типів даних: при спробі зберегти текстові символи у поля, що вимагають числових значень (площа, координати), система перехоплює виняток та блокує транзакцію запису в базу даних, інформуючи користувача модальним вікном.

Після реєстрації ділянки користувач отримує доступ до підсистеми картографування (вкладка Карта полів (GIS)). Інтерфейс цього модуля реалізовано за допомогою віджета TkinterMapView.

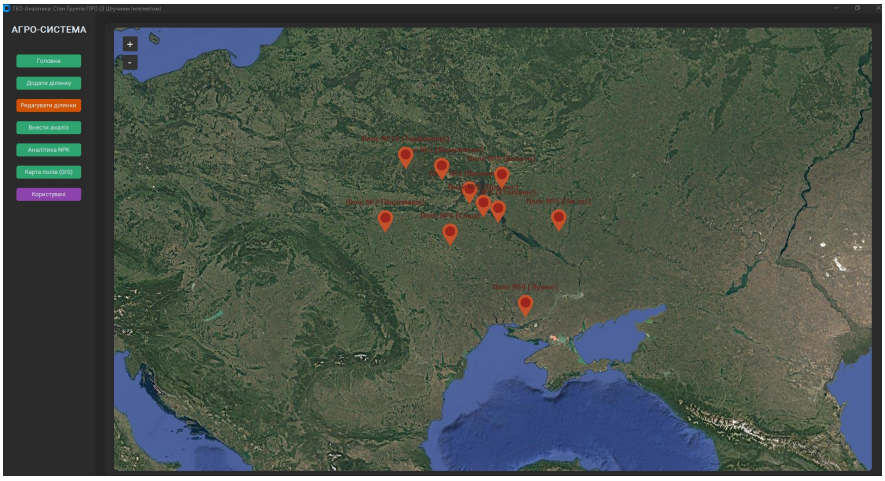


Рисунок 3.3 - Відображення земельних ділянок на інтерактивній супутниковій карті

Особливістю програмної реалізації даного модуля є використання URL-шаблону mt0.google.com/vt/lyrs=s для завантаження реальних супутникових знімків високої роздільної здатності замість схематичних векторних карт. При відкритті вкладки система виконує ітерацію по всіх записах таблиці Fields, де координати не дорівнюють NULL, і динамічно встановлює маркери на карті. Для забезпечення стабільної роботи без перевантаження центрального процесора налаштовано локальне кешування картографічних тайлів у файл map_cache.db. Це дозволяє програмі миттєво завантажувати раніше переглянуті ділянки карти навіть за відсутності активного інтернет з'єднання.

На даному етапі життєвого циклу програмного продукту реалізовано шар точкової просторової прив'язки. Замість складних полігональних меж (які перевантажують локальну базу даних дрібних фермерів), система використовує центроїди – точні географічні координати (широта і довгота) центру кожної ділянки. Це забезпечує жорстку прив'язку до місцевості на супутниковому шарі Google Satellite. Такого "точкового шару" абсолютно достатньо для коректного відправлення HTTP-запитів до метеорологічних API та візуальної орієнтації

агронома на місцевості. Впровадження полігональних векторних шарів винесено як напрямок подальшої модернізації системи.

3.3. Модуль аналітики, прогнозування та експертна система

Найбільш ресурсоємним компонентом розробленої інформаційної системи є підсистема математичної аналітики та підтримки прийняття рішень. Доступ до цього функціоналу здійснюється через вкладку Аналітика NPK бокового навігаційного меню. Інтерфейс модуля поділено на верхню панель управління, текстовий блок виведення метеорологічних та експертних даних, а також область рендерингу графіків.

Для ініціалізації процесу аналізу користувач обирає цільову земельну ділянку з випадаючого списку та натискає кнопку Побудувати графік та Прогноз. На цьому етапі система виконує комплексний алгоритм обробки даних:

1. **Отримання метеорологічних даних:** За координатами обраної ділянки система формує синхронний HTTP-запит до сервера Open-Meteo. Отримані дані (поточна температура повітря та швидкість вітру) одразу відображаються у верхній частині вікна, що дозволяє агроному оцінити сприятливість поточних погодних умов для внесення добрив.

2. **Робота Експертної системи:** Програма звертається до бази даних (БД) та вилучає параметри останнього проведеного лабораторного аналізу (рівень pH, вологість, азот, фосфор, калій). Ці дані пропускаються через блок продукційних правил. Результат роботи алгоритму (наприклад, попередження про критичну кислотність та необхідність вапнування) виводиться у спеціальний текстовий віджет з міткою дати останнього аналізу.

3. **Візуалізація та математичне прогнозування:** Використовуючи бібліотеку matplotlib, система будує три графіки часових рядів для кожного з макроелементів (N, P, K). Одночасно підключається модуль машинного навчання. За допомогою функції polyfit бібліотеки NumPy обчислюються коефіцієнти лінійної регресії, і на графік накладаються пунктирні лінії тренду, які екстраполюють зміну хімічного складу ґрунту на 30 днів у майбутнє.

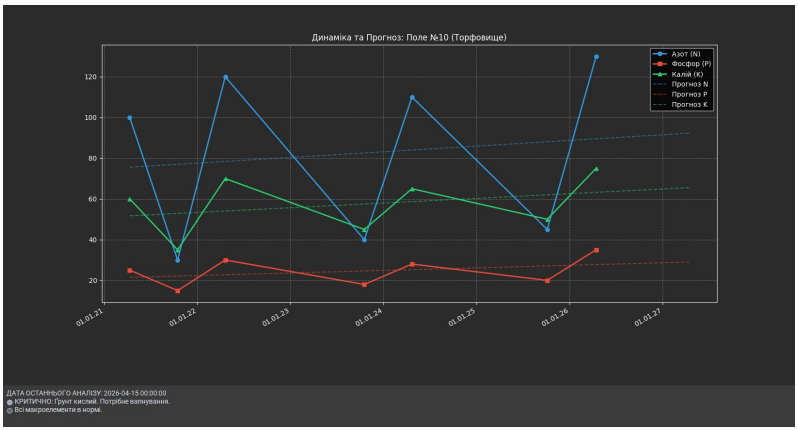


Рисунок 3.4 - Інтерфейс аналітичного модуля з візуалізацією математичного прогнозу та висновками експертної системи

На рисунку 3.4 представлено результат роботи аналітичного ядра системи. Дані для графіка генеруються динамічно: програма виконує SQL-запит до таблиці Soil_Samples, вилучаючи всі історичні показники для обраного поля. Вісь абсцис (X) формується з дат забору проб, а вісь ординат (Y) – з концентрації NPK у мг/кг. Пунктирні лінії прогнозу розраховуються за допомогою методів машинного навчання, а саме алгоритму поліноміальної лінійної регресії (метод найменших квадратів), реалізованого через функцію polyfit бібліотеки NumPy. Система знаходить оптимальне рівняння прямої, продовжує його на 30 днів вперед (вікно прогнозування) та накладає на основний графік за допомогою бібліотеки Matplotlib.

Графік інтегрується безпосередньо у вікно Tkinter за допомогою бекенду FigureCanvasTkAgg. Використання темної підкладки для графіка забезпечує цілісність візуального сприйняття загального інтерфейсу програми.

3.4. Підсистема імпорту/експорту даних та генерації звітності

Для забезпечення інтероперабельності (здатності системи взаємодіяти з іншим програмним забезпеченням) в ІС було імплементовано модулі масового імпорту та генерації офіційної звітності.

Введення великих масивів історичних даних вручну є неефективним. Тому у вкладці Внести аналіз реалізовано кнопку Імпорт бази з Excel. При її натисканні викликається стандартне діалогове вікно операційної системи для вибору файлу. Бібліотека pandas зчитує вміст обраного файлу формату .xlsx у структуру

DataFrame. Далі система ітерує рядки таблиці та автоматично виконує масове додавання записів до таблиці Soil_Samples бази даних SQLite, інформуючи користувача про кількість успішно імпортованих рядків.

