НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК 004.4:631.115

«ПОГОДЖЕНО»	**«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»**
Декан факультету інформаційних технологій	Завідувач кафедри комп'ютерних наук
Болбот І.М., д.т.н., професор	Голуб Б.Л., к.т.н., доцент
_____	_____
____ _____ **2024 р.**	____ _____ **2024 р.**

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Програмне забезпечення системи підтримки прийняття рішень в управлінні фермерським господарством

Спеціальність 121 «Інженерія програмного забезпечення»
(код і назва)

Освітня програма Програмне забезпечення інформаційних систем
(назва)

Орієнтація освітньої програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

Доцент, к.т.н.	_____	Семко В. В.
(науковий ступінь та вчене звання)	(підпис)	(ПІБ)

Керівник магістерської кваліфікаційної роботи

професор, д.т.н.	_____	Бородкіна І. Л.
(науковий ступінь та вчене звання)	(підпис)	(ПІБ)

Виконав _____ Салюк Н. Ю.
(підпис) (ПІБ студента)

КИЇВ-2024

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет (ННІ) _____

ЗАТВЕРДЖУЮ
Завідувач кафедри_____
_____
_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (ПІБ)
"____" _____20_____року

З А В Д А Н Н Я
ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ СТУДЕНТУ

_____Салюк Назар Юрійович _____
(прізвище, ім'я, по батькові)

Спеціальність _____121 «Інженерія програмного забезпечення»_____
(код і назва)

Освітня програма _____Програмне забезпечення інформаційних систем_____
(назва)

Орієнтація освітньої програми_____освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи Програмне забезпечення системи підтримки прийняття рішень в управлінні фермерським господарством
затверджена наказом ректора НУБіП України від "____"_____20___р. №_____

Термін подання завершеної роботи на кафедру _____
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи: наукові публікації та звіти у сфері аграрного господарства_____
_____

Перелік питань, що підлягають дослідженню:

1. Дослідження наукових джерел про сільське господарство, аграрне виробництво.
2. Аналіз існуючих програмних рішень у сфері агробізнесу.
3. Вибір технологій та особливості архітектури програмного забезпечення управління фермерським господарством.
4. Розробка програмних модулів системи підтримки прийняття рішень.
5. Функціональне та модульне тестування.

Перелік графічного матеріалу (за потреби) _____
_____

Дата видачі завдання "_____"_____20___ р.

Керівник магістерської кваліфікаційної роботи _____ __Бородкіна І. Л._
(підпис) (прізвище та ініціали)

Завдання прийняв до виконання _____ _____Салюк Н. Ю._____
(підпис) (прізвище та ініціали студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Ферма – окреме господарство, яке займається виробництвом сільськогосподарської продукції і є основною одиницею обліку в системі.

Власник – фізична або юридична особа, яка володіє фермерським господарством і відповідає за управління його ресурсами.

Поле – земельна ділянка, що належить фермі, і використовується для вирощування певних культур.

Культура – вид сільськогосподарської продукції (наприклад, картопля, кукурудза), що вирощується на полі.

Витрати – фінансові ресурси, що витрачаються на підтримку діяльності ферми, включаючи матеріальні витрати, обслуговування обладнання тощо.

Обладнання – сільськогосподарська техніка та інші інструменти, що використовуються для обробки полів і підтримки фермерських процесів.

Погодні умови – кліматичні показники, такі як температура, вологість, кількість опадів, що впливають на продуктивність ферми.

Трирівнева архітектура – модель архітектури програмного забезпечення, що включає рівень представлення, бізнес-логіки та рівень доступу до даних.

MS SQL Server – система управління реляційними базами даних, яка використовується для зберігання та обробки даних у системі управління фермерським господарством.

C# – мова програмування, на якій розроблено програмне забезпечення для управління фермерським господарством.

Visual Studio 2022 – інтегроване середовище розробки (IDE), що використовується для написання, компіляції та тестування коду додатку.

ВСТУП

Сучасне сільське господарство перебуває в умовах швидкого розвитку технологій, які дозволяють значно підвищити ефективність управління ресурсами та оптимізувати процеси виробництва. У зв'язку з глобальними кліматичними змінами, зростаючими потребами в продовольстві та необхідністю сталого використання природних ресурсів, виникає необхідність в нових підходах до управління фермерськими господарствами. Одним із ключових напрямків у цьому контексті є розробка програмного забезпечення, яке здатне допомогти фермерам приймати оптимальні рішення на основі аналізу великих обсягів даних. Це включає погодні умови, стан ґрунтів, розподіл ресурсів, витрати на вирощування культур та прогнозування врожайності.

Застосування інформаційних технологій у сільському господарстві дає змогу вирішувати низку актуальних проблем, зокрема ефективне управління землекористуванням, зменшення втрат від непередбачених погодних умов, раціоналізацію використання добрив та засобів захисту рослин. Однак, попри досягнення в цій галузі, все ще залишається багато викликів, пов'язаних із недостатньою автоматизацією процесів та використанням інноваційних рішень у малих і середніх фермерських господарствах [1].

Необхідність розробки інноваційного програмного забезпечення для підтримки прийняття рішень у сільському господарстві обумовлена кількома факторами. По-перше, традиційні методи управління, що базуються на ручних розрахунках і досвіді, не завжди можуть ефективно відповідати сучасним вимогам. По-друге, зростає потреба в гнучких системах, які можуть автоматично аналізувати дані в реальному часі та надавати точні прогнози. Існуючі рішення часто є складними у впровадженні або не враховують специфіку локальних умов, що обмежує їх використання, особливо в контексті фермерських господарств України.

Українські фермери стикаються з численними викликами, зокрема економічними труднощами, нестабільністю ринку, змінами клімату та потребою в модернізації інфраструктури. Впровадження новітніх технологій може суттєво підвищити їх конкурентоспроможність, зменшити ризики та оптимізувати виробничі процеси. Програмне забезпечення, яке здатне інтегрувати дані з різних джерел, аналізувати їх за допомогою штучного інтелекту та надавати рекомендації, може стати незамінним інструментом для підтримки прийняття рішень в управлінні фермерськими господарствами [2].

Актуальність цієї роботи полягає у розробці та впровадженні системи, яка зможе автоматизувати процеси управління на основі великих обсягів даних, що надходять від різних джерел, включаючи IoT-сенсори, метеорологічні дані та інформацію про стан ґрунтів. Використання таких систем дозволяє досягати високої точності прогнозування врожайності, оптимізації використання ресурсів та зниження витрат, що є особливо актуальним у контексті сталого розвитку сільського господарства.

Вагомість цієї роботи для України також полягає в тому, що розробка подібного програмного забезпечення дозволить підвищити рівень технологічної підготовки українських фермерів до сучасних умов. Впровадження таких систем дозволить зменшити залежність від погодних умов, знизити ризики втрат і сприяти збереженню природних ресурсів. Крім того, програма може допомогти зменшити витрати на добрива, воду та засоби захисту рослин, що має велике значення для збереження екологічної рівноваги.

Об’єкт дослідження – це процес управління фермерським господарством, який включає в себе комплексне прийняття рішень стосовно планування, вирощування культур, моніторингу ресурсів та аналізу впливу погодних умов і економічних факторів на ефективність господарювання.

Предмет дослідження – система підтримки прийняття рішень в управлінні фермерським господарством, що забезпечує підвищення ефективності господарської діяльності.

Мета дослідження полягає у розробці програмного забезпечення для автоматизації управлінських процесів у фермерському господарстві, що забезпечить полегшення обліку ресурсів, аналіз витрат та управління виробничими показниками.

Для досягнення цієї мети необхідно вирішити такі основні задачі:

– провести аналіз поточного стану інформаційних систем управління фермерськими господарствами, вивчивши особливості діяльності фермерських господарств та наявні потреби в автоматизації процесів;

– оцінити існуючі програмні рішення у сфері агробізнесу з метою визначення їхніх сильних і слабких сторін, а також визначити вимоги до розробки нового програмного забезпечення для фермерських господарств;

– вибрати відповідні технології розробки програмного забезпечення, визначити архітектуру системи, враховуючи особливості проєктування бази даних, бізнес-логіки, інтерфейсу користувача та управління доступом до даних;

– розробити та протестувати програмне забезпечення, яке відповідатиме вимогам фермерських господарств, забезпечуючи автоматизацію процесів, точний аналіз даних та ефективне управління ресурсами.

Для дослідження системи підтримки прийняття рішень в управлінні фермерським господарством були використані такі методи: аналіз літературних джерел, моделювання та порівняння.

Наукова новизна одержаних результатів полягає в удосконаленні методів управління фермерськими господарствами шляхом впровадження системи підтримки прийняття рішень, заснованої на інтеграції бази даних з аналітичними модулями. Вперше реалізовано підхід до автоматизованого формування звітів з використанням бази даних, яка забезпечує зберігання та управління інформацією про поля, культури, погодні умови та витрати в фермерських господарствах.

1 АНАЛІЗ ПОТОЧНОГО СТАНУ ІНФОРМАЦІЙНИХ СИСТЕМ УПРАВЛІННЯ ФЕРМЕРСЬКИМ ГОСПОДАРСТВОМ

1.1 Особливості діяльності фермерських господарств

Сьогодні аграрний бізнес не може досягти успіху лише шляхом збільшення виробництва традиційної сільськогосподарської продукції. Конкурентні переваги сучасних фермерських господарств створюються шляхом ефективного використання наявних ресурсів для виробництва унікальних продуктів або послуг. Важливою умовою є здатність фермерів впроваджувати інновації та швидко адаптувати їх до умов ринку.

Кожне фермерське господарство самостійно обирає напрямок своєї діяльності. Спеціалізація фермерства напряму залежить від якості та цільового призначення землі, якою фермер користується, будь то власність або оренда. Оформлення юридичних аспектів землекористування надає доступ до державних та регіональних програм підтримки агробізнесу. Водночас розмір господарства пов'язаний з обраним напрямом діяльності. Фермер прагне обрати той вид виробництва, який дозволить максимально ефективно використовувати наявні земельні ресурси та капіталовкладення [3].

Багатопрофільний підхід у фермерстві виправданий тим, що дозволяє зменшити ризики фінансових втрат. Вирощування кількох видів продукції дає можливість компенсувати збитки від однієї культури прибутком від інших. Такий підхід є типовим для початкових етапів розвитку фермерського господарства, коли ще не визначено основну спеціалізацію.

Основним видом діяльності фермерських господарств залишається виробництво сільськогосподарської продукції, а також її переробка. Одним зі шляхів підвищення ефективності є інтеграція з переробними підприємствами, якщо організувати переробку всередині господарства неможливо. Однак слід

зазначити, що переробні підприємства часто диктують умови щодо цін, обсягів і строків постачання продукції, а також стандарти якості, що може призвести до нерівномірного розподілу прибутку на користь переробних підприємств [4].

Фермерські господарства можуть існувати у різних формах організації, таких як приватні підприємства, кооперативи, сімейні ферми та корпоративні агрохолдинги (табл. 1.1). Кожна форма організації має свої специфічні структури управління, які впливають на підхід до розподілу ресурсів, планування діяльності та прийняття рішень. Вибір форми організації фермерського господарства значною мірою залежить від розміру господарства, його спеціалізації, а також доступу до фінансових ресурсів і ринкових можливостей.

Таблиця 1.1

Організаційні особливості різних типів фермерських господарств

Тип господарства	Структура управління	Масштаб діяльності	Основні переваги
Приватне підприємство	Індивідуальне управління фермером або власником	Малий або середній	Гнучкість у прийнятті рішень, швидка адаптація до змін
Кооператив	Колективне управління членами кооперативу	Середній або великий	Розподіл ризиків, спільне використання ресурсів
Сімейна ферма	Традиційне сімейне управління, передача з покоління в покоління	Малий або середній	Сталість, спадковість, локалізованість діяльності
Корпоративний агрохолдинг	Центральне управління, багаторівнева ієрархія	Великий	Значні фінансові та технічні ресурси, масштабність виробництва

Приватні підприємства зазвичай характеризуються більш гнучкою структурою, де фермер особисто приймає рішення щодо управління виробництвом і використання ресурсів. Кооперативи базуються на

колективному управлінні, де фермери спільно вирішують питання стратегії та розподілу прибутків. Сімейні ферми мають переважно малий або середній масштаб і часто передаються з покоління в покоління, що впливає на сталість і спадкові традиції управління. Корпоративні агрохолдинги, на відміну від інших, мають складнішу структуру з багаторівневим управлінням, де більшість рішень приймається на рівні центрального офісу [5].

Виробничі особливості фермерських господарств визначаються основними видами діяльності та структурою виробництва, що формується на основі доступних ресурсів, кліматичних умов і ринкових можливостей. Організація виробничих процесів у фермерських господарствах залежить від напрямку діяльності, зокрема рослинництва, тваринництва або змішаного господарювання, яке поєднує обидва види виробництва. Кожен із цих напрямків має свої специфічні вимоги щодо планування, використання ресурсів, управління процесами та технологічних рішень. Табл. 1.2 структуровано демонструє основні виробничі особливості кожного з цих напрямків.

Таблиця 1.2

Виробничі особливості різних типів фермерських господарств

Напрям виробництва	Основна діяльність	Ключові ресурси	Технологічні особливості
Рослинництво	Вирощування зернових, технічних, овочевих культур	Земля, насіння, добрива, вода	Системи зрошення, контроль за якістю ґрунту, збирання врожаю
Тваринництво	Вирощування худоби для м’яса, молока, яєць	Корм, тварини, приміщення, ветеринарні послуги	Системи вентиляції, автоматизоване годування
Змішані господарства	Поєднання рослинництва та тваринництва	Земля, корм, тварини, насіння	Взаємне використання ресурсів, автоматизація процесів
Спеціалізовані господарства	Виробництво окремих видів продукції	Високоспеціалізовані ресурси для одного типу виробництва	Технології для підвищення продуктивності

Фермерські господарства, що займаються рослинництвом, орієнтовані на вирощування культур, таких як зернові, технічні, овочеві чи фруктові. Організація виробничого процесу включає підготовку ґрунту, посів, догляд за культурами та збирання врожаю. Важливу роль у цьому відіграє погодний фактор і використання агротехнологій, таких як системи зрошення та контролю за врожайністю.

Тваринництво, як окремий напрям, включає вирощування худоби та птиці для отримання м’яса, молока та інших продуктів тваринництва. Організація такого типу виробництва вимагає забезпечення оптимальних умов для тварин, включаючи харчування, ветеринарний нагляд і комфортне середовище.

Змішані господарства поєднують рослинництво та тваринництво, забезпечуючи можливість взаємної підтримки між цими напрямками. Такі господарства можуть ефективно використовувати відходи рослинництва для тваринного корму, а гній – як добриво для полів [6].

Сучасні фермерські господарства активно впроваджують технологічні рішення для підвищення ефективності виробничих процесів, оптимізації використання ресурсів та підвищення врожайності. Технології, які використовуються у фермерському господарстві, охоплюють механізацію, автоматизацію та інформаційні системи (табл. 1.3). Ці технології дозволяють не тільки підвищити продуктивність, але й забезпечити більшу точність у виконанні операцій та знизити витрати.

Таблиця 1.3

Технологічні особливості фермерських господарств

Технологія	Опис	Основні переваги	Приклад застосування
1	2	3	4
Механізація	Використання сучасної сільськогосподарської техніки	Підвищення продуктивності, зменшення витрат на ручну працю	Трактори, комбайни, сіялки
Автоматизація	Автоматизовані системи для управління процесами	Зменшення людського втручання, підвищення якості управління	Автоматичні доїльні установки, клімат-контроль у приміщеннях
Інформаційні технології	Використання аналітичного ПО, дронів, GPS-навігації	Оптимізація ресурсів, точне землеробство	Дрони для моніторингу полів, GPS-системи для контролю за полями
Системи моніторингу	Контроль за станом ґрунту, погодними умовами, параметрами середовища	Підвищення точності прогнозування, своєчасна реакція на зміни	Сенсори вологості, температури, показників ґрунту

Механізація охоплює використання сучасної сільськогосподарської техніки для підготовки ґрунту, сівби, догляду за рослинами і збирання врожаю. Високотехнологічні трактори, сіялки, комбайни та інші машини дозволяють виконувати ці операції значно швидше і з меншими витратами праці [7].

Автоматизація полягає у впровадженні систем, що дозволяють контролювати і керувати різними аспектами виробництва без постійного втручання людини. Наприклад, автоматизовані доїльні установки, клімат-контроль у тваринницьких приміщеннях або системи моніторингу стану ґрунту дають змогу підвищити якість управління процесами і зменшити залежність від людського фактора.

Інформаційні технології сприяють впровадженню систем точного землеробства, що базуються на GPS-навігації, дронах для моніторингу полів та аналітичному програмному забезпеченні, яке допомагає фермерам ухвалювати рішення на основі аналізу даних. Ці технології дозволяють не тільки оптимізувати використання ресурсів, але й підвищувати точність операцій, що в кінцевому результаті знижує витрати.

Завдяки використанню цих рішень, фермери отримують можливість краще управляти ресурсами, підвищувати врожайність та знижувати витрати. Технології сприяють впровадженню інноваційних підходів, які дають змогу фермерам бути конкурентоспроможними в умовах сучасного ринку [8].

Фермерські господарства відіграють ключову роль у збереженні природних ресурсів та підтримці екосистем, адже їхня діяльність безпосередньо пов'язана з використанням землі, води та інших природних ресурсів. Організація фермерських процесів з урахуванням екологічних особливостей сприяє забезпеченню сталого розвитку, збереженню родючості ґрунтів та запобіганню виснаженню природних ресурсів.

Таблиця 1.4

Екологічні особливості фермерських господарств

Аспект	Опис	Вплив на навколишнє середовище	Приклади реалізації
Сталий розвиток	Впровадження екологічно чистих методів виробництва	Зниження забруднення, збереження біорізноманіття	Органічне землеробство, агролісомеліорація
Управління водними ресурсами	Раціональне використання води для зрошення	Зменшення виснаження водних ресурсів	Системи крапельного зрошення
Збереження родючості ґрунтів	Підтримка рівня родючості через органічні методи	Підвищення якості ґрунтів, зменшення ерозії	Сівозміна, органічні добрива
Мінімізація хімічних речовин	Використання природних засобів захисту рослин і добрив	Зменшення хімічного забруднення ґрунтів і водних ресурсів	Біологічні засоби захисту рослин, компостування

Екологічні особливості фермерських господарств зосереджені на принципах сталого розвитку, що передбачають довгострокову екологічну стабільність та збереження природних ресурсів для майбутніх поколінь. Основою цього підходу є впровадження методів, що мінімізують негативний вплив на навколишнє середовище і водночас забезпечують ефективне використання земельних і водних ресурсів. Сільське господарство, яке орієнтоване на сталий розвиток, прагне досягти балансу між економічною вигодою, екологічною відповідальністю та соціальною користю.

Одним із ключових аспектів сталого розвитку є використання екологічно безпечних методів виробництва, таких як органічне землеробство. Це підхід, що передбачає зведення до мінімуму використання хімічних пестицидів і синтетичних добрив, замінюючи їх на природні аналоги, такі як компост або біологічні засоби захисту рослин. Завдяки цьому фермери не лише знижують забруднення ґрунтів і водних ресурсів, але й підвищують якість продукції, яка стає екологічно чистішою та безпечнішою для споживачів. Такий підхід сприяє розвитку ринків органічної продукції та підвищенню конкурентоспроможності фермерів.

Зменшення залежності від хімічних засобів у сільському господарстві є важливим напрямом екологічної практики. Замість інтенсивного використання агрохімікатів фермери переходять до застосування біологічних методів боротьби з шкідниками та хворобами, а також до органічних добрив. Це знижує ризики негативного впливу на екосистеми, зменшує забруднення ґрунтів і вод, а також сприяє відновленню природного балансу в агроландшафтах [9].

В умовах змін клімату та збільшення посухових періодів фермерські господарства все більше орієнтуються на ефективні технології використання води, такі як крапельне зрошення або системи збору та збереження дощової води. Це дозволяє зменшити втрати води та забезпечити рослини необхідною кількістю вологи без надмірного виснаження водних ресурсів. Окрім того, збереження родючості ґрунтів є критично важливим завданням. Тривала експлуатація ґрунтів без належного догляду може призвести до їх виснаження

та деградації. Сталий підхід включає методи сівозміни, застосування органічних добрив, а також агролісомеліорацію – створення захисних лісосмуг і зелених насаджень, які допомагають зберегти структуру ґрунтів, запобігають ерозії та підвищують їх родючість.

Особливості діяльності фермерських господарств формують їх ефективність, стійкість та здатність адаптуватися до змінних умов ринку і навколишнього середовища. Вони є ключовими елементами для забезпечення продовольчої безпеки та сталого розвитку аграрного сектору [10].

1.2 Потреба в автоматизації процесів управління фермерським господарством

Автоматизація процесів управління фермерським господарством стає все більш актуальною через зростаючу складність сучасного сільськогосподарського виробництва та потребу в оптимізації використання ресурсів. Сучасні фермери стикаються з низкою викликів, таких як зміни клімату, нестабільність ринку, зростання витрат на виробництво та дефіцит трудових ресурсів. У таких умовах традиційні методи управління втрачають ефективність, що створює необхідність впровадження автоматизованих систем, здатних підтримувати оперативне управління та прийняття рішень.

З огляду на зростаючі виклики в аграрній сфері, виникає нагальна потреба в інтеграції сучасних технологій для забезпечення стабільного розвитку фермерських господарств [11]. Автоматизовані системи управління дозволяють не тільки знизити навантаження на фермерів, але й оптимізувати використання ресурсів, оперативно реагувати на зміни зовнішнього середовища та ефективно управляти процесами виробництва. Рис. 1.1 наочно демонструє ключові потреби в управлінні фермерським господарством, які можна вирішити за допомогою автоматизації та впровадження сучасних інформаційних систем.