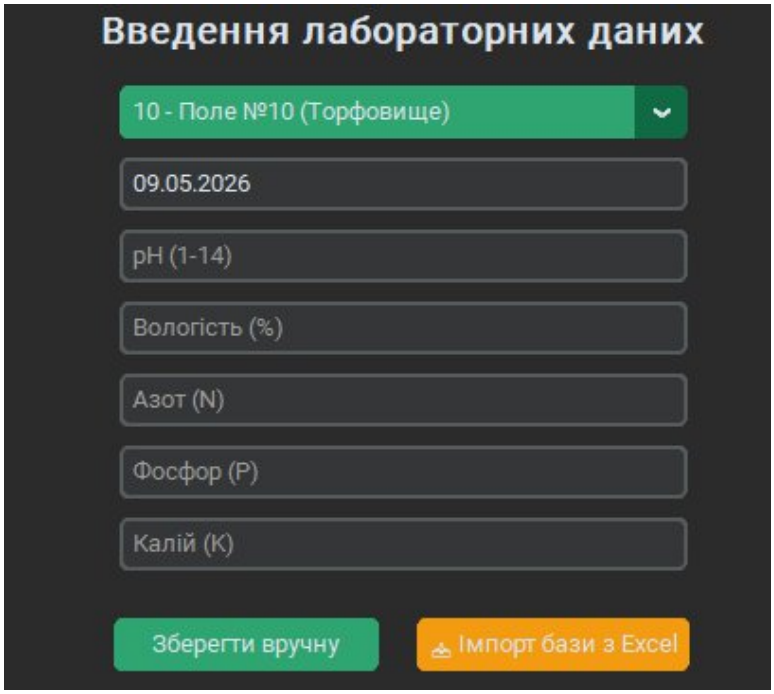


Рисунок 3.5 - Інтерфейс модуля реєстрації лабораторних досліджень та імпорту зовнішніх даних

Інтерфейс реєстрації лабораторних досліджень (рис. 3.5) є головним інструментом наповнення бази даних. Користувач вводить ключові параметри, визначені предметною областю: водневий показник (pH), відсоток вологості та наявність макроелементів (N, P, K). Ці дослідження є критично необхідними, оскільки саме вони є вхідними параметрами для роботи Експертної системи. Якщо не зафіксувати ці дані у БД, система не зможе виявити деградацію ґрунту, розрахувати регресійний прогноз та згенерувати поради щодо внесення меліорантів.

Кінцевим етапом роботи агронома є формування звітних документів для керівництва або архіву. Цей функціонал винесено у верхню панель вкладки Аналітика. Система пропонує два формати експорту:

Експорт в Excel (Табличний формат): Дозволяє вивантажити всі "сирі" дані по обраній ділянці з БД у новий файл .xlsx для подальшої обробки в зовнішніх програмах.

Звіт у PDF (Офіційний документ): При натисканні цієї кнопки система генерує комплексний документ формату А4, готовий до друку. Особливістю реалізації є те, що програма автоматично "перемальовує" темний графік у світлий стиль (для економії фарби принтера) у фоновому режимі. Згенерований PDF-файл містить заголовок, назву ділянки, дату створення, повний текст рекомендацій Експертної системи та вбудоване растрове зображення графіка динаміки з прогнозом.

ОФІЦІЙНИЙ ЗВІТ АНАЛІЗУ ҐРУНТІВ

Об'єкт: Поле №10 (Торфовище)

Дата генерації: 09.05.2026 20:56

1. ВИСНОВОК ЕКСПЕРТНОЇ СИСТЕМИ ТА РЕКОМЕНДАЦІЇ:

ДАТА ОСТАННЬОГО АНАЛІЗУ: 2026-04-15 00:00:00
 КРИТИЧНО: Ґрунт кислий. Потрібне вапнування.
 Всі макроелементи в нормі.

2. ГРАФІК ДИНАМІКИ ТА ПРОГНОЗУ НА 30 ДНІВ:

Рисунок 3.6 - Приклад згенерованого системою комплексного PDF-звіту

Успішна програмна імплементація описаних модулів підтверджує повноту виконання поставленого технічного завдання. Розроблений графічний інтерфейс є інтуїтивно зрозумілим для кінцевого користувача, забезпечуючи високу швидкість обробки агрономічних даних.

3.4 Підсистема адміністрування та управління доступом

Для забезпечення безпеки та розмежування прав у багатокористувацьких інформаційних системах критично важливою складовою є модуль адміністрування. У розробленій системі реалізовано механізм реєстрації нових користувачів та видалення існуючих облікових записів.

При створенні нового профілю система проводить валідацію введених даних (мінімальна довжина логіна та пароля). З метою забезпечення конфіденційності, паролі не зберігаються у відкритому вигляді. Перед записом у базу даних (SQLite) пароль проходить незворотне криптографічне хешування за алгоритмом SHA-256.

Процес видалення користувачів також супроводжується логічними перевірками. Зокрема, реалізовано програмний захист (запобіжник) від випадкового або навмисного видалення головного адміністратора системи (користувача admin). Це гарантує безперебійний доступ до панелі управління системою навіть у випадку помилкових дій персоналу.

Візуальне представлення модуля управління користувачами з полями введення та випадаючим списком існуючих облікових записів наведено на рисунку 3.7.

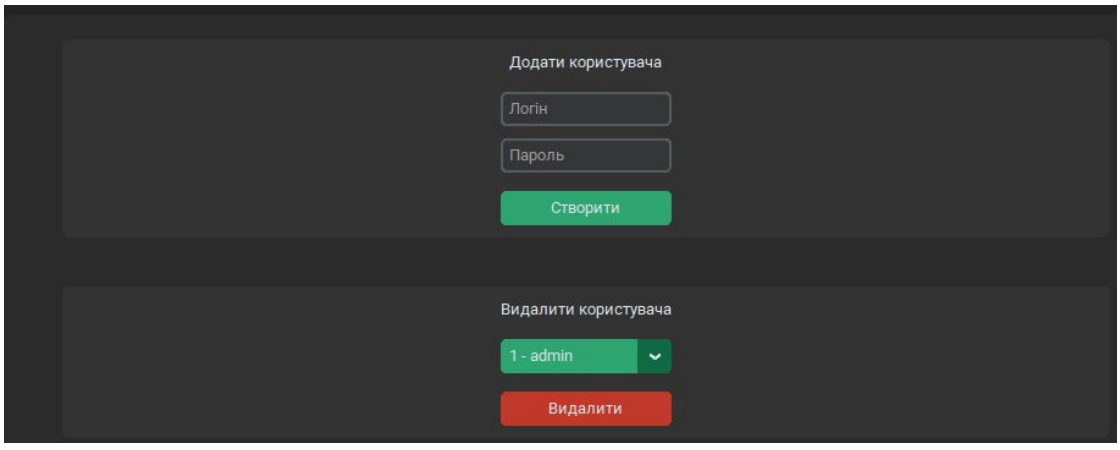


Рисунок 3.7 - Інтерфейс панелі адміністрування та управління користувачами

ВИСНОВОК

Під час виконання дипломної роботи було розроблено спеціалізовану автономну інформаційну систему для моніторингу, аналізу та прогнозування агрохімічних показників ґрунтів. Проведено системний аналіз предметної області та існуючих програмних рішень, табличних процесорів, ГІС). Доведено, що для потреб середніх та дрібних агропідприємств оптимальним архітектурним рішенням є створення автономного товстого клієнта, який не залежить від хмарної інфраструктури та абонентської плати. Також сформульовано функціональні та нефункціональні вимоги до програмного продукту. На їх основі спроектовано логічну та фізичну моделі локальної реляційної бази даних у середовищі SQLite3, яка забезпечує транзакційну надійність та посилальну цілісність збереження просторових і хімічних даних.

Здійснено об'єктно-орієнтоване моделювання архітектури системи із застосуванням мови UML (побудовано діаграми класів, послідовності, діяльності) та розроблено діаграму інформаційних потоків (DFD). Це забезпечило високий рівень модульності та масштабованості програмного коду. Засобами мови програмування Python та фреймворку CustomTkinter розроблено сучасний ергономічний графічний інтерфейс користувача (GUI). Інтерфейс підтримує принцип єдиного вікна та містить вбудовану підсистему адміністрування з криптографічним захистом паролів (SHA-256). Імплементовано геоінформаційний модуль на базі TkinterMapView, який дозволяє візуалізувати земельні ділянки на супутникових картах із використанням алгоритмів локального кешування тайлів. Розроблено та інтегровано модуль математичної аналітики і машинного навчання. Застосування алгоритмів поліноміальної регресії (бібліотека NumPy) дозволило системі автоматично екстраполювати історичні дані та будувати графіки динаміки макроелементів (NPK) з прогнозом на майбутні періоди. Створено експертну підсистему підтримки прийняття рішень на базі продукційних правил, яка автоматично генерує агротехнічні рекомендації щодо внесення меліорантів

та добрив. Додатково реалізовано механізми масового імпорту історичних даних та генерації комплексної офіційної звітності у форматах PDF і Excel.

Розроблена інформаційна система "ІС Аналізу Стану Ґрунтів ПРО" пройшла успішне тестування, є повністю автономною, економічно доцільною для впровадження у виробничі процеси агрономічних лабораторій та фермерських господарств, і повністю відповідає вимогам технічного завдання.

Список використаних джерел

1. Про державний контроль за використанням та охороною земель : Закон України від 19.06.2003 № 963-IV / Верховна Рада України. URL: https://zakon.rada.gov.ua/laws/show/963-15
2. ДСТУ 4362:2004. Якість ґрунту. Показники родючості ґрунтів. Київ : Держспоживстандарт України, 2005. 32 с.
3. ДСТУ 3987-2000. Якість ґрунту. Визначення рухомих сполук фосфору і калію за модифікованим методом Мачигіна. Київ : Держстандарт України, 2000. 12 с.
4. Гнатенко О. Ф., Капштик М. В., Петренко Л. Р., Вітвіцький С. В. Ґрунтознавство з основами геології : навч. посіб. Київ : Оранта, 2005. 648 с.
5. Полупан М. І., Соловей В. Б., Величко В. А. Класифікація ґрунтів України. Київ : Аграрна наука, 2005. 556 с.
6. Тараріко Ю. Г. Формування сталих агроекосистем: теорія і практика. Київ : Аграрна наука, 2005. 508 с.
7. Демиденко О. В. Моніторинг родючості ґрунтів України в умовах змін клімату. *Вісник аграрної науки*. 2018. № 10. С. 15-22.
8. Медведєв В. В. Моніторинг ґрунтів України. Концепція, попередня програма, результати. Харків : 14-та друкарня, 2002. 428 с.
9. Лук'янова В. В. Комп’ютерні мережі та системи : підручник. Київ : Академвидав, 2013. 416 с.
10. Співаковський О. В., Щедролосьєв Д. Є. Проектування інформаційних систем : навч. посіб. Херсон : ХДУ, 2005. 164 с.
11. Гвоздецька Н. О. Побудова інтелектуальних систем підтримки прийняття рішень. *Інформаційні технології та комп’ютерна інженерія*. 2019. № 2. С. 45-52.
12. Пасічник В. В., Жежнич П. І., Кравець Р. Б. Глобальні інформаційні системи та технології (моделі ефективного аналізу даних) : навч. посіб. Львів : Видавництво Львівської політехніки, 2014. 488 с.
13. Lutz M. Learning Python. 5th ed. Sebastopol : O'Reilly Media, 2013. 1600 p.
14. Freeman E., Robson E. Head First Design Patterns. 2nd ed. Sebastopol : O'Reilly Media, 2020. 672 p.
15. McKinney W. Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython. 2nd ed. Sebastopol : O'Reilly Media, 2017. 547 p.
16. Sedgewick R., Wayne K. Algorithms. 4th ed. Upper Saddle River : Addison-Wesley, 2011. 976 p.
17. Albon C. Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning. Sebastopol : O'Reilly Media, 2018. 366 p.
18. CustomTkinter Documentation. URL: https://customtkinter.tomschimansky.com/documentation/
19. SQLite Home Page. Structured Query Language in SQLite. URL: https://www.sqlite.org/lang.html (дата звернення: 05.04.2026).
20. Matplotlib: Visualization with Python. URL: https://matplotlib.org/stable/contents.html (дата звернення: 10.04.2026).
21. Pandas Documentation. URL: https://pandas.pydata.org/docs/
22. TkinterMapView Documentation. URL: https://github.com/TomSchimansky/TkinterMapView
23. Open-Meteo Free Weather API. URL: https://open-meteo.com/en/docs
24. NumPy Documentation. Polynomial regression using polyfit. URL: https://numpy.org/doc/stable/reference/generated/numpy.polyfit.html

25. ReportLab PDF Library. User Guide. URL: https://www.reportlab.com/docs/reportlab-userguide.pdf
26. Булигін С. Ю. Точне землеробство — шлях до сталого розвитку. *Агроном*. 2020. № 1. С. 10-14.
27. Коваль П. В. ГІС в агрономії: сучасний стан та перспективи. *Геоінформатика*. 2021. № 3. С. 56-61.
28. Ткаченко М. А. Оптимізація живлення рослин на основі агрохімічних картограм. *Землеробство*. 2019. Вип. 94. С. 33-40.
29. Агрохімічний стан ґрунтів України : монографія / за ред. С. А. Балюка, В. В. Медведєва. Київ : Аграрна наука, 2012. 248 с.
30. Хом'як О. В. Тестування та верифікація програмного забезпечення : навч. посіб. Львів : Магнолія 2006, 2015. 256 с.
31. ДСТУ ISO/IEC 12207:2015 Інженерія систем і програмного забезпечення. Процеси життєвого циклу програмного забезпечення. Київ : ДП УкрНДНЦ. 2016. 104 с.
32. Python Software Foundation. Python 3.14 Language Reference. URL: https://docs.python.org/3.14/

ДОДАТКИ

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Декларація про академічну доброчесність та використання ШІ

ДЕКЛАРАЦІЯ

Я, Єсіп Вікторія Іванівна, здобувач(ка) НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

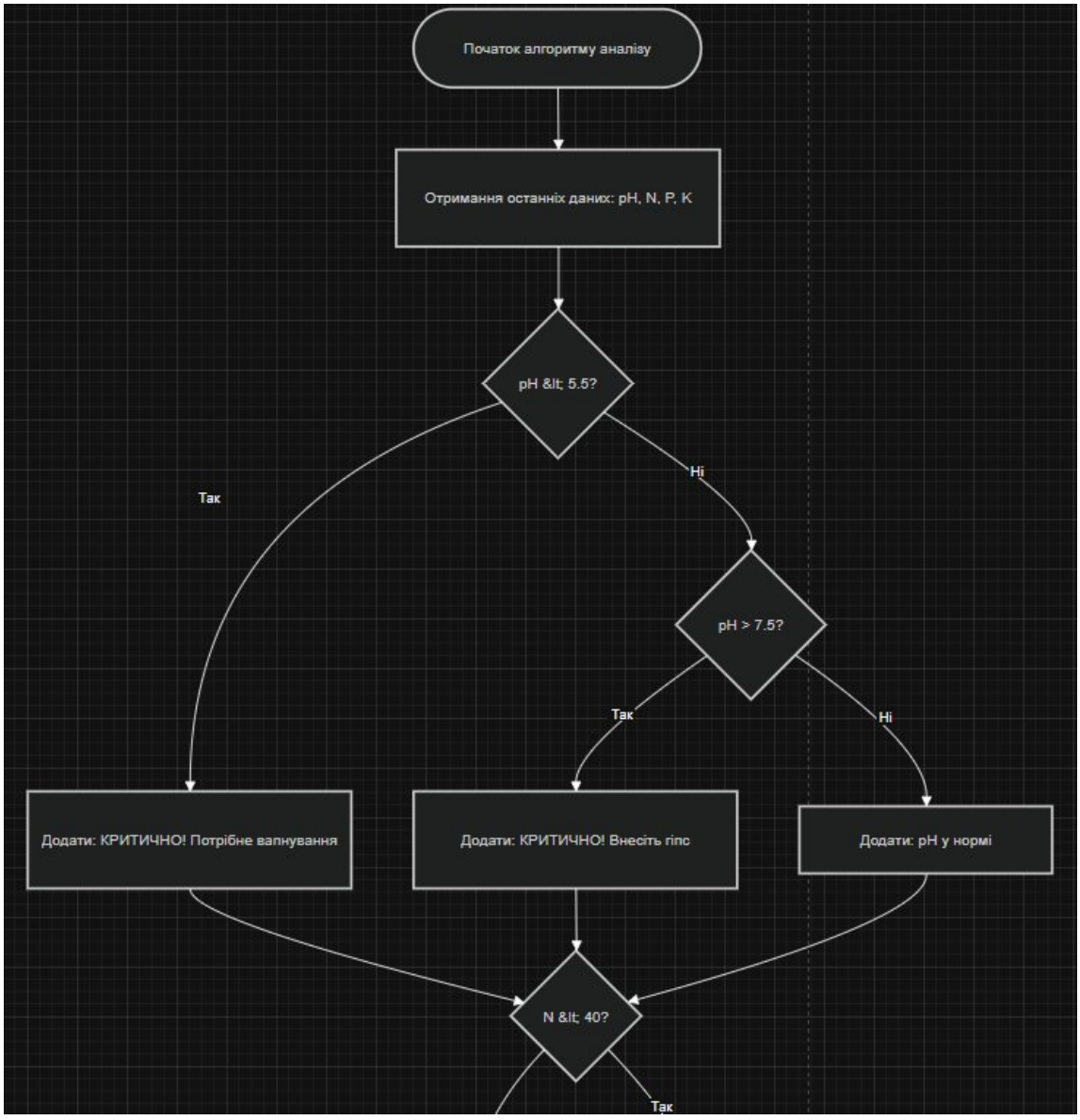
1. Моя бакалаврська кваліфікаційна робота (проєкт) на тему «Розробка web орієнтованої інформаційної системи управління малим торговельним підприємством» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