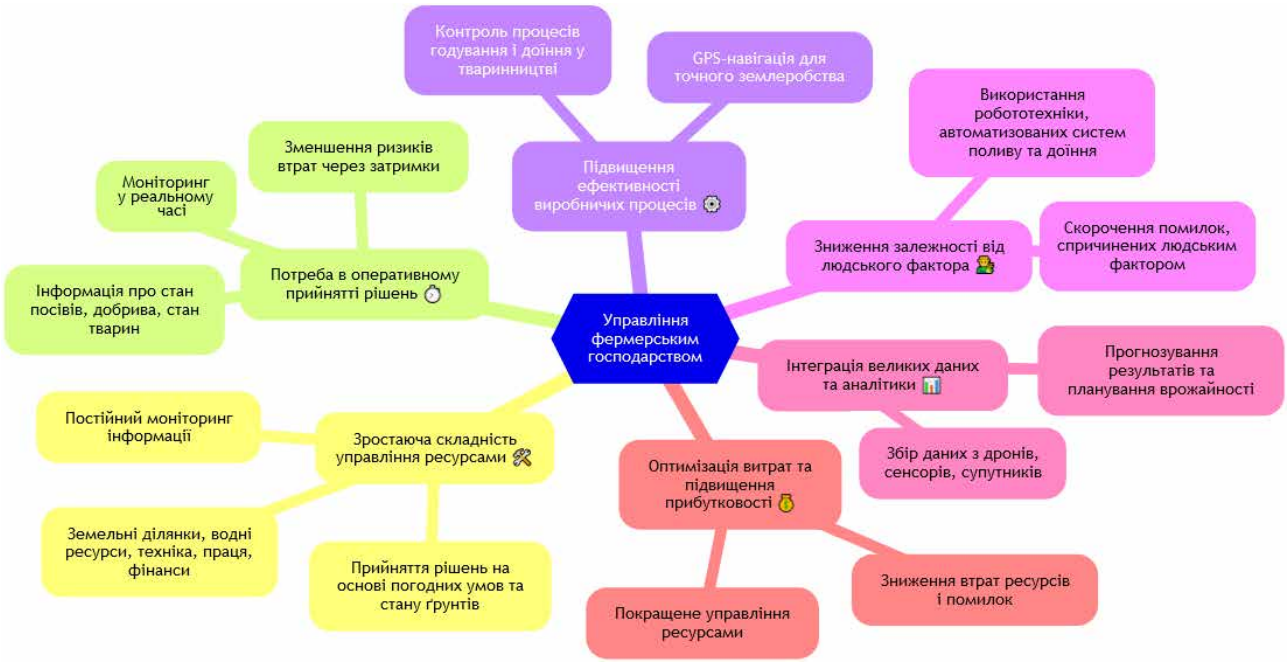

Рис. 1.1 – Потреби в управлінні фермерським господарством

До ключових потреб в управлінні фермерським господарством належать:

– зростаюча складність управління ресурсами. Фермерські господарства мають справу з різноманітними ресурсами: земельними ділянками, водними ресурсами, технікою, працею та фінансами. Ефективне управління цими ресурсами є ключовим фактором для зниження витрат і підвищення продуктивності. Без автоматизації цей процес стає громіздким, оскільки фермери повинні постійно відстежувати інформацію, приймати рішення на основі різних факторів, таких як погодні умови, стан ґрунтів та потреби рослин або тварин;

– потреба в оперативному прийнятті рішень. У сучасному сільському господарстві, де результативність залежить від точного та своєчасного прийняття рішень, автоматизовані системи можуть значно полегшити цей процес. Такі системи дозволяють фермеру оперативно отримувати інформацію про стан посівів, погодні умови, використання добрив або стан тваринництва. Наприклад, системи моніторингу ґрунтів або кліматичних умов у реальному часі можуть повідомляти про необхідність поливу, внесення добрив або інших заходів, що підвищують продуктивність і знижують ризики втрат; [12]

– підвищення ефективності виробничих процесів. Автоматизація дозволяє інтегрувати сучасні технології у всі етапи сільськогосподарського виробництва – від підготовки ґрунту та посіву до збору врожаю та його зберігання. Наприклад, використання GPS-навігації для точного землеробства дозволяє зменшити втрати через недоцільне використання добрив або води, підвищуючи ефективність використання ресурсів. Автоматизовані системи управління тваринництвом можуть контролювати процеси годування, доїння та моніторинг стану здоров'я тварин, що зменшує людський фактор і підвищує точність виконання операцій;

– зниження залежності від людського фактору та трудових ресурсів. Сільськогосподарське виробництво зазвичай є трудомістким процесом, а дефіцит робочої сили в сільській місцевості ставить перед фермерами додаткові виклики. Автоматизація дає змогу знизити залежність від людського фактора, підвищуючи продуктивність навіть за умов обмежених трудових ресурсів. Використання робототехніки, автоматизованих доїльних установок, систем поливу та інших технологічних рішень допомагає скоротити витрати на наймання персоналу та зменшити помилки, спричинені людським фактором.

– інтеграція великих даних та аналітики. Автоматизовані системи управління фермерським господарством дозволяють збирати великі обсяги даних з різних джерел: сенсори, дрони, супутникові дані тощо. На основі цих даних можна здійснювати точний аналіз стану господарства, виявляти тренди та прогнозувати результати. Аналітичні інструменти допомагають фермерам оцінювати ефективність використання ресурсів, планувати врожайність і оптимізувати процеси виробництва. Інтеграція таких рішень в управління дає можливість оперативно реагувати на зміни та ухвалювати стратегічні рішення на основі надійної інформації .

– оптимізація витрат та підвищення прибутковості. Один із ключових факторів, що стимулюють автоматизацію, – це економічна ефективність. Автоматизація допомагає зменшити втрати ресурсів, мінімізувати вплив людських помилок і підвищити точність операцій, що в свою чергу сприяє

зниженню витрат. За рахунок кращого управління ресурсами та більш точного прогнозування, фермери можуть оптимізувати свої витрати та підвищити прибутковість виробництва [13].

Автоматизація процесів управління фермерським господарством є важливим інструментом для підвищення продуктивності, зниження витрат та забезпечення довгострокової стійкості виробництва. Впровадження сучасних технологій дозволяє фермерам ефективніше використовувати ресурси, оперативно ухвалювати рішення на основі даних та мінімізувати вплив людських помилок. Завдяки автоматизації, фермерські господарства стають більш конкурентоспроможними та готовими до викликів сучасного ринку.

1.3 Аналіз існуючих програмних рішень у сфері агробізнесу

У сучасному агробізнесі програмні рішення відіграють ключову роль у підвищенні ефективності управління фермерськими господарствами. Впровадження таких рішень дозволяє автоматизувати рутинні операції, покращити процеси прийняття рішень, зменшити втрати ресурсів та підвищити продуктивність. Програмні продукти, які використовуються в аграрному секторі, охоплюють широкий спектр завдань – від управління земельними ресурсами до контролю за станом посівів і тварин. Існує велика кількість систем, які дозволяють фермерам керувати своїм господарством більш ефективно, використовуючи аналітичні інструменти, прогнози врожайності, дані про погодні умови, стан ґрунтів, використання добрив та багато іншого.

Trimble Ag Software призначений для управління сільськогосподарськими процесами, зокрема планування польових робіт, моніторингу врожайності та аналізу використання ресурсів (рис. 1.2). Його мета – допомогти фермерам оптимізувати свою діяльність за рахунок точного аналізу даних, що надходять від різних джерел, таких як сенсори, супутники та техніка, оснащена GPS-навігацією.

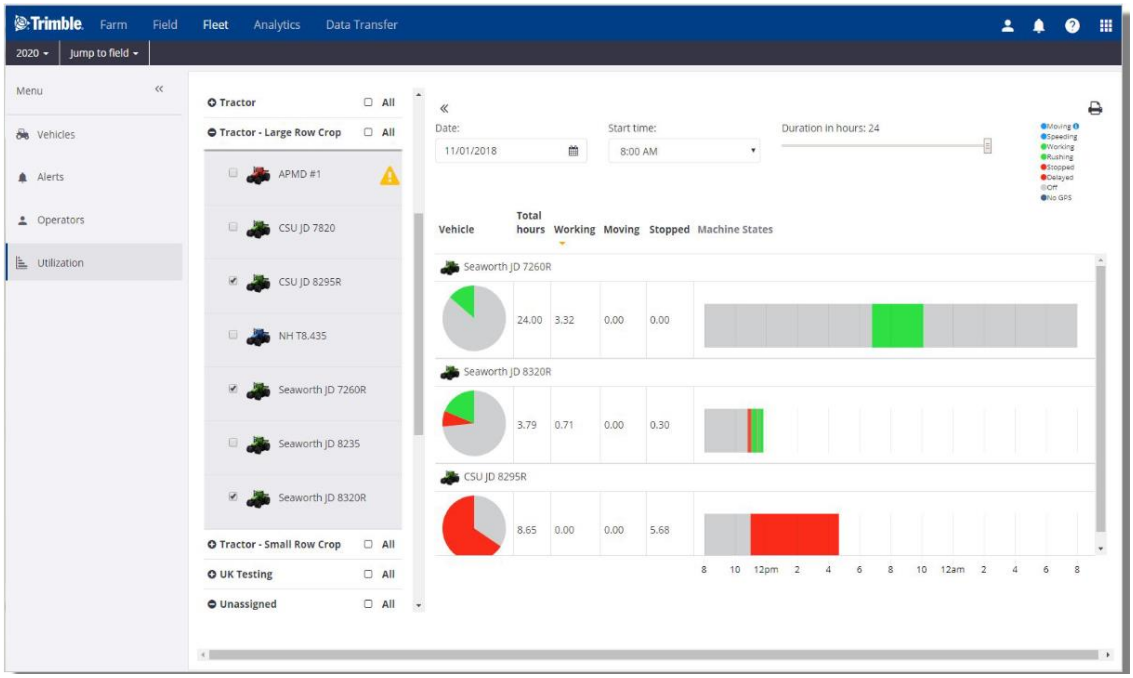

Рис. 1.2 – Приклад інтерфейсу системи «Trimble Ag Software» [14]

Архітектура системи побудована на основі хмарних технологій, що забезпечує доступ до даних у будь-який момент та з будь-якого пристрою. Це дозволяє фермерам відстежувати стан полів і приймати рішення в режимі реального часу. Система інтегрує аналітичні інструменти для прогнозування врожайності та оптимізації використання добрив і води, а також підтримує зв'язок із польовою технікою, забезпечуючи автоматизоване управління її роботою.

Переваги Trimble Ag Software:

– хмарна архітектура – забезпечує доступ до даних у будь-який час і з будь-якого пристрою;

– інтеграція з польовою технікою – дозволяє автоматизувати процеси, пов'язані з обробкою полів;

– аналітичні інструменти – забезпечують точне прогнозування врожайності та оптимізацію використання ресурсів;

– GPS-навігація – покращує точність польових операцій, зменшуючи втрати та підвищуючи продуктивність.

Недоліки Trimble Ag Software:

– висока вартість – для деяких фермерів рішення може бути занадто дорогим через підписки та обслуговування;

– складність інтеграції – вимагає наявності техніки, що підтримує GPS та інші сумісні пристрої;

– потреба в постійному доступі до інтернету – у віддалених регіонах це може бути проблемою;

– залежність від технічної підтримки – складні налаштування та робота можуть вимагати постійної підтримки від виробника [15].

Отже, Trimble Ag Software є потужним інструментом для управління сільськогосподарськими процесами, що пропонує широкий спектр функцій для оптимізації виробництва, зокрема завдяки хмарній архітектурі та інтеграції з технікою. Проте його висока вартість та складність інтеграції можуть обмежити використання в менших фермерських господарствах, особливо в регіонах із поганою інтернет-інфраструктурою.

FarmLogs призначений для допомоги фермерам в управлінні аграрними процесами, включаючи моніторинг стану полів, планування робіт, контроль витрат і аналіз врожайності (рис. 1.3). Основна мета програми – надати фермерам інструменти для прийняття обґрунтованих рішень на основі даних про погоду, стан ґрунтів, використання добрив та обробку полів.

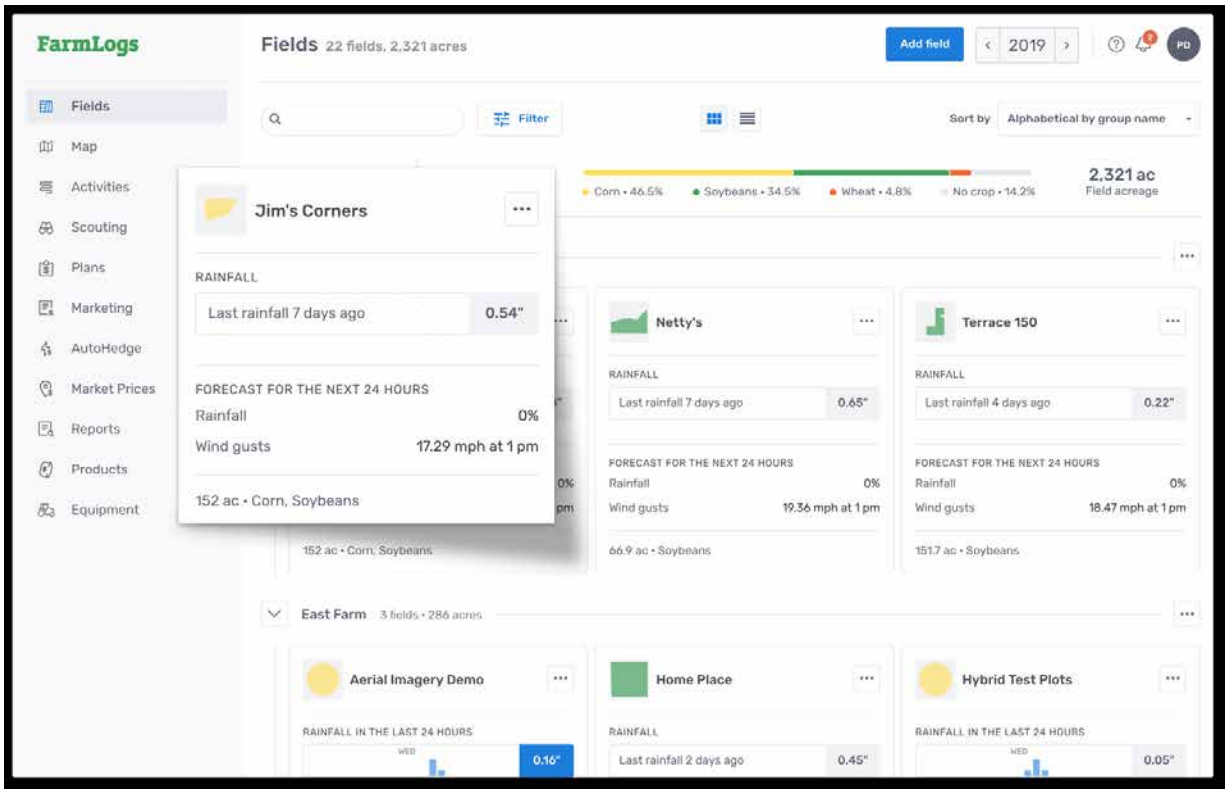

Рис. 1.3 – Приклад інтерфейсу системи «FarmLogs» [16]

Архітектура FarmLogs побудована на основі хмарних технологій, що забезпечує зберігання даних та доступ до них через веб-інтерфейс або мобільні додатки. Дані надходять із різних джерел, включаючи супутникові зображення, погодні сервіси та польову техніку. Система інтегрує ці дані, забезпечуючи фермерам комплексне бачення їхньої діяльності в реальному часі. Завдяки хмарній архітектурі користувачі можуть легко отримати доступ до своїх даних з будь-якого місця, що спрощує управління фермерським господарством.

Переваги FarmLogs:

– простота використання – інтуїтивно зрозумілий інтерфейс, що дозволяє швидко освоїти роботу із системою;

– хмарна архітектура – надає доступ до даних у будь-який час і з будь-якого пристрою, забезпечуючи мобільність і зручність;

– супутникові зображення – дозволяють моніторити стан полів у реальному часі, що допомагає краще оцінювати потреби в обробці або внесенні добрив;

– інтеграція з погодними сервісами – дозволяє точніше планувати роботи на основі прогнозів погоди та поточних умов.

Недоліки FarmLogs:

– обмежений функціонал для великих господарств – може бути недостатньо потужним для великих агропідприємств із складними виробничими процесами;

– залежність від інтернету – для роботи програми потрібне постійне підключення до мережі, що може бути проблемою в віддалених регіонах;

– вартість преміум-функцій – деякі корисні функції доступні лише в платних підписках, що може бути обмеженням для невеликих фермерів;

– обмежена інтеграція з технікою – не всі моделі польової техніки підтримують повну інтеграцію з додатком [17].

Отже, FarmLogs є ефективним інструментом для управління фермерським господарством, особливо для малих і середніх підприємств. Простий у використанні інтерфейс та хмарна архітектура роблять його доступним для фермерів, які прагнуть покращити моніторинг своїх полів і планування робіт. Проте його функціональні обмеження та вартість преміум-підписок можуть не відповідати потребам великих господарств або фермерів у віддалених місцевостях.

AgroOffice – це програмне рішення для управління сільськогосподарськими процесами, спрямоване на підвищення ефективності планування, моніторингу та аналізу діяльності фермерських господарств (рис. 1.4). Програма дозволяє фермерам управляти посівними роботами, відстежувати використання ресурсів, вести облік врожайності, контролювати стан полів та отримувати аналітичні дані для оптимізації господарських операцій.

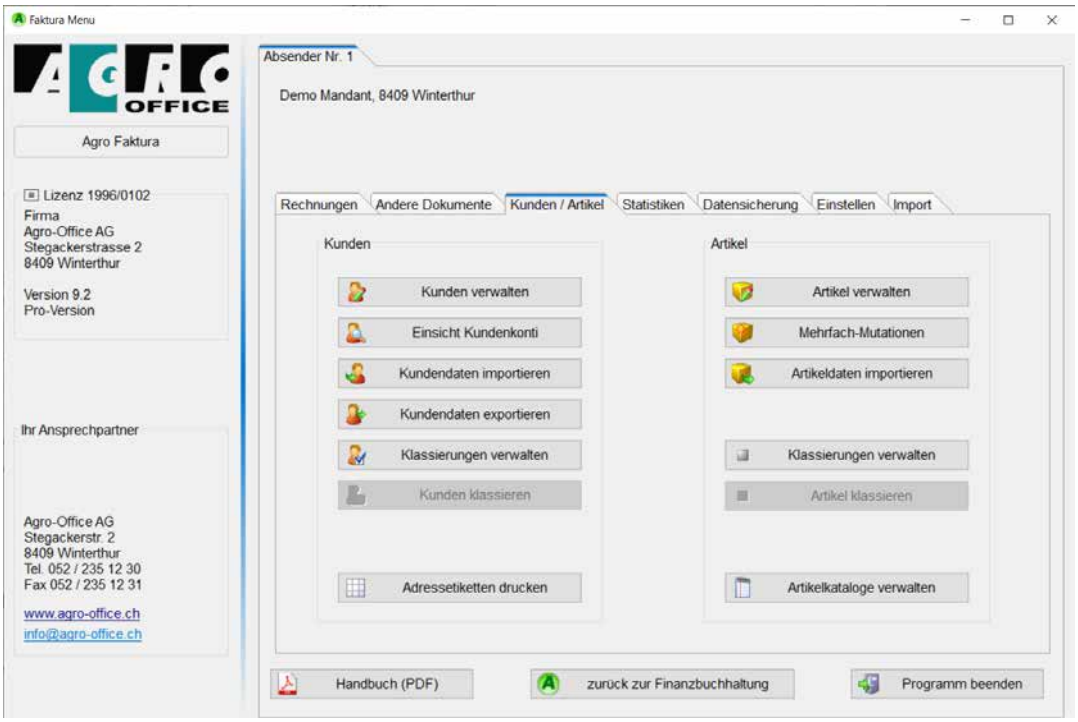

Рис. 1.4 – Приклад інтерфейсу системи «AgroOffice» [18]

Архітектура AgroOffice базується на хмарних технологіях, що забезпечує зручний доступ до інформації через інтернет з будь-якого пристрою. Система обробляє дані з різних джерел, таких як польова техніка, сенсори та інші програмні модулі. Ці дані синхронізуються у централізованій базі, яка дозволяє фермерам бачити загальну картину свого господарства, аналізувати історичні дані та прогнозувати ефективність наступних кроків. Хмарна архітектура також забезпечує високу гнучкість у використанні і можливість інтеграції з іншими системами управління.

Переваги AgroOffice:

– хмарна архітектура – забезпечує доступ до даних з будь-якого пристрою в будь-який час, дозволяючи фермерам контролювати господарство дистанційно;

– широка функціональність – підтримує управління ресурсами, посівними роботами, облік врожайності та моніторинг стану полів;

– аналітичні інструменти – надає можливість детального аналізу даних для оптимізації діяльності та прогнозування майбутніх показників;

– інтеграція з технікою та сенсорами – забезпечує збирання реальних даних з поля для кращого контролю та управління.

Недоліки AgroOffice:

– складність для новачків – через велику кількість функцій програма може бути складною для освоєння користувачами без досвіду роботи з подібними системами;

– постійна потреба в інтернеті – хмарна архітектура вимагає постійного підключення до мережі, що може бути проблемою в регіонах із поганим інтернетом;

– вартість для малих господарств – ціна програмного забезпечення може бути високою для дрібних фермерів, враховуючи всі додаткові функції та підтримку;

– обмежені можливості адаптації – може потребувати часу на налаштування і не завжди підходить для специфічних видів господарств через стандартні модулі.

Отже, AgroOffice є потужним інструментом для фермерів, що прагнуть автоматизувати та оптимізувати свої господарські процеси. Завдяки хмарній архітектурі та широкому набору функцій, програма дозволяє ефективно контролювати всі аспекти діяльності господарства. Однак складність освоєння, постійна залежність від інтернету та висока вартість можуть обмежити її використання для малих фермерських підприємств або новачків.

Аналіз існуючих програмних рішень у сфері агробізнесу вказує на те, що більшість з них мають складний функціонал, який важко освоїти новачкам, а також значні витрати на впровадження та обслуговування, що є перешкодою для малих і середніх фермерських господарств. Це створює потребу в розробці простішого програмного рішення з базовим функціоналом, яке було б легким у використанні та інтеграції, особливо для фермерів без попереднього досвіду роботи з подібними системами. Важливо також забезпечити безкоштовну версію, яка дозволить користувачам отримати доступ до ключових можливостей без необхідності значних фінансових вкладень. Такий підхід

зробить технології управління доступнішими для ширшого кола фермерів, сприяючи їх цифровій трансформації.

1.4 Постановка задачі для програмного забезпечення

Розробка програмного забезпечення для підтримки прийняття рішень в управлінні фермерським господарством ґрунтується на необхідності автоматизації складних процесів, зокрема управління ресурсами, моніторингу стану полів, аналізу погодних умов, управління витратами та оптимізації врожайності. Фермерські господарства, особливо малі та середні, потребують простих, інтуїтивно зрозумілих інструментів, які дозволять підвищити ефективність управління та прийняття рішень на основі реальних даних.