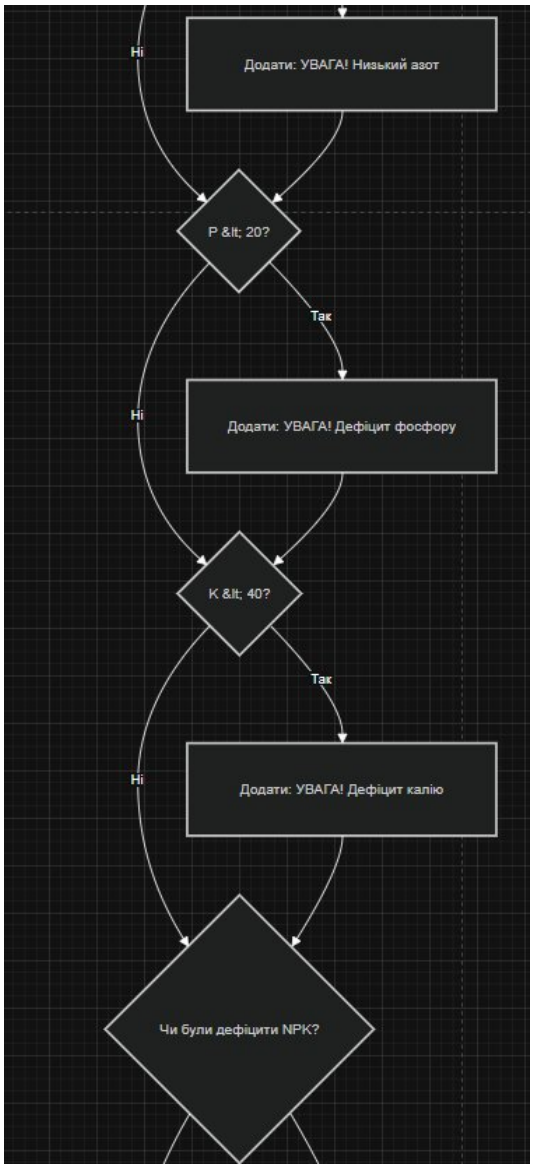
- [0] інструменти генеративного ШІ не використовувалися.
- [0] інструменти ШІ (ChatGPT, Claude) були використані для:
  - [0] перекладу іншомовних джерел;
  - [ ] стилістичного редагування та перевірки граматики;
  - [ ] допомоги у написанні/відлагодженні програмного коду;
  - [ ] інше: _____.

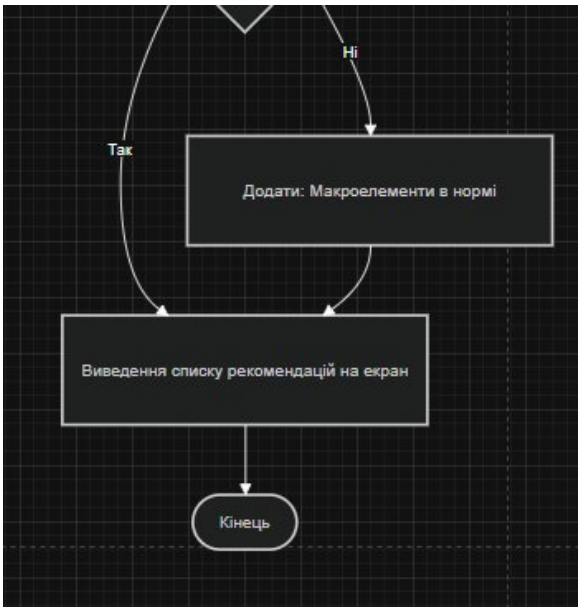
Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«__» _____2026р. _____ **Олександр Кондратюк**

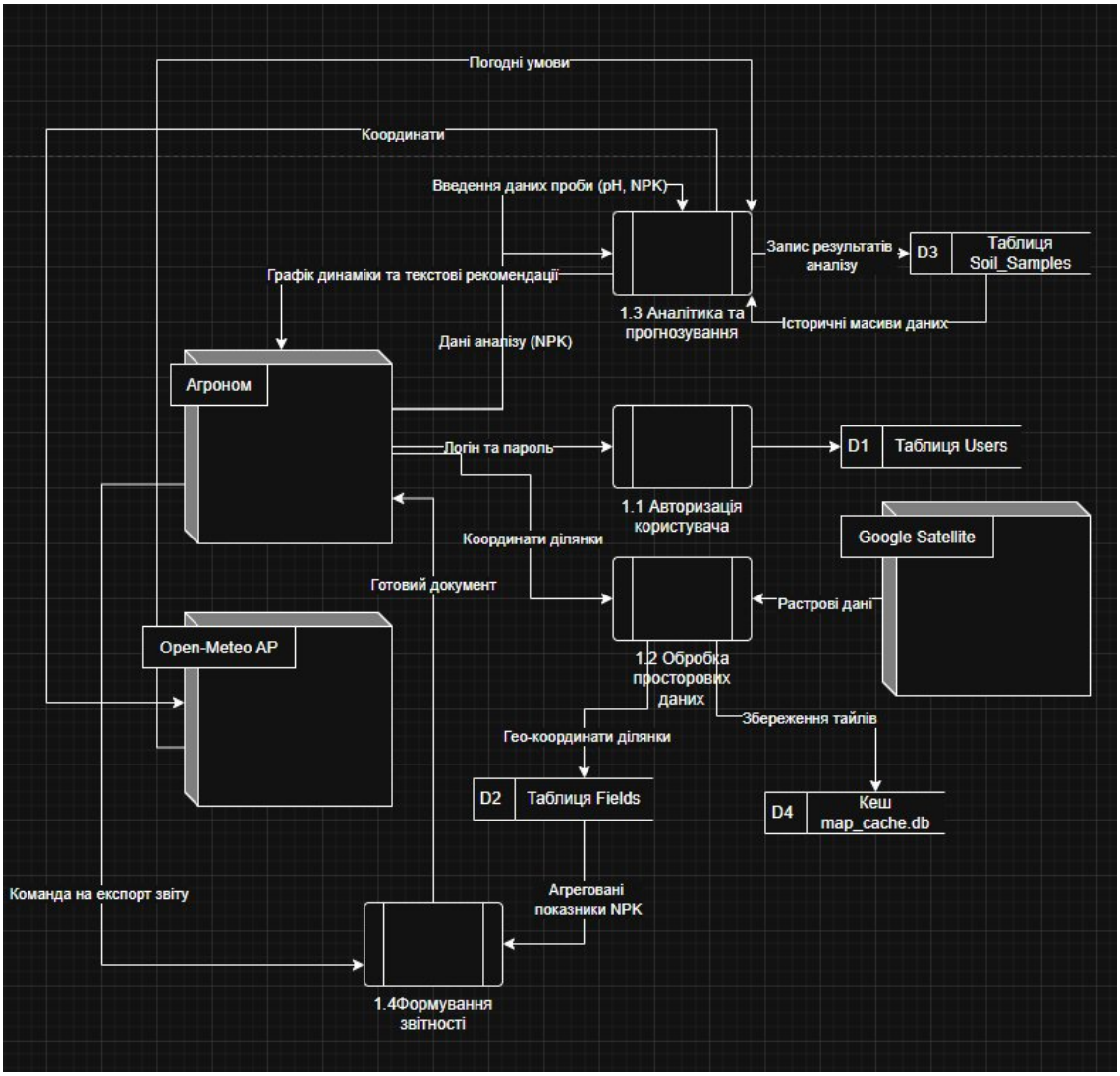
Додаток А: UML-діаграма діяльності алгоритму експертної системи



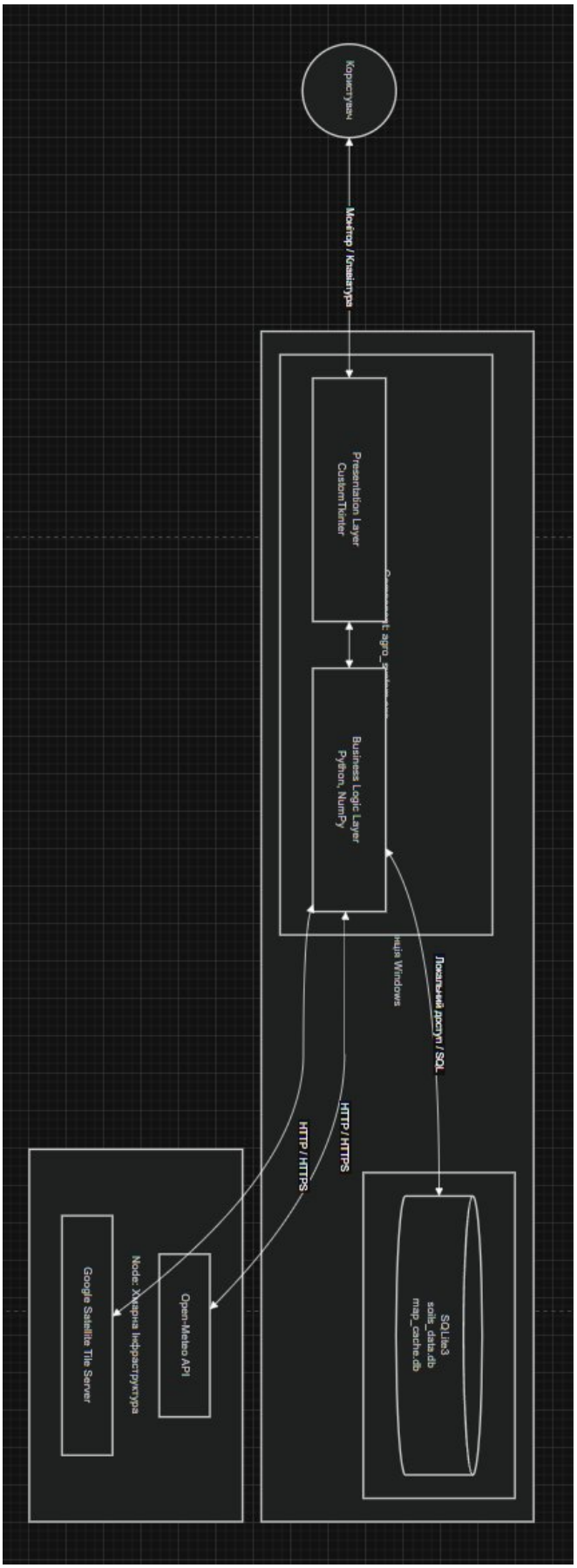

Ні
Додати: УВАГА! Низький азот
P < 20?
Так
Ні
Додати: УВАГА! Дефіцит фосфору
K < 40?
Так
Ні
Додати: УВАГА! Дефіцит калію
Чи були дефіцити NPK?


Ні
Так
Додати: Макроелементи в нормі
Виведення списку рекомендацій на екран
Кінець

Додаток Б Діаграма інформаційних потоків (DFD)