Задача програмного забезпечення полягає в тому, щоб забезпечити фермерів зручним інструментом для збору, зберігання та аналізу даних, які стосуються господарської діяльності. Для досягнення цієї мети програмне забезпечення повинне:

– надавати можливість управління інформацією про ферми, поля, культури та погодні умови;

– автоматизувати обробку даних для генерування звітів про стан врожайності, витрати, погодні умови та рекомендації для подальших дій;

– забезпечити можливість введення та зберігання даних про стан полів, культури та погодні умови. Це дозволить фермерам самостійно фіксувати ключові показники врожайності, погодних умов та обробки полів для подальшого аналізу;

– забезпечити можливість ведення обліку витрат, використання ресурсів і стану обладнання, що дозволить оптимізувати використання фінансів і підвищити ефективність виробництва;

– можливість формування звітності. Система повинна автоматично генерувати звіти на основі введених даних про посіви, витрати, врожайність та

інші ключові показники. Це дозволить фермерам отримувати зрозумілі та детальні аналітичні матеріали для прийняття обґрунтованих рішень. Звіти повинні містити інформацію про стан господарства, тенденції розвитку, можливі ризики та рекомендації для подальших дій, що значно спростить планування і управління процесами у фермерському господарстві.

Описані вимоги диктують необхідність створення централізованої бази даних, яка зберігатиме всі дані, пов'язані з діяльністю фермерського господарства. Важливо, щоб система була простою у впровадженні та використанні, щоб вона могла бути корисною не лише для великих, а й для малих фермерських господарств.

1.5 Висновок

У рамках даного розділу було проведено детальний аналіз поточного стану інформаційних систем управління фермерським господарством. Розглянуто організаційні, виробничі, технологічні та екологічні особливості різних типів фермерських господарств, що дозволило зрозуміти специфіку управління фермерською діяльністю. Виявлено, що автоматизація управлінських процесів є необхідною через зростаючу складність у керуванні ресурсами, потребу в оперативному прийнятті рішень та оптимізації витрат. Проаналізовано існуючі програмні рішення у сфері агробізнесу, зокрема Trimble Ag Software, FarmLogs і AgroOffice, з визначенням їхніх переваг та недоліків. На основі цього аналізу було обґрунтовано необхідність розробки нового програмного забезпечення, яке буде простішим у використанні, матиме легкість впровадження та забезпечуватиме базові функції у безкоштовній версії. Отримані результати слугуватимуть основою для вибору технологій і проєктування програмного забезпечення в наступному розділі, де буде розглянуто конкретні технічні рішення для розробки інструментів автоматизації управління фермерськими господарствами.

2 ВИБІР ТЕХНОЛОГІЙ ТА ОСОБЛИВОСТІ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ УПРАВЛІННЯ ФЕРМЕРСЬКИМ ГОСПОДАРСТВОМ

2.1 Вибір технологій розробки

Розробка програмного забезпечення для системи підтримки прийняття рішень у сфері агробізнесу вимагає ретельного підбору технологій, що забезпечать ефективну обробку великих обсягів даних, гнучкість у розширенні та інтеграцію з різноманітними джерелами інформації. Основні аспекти вибору технологій включають мову програмування, середовище розробки та систему управління базами даних (СУБД).

2.1.1 Мова програмування для системи підтримки рішенью. Під час вибору мови програмування для розробки системи важливо враховувати її здатність до роботи з великими обсягами даних, гнучкість інтеграції з іншими технологіями та зручність розробки інтерфейсу користувача. Крім цього, мова повинна забезпечувати продуктивність та надійність, щоб система могла ефективно обробляти запити, виконувати аналітику та підтримувати сучасні бібліотеки для роботи з даними. Тому розглянемо можливості популярних мов програмування – Python, Java та C# – для оцінки їх потенціалу в подібних розробках.

Python є динамічною та інтерпретованою мовою, що ідеально підходить для обробки даних, аналізу та машинного навчання завдяки багатому набору бібліотек (наприклад, Pandas, NumPy та TensorFlow) [19]. Мова є простою у використанні, що пришвидшує процес розробки ідеального рішення, але зазвичай програє у продуктивності через інтерпретовану природу [20]. Python також популярний серед розробників наукового програмного забезпечення, завдяки чому має велику кількість готових рішень та бібліотек.

Java – це мова, відома своєю незалежністю від платформи завдяки JVM (Java Virtual Machine), що забезпечує її універсальність і зручність для великих масштабованих систем [21]. Вона забезпечує високий рівень безпеки, стабільності та підтримує багатозадачність, що важливо для систем управління підприємствами [22]. Java має численні інструменти для роботи з даними та інтеграції в мережеві середовища, хоча її синтаксис є складнішим для новачків.

Мова C# відома своїм зручним синтаксисом і високою сумісністю з платформою .NET, що робить її популярним вибором для створення корпоративних застосунків та веб-сервісів [23]. Вона поєднує об'єктно-орієнтовану модель з можливостями багатопотокової обробки, що дозволяє створювати продуктивні та стійкі системи [24]. C# підтримує широкий спектр бібліотек для роботи з даними та інструментів для інтеграції з різними джерелами інформації, що робить її ефективним вибором для багатофункціональних систем управління.

У табл. 2.1 представлено порівняння основних характеристик зазначених мов програмування.

Таблиця 2.1

Порівняння мов програмування

Характеристика	Python	Java	C#
Швидкість виконання (кількісно)	Низька (інтерпретована мова)	Висока (JVM)	Висока (компіляція під .NET)
Продуктивність	Оптимальна для обробки даних	Висока для серверних рішень	Висока для десктопних та веб-додатків
Простота навчання	Висока, зрозумілий синтаксис	Середня, строгий синтаксис	Середня, інтуїтивний інтерфейс
Мережеві можливості	Широкі, але вимагають зовнішніх бібліотек	Добра підтримка через Java EE	Вбудована підтримка .NET Core
Безпека	Обмежена на рівні інтерпретатора	Висока, є перевірка безпеки на рівні JVM	Висока, підтримка захисту через .NET безпеку
Спільнота розробників	Величезна, широко використовується	Висока, особливо у веб-розробці	Висока, особливо у розробці корпоративних рішень

Виходячи з проведеного аналізу, мову C# обрано для розробки системи завдяки її високій продуктивності, особливо в рамках платформи .NET, що забезпечує швидке виконання та надійність коду. Повна підтримка багатопотоковості дозволяє системі обробляти кілька процесів одночасно, що є критично важливим для інтерактивних та масштабованих рішень. Інтеграція C# з Windows та екосистемою .NET забезпечує просту і швидку реалізацію мережевих можливостей та високий рівень безпеки даних, що відповідає вимогам до захищених корпоративних додатків. Окрім того, C# пропонує широкий спектр інструментів для створення інтерфейсу користувача та управління даними, що сприяє зручності розробки та оптимізації взаємодії з кінцевими користувачами.

2.1.2 Середовище розробки програмного продукту. Оскільки для розробки обрано мову C#, важливо вибрати оптимальне середовище розробки, яке забезпечить зручність роботи з .NET платформою, надійний інструментарій для тестування і налагодження, а також підтримку інтеграції з системами контролю версій. Найбільш відповідними варіантами є Visual Studio 2022, Visual Studio Code та Rider, кожне з яких має свої особливості і може запропонувати певні переваги для різних аспектів розробки на C#. Нижче наведено короткий огляд можливостей кожного з цих середовищ, що дозволяє краще зрозуміти їх придатність для проекту.

Visual Studio 2022 – повноцінне інтегроване середовище розробки (IDE), яке має потужний набір інструментів для створення, налагодження та розгортання додатків на .NET [25]. Воно підтримує роботу з великими проєктами, пропонує ефективне управління багатьма компонентами та підтримує різні версії C# для зворотної сумісності. Visual Studio 2022 також забезпечує інтеграцію з хмарними сервісами Microsoft Azure, що є корисним для розробки веб-орієнтованих додатків.

Visual Studio Code – легке редакторське середовище, орієнтоване на швидкість і гнучкість, із широкою підтримкою плагінів, що дозволяють додати функціональність, потрібну для розробки на C# [26]. Visual Studio Code підтримує інтеграцію з Git і є зручним для швидких змін у коді та роботи в різних мовах програмування. Це середовище добре підходить для невеликих проєктів або етапів розробки, що потребують високої швидкості редагування.

Rider – середовище розробки від JetBrains, яке спеціалізується на .NET і надає потужні інструменти для рефакторингу та аналізу коду, що є особливо корисним для великих проєктів [27]. Rider відрізняється відмінною оптимізацією та продуктивністю, швидким запуском і зручною інтеграцією з іншими продуктами JetBrains. Це IDE підтримує багатоплатформенну розробку, що робить його корисним для команд, які працюють у різних операційних середовищах.

У табл. 2.2 представлено порівняний аналіз середовищ розробки ПЗ.

Таблиця 2.2

Порівняння середовищ розробки

Характеристика	Visual Studio 2022	Visual Studio Code	Rider
Тип середовища	Повноцінне IDE для розробки великих проєктів	Легке редакторське середовище	Інтегроване IDE з фокусом на .NET
Підтримка мов програмування	Широка підтримка, особливо для .NET, C++, Python	Багатомовна підтримка через розширення	Основна підтримка .NET, JavaScript, TypeScript
Підтримка C# та .NET	Повна, з нативною інтеграцією .NET	Доступна через розширення, з базовою функціональністю	Повна, з розширеним інструментарієм для .NET
Інтеграція з Git	Повна інтеграція з Git та іншими системами контролю	Вбудована підтримка Git	Повна підтримка Git і інтеграція з JetBrains Space
Швидкодія	Високий рівень для великих проєктів	Висока для невеликих проєктів	Оптимізована для великих проєктів
Вартість	Безкоштовно (Community Edition) або за передплатою	Безкоштовно	За передплатою

Visual Studio 2022 обрано для розробки системи завдяки його потужним інструментам для комплексного управління великими проектами та нативній підтримці C# і .NET, що дозволяє максимально ефективно реалізовувати функціональні можливості системи. Це середовище надає широкий набір інструментів для налагодження, профілювання та аналізу коду, що сприяє підвищенню продуктивності розробки та полегшує підтримку стабільності проєкту. Visual Studio 2022 інтегрується з хмарними сервісами Azure, що розширює можливості для масштабування та зберігання даних, а також дозволяє ефективно працювати з системами контролю версій, зокрема Git. Крім того, Visual Studio 2022 має добре структуровані інструменти для створення користувацького інтерфейсу, що робить його ідеальним вибором для розробки надійної та зручної системи підтримки прийняття рішень.

2.1.3 Система управління базами даних. Для розробки надійної та масштабованої системи підтримки прийняття рішень важливо обрати відповідну СУБД, яка забезпечить стабільну роботу з великими обсягами даних, високу швидкість обробки запитів і надійність у питаннях безпеки та резервного копіювання. Вибір СУБД повинен враховувати не лише сумісність з платформою .NET, але й можливості інтеграції з іншими компонентами системи, простоту адміністрування та підтримку функцій для аналітики.

MS SQL Server є корпоративною реляційною СУБД, яка надає розширені можливості для роботи з даними, включаючи інструменти аналітики та підтримку хмарних рішень Azure [28]. Вона оптимізована для інтеграції з екосистемою Microsoft, що забезпечує надійну роботу та просту взаємодію з C# і .NET. MS SQL Server має розширені можливості безпеки, що робить її популярним вибором для корпоративних і високонавантажених систем.

MySQL – це відкрита СУБД, що забезпечує високу продуктивність та простоту використання для веб-додатків та проєктів середнього розміру [29]. Вона має гнучкі налаштування та підтримку широкого кола операційних систем, що

робить її популярною серед розробників веб-застосунків. MySQL має обмежену підтримку складних аналітичних функцій порівняно з іншими корпоративними СУБД, але відрізняється простотою адміністрування та легкістю інтеграції.

FireBird – це безкоштовна реляційна СУБД, що має легкий інсталяційний процес та може працювати на різних платформах, включаючи Windows і Linux [30]. Вона забезпечує високий рівень безпеки даних і є зручною для проектів із середнім та малим навантаженням, де потрібна стабільність та надійність. FireBird пропонує простий у використанні набір функцій для обробки даних, але не завжди підтримує такі ж розширені аналітичні можливості, як комерційні рішення.

У табл. 3.2 наведено порівняння основних характеристик цих СУБД.

Табл. 2.3 відображає основні характеристики розглянутих СУБД.

Таблиця 2.3

Порівняння СУБД

Характеристика	MS SQL Server	MySQL	FireBird
Продуктивність	Висока, оптимізована для роботи з великими обсягами даних	Висока, але залежить від конфігурації	Середня, оптимізована для середніх навантажень
Масштабованість	Відмінна, підтримка великих корпоративних систем	Висока, для веб-застосунків і середніх проєктів	Обмежена, підходить для малих і середніх проектів
Функції безпеки	Розширені можливості, з підтримкою шифрування та ролей	Базові функції, потребує додаткових налаштувань	Основні функції безпеки
Резервне копіювання	Автоматичне, з можливістю інкрементного відновлення	Обмежені можливості, налаштовується вручну	Базова підтримка, не має вбудованого інкрементного
Інтеграція з .NET	Повна, нативна інтеграція	Обмежена, через драйвери та розширення	Часткова, потребує налаштування

Вибір MS SQL Server для розробки системи обумовлений його високою продуктивністю і надійністю в роботі з великими обсягами даних, що є ключовим для забезпечення стабільності та швидкості виконання запитів у корпоративному середовищі. MS SQL Server пропонує розширені функції безпеки, включаючи підтримку шифрування даних і налаштування ролей, що забезпечує захист конфіденційної інформації. Крім того, повна інтеграція з платформою .NET та підтримка аналітичних інструментів роблять цю СУБД ідеальною для глибокого аналізу даних і побудови комплексних бізнес-логік. Можливість інтеграції з хмарними рішеннями Azure дозволяє легко масштабувати систему та оптимізувати роботу у хмарному середовищі, що розширює можливості розвитку проєкту в майбутньому.

2.2 Аналіз вимог до системи. Діаграми сценаріїв використання

Аналізуючи вимоги до системи підтримки прийняття рішень в управлінні фермерським господарством, слід виділити основні категорії вимог, які включають функціональні, нефункціональні та вимоги до інтерфейсу користувача. Вимоги цієї системи розроблені так, щоб забезпечити зручність і надійність у використанні як для кінцевих користувачів, так і для адміністраторів. Функціональні вимоги визначають специфічні завдання, які система повинна забезпечувати, окреслюючи основні функції для підтримки фермерських процесів і ефективного управління ними. Ці вимоги описують очікувану поведінку системи при взаємодії з користувачами або іншими компонентами, а також реакції на певні події чи запити [31]. Вони конкретизують функціональні можливості, необхідні для роботи з даними про ферми, поля, витрати та інше, що формує основу для створення зручного інструменту управління ресурсами господарства. У табл. 2.4 наведено перелік основних функціональних вимог, які мають забезпечити надійну та ефективну роботу системи і підтримувати всі необхідні процеси.

Таблиця 2.4

Функціональні вимоги до застосунку

Вимоги	Опис
REQ-1	Реєстрація та автентифікація користувачів для забезпечення контролю доступу до системи.
REQ-2	Ведення журналу активності користувачів для моніторингу та аудиту подій у системі.
REQ-3	Можливість додавати та оновлювати дані про ферми, включаючи їх основні характеристики та розташування.
REQ-4	Можливість додавання та редагування інформації про фермерські поля з урахуванням їх площі та розташування.
REQ-5	Додавання та редагування даних про власників фермерських господарств, зокрема контактної інформації.
REQ-6	Можливість додавати та оновлювати інформацію про сільськогосподарські культури, що вирощуються на фермах.
REQ-7	Додавання та редагування даних про обладнання, яке використовується на фермі, включаючи тип, стан та термін служби.
REQ-8	Фіксація рекомендацій і рішень для користувачів з метою підтримки ефективного управління процесами.
REQ-9	Ведення обліку витрат на ферму для аналізу фінансових витрат і оптимізації ресурсів.
REQ-10	Фіксація виконаних робіт на фермі для подальшого аналізу ефективності управління та формування звітності.
REQ-11	Можливість генерувати звіти за показниками, такими як урожайність по полях, погодні умови, витрати та обслуговування обладнання.

Ці функціональні вимоги створюють основу для системи, яка забезпечує управління ресурсами фермерського господарства, дозволяючи користувачам ефективно контролювати процеси та приймати обґрунтовані рішення.

Нефункціональні вимоги визначають характеристики системи, які забезпечують належну якість її роботи та відповідають критеріям надійності, продуктивності, безпеки та масштабованості. Вони стосуються не тільки стабільності та швидкодії, а й узгодженості інтерфейсу та можливості інтеграції

з іншими компонентами, що є важливим для зручності користувачів. Ці вимоги забезпечують підтримку функціональних можливостей системи, сприяючи її ефективній і безперебійній роботі навіть за умов зростання обсягів даних і кількості користувачів. У табл. 2.5 наведено основні нефункціональні вимоги до застосунку для підтримки прийняття рішень.

Таблиця 2.5

Нефункціональні вимоги застосунку

Вимоги	Опис
NFR-1	Система повинна забезпечувати стабільну роботу з високим рівнем надійності та бути захищеною від збоїв.
NFR-2	Система має бути здатна обробляти великі обсяги даних та підтримувати збільшення кількості записів без втрати продуктивності.
NFR-3	Забезпечення швидкої обробки запитів і мінімального часу відповіді, особливо під час роботи з великими обсягами даних.
NFR-4	Інтерфейс системи повинен бути інтуїтивно зрозумілим і однаковим для всіх користувачів, незалежно від їхніх ролей.
NFR-5	Система повинна підтримувати високу доступність, з мінімальним часом простою, навіть під час оновлень чи технічного обслуговування.
NFR-6	Забезпечення захищеного доступу до даних через механізми автентифікації та шифрування інформації.
NFR-7	Система повинна підтримувати масштабування, дозволяючи збільшувати ресурси та функціональні можливості без втрати продуктивності.

Перелічені нефункціональні вимоги гарантують, що система працюватиме стабільно й надійно, навіть за умов інтенсивної експлуатації, забезпечуючи безпеку даних і швидкий доступ до них, що є критичними факторами для ефективного управління фермерським господарством.

Актори – це особи чи сутності, які взаємодіють із системою для досягнення конкретних цілей, що визначаються їхніми функціональними завданнями та ролями. У цій системі основними акторами є адміністратор, користувачі фермерських господарств і база даних, кожен із яких відіграє

важливу роль у забезпеченні функціонування застосунку. Метою кожного актора є оптимізація його взаємодії із системою, що дозволяє ефективно підтримувати процеси управління фермою, захист і збереження даних, а також зручний доступ до потрібної інформації.

Таблиця 2.6

Актори та цілі застосунку

Актори	Цілі
Адміністратор	Адміністратор відповідає за забезпечення безпеки системи та контроль над доступом користувачів, підтримуючи цілісність і конфіденційність даних, а також стабільну роботу системи.
Користувач	Користувачі фермерських господарств використовують систему для отримання оперативної інформації про поля, врожайність, обладнання і витрати, оптимізуючи управління фермою.
База даних	База даних забезпечує збереження, структурування та надійний доступ до даних системи, підтримуючи цілісність і доступність інформації для інших акторів системи.

Варіанти використання представляють собою сценарії взаємодії акторів із системою, які спрямовані на досягнення конкретних цілей та забезпечення певних функцій, необхідних для ефективного управління фермерським господарством. Кожен варіант використання описує кроки, які повинен виконати користувач або адміністратор для отримання бажаних результатів, наприклад, управління даними про ферми, моніторинг обладнання, додавання нових полів або реєстрацію витрат. Залежно від ролі користувача, варіанти використання можуть відрізнятися як за рівнем доступу до інформації, так і за функціональністю. Наприклад, адміністратор може мати змогу редагувати дані про користувачів, в той час як звичайний користувач обмежується переглядом або внесенням інформації про врожайність та погодні умови. Кожен сценарій покликаний відображати конкретну послідовність дій і реакцію системи, що є важливими для забезпечення зручного та інтуїтивного інтерфейсу. У табл. 2.7

наведено основні варіанти використання, що підкреслюють ключові процеси взаємодії користувачів з системою у контексті управління фермерським господарством. Такий підхід до визначення сценаріїв допомагає краще розуміти потреби кожного користувача і оптимізувати функціональність застосунку, роблячи його інструментом ефективного управління господарськими ресурсами та процесами.

Таблиця 2.7

Опис варіантів використання системи

№	Ім’я	Опис
1	2	3
UC1	Реєстрація користувача	Реєстрація нового користувача в системі, що включає введення необхідних особистих даних і створення облікового запису.
UC2	Автентифікація в системі	Перевірка особи користувача при вході в систему для отримання доступу до її функціоналу.
UC3	Вивід каталогу користувачів	Надання адміністраторам можливості перегляду списку всіх користувачів системи.
UC4	Додати користувача	Додавання нового користувача до системи, створення його профілю з визначеними правами доступу.
UC5	Редагувати користувача	Зміна даних існуючого користувача, включаючи особисту інформацію та права доступу.
UC6	Вивід каталогу ферм	Відображення списку всіх зареєстрованих ферм у системі для перегляду користувачами з відповідними правами доступу.
UC7	Додати ферму	Додавання нової ферми до системи, із зазначенням її характеристик та параметрів.
UC8	Редагувати ферму	Оновлення інформації про ферму, включаючи її розташування, розмір, або види сільськогосподарських культур.
UC9	Вивід каталогу фермерських полів	Відображення детального списку всіх полів, що належать певній фермі.
UC10	Додати поле	Додавання нового поля до ферми із зазначенням його площі, розташування та виду культури.