Додаток В UML-Діаграма розгортання



Додаток Г Лістинг програми

```
import sqlite3
import customtkinter as ctk
from tkinter import messagebox, filedialog
from datetime import datetime
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import pandas as pd
import numpy as np
import tkintermapview
import hashlib
import os
import requests
import io
from matplotlib.image import imread

def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()

def init_db():
    conn = sqlite3.connect('soils_data.db')
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS Users (
                      id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT UNIQUE, password
TEXT)''')
    cursor.execute('''CREATE TABLE IF NOT EXISTS Fields (
                      id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT NOT NULL, area REAL,
soil_type TEXT, lat REAL, lon REAL)''')
    cursor.execute('''CREATE TABLE IF NOT EXISTS Soil_Samples (
                      id INTEGER PRIMARY KEY AUTOINCREMENT, field_id INTEGER, sample_date TEXT,
ph_level REAL, moisture REAL, nitrogen REAL, phosphorus REAL, potassium REAL, FOREIGN
KEY(field_id) REFERENCES Fields(id))''')
    cursor.execute("SELECT * FROM Users WHERE username='admin'")
    if not cursor.fetchone():
        cursor.execute("INSERT INTO Users (username, password) VALUES (?, ?)", ('admin',
hash_password('admin')))
    conn.commit()
    conn.close()

def check_login(username, password):
    conn = sqlite3.connect('soils_data.db')
    c = conn.cursor()
    c.execute("SELECT * FROM Users WHERE username=? AND password=?", (username,
hash_password(password)))
    user = c.fetchone()
    conn.close()
    return user is not None

def add_user_to_db(username, password):
    try:
```

```
        conn = sqlite3.connect('soils_data.db')
        c = conn.cursor()
        c.execute("INSERT INTO Users (username, password) VALUES (?, ?)", (username,
hash_password(password)))
        conn.commit()
        conn.close()
        return True
    except sqlite3.IntegrityError:
        return False

def delete_user_from_db(user_id):
    conn = sqlite3.connect('soils_data.db')
    c = conn.cursor()
    c.execute('DELETE FROM Users WHERE id=?', (user_id,))
    conn.commit()
    conn.close()

def get_current_weather(lat, lon):
    if not lat or not lon: return "Координати не задані"
    try:
        url = f"https://api.open-
meteo.com/v1/forecast?latitude={lat}&longitude={lon}&current_weather=true"
        r = requests.get(url, timeout=3).json()
        cw = r.get("current_weather", {})
        temp = cw.get("temperature", "?")
        wind = cw.get("windspeed", "?")
        return f"Погода зараз: {temp}°C, Вітер: {wind} км/год"
    except:
        return "Немає зв'язку з метеостанцією"

def export_to_excel(field_id, field_name):
    try:
        conn = sqlite3.connect('soils_data.db')
        df = pd.read_sql_query(f"SELECT sample_date as 'Дата', ph_level as 'pH', moisture as
'Вологість (%)', nitrogen as 'Азот (N)', phosphorus as 'Фосфор (P)', potassium as 'Калій (K)' FROM
Soil_Samples WHERE field_id={field_id}", conn)
        conn.close()
        if df.empty: return False, "Немає даних."
        filename = f"Звіт_Агро_{field_name.replace(' ', '_')}.xlsx"
        df.to_excel(filename, index=False)
        return True, f"Збережено у файл:\n{os.path.abspath(filename)}"
    except Exception as e:
        return False, str(e)

def generate_recommendations(ph, n, p, k):
    recs = []
    if ph < 5.5: recs.append("🔴 КРИТИЧНО: Ґрунт кислий. Потрібне вапнування.")
    elif ph > 7.5: recs.append("🔴 КРИТИЧНО: Ґрунт лужний. Внесіть гіпс.")
    else: recs.append("   pH у нормі.")
```

```
    if n < 40: recs.append("    УВАГА: Низький азот (N). Потрібне підживлення.")
    if p < 20: recs.append("    УВАГА: Дефіцит фосфору (P). Внесіть суперфосфат.")
    if k < 40: recs.append("    УВАГА: Дефіцит калію (K). Внесіть калійну сіль.")
    if len(recs) == 1: recs.append("    Всі макроелементи в нормі.")
    return "\n".join(recs)

ctk.set_appearance_mode("Dark")
ctk.set_default_color_theme("green")

class MainApp(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("АГРО-СИСТЕМА: Авторизація")
        self.geometry("400x350")
        self.eval('tk::PlaceWindow . center')
        self.build_login_screen()

    def build_login_screen(self):
        self.login_frame = ctk.CTkFrame(self, corner_radius=15)
        self.login_frame.pack(pady=40, padx=40, fill="both", expand=True)
        ctk.CTkLabel(self.login_frame, text="Вхід у систему", font=ctk.CTkFont(size=24,
weight="bold")).pack(pady=(20, 20))
        self.entry_user = ctk.CTkEntry(self.login_frame, placeholder_text="Логін", width=200)
        self.entry_user.pack(pady=10)
        self.entry_pass = ctk.CTkEntry(self.login_frame, placeholder_text="Пароль", width=200,
show="*")
        self.entry_pass.pack(pady=10)
        ctk.CTkButton(self.login_frame, text="Увійти", command=self.do_login).pack(pady=20)
        ctk.CTkLabel(self.login_frame, text="(Логін: admin | Пароль: admin)", text_color="gray",
font=("Arial", 10)).pack()

    def do_login(self):
        if check_login(self.entry_user.get(), self.entry_pass.get()):
            self.login_frame.destroy()
            self.build_main_dashboard()
        else:
            messagebox.showerror("Помилка", "Невірний логін або пароль!")

    def build_main_dashboard(self):
        self.title("ГЕО-Аналітика: Стан Ґрунтів ПРО (З Штучним Інтелектом)")
        self.geometry("1200x800")
        self.minsize(1000, 700)
        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(1, weight=1)

        self.sidebar = ctk.CTkFrame(self, width=220, corner_radius=0)
        self.sidebar.grid(row=0, column=0, sticky="nsew")
        self.sidebar.grid_rowconfigure(9, weight=1)
```

```
        ctk.CTkLabel(self.sidebar, text="АГРО-СИСТЕМА", font=ctk.CTkFont(size=20,
weight="bold")).grid(row=0, column=0, padx=20, pady=(20, 30))

        buttons = [("Головна", self.show_home), ("Додати ділянку", self.show_add_field),
("Редагувати ділянки", self.show_manage_fields),
                   ("Внести аналіз", self.show_add_sample), ("Аналітика NPK", self.show_analytics),
("Карта полів (GIS)", self.show_map), ("Користувачі", self.show_manage_users)]
        for i, (text, cmd) in enumerate(buttons, start=1):
            btn = ctk.CTkButton(self.sidebar, text=text, command=cmd)
            if text == "Користувачі": btn.configure(fg_color="#8e44ad", hover_color="#9b59b6")
            if text == "Редагувати ділянки": btn.configure(fg_color="#d35400", hover_color="#e67e22")
            btn.grid(row=i, column=0, padx=20, pady=10)

        self.main_frame = ctk.CTkFrame(self, corner_radius=15, fg_color="transparent")
        self.main_frame.grid(row=0, column=1, sticky="nsew", padx=20, pady=20)
        self.main_frame.grid_rowconfigure(0, weight=1)
        self.main_frame.grid_columnconfigure(0, weight=1)

        self.frames = {}
        self.create_home_frame()
        self.create_add_field_frame()
        self.create_manage_fields_frame()
        self.create_add_sample_frame()
        self.create_analytics_frame()
        self.create_map_frame()
        self.create_manage_users_frame()
        self.show_home()

    def create_home_frame(self):
        frame = ctk.CTkFrame(self.main_frame, corner_radius=10)
        self.frames["home"] = frame
        ctk.CTkLabel(frame, text="Панель управління", font=ctk.CTkFont(size=28,
weight="bold")).pack(pady=40)
        self.lbl_stats = ctk.CTkLabel(frame, text="Завантаження статистики...",
font=ctk.CTkFont(size=18))
        self.lbl_stats.pack(pady=20)

    def create_add_field_frame(self):
        frame = ctk.CTkFrame(self.main_frame, corner_radius=10)
        self.frames["add_field"] = frame
        ctk.CTkLabel(frame, text="Реєстрація ділянки", font=ctk.CTkFont(size=20,
weight="bold")).pack(pady=20)
        self.f_name = ctk.CTkEntry(frame, width=300, placeholder_text="Назва ділянки")
        self.f_name.pack(pady=10)
        self.f_area = ctk.CTkEntry(frame, width=300, placeholder_text="Площа (га)")
        self.f_area.pack(pady=10)
        self.f_soil = ctk.CTkEntry(frame, width=300, placeholder_text="Тип ґрунту")
        self.f_soil.pack(pady=10)
        self.f_lat = ctk.CTkEntry(frame, width=300, placeholder_text="Широта (напр. 49.8397)")
        self.f_lat.pack(pady=10)
```

```
        self.f_lon = ctk.CTkEntry(frame, width=300, placeholder_text="Довгота (напр. 24.0297)")
        self.f_lon.pack(pady=10)
        ctk.CTkButton(frame, text="Зберегти", command=self.save_field).pack(pady=20)

    def create_manage_fields_frame(self):
        frame = ctk.CTkFrame(self.main_frame, corner_radius=10)
        self.frames["manage_fields"] = frame
        ctk.CTkLabel(frame, text="Редагування ділянок", font=ctk.CTkFont(size=20,
weight="bold")).pack(pady=20)
        self.combo_edit_field = ctk.CTkOptionMenu(frame, values=[""], width=300,
command=self.load_field_data)
        self.combo_edit_field.pack(pady=10)
        self.ef_name = ctk.CTkEntry(frame, width=300)
        self.ef_name.pack(pady=5)
        self.ef_area = ctk.CTkEntry(frame, width=300)
        self.ef_area.pack(pady=5)
        self.ef_soil = ctk.CTkEntry(frame, width=300)
        self.ef_soil.pack(pady=5)
        self.ef_lat = ctk.CTkEntry(frame, width=300)
        self.ef_lat.pack(pady=5)
        self.ef_lon = ctk.CTkEntry(frame, width=300)
        self.ef_lon.pack(pady=5)
        btn_frame = ctk.CTkFrame(frame, fg_color="transparent")
        btn_frame.pack(pady=20)
        ctk.CTkButton(btn_frame, text="Оновити дані", command=self.update_field).pack(side="left",
padx=10)
        ctk.CTkButton(btn_frame, text="Видалити ділянку", fg_color="#c0392b",
hover_color="#e74c3c", command=self.delete_field).pack(side="left", padx=10)

    def create_add_sample_frame(self):
        frame = ctk.CTkFrame(self.main_frame, corner_radius=10)
        self.frames["add_sample"] = frame
        ctk.CTkLabel(frame, text="Введення лабораторних даних", font=ctk.CTkFont(size=20,
weight="bold")).pack(pady=10)
        self.combo_field = ctk.CTkOptionMenu(frame, values=[""], width=300)
        self.combo_field.pack(pady=5)
        self.s_date = ctk.CTkEntry(frame, width=300)
        self.s_date.insert(0, datetime.now().strftime("%d.%m.%Y"))
        self.s_date.pack(pady=5)
        self.s_ph = ctk.CTkEntry(frame, width=300, placeholder_text="pH (1-14)")
        self.s_ph.pack(pady=5)
        self.s_moist = ctk.CTkEntry(frame, width=300, placeholder_text="Вологість (%)")
        self.s_moist.pack(pady=5)
        self.s_n = ctk.CTkEntry(frame, width=300, placeholder_text="Азот (N)")
        self.s_n.pack(pady=5)
        self.s_p = ctk.CTkEntry(frame, width=300, placeholder_text="Фосфор (P)")
        self.s_p.pack(pady=5)
        self.s_k = ctk.CTkEntry(frame, width=300, placeholder_text="Калій (K)")
        self.s_k.pack(pady=5)
```

```
        btn_f = ctk.CTkFrame(frame, fg_color="transparent")
        btn_f.pack(pady=20)
        ctk.CTkButton(btn_f, text="Зберегти вручну", command=self.save_sample).pack(side="left",
padx=10)
        ctk.CTkButton(btn_f, text="📥 Імпорт бази з Excel", fg_color="#f39c12",
hover_color="#d68910", command=self.import_from_excel).pack(side="left", padx=10)

    def create_analytics_frame(self):
        frame = ctk.CTkFrame(self.main_frame, corner_radius=10)
        self.frames["analytics"] = frame
        top_bar = ctk.CTkFrame(frame, fg_color="transparent")
        top_bar.pack(fill="x", pady=10, padx=20)
        self.combo_analytics = ctk.CTkOptionMenu(top_bar, values=[""], width=200)
        self.combo_analytics.pack(side="left", padx=5)

        ctk.CTkButton(top_bar, text="Побудувати графік та Прогноз",
command=self.plot_analytics).pack(side="left", padx=5)
        ctk.CTkButton(top_bar, text="📄 Звіт у PDF", fg_color="#c0392b", hover_color="#e74c3c",
width=100, command=self.export_pdf).pack(side="right", padx=5)
        ctk.CTkButton(top_bar, text="📊 В Excel", fg_color="#27ae60", hover_color="#2ecc71",
width=100, command=self.export_excel_btn).pack(side="right", padx=5)

        self.lbl_weather = ctk.CTkLabel(frame, text="Погода: очікування...", text_color="#f1c40f",
font=("Arial", 14, "bold"))
        self.lbl_weather.pack()

        self.graph_frame = ctk.CTkFrame(frame, height=350, corner_radius=10)
        self.graph_frame.pack(fill="both", expand=True, padx=20, pady=5)
        self.canvas_widget = None

        self.recs_box = ctk.CTkTextbox(frame, height=90, font=ctk.CTkFont(size=14))
        self.recs_box.pack(fill="x", padx=20, pady=10)
        self.recs_box.insert("0.0", "Експертна система...")
        self.recs_box.configure(state="disabled")

    def create_map_frame(self):
        frame = ctk.CTkFrame(self.main_frame, corner_radius=10)
        self.frames["map"] = frame
        db_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), "map_cache.db")
        self.map_widget = tkintermapview.TkinterMapView(frame, corner_radius=10,
database_path=db_path)
        self.map_widget.set_tile_server("https://mt0.google.com/vt/lyrs=s&hl=en&x={x}&y={y}&z={z}&
s=Ga", max_zoom=19)
        self.map_widget.pack(fill="both", expand=True, padx=20, pady=10)
        self.map_widget.set_position(49.0, 31.0)
        self.map_widget.set_zoom(6)

    def create_manage_users_frame(self):
        frame = ctk.CTkFrame(self.main_frame, corner_radius=10)
        self.frames["manage_users"] = frame
```

```
        add_f = ctk.CTkFrame(frame)
        add_f.pack(pady=20, padx=50, fill="x")
        ctk.CTkLabel(add_f, text="Додати користувача").pack(pady=5)
        self.u_name = ctk.CTkEntry(add_f, placeholder_text="Логін")
        self.u_name.pack(pady=5)
        self.u_pass = ctk.CTkEntry(add_f, placeholder_text="Пароль")
        self.u_pass.pack(pady=5)
        ctk.CTkButton(add_f, text="Створити", command=self.add_user).pack(pady=10)

        del_f = ctk.CTkFrame(frame)
        del_f.pack(pady=20, padx=50, fill="x")
        ctk.CTkLabel(del_f, text="Видалити користувача").pack(pady=5)
        self.combo_users = ctk.CTkOptionMenu(del_f, values=[""])
        self.combo_users.pack(pady=5)
        ctk.CTkButton(del_f, text="Видалити", fg_color="#c0392b", hover_color="#e74c3c",
command=self.delete_user).pack(pady=10)

    def show_frame(self, name):
        for f in self.frames.values(): f.grid_forget()
        self.frames[name].grid(row=0, column=0, sticky="nsew")

    def show_home(self):
        conn = sqlite3.connect('soils_data.db')
        c = conn.cursor()
        c.execute('SELECT COUNT(*) FROM Fields')
        f_cnt = c.fetchone()[0]
        c.execute('SELECT COUNT(*) FROM Soil_Samples')
        s_cnt = c.fetchone()[0]
        conn.close()
        self.lbl_stats.configure(text=f"Ділянок у базі: {f_cnt}  |  Проведено аналізів: {s_cnt}")
        self.show_frame("home")

    def update_combos(self):
        conn = sqlite3.connect('soils_data.db')
        c = conn.cursor()
        c.execute('SELECT id, name FROM Fields')
        fields = c.fetchall()
        c.execute('SELECT id, username FROM Users')
        users = c.fetchall()
        conn.close()
        names = [f"{f[0]} - {f[1]}" for f in fields] if fields else ["Немає"]

        for combo in ['combo_field', 'combo_analytics', 'combo_edit_field']:
            if hasattr(self, combo):
                getattr(self, combo).configure(values=names)
                if fields: getattr(self, combo).set(names[-1])

        if fields and hasattr(self, 'ef_name'):
            self.load_field_data(names[-1])
```

```
        unames = [f"{u[0]} - {u[1]}" for u in users] if users else ["Немає"]
        if hasattr(self, 'combo_users'):
            self.combo_users.configure(values=unames)
            if users: self.combo_users.set(unames[-1])

    def show_add_field(self): self.show_frame("add_field")
    def show_manage_fields(self): self.update_combos(); self.show_frame("manage_fields")
    def show_add_sample(self): self.update_combos(); self.show_frame("add_sample")
    def show_analytics(self): self.update_combos(); self.show_frame("analytics")
    def show_manage_users(self): self.update_combos(); self.show_frame("manage_users")

    def show_map(self):
        self.show_frame("map")
        if hasattr(self, 'map_widget'):
            self.map_widget.delete_all_marker()
            conn = sqlite3.connect('soils_data.db')
            c = conn.cursor()
            c.execute('SELECT name, lat, lon FROM Fields WHERE lat IS NOT NULL AND lon IS NOT
NULL')
            for name, lat, lon in c.fetchall():
                self.map_widget.set_marker(lat, lon, text=name)
            conn.close()

    def save_field(self):
        try:
            conn = sqlite3.connect('soils_data.db')
            c = conn.cursor()
            c.execute('INSERT INTO Fields (name, area, soil_type, lat, lon) VALUES (?,?,?,?,?)',
                      (self.f_name.get(), float(self.f_area.get()), self.f_soil.get(), float(self.f_lat.get() or 0),
float(self.f_lon.get() or 0)))
            conn.commit()
            conn.close()
            messagebox.showinfo("Успіх", "Ділянку збережено!")
        except:
            messagebox.showerror("Помилка", "Перевірте числа.")

    def load_field_data(self, sel):
        if sel == "Немає" or " - " not in sel: return
        conn = sqlite3.connect('soils_data.db')
        c = conn.cursor()
        c.execute('SELECT name, area, soil_type, lat, lon FROM Fields WHERE id=?', (int(sel.split(" -
")[0]),))
        data = c.fetchone()
        conn.close()
        if data:
            for e in [self.ef_name, self.ef_area, self.ef_soil, self.ef_lat, self.ef_lon]: e.delete(0, 'end')
            self.ef_name.insert(0, data[0])
            self.ef_area.insert(0, data[1])
            self.ef_soil.insert(0, data[2])
            self.ef_lat.insert(0, data[3])
```

```
        self.ef_lon.insert(0, data[4])

    def update_field(self):
        try:
            conn = sqlite3.connect('soils_data.db')
            c = conn.cursor()
            c.execute('UPDATE Fields SET name=?, area=?, soil_type=?, lat=?, lon=? WHERE id=?',
                      (self.ef_name.get(), float(self.ef_area.get()), self.ef_soil.get(), float(self.ef_lat.get() or 0),
float(self.ef_lon.get() or 0), int(self.combo_edit_field.get().split(" - ")[0])))
            conn.commit()
            conn.close()
            messagebox.showinfo("Успіх", "Оновлено!")
            self.update_combos()
        except:
            messagebox.showerror("Помилка", "Перевірте числа.")

    def delete_field(self):
        if not hasattr(self, 'combo_edit_field'): return
        sel = self.combo_edit_field.get()
        if sel == "Немає": return
        if messagebox.askyesno("Увага", f"Видалити '{sel}'?"):
            fid = int(sel.split(" - ")[0])
            conn = sqlite3.connect('soils_data.db')
            c = conn.cursor()
            c.execute('DELETE FROM Soil_Samples WHERE field_id=?', (fid,))
            c.execute('DELETE FROM Fields WHERE id=?', (fid,))
            conn.commit()
            conn.close()
            messagebox.showinfo("Видалено", "Видалено.")
            self.update_combos()

    def save_sample(self):
        if not hasattr(self, 'combo_field'): return
        sel = self.combo_field.get()
        if not sel or sel == "Немає" or " - " not in sel:
            messagebox.showwarning("Увага", "Оберіть ділянку!")
            return
        fid = int(sel.split(" - ")[0])
        try:
            conn = sqlite3.connect('soils_data.db')
            c = conn.cursor()
            c.execute('''INSERT INTO Soil_Samples (field_id, sample_date, ph_level, moisture, nitrogen,