Закінчення табл. 2.7

UC11	Редагувати поле	Модифікація даних про поле, включаючи зміну виду культури, розміру чи інших параметрів.
UC12	Вивід каталогу власників	Відображення списку всіх власників, зареєстрованих у системі.
UC13	Додати власника	Додавання нового власника до системи із зазначенням основної інформації про нього.
UC14	Редагувати власника	Оновлення інформації про власника, включаючи зміну контактних даних чи інших характеристик.
UC15	Вивід списку фермерських культур	Відображення наявних видів сільськогосподарських культур, що вирощуються на фермах, зареєстрованих у системі.
UC16	Додати культуру	Додавання нового виду культури до списку з наданням основної інформації про неї.
UC17	Редагувати культуру	Зміна даних про певний вид культури, таких як сезонність або опис.
UC18	Вивід списку обладнання	Перегляд каталогу всього обладнання, зареєстрованого для використання на фермах.
UC19	Додати обладнання	Додавання нового обладнання із зазначенням його назви, типу та інших характеристик.
UC20	Редагувати обладнання	Оновлення інформації про обладнання, наприклад, зміну назви, типу або стану обслуговування.
UC21	Фіксація підтримки рішень	Збереження підтримки та рекомендацій для користувачів щодо прийняття оптимальних рішень.
UC22	Вивід каталогу витрат	Відображення всіх витрат, пов’язаних з управлінням фермою, що дозволяє здійснити аналіз витрат.
UC23	Додати витрати	Додавання нової статті витрат, включаючи опис, суму та дату.
UC24	Редагувати витрати	Оновлення існуючих записів витрат, включаючи зміну опису чи суми.
UC25	Звітність	Генерація звітів, таких як урожайність по полях, погодні умови за обраний період, витрати на ферму, обслуговування обладнання, та аналіз урожайності.

На основі проаналізованих даних було розроблено діаграми варіантів використання (use-case) для ролей "адміністратор" та "користувач", які відображені на рис. 2.1 та рис. 2.2 відповідно.

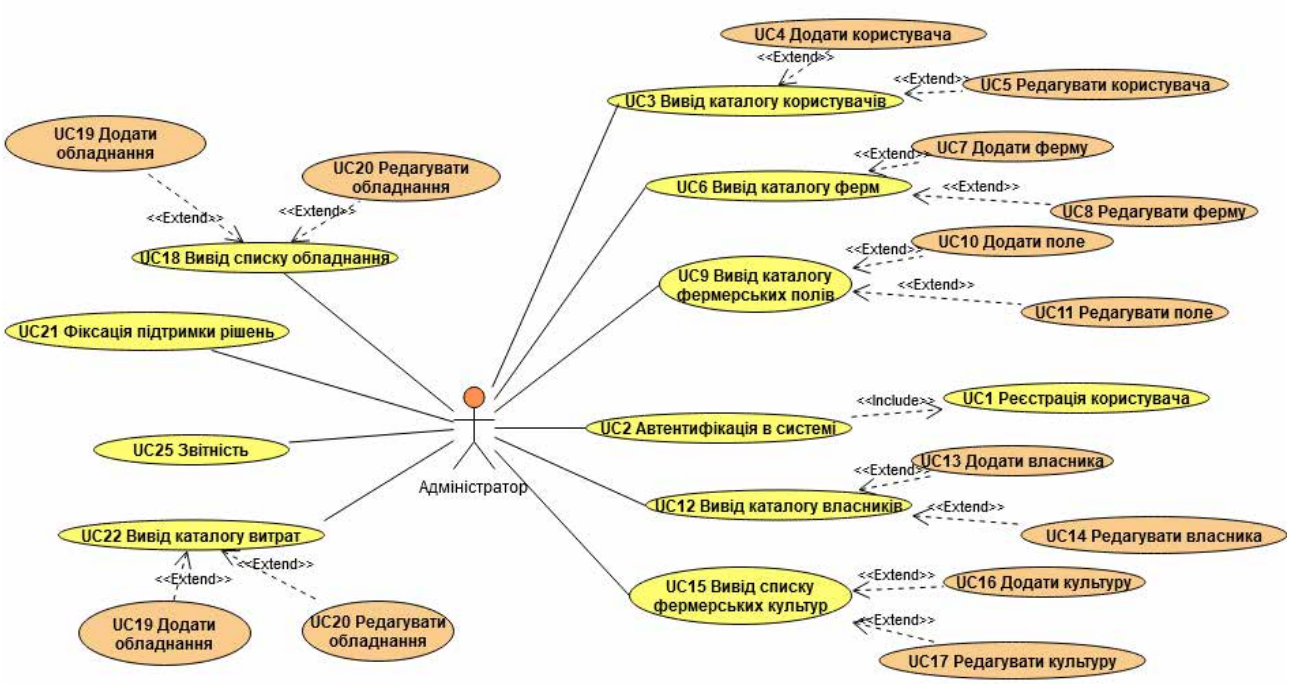

Рис. 2.1 Use-case діаграма для ролі «адміністратор»

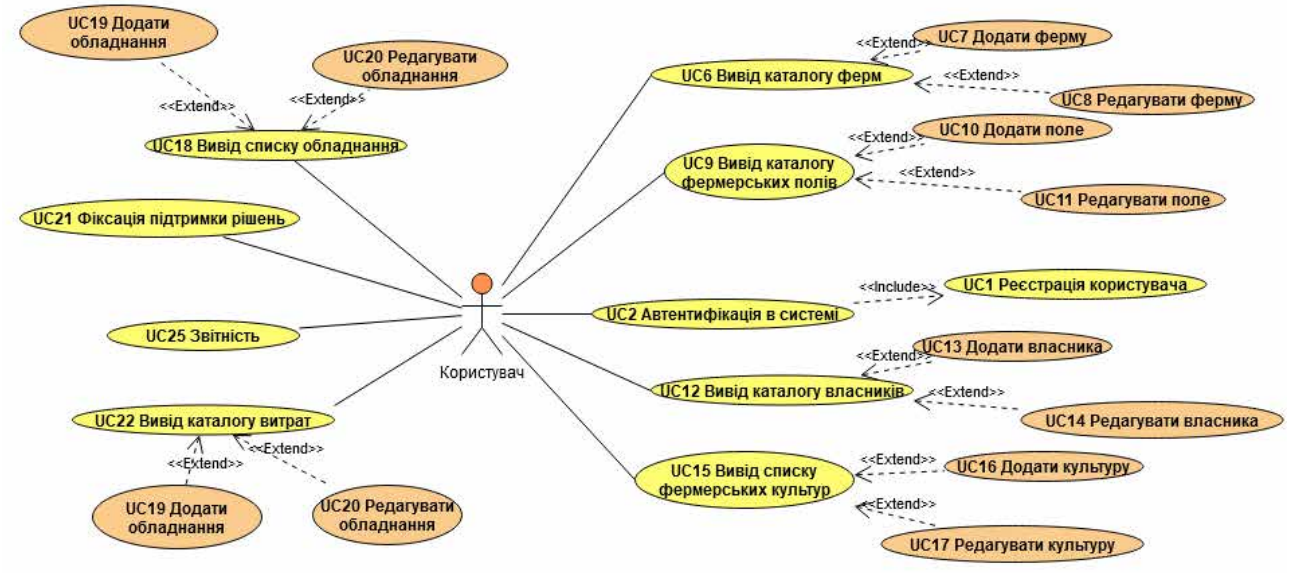

Рис. 2.2 Use-case діаграма для ролі «користувач»

Діаграми показують послідовність дій та варіанти взаємодії кожного актора із системою, що дозволяє зрозуміти основні функції та сценарії, необхідні для ефективного виконання завдань. Побудовані діаграми надають

наочне уявлення про структуру взаємодії, допомагаючи розробникам і користувачам краще зрозуміти ключові процеси та ролі в системі.

2.3 Особливості проєктування бази даних

Під час проектування бази даних для предметної області застосовувався метод «сутність-зв'язок» (ER-метод). Використання цього методу дозволяє структурувати дані, окреслити їхню логічну взаємодію і створити чітку модель, що відображає ключові елементи системи та їх зв'язки перед деталізацією загальної структури [32]. ER-метод забезпечує послідовне виконання завдань на базових рівнях, що сприяє кращому розумінню структурних зв'язків між даними в обраній предметній області.

На основі цього підходу була створена логічна модель бази даних із використанням ER-діаграми, яка наочно представляє сутності (entities) разом з їх атрибутами і зв’язками (relationships) між ними. ER-діаграма допомагає забезпечити цілісність і логічність структури бази даних, відображаючи всі ключові компоненти системи управління фермерським господарством. У рамках проекту бази даних для цієї системи було виділено наступні основні сутності:

Таблиця «Users» (Користувачі) створена для зберігання інформації про користувачів системи, включаючи особисті дані, логін і пароль, а також ідентифікатор ролі для визначення рівня доступу.

Таблиця 2.8

Структура таблиці «Users»

Назва поля	Тип даних	ПК	ЗК	Опис поля
UsersId	int	+	-	Унікальний ідентифікатор користувача, первинний ключ
FirstName	nvarchar(60)	-	-	Ім'я користувача
LastName	nvarchar(60)	-	-	Прізвище користувача

Закінчення таблиці 2.8

UsersName	nvarchar(60)	-	-	Логін користувача для входу в систему
UsersPassword	nvarchar(250)	-	-	Пароль користувача
RoleId	int	-	-	Ідентифікатор ролі, що вказує на рівень доступу користувача
Description	nvarchar(1200)	-	-	Додатковий опис користувача або примітки
Email	nvarchar(150)	-	-	Електронна пошта користувача

Таблиця «Logs» (Журнал подій) створена для зберігання інформації про події, які відбуваються у системі, зокрема дії користувачів. Ця таблиця дозволяє відслідковувати активність користувачів, що є корисним для аудиту та аналізу подій.

Таблиця 2.9

Структура таблиці «Logs»

Назва поля	Тип даних	ПК	ЗК	Опис поля
LogsId	int	+	-	Ідентифікатор події, первинний ключ
UsersId	int	-	-	Ідентифікатор користувача, який виконав подію
EventNameShow	nvarchar(MAX)	-	-	Опис події або її назва
EventDate	datetime	-	-	Дата і час, коли відбулась подія

Таблиця «Crop» (Сільськогосподарські культури) призначена для зберігання даних про види культур, що вирощуються на фермі, зокрема назву культури, сезон її вирощування та опис. Ця таблиця є базовою для управління інформацією про врожайність та оптимізацію аграрного процесу.

Таблиця 2.10

Структура таблиці «Crop»

Назва поля	Тип даних	ПК	ЗК	Опис поля
CropId	int	+	-	Ідентифікатор культури, первинний ключ
CropName	nvarchar(250)	-	-	Назва сільськогосподарської культури
Season	nvarchar(50)	-	-	Сезон, у який вирощується культура
Description	nvarchar(MAX)	-	-	Опис або додаткова інформація про культуру

Таблиця «DecisionSupport» (Підтримка прийняття рішень) призначена для зберігання даних про рекомендації та дії, які здійснюються на основі аналізу полів фермерських господарств. Ця таблиця використовується для відстеження прийнятих рішень і поточного стану виконання дій, що дозволяє оптимізувати процес управління фермою.

Таблиця 2.11

Структура таблиці «DecisionSupport»

Назва поля	Тип даних	ПК	ЗК	Опис поля
SupportId	int	+	-	Унікальний ідентифікатор рекомендації, первинний ключ
FarmFieldId	int	-	-	Ідентифікатор поля ферми, для якого надано рекомендацію
DecisionSupportDate	datetime	-	-	Дата створення рекомендації або рішення
Recommendation	nvarchar(MAX)	-	-	Текст рекомендації або вказівки для прийняття рішення
ActionStatus	nvarchar(50)	-	-	Статус виконання рекомендації (наприклад, виконано/в процесі)

Таблиця «Equipment» (Обладнання) призначена для зберігання даних про обладнання, яке використовується на фермі, включаючи його тип, останню дату обслуговування та приналежність до певного господарства. Ця таблиця

важлива для підтримки ефективності роботи обладнання та забезпечення його своєчасного обслуговування.

Таблиця 2.12

Структура таблиці «Equipment»

Назва поля	Тип даних	ПК	ЗК	Опис поля
EquipmentId	int	+	-	Ідентифікатор обладнання
EquipmentName	nvarchar(250)	-	-	Назва обладнання
EquipmentType	nvarchar(100)	-	-	Тип обладнання
FarmId	int	-	-	Ідентифікатор ферми, до якої належить обладнання
LastServiceDate	datetime	-	-	Дата останнього обслуговування обладнання

Таблиця «Expense» (Витрати) призначена для зберігання інформації про фінансові витрати, пов’язані з управлінням фермерським господарством, включаючи опис витрат, суму та дату здійснення. Вона дозволяє контролювати та аналізувати фінансові витрати на фермі, що є важливим для оцінки рентабельності та оптимізації ресурсів.

Таблиця 2.13

Структура таблиці «Expense»

Назва поля	Тип даних	ПК	ЗК	Опис поля
ExpenseId	int	+	-	Унікальний ідентифікатор витрат
Description	nvarchar(MAX)	-	-	Опис статті витрат
Amount	float	-	-	Сума витрат
ExpenseDate	datetime	-	-	Дата здійснення витрат
FarmId	int	-	-	Ідентифікатор ферми, до якої належать витрати

Таблиця «FarmField» (Поля ферми) призначена для зберігання даних про фермерські поля, включаючи їх розмір, назву, опис і вирощувану культуру. Ця

таблиця є основою для відстеження посівів, контролю врожайності та планування сільськогосподарських робіт на різних полях ферми.

Таблиця 2.14

Структура таблиці «Expense»

Назва поля	Тип даних	ПК	ЗК	Опис поля
FarmFieldId	int	+	-	Ідентифікатор поля, первинний ключ
FarmFieldName	nvarchar(250)	-	-	Назва поля
FieldSize	float	-	-	Розмір поля (вказується в гектарах або іншій одиниці)
Description	nvarchar(MAX)	-	-	Додатковий опис поля
CropId	int	-	-	Ідентифікатор культури, яка вирощується на полі
FarmId	int	-	-	Ідентифікатор ферми, до якої належить поле

Таблиця «Farm» (Ферма) призначена для зберігання основної інформації про фермерські господарства, зокрема їх назву, розташування та опис. Ця таблиця дозволяє об’єднати інформацію про різні ферми та забезпечити зв’язок із власниками, що допомагає в управлінні та обліку господарських ресурсів.

Таблиця 2.15

Структура таблиці «Farm»

Назва поля	Тип даних	ПК	ЗК	Опис поля
FarmId	int	+	-	Унікальний ідентифікатор ферми
FarmName	nvarchar(250)	-	-	Назва ферми
Location	nvarchar(250)	-	-	Розташування ферми
Description	nvarchar(MAX)	-	-	Додаткова інформація або опис ферми
OwnerId	int	-	-	Ідентифікатор власника, який володіє фермою

Таблиця «Owner» (Власник) призначена для зберігання даних про власників фермерських господарств, включаючи ім'я, контактну інформацію та адресу. Ця таблиця є важливою для відстеження прав власності та забезпечує зв’язок власників із відповідними господарствами.

Таблиця 2.16

Структура таблиці «Owner»

Назва поля	Тип даних	ПК	ЗК	Опис поля
OwnerId	int	+	-	Унікальний ідентифікатор власника
FirstName	nvarchar(50)	-	-	Ім'я власника
LastName	nvarchar(50)	-	-	Прізвище власника
Phone	nvarchar(20)	-	-	Контактний номер телефону власника
Address	nvarchar(MAX)	-	-	Адреса власника
Email	nvarchar(120)	-	-	Електронна пошта власника

Таблиця «WeatherData» (Метеодані) призначена для зберігання інформації про погодні умови, що можуть впливати на сільськогосподарські процеси. Вона включає дані про температуру, вологість, опади та дату спостереження, що дозволяє здійснювати аналіз впливу кліматичних умов на врожайність та інші агротехнічні процеси на фермі.

Таблиця 2.17

Структура таблиці «Owner»

Назва поля	Тип даних	ПК	ЗК	Опис поля
WeatherDataId	int	+	-	Унікальний ідентифікатор запису метеоданих
WeatherDate	datetime	-	-	Дата фіксації погодних умов
Temperature	float	-	-	Температура повітря на дату спостереження
Humidity	float	-	-	Вологість повітря на дату спостереження
Precipitation	float	-	-	Кількість опадів на дату спостереження
FarmId	int	-	-	Ідентифікатор ферми

Під час розробки бази даних для інформаційної системи була створена ER-діаграма, яка наочно представляє всі сутності та їхні взаємозв'язки в базі даних (рис. 2.3).

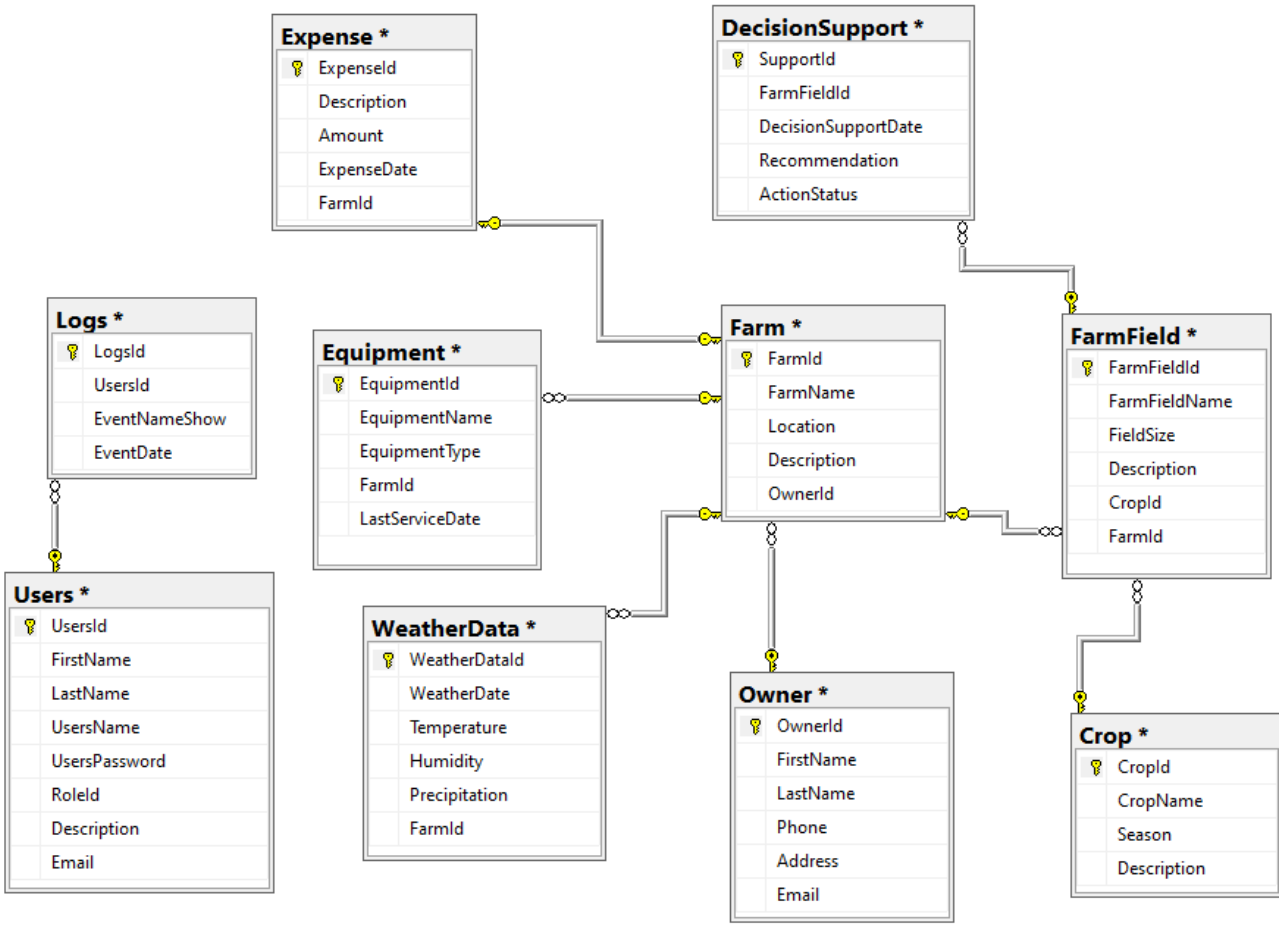

Рис. 2.3 Діаграма бази даних

ER-діаграма є важливим інструментом для подальшого впровадження функціоналу системи, оскільки вона відображає структуру даних і логічні зв'язки між ними. Завдяки цій діаграмі розробники можуть чітко бачити, як мають взаємодіяти сутності, що сприяє коректному проектуванню зберігання та обробки інформації. ER-діаграма також забезпечує ефективну комунікацію між розробниками та замовниками, дозволяючи їм узгодити всі деталі структури даних і уникнути можливих помилок або непорозумінь. Вона виступає основою для побудови функціональних можливостей системи, відповідно до вимог проекту.

2.4 Архітектура програмного забезпечення

Для розробки системи підтримки прийняття рішень в управлінні фермерським господарством обрано трирівневу архітектуру, яка є ефективним підходом для побудови масштабованих та надійних інформаційних систем [33]. Ця архітектура складається з трьох основних рівнів: рівня представлення, рівня бізнес-логіки та рівня доступу до даних.

Трирівнева архітектура дозволяє розділити функціональність системи на незалежні логічні частини, забезпечуючи гнучкість та можливість швидкого внесення змін або розширень. Кожен рівень має свою специфічну функцію: рівень представлення призначений для взаємодії з користувачем, відображення інтерфейсу та отримання введених даних; рівень бізнес-логіки реалізує обробку інформації та виконання основних операцій системи, що є основою для прийняття рішень; рівень доступу до даних забезпечує зберігання, отримання та безпечний доступ до даних, необхідних для роботи системи.

Обрна архітектура підходить для побудови програмного забезпечення, яке потребує високого рівня масштабованості та зручності в обслуговуванні, адже кожен рівень може бути змінений або масштабований незалежно від інших. Впровадження трирівневої архітектури також сприяє підвищенню безпеки даних, оскільки доступ до них здійснюється лише через рівень доступу до даних, що контролює всі операції з інформацією. Крім того, ця структура дозволяє команді розробників ефективно розподілити завдання, зосереджуючи увагу на окремих компонентах системи та їхніх відповідних функціях.

2.4.1 Особливості реалізації рівня бізнес-логіки. Рівень бізнес-логіки розробленої системи відповідає за обробку даних, здійснення основних операцій та реалізацію ключових функцій, що забезпечують прийняття рішень у фермерському господарстві. Цей рівень включає класи, які виконують специфічні завдання для аналітики, обробки даних і підтримки функцій системи. На рис. 2.4 представлена діаграма класів бізнес-логіки, яка відображає структуру і взаємозв'язки класів цього рівня.