phosphorus, potassium)
                         VALUES (?,?,?,?,?,?,?)''',
                      (fid, self.s_date.get(), float(self.s_ph.get()), float(self.s_moist.get()), float(self.s_n.get()),
float(self.s_p.get()), float(self.s_k.get())))
            conn.commit()
            conn.close()
            messagebox.showinfo("Успіх", "Аналіз збережено!")
        except ValueError:
```

```
            messagebox.showerror("Помилка", "Дані (pH, NPK) мають бути числами!")

    def import_from_excel(self):
        file_path = filedialog.askopenfilename(title="Оберіть великий Excel файл", filetypes=[("Excel files", "*.xlsx *.xls")])
        if not file_path: return
        try:
            df = pd.read_excel(file_path)
            conn = sqlite3.connect('soils_data.db')
            c = conn.cursor()
            count_samples = 0
            count_fields = 0

            for index, row in df.iterrows():
                field_name = str(row['Назва_ділянки'])
                c.execute("SELECT id FROM Fields WHERE name=?", (field_name,))
                result = c.fetchone()

                if result:
                    fid = result[0]
                else:
                    c.execute("INSERT INTO Fields (name, area, soil_type, lat, lon) VALUES (?, ?, ?, ?, ?)",
                              (field_name, float(row['Площа_га']), str(row['Тип_ґрунту']), float(row['Широта']), float(row['Довгота'])))
                    fid = c.lastrowid
                    count_fields += 1

                c.execute('''INSERT INTO Soil_Samples (field_id, sample_date, ph_level, moisture, nitrogen, phosphorus, potassium)
                             VALUES (?,?,?,?,?,?,?)''',
                          (fid, str(row['Дата']), float(row['pH']), float(row['Вологість_%']), float(row['Азот_мг/кг']), float(row['Фосфор_мг/кг']), float(row['Калій_мг/кг'])))
                count_samples += 1

            conn.commit()
            conn.close()
            self.update_combos()
            messagebox.showinfo("Імпорт завершено", f"Успішно завантажено!\nСтворено нових полів: {count_fields}\nДодано аналізів: {count_samples}")
        except KeyError as ke:
            messagebox.showerror("Помилка колонок", f"У файлі немає колонки: {ke}")
        except Exception as e:
            messagebox.showerror("Помилка", f"Помилка: {e}")

    def export_excel_btn(self):
        if not hasattr(self, 'combo_analytics'): return
        sel = self.combo_analytics.get()
        if sel != "Немає" and " - " in sel:
            success, msg = export_to_excel(int(sel.split(" - ")[0]), sel.split(" - ")[1])
            if success:
```

```
            messagebox.showinfo("Експорт", msg)
        else:
            messagebox.showerror("Помилка", msg)

    def plot_analytics(self):
        sel = self.combo_analytics.get()
        if not sel or sel == "Немає" or " - " not in sel: return
        fid = int(sel.split(" - ")[0])

        conn = sqlite3.connect('soils_data.db')
        c = conn.cursor()
        c.execute('SELECT lat, lon FROM Fields WHERE id=?', (fid,))
        field_data = c.fetchone()
        if field_data:
            weather = get_current_weather(field_data[0], field_data[1])
            self.lbl_weather.configure(text=weather)

        c.execute('SELECT sample_date, ph_level, nitrogen, phosphorus, potassium FROM
Soil_Samples WHERE field_id=? ORDER BY sample_date', (fid,))
        data = c.fetchall()
        conn.close()

        if not data:
            messagebox.showinfo("Пусто", "Немає даних аналізів.")
            return

        str_dates, ph_vals, n_vals, p_vals, k_vals = zip(*data)

        recs = generate_recommendations(ph_vals[-1], n_vals[-1], p_vals[-1], k_vals[-1])
        self.recs_box.configure(state="normal")
        self.recs_box.delete("0.0", "end")
        self.recs_box.insert("0.0", f"ДАТА ОСТАННЬОГО АНАЛІЗУ: {str_dates[-1]}\n{recs}")
        self.recs_box.configure(state="disabled")

        if self.canvas_widget: self.canvas_widget.destroy()
        plt.style.use('dark_background')
        fig, ax = plt.subplots(figsize=(8, 4), dpi=100)
        fig.patch.set_facecolor('#2b2b2b')
        ax.set_facecolor('#2b2b2b')

        parsed_dates = []
        for d in str_dates:
            try:
                parsed_dates.append(datetime.strptime(str(d).strip(), "%d.%m.%Y"))
            except ValueError:
                try:
                    clean_date = str(d).split()[0]
                    parsed_dates.append(datetime.strptime(clean_date, "%Y-%m-%d"))
                except:
                    parsed_dates.append(datetime.now())
```

```
        x_nums = mdates.date2num(parsed_dates)

        ax.plot(parsed_dates, n_vals, marker='o', label='Азот (N)', color='#3498db', linewidth=2)
        ax.plot(parsed_dates, p_vals, marker='s', label='Фосфор (P)', color='#e74c3c', linewidth=2)
        ax.plot(parsed_dates, k_vals, marker='^', label='Калій (K)', color='#2ecc71', linewidth=2)

        if len(x_nums) > 1:
            try:
                future_x = np.append(x_nums, x_nums[-1] + 365)
                future_dates = mdates.num2date(future_x)

                z_n = np.polyfit(x_nums, n_vals, 1)
                p_n = np.poly1d(z_n)
                ax.plot(future_dates, p_n(future_x), linestyle="--", color='#3498db', alpha=0.5,
label='Прогноз N')

                z_p = np.polyfit(x_nums, p_vals, 1)
                p_p = np.poly1d(z_p)
                ax.plot(future_dates, p_p(future_x), linestyle="--", color='#e74c3c', alpha=0.5,
label='Прогноз P')

                z_k = np.polyfit(x_nums, k_vals, 1)
                p_k = np.poly1d(z_k)
                ax.plot(future_dates, p_k(future_x), linestyle="--", color='#2ecc71', alpha=0.5,
label='Прогноз K')
            except: pass

        ax.xaxis.set_major_formatter(mdates.DateFormatter('%d.%m.%y'))
        fig.autofmt_xdate()
        ax.set_title(f"Динаміка та Прогноз: {sel.split(' - ')[1]}")
        ax.legend()
        ax.grid(True, linestyle='--', alpha=0.3)

        self.current_fig = fig
        self.current_field_name = sel.split(' - ')[1]

        canvas = FigureCanvasTkAgg(fig, master=self.graph_frame)
        canvas.draw()
        self.canvas_widget = canvas.get_tk_widget()
        self.canvas_widget.pack(fill="both", expand=True)

    def export_pdf(self):
        if not hasattr(self, 'current_fig') or self.combo_analytics.get() == "Немає":
            messagebox.showwarning("Увага", "Спочатку побудуйте графік!")
            return

        try:
            plt.style.use('default')
            pdf_fig, pdf_ax = plt.subplots(figsize=(8.27, 11.69), dpi=150)
```

```
            pdf_fig.patch.set_facecolor('white')
            pdf_ax.axis('off')

            pdf_fig.text(0.5, 0.92, f"ОФІЦІЙНИЙ ЗВІТ АНАЛІЗУ ҐРУНТІВ", fontsize=18, fontweight='bold',
ha='center')
            pdf_fig.text(0.5, 0.88, f"Об'єкт: {self.current_field_name}", fontsize=14, ha='center')
            pdf_fig.text(0.5, 0.85, f"Дата генерації: {datetime.now().strftime('%d.%m.%Y %H:%M')}",
fontsize=10, ha='center', color='gray')

            pdf_fig.text(0.1, 0.78, "1. ВИСНОВОК ЕКСПЕРТНОЇ СИСТЕМИ ТА РЕКОМЕНДАЦІЇ:",
fontsize=12, fontweight='bold')
            recs = self.recs_box.get("0.0", "end")
            pdf_fig.text(0.1, 0.68, recs, fontsize=11, verticalalignment='top', wrap=True)

            pdf_fig.text(0.1, 0.58, "2. ГРАФІК ДИНАМІКИ ТА ПРОГНОЗУ НА 30 ДНІВ:", fontsize=12,
fontweight='bold')

            buf = io.BytesIO()
            self.current_fig.savefig(buf, format='png', bbox_inches='tight')
            buf.seek(0)
            img = imread(buf)

            new_ax = pdf_fig.add_axes([0.1, 0.15, 0.8, 0.4])
            new_ax.imshow(img)
            new_ax.axis('off')

            pdf_filename = f"PDF_Звіт_{self.current_field_name.replace(' ', '_')}.pdf"
            pdf_fig.savefig(pdf_filename, format='pdf')
            plt.close(pdf_fig)

            plt.style.use('dark_background')
            messagebox.showinfo("Успіх", f"Офіційний PDF-звіт
створено:\n{os.path.abspath(pdf_filename)}")
        except Exception as e:
            messagebox.showerror("Помилка", f"Не вдалося згенерувати PDF:\n{e}")

    def add_user(self):
        username = self.u_name.get()
        pwd = self.u_pass.get()

        if len(username) < 3 or len(pwd) < 3:
            messagebox.showwarning("Увага", "Логін та пароль мають містити більше 3 символів!")
            return

        try:
            conn = sqlite3.connect('soils_data.db')
            c = conn.cursor()
            hashed_pwd = hashlib.sha256(pwd.encode()).hexdigest()

            c.execute("INSERT INTO Users (username, password) VALUES (?, ?)",
```

```
                (username, hashed_pwd))
            conn.commit()
            conn.close()

            messagebox.showinfo("Успіх", f"Користувача {username} створено!")
            self.update_combos()
        except sqlite3.IntegrityError:
            messagebox.showerror("Помилка", "Користувач з таким логіном вже існує!")

    def delete_user(self):
        sel = self.combo_users.get()
        if sel == "Немає" or not sel:
            return

        uid = int(sel.split(" - ")[0])
        uname = sel.split(" - ")[1]

        if uname == "admin":
            messagebox.showerror("Помилка", "Критична дія: не можна видалити адміністратора!")
            return

        if messagebox.askyesno("Підтвердження", f"Ви впевнені, що хочете видалити '{uname}'?"):
            conn = sqlite3.connect('soils_data.db')
            c = conn.cursor()
            c.execute('DELETE FROM Users WHERE id=?', (uid,))
            conn.commit()
            conn.close()

            messagebox.showinfo("Видалено", "Користувача успішно видалено.")
            self.update_combos()

if __name__ == "__main__":
    init_db()
    app = MainApp()
    app.mainloop()
```