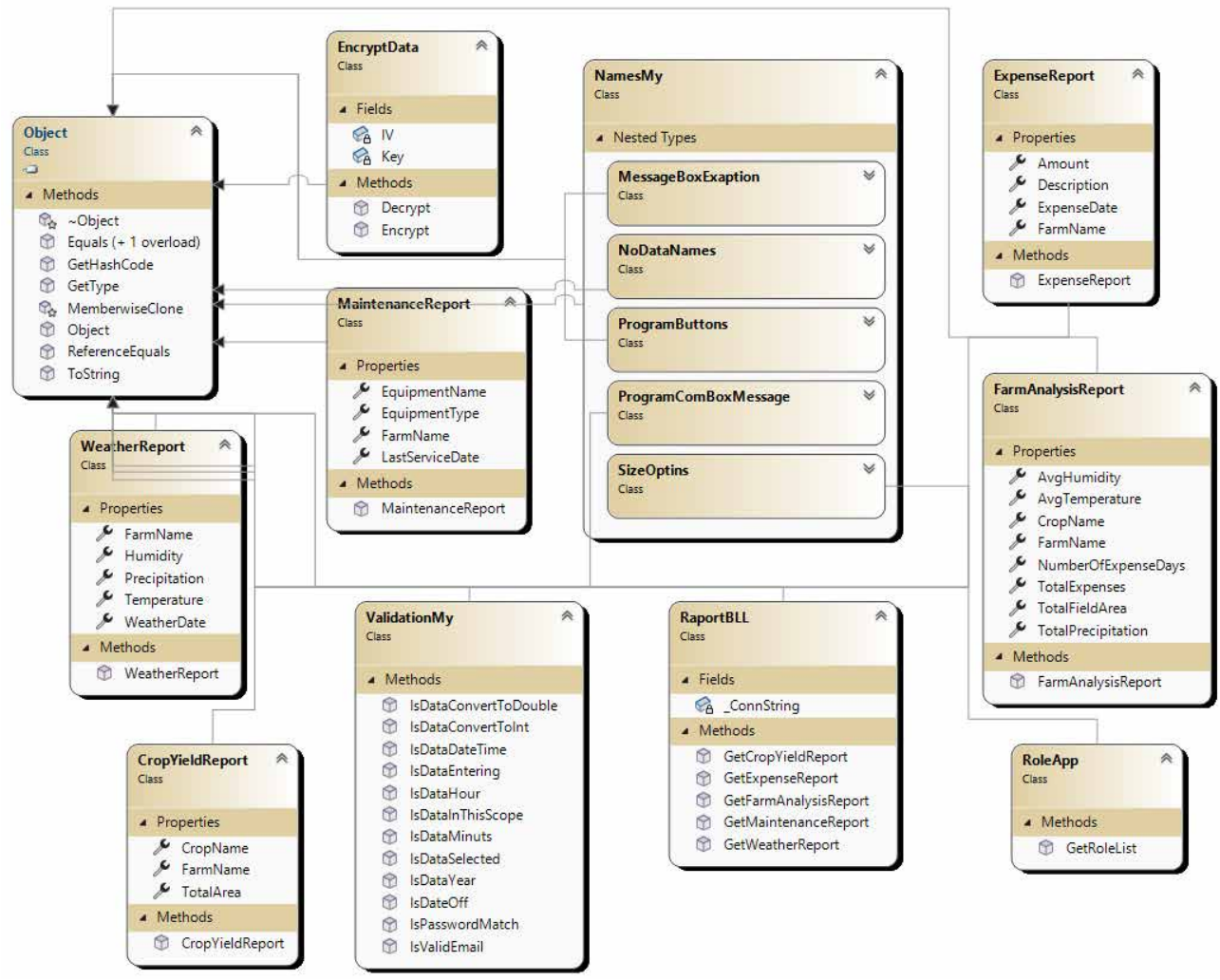

Рис. 2.4 Діаграма класів бізнес-логіки

Як відображено на рис. 2.4, діаграма класів цього рівня логіки складається із наступних класів:

– клас EncryptData відповідає за шифрування та розшифрування даних у системі, забезпечуючи захист конфіденційної інформації під час зберігання та передачі. Він містить методи для реалізації алгоритмів шифрування, що підвищує безпеку даних у системі;

– клас NamesMy виконує обробку та зберігання імен користувачів, а також може містити додаткові методи для форматування імен або створення унікальних ідентифікаторів. Цей клас забезпечує зручне управління даними про користувачів для інших компонентів системи;

– клас RoleApp відповідає за управління ролями користувачів у системі, надаючи доступ до певних функцій залежно від ролі. Він включає методи для

перевірки прав доступу та зміни ролей, забезпечуючи контроль за доступом до функціональності;

– клас ValidationMy забезпечує механізми для перевірки вхідних даних, що надходять у систему, допомагаючи уникнути помилок або небажаних дій. Клас включає методи для перевірки коректності введених значень та відповідності бізнес-правилам системи;

– клас RaportBLL відповідає за генерацію загальних звітів у системі, інтегруючи різні модулі для отримання та обробки даних. Він служить основою для створення звітів та їх подальшого експорту в різних форматах;

– клас CropYieldReport розраховує врожайність на полях, ґрунтуючись на даних про посіви та обсяг зібраного врожаю;

– клас WeatherReport збирає і обробляє дані про погодні умови на фермі за визначений період, включаючи температуру, вологість та опади;

– клас ExpenseReport призначений для обчислення загальних витрат на ферму протягом певного періоду, включаючи фінансові звіти та аналіз витрат;

– клас MaintenanceReport відстежує дані про обслуговування обладнання, включаючи дати останнього обслуговування та рекомендації щодо наступних заходів;

– клас FarmAnalysisReport проводить аналіз впливу витрат і погодних умов на врожайність для забезпечення комплексного підходу до оцінки результатів господарської діяльності.

Представлена структура класів забезпечує необхідний рівень абстракції та організації для ефективного функціонування рівня бізнес-логіки та підтримує різноманітні аналітичні і управлінські функції системи.

2.4.2 Особливості розробки інтерфейсу користувача. Рівень користувацького інтерфейсу розробленої системи забезпечує зручну взаємодію користувачів із програмним забезпеченням, дозволяючи легко керувати даними та виконувати необхідні операції. Інтерфейс організований у вигляді окремих форм для роботи з ключовими компонентами системи, що підвищує

ефективність та інтуїтивність роботи користувача. На рис. 2.3 представлена діаграма класів цього рівня, що відображає основні компоненти інтерфейсу та їх призначення.

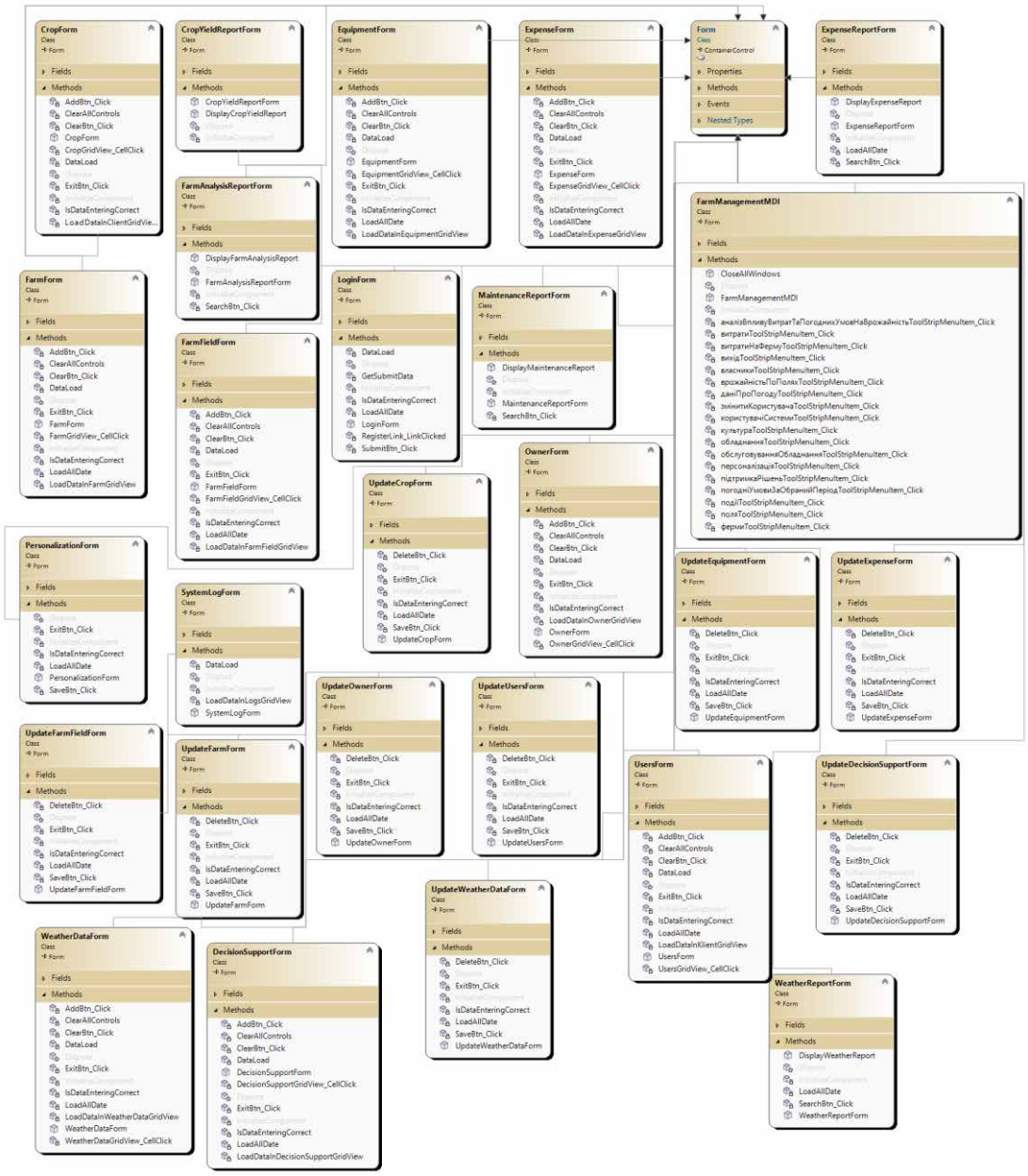

Рис. 2.5 Діаграма класів інтерфейсу системи

Даний рівень охоплює наступні класи:

– LoginForm забезпечує форму авторизації, де користувачі вводять свої облікові дані для доступу до системи. Він включає функції перевірки логіну та пароля, дозволяючи забезпечити безпеку доступу до системи;

– FarmManagementMDI – головне вікно системи, яке слугує стартовою точкою для навігації між основними функціями та формами інтерфейсу. Клас

організовує робоче середовище користувача, надаючи доступ до всіх модулів системи;

– CropForm відповідає за інтерфейс для перегляду та додавання нових фермерських культур. Він дозволяє користувачам вводити інформацію про культури та редагувати наявні записи;

– EquipmentForm – інтерфейс для управління даними про обладнання на фермі, включаючи його додавання, редагування та перегляд. Клас підтримує введення інформації про тип обладнання, його стан та дати обслуговування;

– FarmFieldForm надає користувачам можливість додавати нові фермерські поля або переглядати дані про існуючі. Він включає інформацію про розміри поля, культуру, яка вирощується, та інші характеристики;

– FarmForm забезпечує інтерфейс для управління даними про ферми, дозволяючи переглядати й редагувати інформацію про ферму, її розташування та інші основні параметри;

– OwnerForm – інтерфейс для введення та перегляду даних про власників ферми. Клас дозволяє вводити контактну інформацію власника, а також редагувати вже наявні дані;

– DecisionSupportForm відповідає за введення та перегляд рішень, що підтримують аграрну діяльність, допомагаючи у відстеженні прийнятих рекомендацій та їх виконання;

– ExpenseForm – інтерфейс для управління фінансовими витратами на фермі, що дозволяє додавати нові витрати, редагувати та контролювати витрати за певний період. Клас сприяє веденню фінансового обліку та аналізу витрат на основі введених даних;

– UpdateCropForm забезпечує інтерфейс для редагування або видалення інформації про сільськогосподарські культури. Користувач може змінювати дані про культуру, такі як її назва, сезон вирощування, а також видаляти записи за потреби;

– UpdateEquipmentForm – інтерфейс, призначений для оновлення та видалення записів про обладнання, що використовується на фермі. Клас дозволяє редагувати інформацію про тип обладнання, дату останнього обслуговування та його поточний стан;

– UpdateFarmFieldForm забезпечує можливість редагувати дані про фермерські поля, включаючи розмір, культуру та інші характеристики, або видаляти записи, якщо поле більше не використовується. Інтерфейс сприяє підтримці актуальності інформації про земельні ресурси;

– UpdateFarmForm надає інтерфейс для редагування та видалення записів про ферми, дозволяючи користувачам оновлювати інформацію про назву, розташування та власника ферми або видаляти записи при необхідності;

– UpdateOwnerForm забезпечує редагування та видалення записів про власників фермерських господарств. Інтерфейс дозволяє змінювати контактну інформацію або видаляти записи про власників;

– UpdateWeatherDataForm відповідає за редагування та видалення даних про погодні умови для певного періоду. Це дозволяє коригувати дані про температуру, вологість, опади, зберігаючи точність інформації;

– WeatherDataForm – інтерфейс для введення нових або перегляду наявних даних про погодні умови, що впливають на сільськогосподарські процеси. Клас включає можливості введення температури, вологості та кількості опадів для певного періоду;

– CropYieldReportForm відповідає за генерацію звіту про врожайність на полях, дозволяючи користувачам отримувати аналітичні дані щодо ефективності землекористування. Інтерфейс дозволяє переглядати і аналізувати врожайність по кожному полю;

– ExpenseReportForm – інтерфейс для створення звітів про витрати на ферму за визначений період. Клас забезпечує перегляд і аналіз витрат, дозволяючи здійснювати фінансовий контроль і планування бюджету;

– FarmAnalysisReportForm призначений для аналізу взаємозв'язку між витратами, погодними умовами та врожайністю. Інтерфейс надає можливість отримувати комплексні аналітичні дані, що підтримують прийняття рішень у фермерському господарстві;

– MaintenanceReportForm створює звіти про обслуговування обладнання, дозволяючи користувачам відстежувати останні дати обслуговування та планувати наступні. Інтерфейс сприяє підтримці ефективного функціонування обладнання;

– WeatherReportForm призначений для створення звітів про погодні умови на фермі за певний період. Інтерфейс дозволяє переглядати погодні дані, які впливають на аграрні процеси, що допомагає у плануванні робіт та аналізі умов для вирощування культур.

Описані класи формують функціональну частину інтерфейсу для управління даними та звітності, забезпечуючи зручний доступ до важливої інформації та підтримуючи ефективне прийняття рішень у фермерському господарстві.

2.4.3 Особливості роботи з рівнем доступу до даних. Рівень доступу до даних у розробленій системі відповідає за зберігання та управління інформацією в базі даних, забезпечуючи надійний зв'язок між бізнес-логікою та фізичним сховищем даних. Цей рівень дозволяє виконувати операції з додавання, оновлення, видалення та пошуку даних, необхідних для функціонування системи. Використання окремих класів для доступу до кожної категорії даних сприяє організованій і ефективній роботі з інформацією, підвищуючи модульність та гнучкість системи. На рис. 2.6 представлена діаграма класів рівня доступу до даних, що демонструє структуру та призначення кожного класу, відповідального за взаємодію з базою даних.

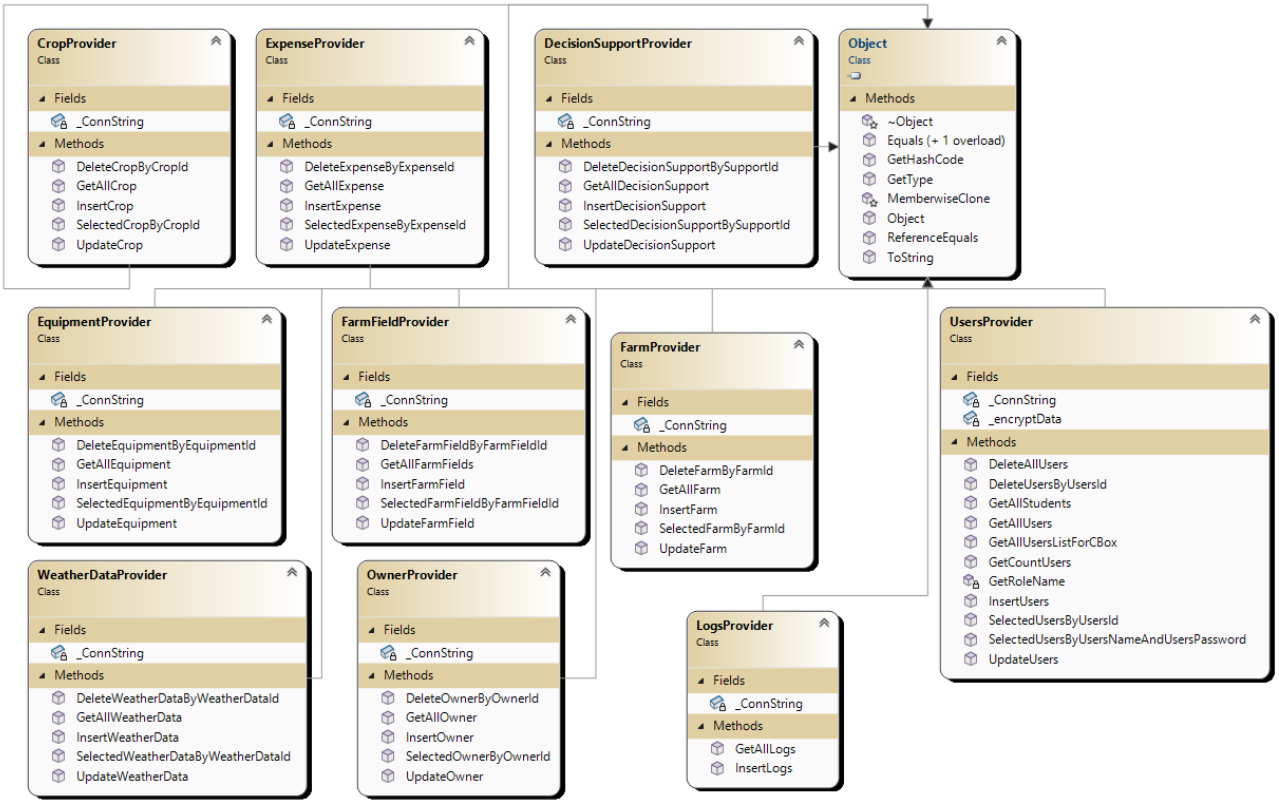

Рис. 2.6 Діаграма класів рівня даних

Діаграма класів рівня даних складається із:

– класу CropProvider, який відповідає за доступ до даних про сільськогосподарські культури. Він містить методи для додавання, оновлення, видалення та пошуку інформації про культури, що вирощуються на фермі;

– класу DecisionSupportProvider що, призначений для зберігання та отримання даних, що стосуються підтримки прийняття рішень. Він забезпечує доступ до рекомендацій та рішень, зберігаючи інформацію про дати і статус виконання;

– класу EquipmentProvider, який дозволяє взаємодіяти з інформацією про обладнання, яке використовується на фермі. Він включає методи для додавання, редагування та видалення даних про тип обладнання, стан і дати обслуговування;

– класу ExpenseProvider, що забезпечує доступ до даних про витрати ферми. Він підтримує операції для внесення нових витрат, редагування наявних та перегляд історії витрат, що важливо для фінансового обліку;

– класу FarmFieldProvider, який обробляє інформацію про фермерські поля, включаючи їх розмір, розташування та тип культури, що вирощується. Він дозволяє додавати та редагувати дані полів, а також виконувати запити для перегляду полів;

– класу FarmProvider, який відповідальний за доступ до інформації про ферми, зокрема їх назву, розташування та власника. Він підтримує операції для управління записами про ферми в базі даних.

– класу LogsProvider, що забезпечує доступ до журналу подій, зберігаючи інформацію про дії користувачів. Він дозволяє додавати нові записи до журналу, відстежуючи історію активності для моніторингу та аудиту;

– класу OwnerProvider, призначеного для роботи з даними про власників фермерських господарств. Він підтримує операції додавання, оновлення та видалення записів, забезпечуючи доступ до контактної інформації власників;

– класу UsersProvider, що відповідає за доступ до інформації про користувачів системи, зокрема їх ролі та облікові дані. Він підтримує функції для управління користувачами, включаючи перевірку даних при авторизації;

– класу WeatherDataProvider, що призначений для роботи з метеоданими, такими як температура, вологість та опади. Він забезпечує доступ до даних про погодні умови, що дозволяє аналізувати їхній вплив на агротехнічні процеси.

Класи рівня даних реалізують необхідні механізми для управління даними в базі та забезпечують надійний і ефективний доступ до інформації на рівні системи.

2.5 Висновок

У рамках даного розділу було проведено детальний аналіз і обґрунтування вибору технологій для розробки програмного забезпечення управління фермерським господарством, визначено вимоги до системи, спроєктовано базу даних та розроблено архітектуру. Для забезпечення

стабільності, продуктивності та сумісності з іншими компонентами системи обрано мову програмування C# як оптимальне рішення для роботи з .NET-платформою. В якості середовища розробки було проаналізовано кілька варіантів, із яких обрано Visual Studio 2022, що надає необхідний набір інструментів для продуктивної роботи та інтеграції з C#. Для зберігання даних було обрано MS SQL Server, оскільки ця система управління базами даних забезпечує надійне зберігання та доступ до інформації, що відповідає вимогам масштабованості й безпеки.

Визначено функціональні та нефункціональні вимоги до системи, що дозволило розробити діаграми сценаріїв використання для основних ролей – адміністратора та користувача. Це дало змогу чітко відобразити сценарії взаємодії користувачів із системою та визначити основні завдання, які система повинна виконувати. У розділі проєктування бази даних описані основні таблиці та взаємозв’язки між ними, а також побудована фізична модель даних, що забезпечує організацію зберігання інформації про ферми, обладнання, витрати, культури та інші ключові елементи системи.

Архітектура системи була розроблена на основі трирівневої моделі, що включає рівні представлення, бізнес-логіки та доступу до даних. На рівні бізнес-логіки було визначено та описано основні класи, які виконують функції обробки інформації та забезпечують роботу аналітичних модулів системи, таких як звітність про врожайність та аналіз витрат. Рівень інтерфейсу користувача був реалізований у вигляді класів, що забезпечують зручну взаємодію користувачів із системою через різні форми для перегляду, додавання, редагування даних. Рівень доступу до даних містить класи для управління даними в базі, що забезпечує надійний зв'язок з бізнес-логікою та підтримує виконання CRUD-операцій для основних таблиць.

Отримані результати розгляду та проєктування забезпечують підґрунтя для наступного етапу роботи – розробки, тестування та впровадження програмного забезпечення управління фермерським господарством, де всі спроєктовані компоненти будуть об’єднані у функціональну систему.

3 РОЗРОБКА, ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ УПРАВЛІННЯ ФЕРМЕРСЬКИМ ГОСПОДАРСТВОМ

3.1 Розробка програмних модулів системи підтримки рішень

Розробка програмних модулів для системи підтримки рішень спрямована на створення інтерфейсу та функціональних елементів, які дозволяють здійснювати аналіз даних, формувати рекомендації та оптимізувати управлінські рішення на основі аграрних показників. Зокрема, кожен модуль забезпечує автоматизацію окремих процесів, пов’язаних із моніторингом стану посівів, прогнозуванням врожайності та управлінням ресурсами. Взаємодія між модулями дозволяє ефективно інтегрувати інформацію з різних джерел та полегшити процес прийняття рішень.

Після створення бази даних необхідно здійснити її підключення до системи для забезпечення збереження та обробки інформації в реальному часі (рис. 3.1).

```
<appSettings>
  <add key="CONNECT"
       value="Data Source=(LocalDB)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\DB.mdf;
       Integrated Security=True" />
</appSettings>
```

Рис. 3.1 Під’єднання до бази даних

Цей фрагмент коду налаштовує конфігурацію підключення до бази даних у додатку на C#. Він використовує елемент <appSettings>, що дозволяє зберігати конфігураційні параметри, які можна легко змінювати без потреби перекомпіляції програми. Параметр key="CONNECT" визначає ім'я налаштування підключення, а value задає рядок підключення до локальної бази даних, що розміщена на сервері LocalDB. Рядок включає вказівку на використання AttachDbFilename, щоб підключити файл бази даних з іменем DB.mdf, який знаходиться в каталозі додатка. Параметр Integrated Security=True

дозволяє використовувати облікові дані поточного користувача Windows для автентифікації, що додає рівень безпеки.

На рис. 3.2 представлено головне вікно системи управління фермерським господарством, яке включає меню навігації для доступу до основних розділів програми. Користувач має можливість обирати серед таких розділів, як «Ферми», «Фермерські поля», «Власники», «Культура», «Дані про погоду» та «Обладнання», що спрощує роботу з інформацією та доступ до необхідних функцій.

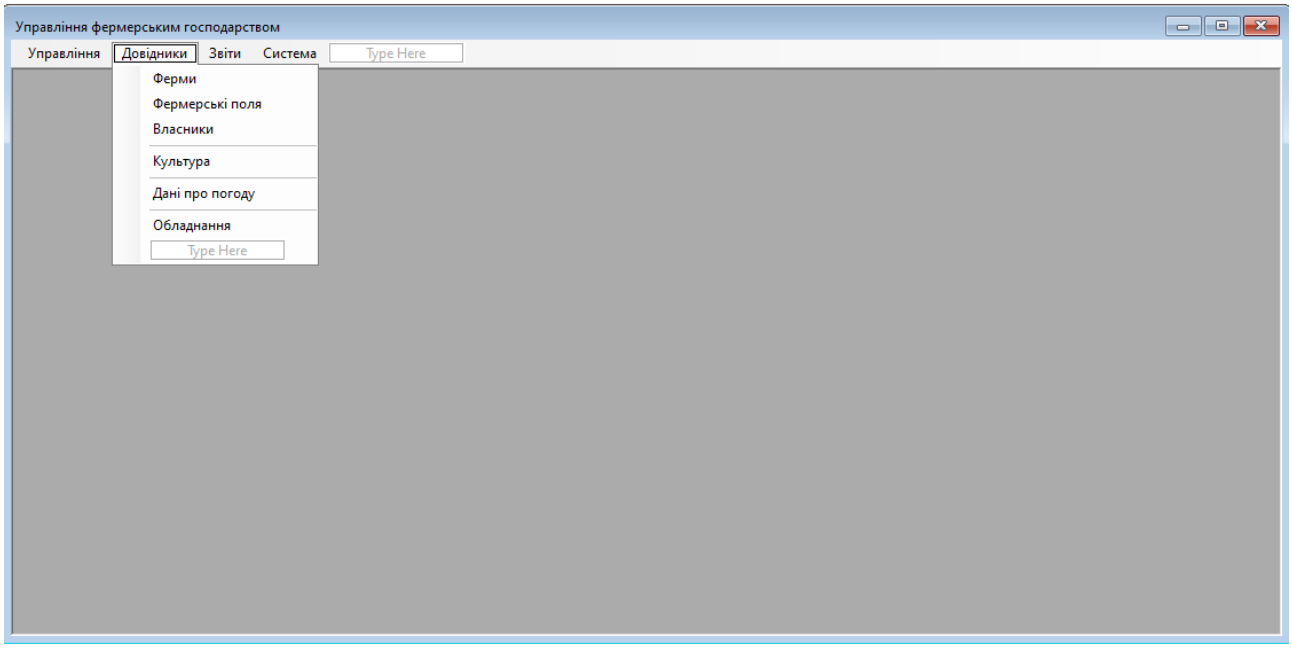

Рис. 3.2 Головне меню програми

Інтерфейс орієнтований на забезпечення інтуїтивної взаємодії з системою, дозволяючи швидкий доступ до ключових даних для управління фермерськими процесами.

Код на рис. 3.3 реалізує подію при натисканні на елемент меню «Ферми» у графічному інтерфейсі програми. При виклику цієї події спершу закриваються всі відкриті дочірні вікна за допомогою методу «CloseAllWindows», що забезпечує фокус на новому вікні. Потім створюється новий об'єкт форми FarmForm, який відображає інформацію про ферми. Ця форма призначається як дочірнє вікно (MdiParent = this) головної форми програми та встановлюється у максимізованому стані для повного

використання простору екрану. В кінці форма відображається, відкриваючи користувачу розділ для роботи з фермами.

```
private void фермиToolStripMenuItem_Click(object sender, EventArgs e) {
  CloseAllWindows();
  FarmForm farmForm = new FarmForm();
  farmForm.MdiParent = this;
  farmForm.WindowState = FormWindowState.Maximized;
  farmForm.Show();
}
```

Рис. 3.3 Програмування пунктів меню

Для взаємодії з базою даних у проекті було створено спеціалізовані класи, які відповідають за роботу з основними сутностями системи. Усі класи структуровані в окремій папці, що забезпечує організовану архітектуру проекту та полегшує доступ до них під час розробки і підтримки коду.

Код на рис. 3.4 реалізує метод «GetAllFarm», який отримує список усіх ферм з бази даних.

```
11 references
public List<Farm> GetAllFarm() {
  int num = 0;
  string SqlString = @"
    SELECT f.FarmId, f.FarmName, f.Location, f.Description, f.OwnerId,
           (o.FirstName + ' ' + o.LastName) AS FIO
    FROM Farm f
    LEFT JOIN Owner o ON f.OwnerId = o.OwnerId
    ORDER BY f.FarmName";
  List<Farm> listAllFarm = new List<Farm>();
  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
      conn.Open();
      using (SqlDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          Farm oneFarm = new Farm();
          oneFarm.Number = ++num;
          oneFarm.FarmId = Convert.ToInt32(reader["FarmId"]);
          oneFarm.FarmName = reader["FarmName"].ToString();
          oneFarm.Location = reader["Location"].ToString();
          oneFarm.Description = reader["Description"].ToString();
          oneFarm.OwnerId = Convert.ToInt32(reader["OwnerId"]);
          oneFarm.FIO = reader["FIO"].ToString();

          listAllFarm.Add(oneFarm);
        }
      }
      conn.Close();
    }
  }
  if (listAllFarm.Count == 0) {
    Farm noFarm = new Farm();
    noFarm.FarmId = 0;
    noFarm.Message = NamesMy.NoDataNames.NoDataInFarm;
    listAllFarm.Add(noFarm);
  }
  return listAllFarm;
}
```

Рис. 3.4 – Код методу «GetAllFarm»

На початку методу формується SQL-запит, який вибирає основні дані про ферми разом із ім'ям власника, об’єднуючи таблицю Farm з таблицею Owner. За допомогою класу SqlConnection здійснюється підключення до бази даних, і виконується SQL-запит через SqlCommand. Отримані результати зчитуються через SqlDataReader, і для кожного запису створюється об'єкт Farm, який додається до списку. У випадку, якщо даних немає, створюється об'єкт з повідомленням про відсутність даних. Метод повертає список всіх ферм або повідомлення, якщо дані відсутні.

Метод «InsertExpense» забезпечує додавання нового запису про витрати до бази даних (рис. 3.4). Він формує SQL-запит для вставки даних у таблицю Expense, де параметри Description, Amount, ExpenseDate і FarmId задаються як вхідні аргументи методу. За допомогою SqlConnection створюється з'єднання з базою, а SqlCommand виконує команду на вставку даних. Для уникнення SQL-ін'єкцій усі значення передаються як параметри запиту. Після відкриття з'єднання виконується команда ExecuteNonQuery, яка додає новий запис до таблиці, і з'єднання закривається, що забезпечує збереження даних у базі.

```
public void InsertExpense(string Description, double Amount, DateTime ExpenseDate, int FarmId) {
  string SqlString = "INSERT INTO Expense (Description, Amount, ExpenseDate, FarmId) " +
    "Values(@Description, @Amount, @ExpenseDate, @FarmId)";
  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("@Description", Description);
      cmd.Parameters.AddWithValue("@Amount", Amount);
      cmd.Parameters.AddWithValue("@ExpenseDate", ExpenseDate);
      cmd.Parameters.AddWithValue("@FarmId", FarmId);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рис. 3.5 Код методу «InsertExpense»

На рис. 3.6 зображено форму для введення даних про ферми, яка дозволяє користувачу додавати нові записи до системи. Форма включає поля для назви ферми, місцезнаходження, вибору власника та опису, що надає можливість деталізувати інформацію про кожну ферму. Додатково реалізовані кнопки «Додати», «Очистити» та «Вихід», що спрощують керування процесом введення і редагування даних.

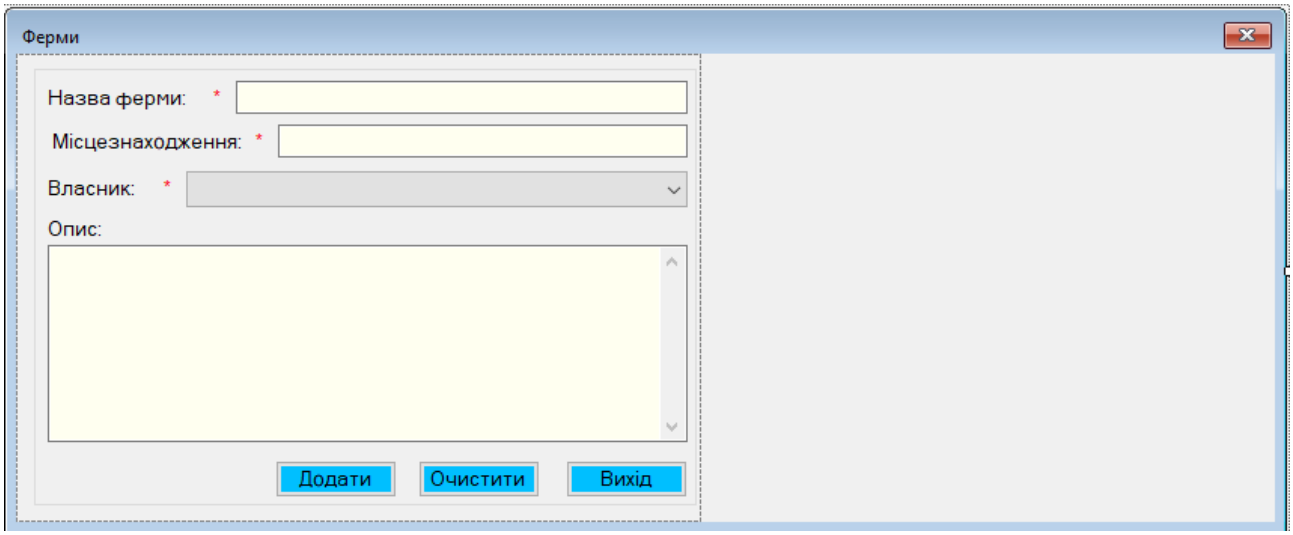

Рис. 3.6 Реалізація форми «Ферми»

Метод «AddBtn_Click» виконується при натисканні на кнопку «Додати» у формі, щоб додати новий запис про ферму (рис. 3.7). Спочатку виконується перевірка правильності введених даних за допомогою методу «IsDataEnteringCorrect». Якщо дані введені коректно, метод InsertFarm з класу _FarmProvider додає нову ферму до бази даних, використовуючи значення з полів введення. Після успішного додавання викликається метод «DataLoad» для оновлення даних у системі, а також «ClearAllControls», який очищує всі елементи введення, щоб підготувати форму до наступного вводу.

```
private void AddBtn_Click(object sender, EventArgs e) {
    if (IsDataEnteringCorrect()) {
        _FarmPrivider.InsertFarm(FarmNameTBox.Text,
            LocationTBox.Text, DescriptionTBox.Text,
            Convert.ToInt32(OwnerCBox.SelectedValue));
        DataLoad();
        ClearAllControls();
    }
}
```

Рис. 3.7 Метод для додавання інформації про ферму

Метод «FarmGridView_CellClick» запускається при натисканні на клітинку в таблиці FarmGridView (рис. 3.8). Якщо користувач клацнув на рядок, що містить дійсні дані про ферму (перевірка за індексом рядка та наявністю даних), зберігається індекс вибраного рядка у змінну _selectedRowIndex. Далі створюється екземпляр форми UpdateFarmForm, яка відкривається в режимі діалогу для редагування даних обраної ферми, і передається її унікальний

ідентифікатор. Після завершення редагування оновлюються дані у таблиці за допомогою методу DataLoad().

```
private void FarmGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
  if (e.RowIndex >= 0 && FarmGridView[0, e.RowIndex].Value.ToString()
    != _FarmList[0].Message) {
    _selectedRowIndex = e.RowIndex;
    UpdateFarmForm updateFarmForm =
      new UpdateFarmForm(Convert.ToInt32(FarmGridView[0, e.RowIndex].Value.ToString()));
    updateFarmForm.ShowDialog();
    DataLoad();
  }
}
```

Рис. 3.8 Код методу «FarmGridView_CellClick»

Клас RaportBLL був розроблений для обробки та генерації звітів у системі, що забезпечує логіку бізнес-рівня для роботи зі звітними даними. Він об'єднує методи, які формують різноманітні звіти, наприклад метод «GetCropYieldReport» формує і повертає список звітів про врожайність для кожної ферми (рис. 3.9).

```
public List<CropYieldReport> GetCropYieldReport() {
  var cropYieldReports = new List<CropYieldReport>();
  string query = @"
    SELECT f.FarmName AS FarmName, cr.CropName AS CropName, SUM(ff.FieldSize) AS TotalArea
    FROM Farm f
    JOIN FarmField ff ON f.FarmId = ff.FarmId
    JOIN Crop cr ON ff.CropId = cr.CropId
    GROUP BY f.FarmName, cr.CropName
    ORDER BY f.FarmName, cr.CropName";
  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    SqlCommand command = new SqlCommand(query, conn);
    conn.Open();

    using (SqlDataReader reader = command.ExecuteReader()) {
      while (reader.Read()) {
        string farmName = reader["FarmName"].ToString();
        string cropName = reader["CropName"].ToString();
        double totalArea = Convert.ToDouble(reader["TotalArea"]);

        cropYieldReports.Add(new CropYieldReport(farmName, cropName, totalArea));
      }
    }
  }
  return cropYieldReports;
}
```

Рис. 3.9 Код методу «GetCropYieldReport»

На початку створюється пустий список cropYieldReports, який буде заповнений даними про врожайність. SQL-запит отримує назви ферм і культур, а також обчислює загальну площу полів, на яких вирощується кожна культура, групуючи результати за назвою ферми та культури. Після встановлення

підключення до бази даних виконується цей запит, і отримані результати зчитуються за допомогою SqlDataReader. Для кожного запису створюється новий об'єкт CropYieldReport, який додається до списку, забезпечуючи структуроване представлення даних про врожайність.

Для отримання погодних даних для певної ферми у визначеному діапазоні дат реалізовано SQL-запит, який представлено на рис. 3.10.

```
string query = @"
  SELECT f.FarmName AS FarmName, wd.WeatherDate,
    wd.Temperature, wd.Humidity, wd.Precipitation
  FROM WeatherData wd
  JOIN Farm f ON wd.FarmId = f.FarmId
  WHERE wd.WeatherDate BETWEEN @StartDate AND @EndDate AND wd.FarmId = @FarmId
  ORDER BY f.FarmName, wd.WeatherDate";
```

Рис. 3.10 SQL-запит отримання погодних даних

Запит об'єднує таблиці WeatherData і Farm, щоб отримати назву ферми, дату, температуру, вологість і кількість опадів для обраної ферми. Параметри @StartDate, @EndDate та @FarmId забезпечують гнучкість у виборі діапазону дат і конкретної ферми, а сортування за назвою ферми і датою дозволяє отримати впорядковані результати для аналізу погодних умов у зазначений період.

Також реалізовано запит аналізу витрати для конкретної ферми в обраному часовому проміжку (рис. 3.11).

```
string query = @"
      SELECT f.FarmName AS FarmName, e.ExpenseDate, e.Description, e.Amount
      FROM Expense e
      JOIN Farm f ON e.FarmId = f.FarmId
      WHERE e.ExpenseDate BETWEEN @StartDate AND @EndDate AND e.FarmId = @FarmId
      ORDER BY e.ExpenseDate";
```

Рис. 3.11 Аналіз витрат ферми у заданому діапазоні

Він об'єднує таблицю Expense з таблицею Farm, щоб відобразити назву ферми, дату витрати, її опис та суму. Використання параметрів @StartDate, @EndDate і @FarmId дозволяє гнучко вибирати витрати для конкретної ферми та діапазону дат. Сортування результатів за датою витрати допомагає побачити хронологію витрат, що полегшує аналіз фінансових даних ферми.

3.2 Результати функціонального та модульного тестування

Тестування програмного забезпечення дозволяє виявити можливі помилки та перевірити відповідність програмного продукту функціональним і нефункціональним вимогам, забезпечуючи тим самим його якість та відповідність очікуванням користувача. Для даної системи тестування було зосереджене на оцінці точності обробки даних, перевірці коректності виконання операцій над базовими компонентами, а також стійкості системи до навантажень і виняткових ситуацій.

Тестування форми є важливим етапом для перевірки коректності функціонування системи управління фермерським господарством, оскільки воно дозволяє забезпечити точність та надійність введення та обробки інформації про ферми. Наприклад у табл. 3.1 описано тестові сценарії охоплюють перевірку основних елементів інтерфейсу та операцій із введенням даних, включаючи перевірку обов’язкових полів, коректну роботу кнопки «Очистити» та інших ключових функцій.

Таблиця 3.1

Тестові сценарії для форми «Ферми»

№ з/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка проходження
1	Ввести всі дані, натиснути «Додати»	Ферму успішно додано до бази даних	Ферму додано	Так
2	Не ввести значення обов'язкових полів (назва ферми, місцезнаходження)	Повідомлення про помилку, дані не збережено	Повідомлення виведено, дані не додано	Так
3	Ввести неправильний формат даних у полі (наприклад, власник)	Повідомлення про некоректний формат введення	Повідомлення виведено	Так
4	Натиснути «Очистити» після введення даних	Усі поля очищені	Поля очищені	Так
5	Ввести тільки обов'язкові поля та натиснути «Додати»	Ферму успішно додано до бази	Ферму додано	Так

На рис. 3.12 наведено приклад форми «Ферми», яка демонструє проведення тестових сценаріїв.

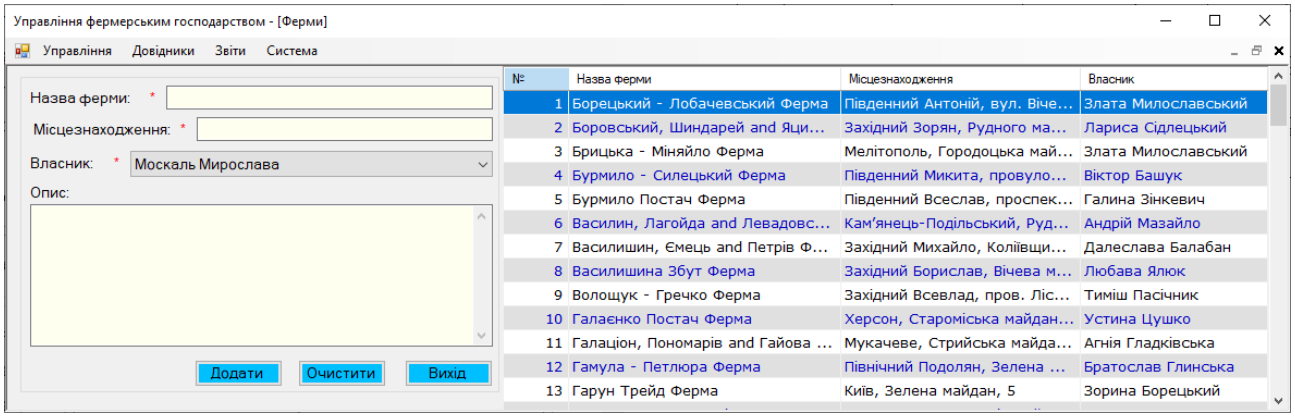

Рис. 3.12 Приклад тестування форми «Ферми»

Тестування форми «Власники» є критичним для перевірки коректності введення і зберігання контактної інформації власників, що забезпечує цілісність даних та їх відповідність вимогам системи. Тестові сценарії у табл. 3.13 зосереджені на перевірці обов'язкових полів, таких як прізвище, ім'я та номер телефону, а також на перевірці коректного введення та валідації формату телефонного номера, можливості очищення форми перед введенням нових даних та інших важливих аспектів.

Таблиця 3.2

Тестові сценарії для форми «Власники»

№ з/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка проходження
1	Ввести всі дані, натиснути «Додати»	Власника успішно додано до бази даних	Власника додано	Так
2	Не ввести значення обов'язкових полів (прізвище, ім'я, № тел.)	Повідомлення про помилку, дані не збережено	Повідомлення виведено, дані не додано	Так
3	Ввести неправильний формат номеру телефону	Повідомлення про некоректний формат номера	Повідомлення виведено	Так
4	Натиснути «Очистити» після введення даних	Усі поля очищені	Поля очищені	Так
5	Ввести тільки обов'язкові поля та натиснути «Додати»	Власника успішно додано до бази	Власника додано	Так

На рис. 3.13 представлено форму «Власники», яка ілюструє процес тестування сценаріїв для додавання нового власника, перевірку обов'язкових полів, та ін. відповідно до вимог системи.

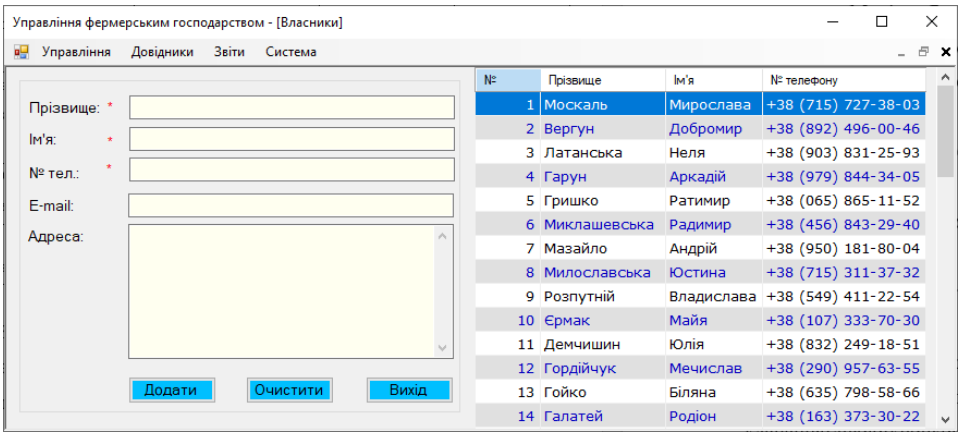

Рис. 3.13 Приклад тестування форми «Власники»

Тестування форми «Витрати» є необхідним для забезпечення правильного введення та зберігання фінансових даних, пов'язаних з управлінням фермами (табл. 3.3).

Таблиця 3.3

Тестові сценарії для форми «Витрати»

№ з/п	Крок сценарію	Очікуваний результат	Отриманий результат	Відмітка проходження
1	Ввести всі дані, натиснути «Додати»	Запис про витрати успішно додано до бази	Витрати додано	Так
2	Не ввести значення обов'язкових полів (ферма, сума витрат)	Повідомлення про помилку, дані не збережено	Повідомлення виведено, дані не додано	Так
3	Ввести неправильний формат дати	Повідомлення про некоректний формат дати	Повідомлення виведено	Так
4	Натиснути «Очистити» після введення даних	Усі поля очищені	Поля очищені	Так
5	Ввести тільки обов'язкові поля та натиснути «Додати»	Запис про витрати успішно додано до бази	Витрати додано	Так

На рис. 3.14 представлено форму «Витрати», яка ілюструє тестування сценаріїв для додавання нового запису про витрати, перевірки полів, валідації формату даних та очищення введених значень відповідно до вимог системи.

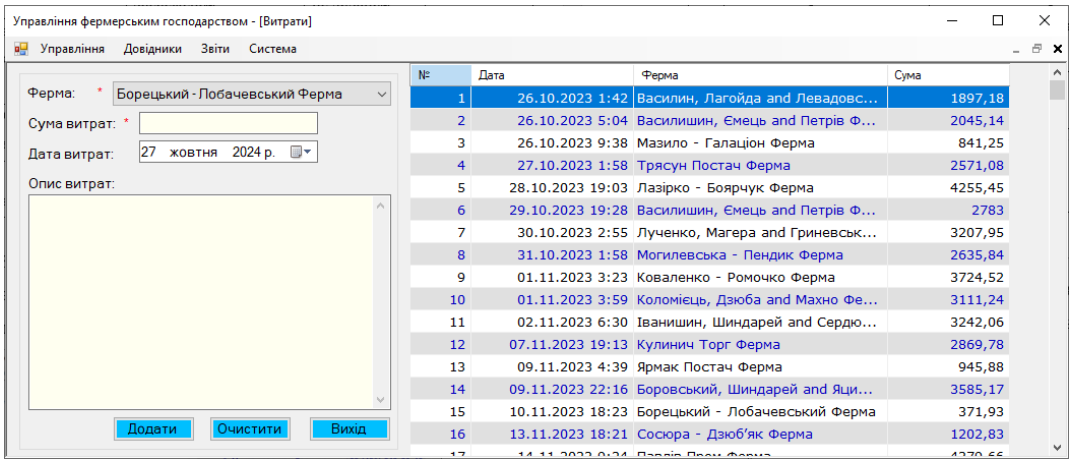

Рис. 3.14 Приклад тестування форми «Витрати»

Тестування з використанням MS Test було застосоване для перевірки коректної роботи інформаційної системи «Управління фермерським господарством». MS Test дозволив реалізувати автоматизоване тестування основних функціональних модулів системи, зокрема додавання, редагування та видалення даних про ферми, витрати та власників. Використання цього інструменту забезпечило стандартизацію процесу тестування та підвищило точність перевірки програмного забезпечення на предмет помилок. Результати успішного проходження тестів, що підтверджують відповідність системи вимогам, наведено на рис. 3.15.

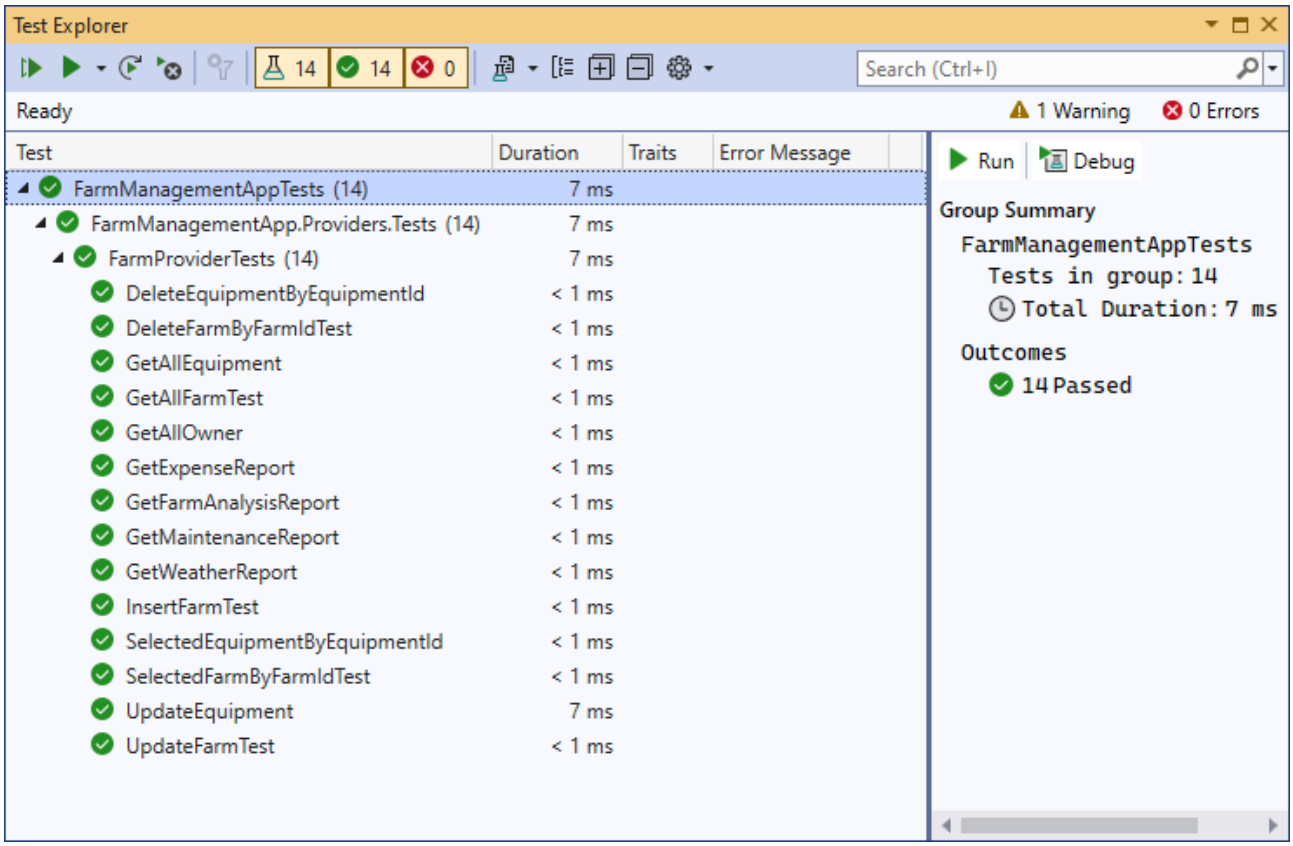

Рис. 3.15 Результати модульного тестування

Комбінація автоматизованого тестування за допомогою MS Test та ручних тестових сценаріїв забезпечила комплексну перевірку функціональності системи. Це дозволило виявити потенційні помилки, підвищити надійність роботи та впевнитися у відповідності програми бізнес-вимогам.

3.3 Інструкція для користувача

Інструкція для користувача призначена для полегшення процесу роботи з інформаційною системою «Управління фермерським господарством» та забезпечення її ефективного використання. У цьому підрозділі описано основні функції системи, наведено покрокові рекомендації щодо виконання ключових операцій.

3.3.1 Мінімальні вимоги для запуску програмного забезпечення. Для успішного запуску та стабільної роботи програмного забезпечення «Управління

фермерським господарством» важливо врахувати мінімальні вимоги до апаратного та програмного забезпечення, що дозволить забезпечити належну продуктивність та функціональність системи.

Мінімальні апаратні вимоги для запуску системи:

– процесор з частотою не менше 2.0 ГГц (рекомендовано Intel Core i3 або AMD Ryzen 3);

– оперативна пам'ять: щонайменше 4 ГБ;

– вільний дисковий простір: не менше 500 МБ;

– наявність стандартних периферійних пристроїв, таких як миша, клавіатура та монітор.

Програмні вимоги:

– операційна система Windows 10 або новіша версія;

– встановлений .NET Framework версії 4.8 або вище для забезпечення сумісності з усіма функціями додатку.

Для встановлення програмного забезпечення користувачу необхідно створити папку на жорсткому диску, наприклад, «FarmManagement». Після цього потрібно скопіювати файли з каталогу «Debug» у цю папку. Для запуску програми достатньо двічі клацнути на виконуваному файлі, що забезпечить доступ до інтерфейсу програми.

3.3.2 Опис процесу розгортання програмного продукту. Для успішного розгортання програмного продукту «Управління фермерським господарством» рекомендується виконати наступні кроки:

– переконатись, що на комп’ютері встановлено актуальну версію .NET Framework (рекомендовано версію 4.8 або вище), яка забезпечить коректну роботу додатку, розробленого на C#;

– встановити MS SQL Server та налаштувати його для роботи з базою даних додатку. Після завершення встановлення запустити сервер та переконатись, що він активний і доступний для підключення;

– розгорнути базу даних, створивши нову базу за допомогою SQL Server Management Studio (SSMS). Виконати необхідні SQL-скрипти для створення таблиць, індексів, які надаються разом з програмним продуктом. Параметри бази даних, такі як ім’я сервера, ім’я бази та облікові дані, знадобляться для подальшої конфігурації;

– скопіювати файли програми до обраної папки на комп’ютері, наприклад, «FarmManagementSystem». Запустити програмний продукт, двічі клацнувши на виконуваному файлі (.exe), щоб перевірити доступ до основного інтерфейсу та почати взаємодію із системою;

– налаштувати підключення до бази даних у конфігураційному файлі програми (app.config), де потрібно вказати рядок підключення із зазначенням адреси сервера, назви бази, імені користувача та пароля. Це забезпечить коректну інтеграцію додатку з базою даних;

– провести тестування, запустивши програму та перевіривши її функціональність: зокрема, можливість додавання, редагування та видалення записів, а також взаємодію з базою даних. Переконатисяь що всі основні функції працюють коректно та додаток готовий до експлуатації.

Після виконання цих кроків додаток готовий до використання та може бути переданий у розпорядження кінцевих користувачів для повноцінної роботи.

3.3.3 Використання програмного продукту для управління фермерським господарством.

Для початку роботи з додатком «Управління фермерським господарством» необхідно запустити програму, після чого користувачу буде представлено вікно авторизації. У цьому вікні потрібно ввести ім'я користувача та пароль, що забезпечує захист даних від несанкціонованого доступу та гарантує, що лише авторизовані особи зможуть керувати інформацією про ферми (рис. 3.16).

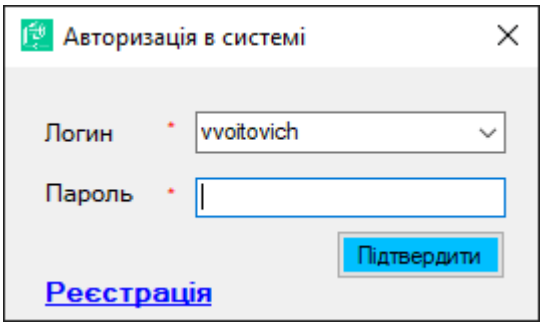

Рис. 3.16 Авторизація користувача в системі

Якщо введені дані будуть некоректними або порожніми, система попередить користувача про помилку, відобразивши відповідне повідомлення, що допоможе уникнути помилкового доступу та забезпечити безпеку облікових записів (рис. 3.17). Це дозволяє зменшити ризик помилок під час авторизації, зберігаючи цілісність даних та захист інформації.

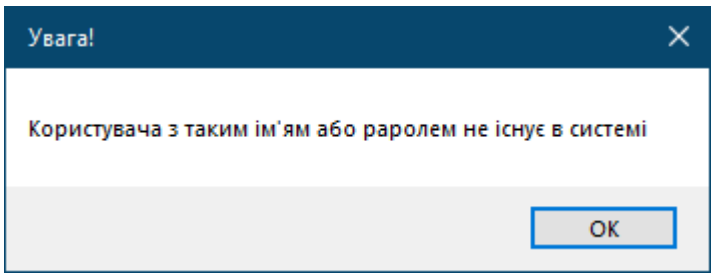

Рис. 3.17 Повідомлення про некоректні дані для авторизації

Після успішної авторизації система перенаправляє користувача до головного вікна програми, яке містить основне меню з доступом до ключових функцій. У головному вікні користувач може обрати необхідні розділи для управління фермерським господарством, таких як ведення обліку витрат, управління власниками, додавання нових ферм і культур, а також перегляд звітів про діяльність господарства (рис. 3.18).

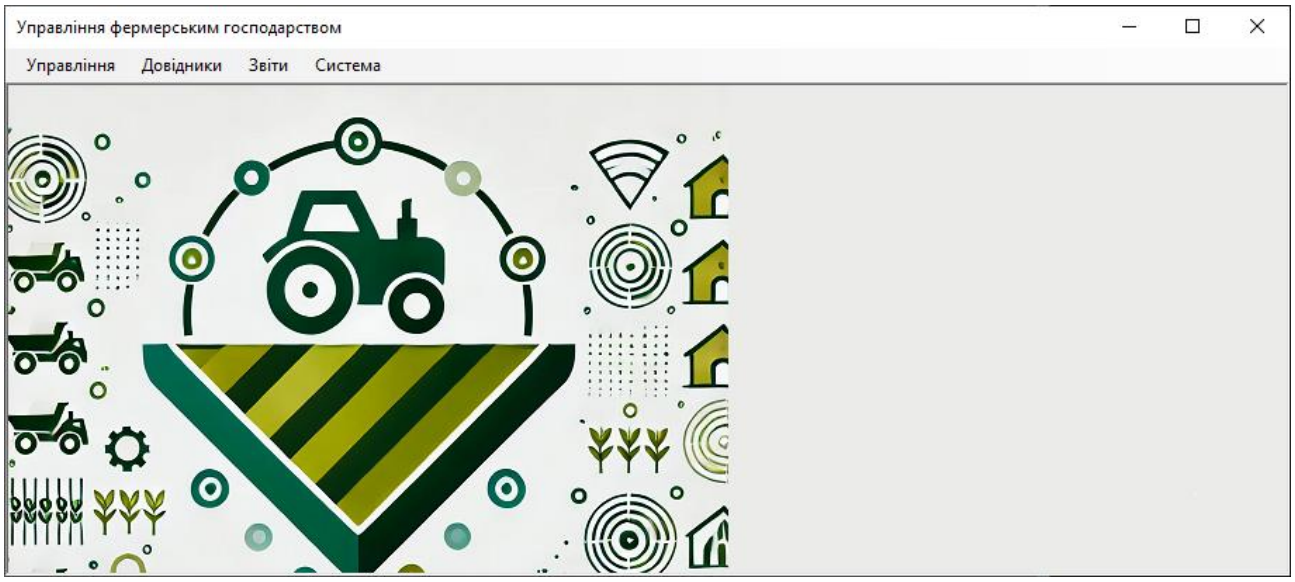

Рис. 3.18 Головне меню програми

Форма для опрацювання даних про ферми, представлена на рис. 3.19, надає користувачеві можливість додавати, редагувати та переглядати інформацію про фермерські господарства. У верхній частині форми розташовані поля для введення основних даних про ферму, таких як назва ферми, місцезнаходження, вибір власника з випадаючого списку та поле для опису. Поля «Назва ферми», «Місцезнаходження» та «Власник» позначені як обов’язкові для заповнення, що забезпечує мінімальну вимогу до внесення інформації. Кнопки «Додати», «Очистити» та «Вихід» розташовані внизу форми, що полегшує управління процесом внесення даних.

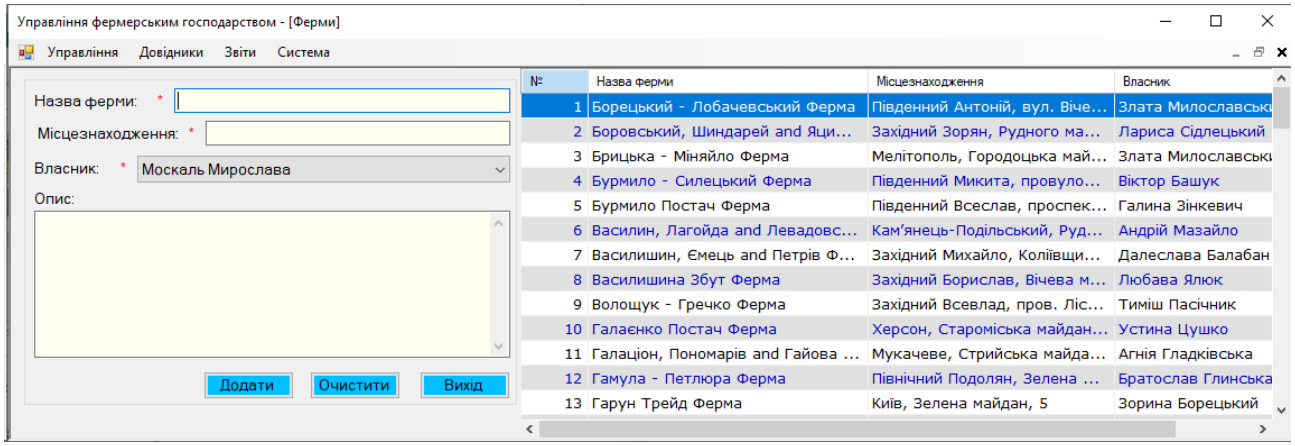

Рис. 3.19 Опрацювання даних «Ферм»

Праворуч від форми представлена таблиця з уже існуючими записами, яка відображає перелік ферм із зазначенням їхніх назв, місцезнаходження та

імен власників. Це дозволяє користувачу зручно переглядати та шукати необхідну інформацію. Для редагування інформації достатньо обрати запис у таблиці та внести зміни у відповідні поля, після чого запис буде оновлено в базі даних.

При натисканні на запис у таблиці з даними про ферми відкривається форма редагування (рис. 3.20), яка дозволяє користувачу вносити зміни до інформації про обрану ферму. У цій формі відображаються всі основні поля для редагування: назва ферми, місцезнаходження, власник та опис.

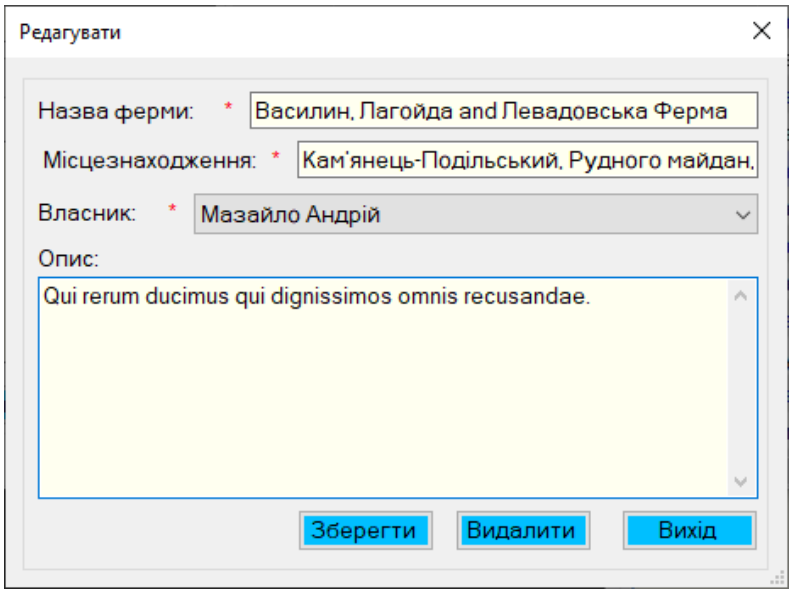

Рис. 3.20 Вікно для редагування даних

Користувач має можливість внести корективи у вже наявні дані або доповнити опис ферми. Після внесення змін необхідно натиснути кнопку «Зберегти», щоб оновити інформацію у базі даних. Крім того, форма надає можливість видалити обрану ферму з бази даних, натиснувши кнопку «Видалити», що дозволяє підтримувати актуальність записів. Кнопка «Вихід» закриває форму редагування без збереження змін, що зручно для відмови від редагування в будь-який момент.

Форма для опрацювання даних про фермерські поля, представлена на рис. 3.21, забезпечує користувачу можливість додавання, редагування та перегляду інформації про земельні ділянки, що належать фермам. У верхній частині форми розташовані поля для введення основних даних: назви поля, його

розміру, вибору ферми, до якої воно належить, культури, що вирощується, та опису. Поля «Назва поля», «Розмір поля» та «Ферма» є обов'язковими, що гарантує внесення найважливішої інформації для кожної ділянки.

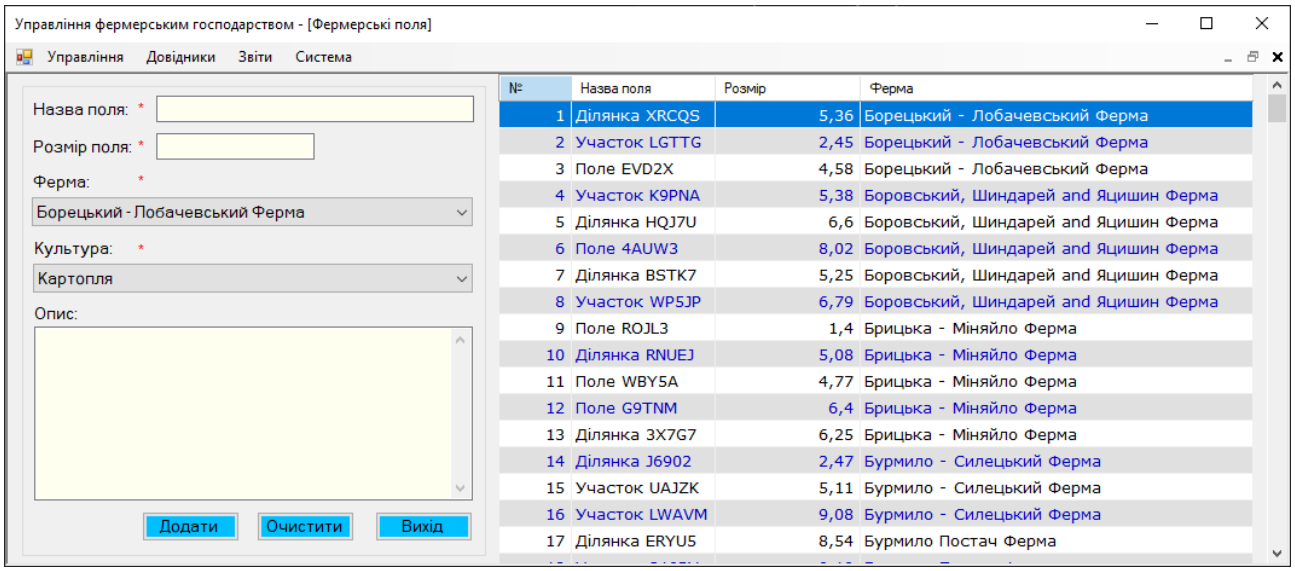

Рис. 3.21 Форма для опрацювання даних фермерських полів

Користувач може вводити інформацію про розмір ділянки у відповідному полі, що забезпечує точність обліку площі кожного поля. У випадаючих списках можна обрати ферму та культуру, що спрощує процес заповнення форми і гарантує правильну категоризацію інформації. Кнопки «Додати», «Очистити» та «Вихід» у нижній частині форми дозволяють зручно керувати даними — додавати нові записи, очищати форму для наступного введення даних або виходити з форми.

Праворуч у формі відображено таблицю зі списком вже наявних полів із зазначенням їхніх назв, розмірів та відповідних ферм. Це дозволяє швидко переглядати і знаходити необхідні записи для аналізу або редагування. Натискання на конкретний запис у таблиці відкриває можливість його редагування, що дозволяє підтримувати актуальність і точність даних у системі.

Форма для опрацювання даних про власників ферм, зображена на рис. 3.22, надає користувачу зручний інструмент для введення, редагування та перегляду інформації про осіб, які володіють фермерськими господарствами. На формі передбачені поля для заповнення основних даних про власника, зокрема прізвища, імені та номера телефону, які є обов'язковими для

заповнення. Це забезпечує надійну ідентифікацію кожного власника та мінімізує можливість дублювання даних. Додатково користувач може внести електронну пошту та адресу власника, що розширює доступну інформацію та полегшує подальшу комунікацію.

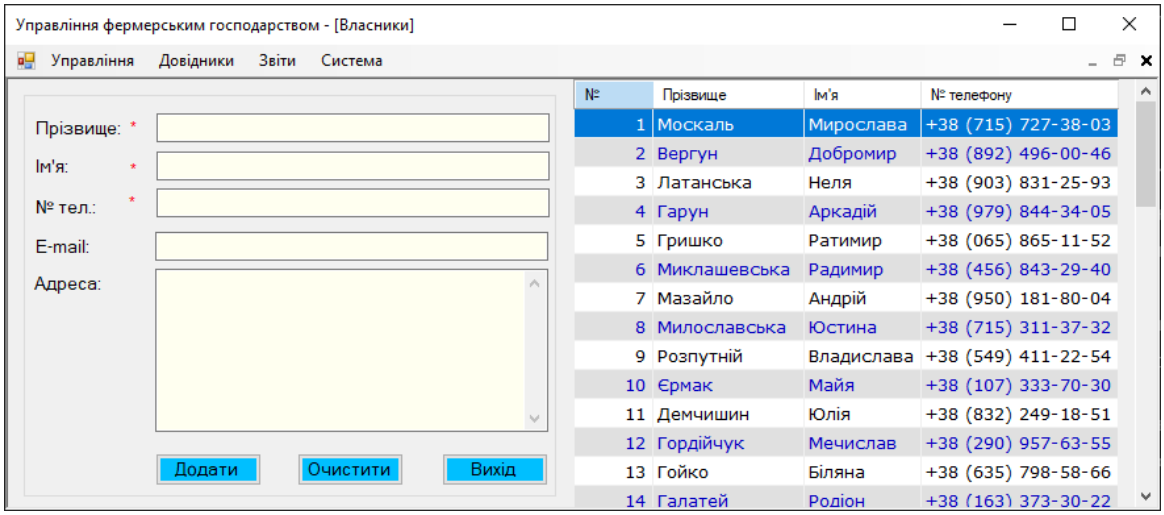

Рис. 3.22 Форма для опрацювання даних власників ферм

Форма для опрацювання даних про витрати, представлена на рис. 3.23, призначена для обліку та управління фінансовими даними, пов’язаними з діяльністю фермерських господарств. У верхній частині форми користувач може ввести основні параметри витрат: вибрати ферму зі списку, вказати суму витрат, дату транзакції та, за потреби, додати детальний опис витрат. Поля «Ферма», «Сума витрат» і «Дата витрат» є обов'язковими, що забезпечує мінімальну кількість необхідної інформації для зберігання кожного запису.

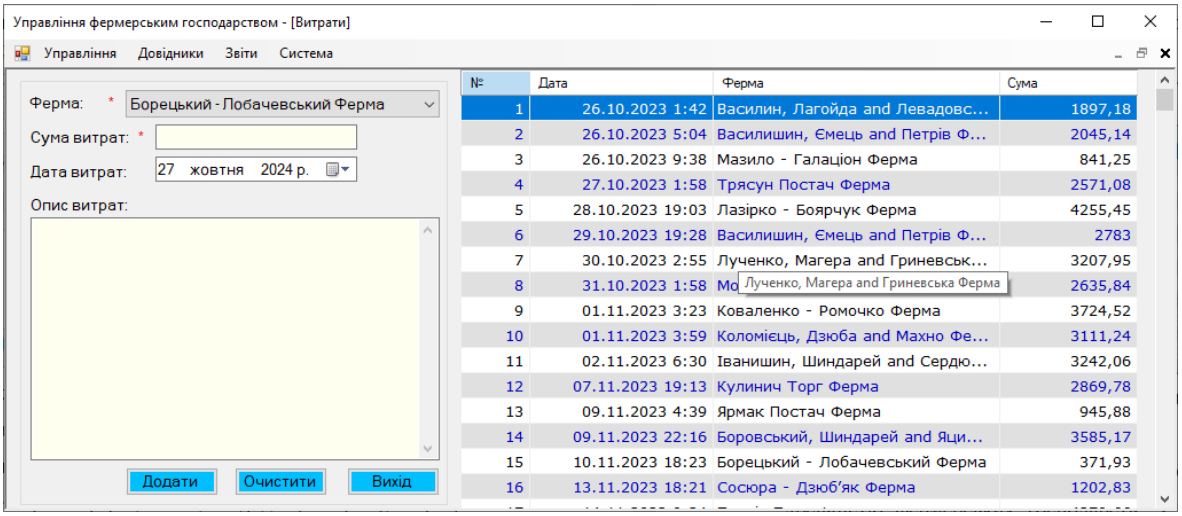

Рис. 3.23 Форма для опрацювання даних витрат

При переході до розділу «Звіти» та виборі пункту «Врожайність по полях» користувачеві надається звіт про врожайність різних культур на полях (рис. 3.24). Цей звіт містить детальну інформацію про кожне фермерське господарство, включаючи назву ферми, культуру, яка вирощується, та загальну площу поля, зайнятого під цю культуру. Така структура даних дозволяє швидко отримати зведення про обсяги посівних площ для кожної культури на різних фермах.

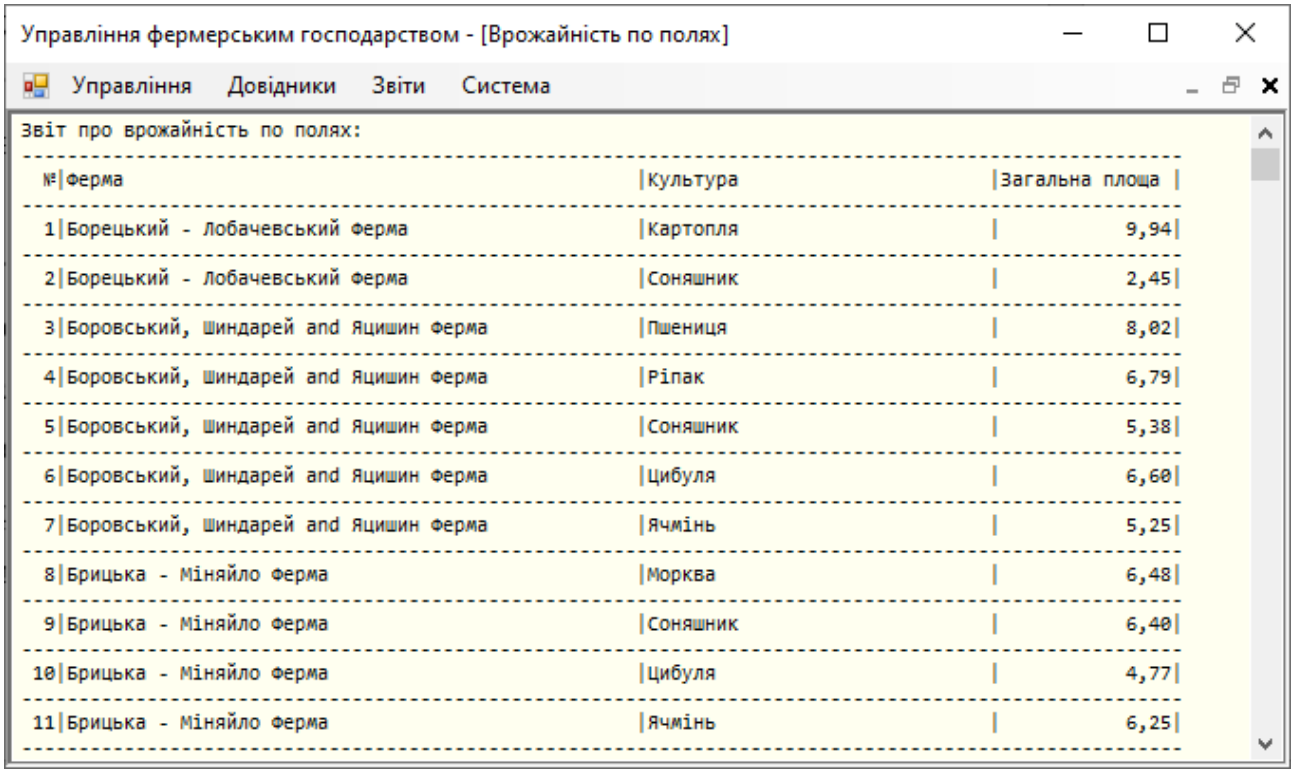

Рис. 3.24 Звітність по врожайності на полях

Такий звіт допомагає приймати обґрунтовані рішення щодо розподілу площ під посіви, оптимізації виробництва та управління ресурсами. Він також може бути використаний для планування майбутніх посівних кампаній або для аналітичного огляду врожайності на рівні окремих ферм і культур.

При виборі пункту звіту «Погодні умови за обраний період» користувач може отримати зведення про погодні показники для конкретної ферми у вказаному часовому проміжку (рис. 3.25). У верхній частині форми надаються елементи для вибору ферми зі списку, а також поля для встановлення початку і кінця періоду, за який буде сформовано звіт. Після введення необхідних параметрів і натискання кнопки «Формувати», система відображає дані про

погодні умови, що включають температуру, вологість та кількість опадів за кожну дату у вибраному діапазоні.

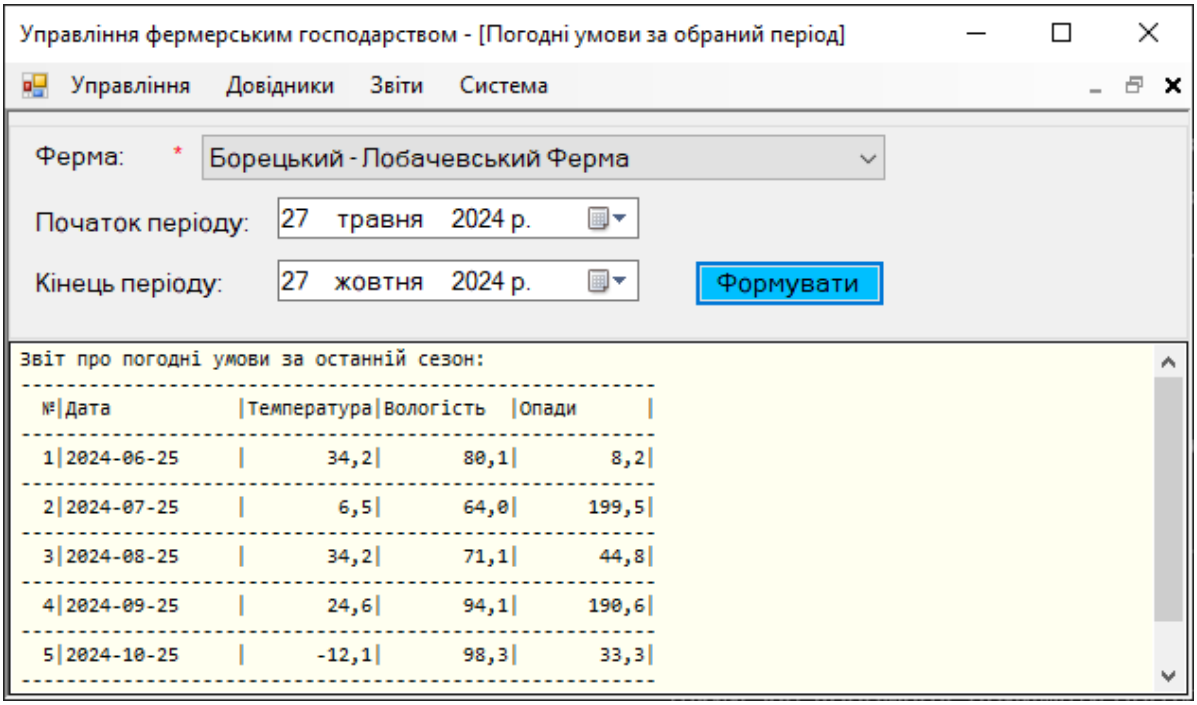

Рис. 3.25 Звітність по погодні умови

Звіт про погодні умови є важливим інструментом для управління фермерським господарством, оскільки дозволяє отримати повне уявлення про кліматичні умови, що впливали на виробництво, і сприяє прийняттю обґрунтованих рішень у подальшій діяльності.

При виборі пункту звіту «Витрати на ферму» користувач може отримати зведення про фінансові витрати, пов’язані з діяльністю конкретного фермерського господарства за обраний період (рис. 3.26). У верхній частині форми розташовані елементи для вибору ферми зі списку та поля для встановлення початкової і кінцевої дати звітного періоду. Після вибору ферми і часових меж користувач натискає кнопку «Формувати», що ініціює процес генерації звіту.

Сформований звіт відображається у табличному вигляді з колонками для дати витрат і суми. Це дозволяє легко аналізувати динаміку фінансових витрат на ферму протягом обраного періоду, надаючи користувачу можливість виявити, у які дати відбувалися найбільші витрати та оцінити загальний рівень

витрат. Така інформація є корисною для фінансового планування, контролю бюджету та оптимізації витрат у подальшій діяльності.

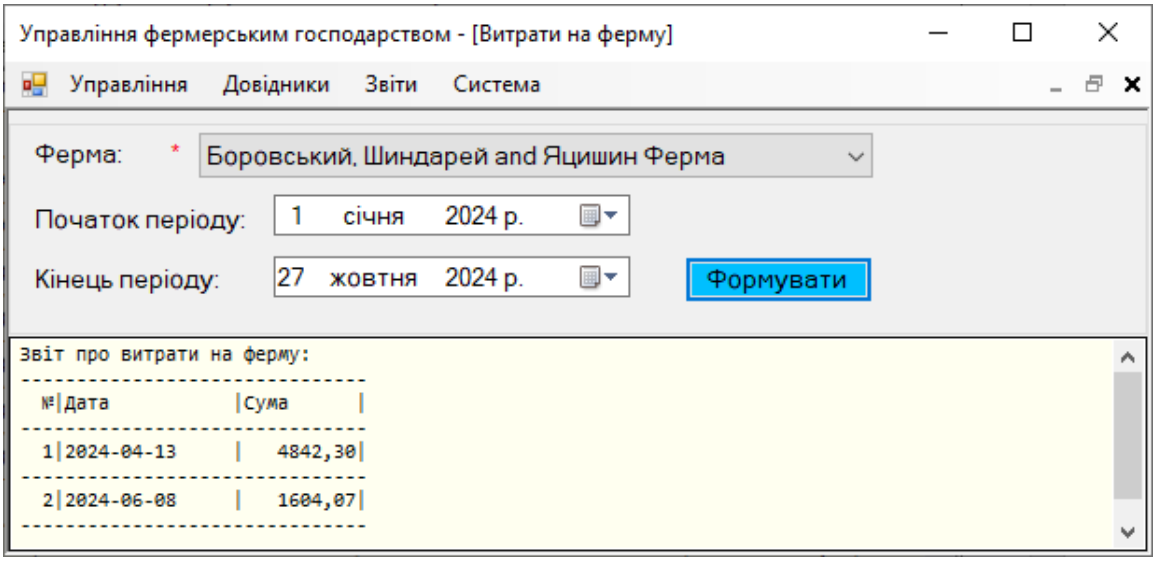

Рис. 3.26 Звітність по витратах на ферму

Форма «Обслуговування обладнання» дозволяє користувачу отримати звіт про проведене обслуговування обладнання на фермерських господарствах за обраний період (рис. 3.27). У верхній частині форми розташовані поля для вибору початкової та кінцевої дати періоду, за який буде сформовано звіт. Після вибору відповідних дат користувач натискає кнопку «Формувати», що запускає процес генерації звіту, відображеного в табличному форматі.

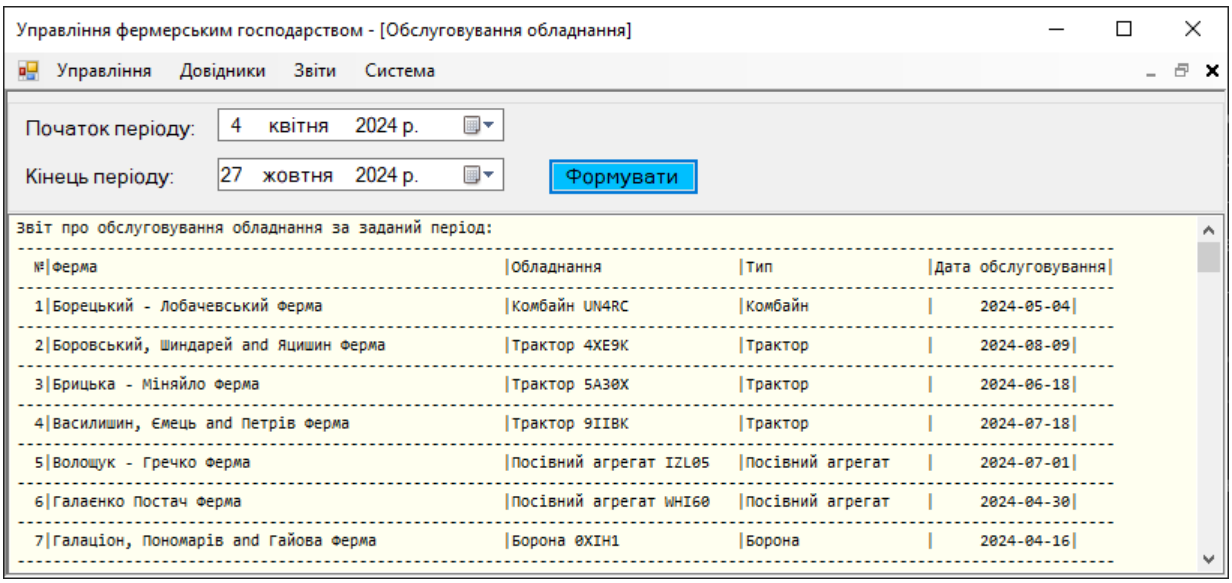

Рис. 3.27 Звітність про обслуговування обладнання

Дана звітність є корисною для планування подальшого обслуговування та забезпечення надійної роботи техніки, що є критичним фактором для

безперервного функціонування господарства. Користувач може своєчасно відстежувати, коли було проведено останнє обслуговування та які види обладнання потребують уваги, що допомагає оптимізувати процеси управління технічними ресурсами.

Форма «Аналіз впливу витрат та погодних умов на врожайність» дозволяє користувачу отримати комплексний звіт про вплив фінансових витрат та кліматичних показників на врожайність для кожного фермерського господарства за заданий період (рис. 3.28). У верхній частині форми користувач може обрати початкову та кінцеву дати періоду, за який необхідно згенерувати звіт. Після вибору періоду натискання кнопки «Формувати» запускає процес створення звіту, який представлений у табличному форматі.

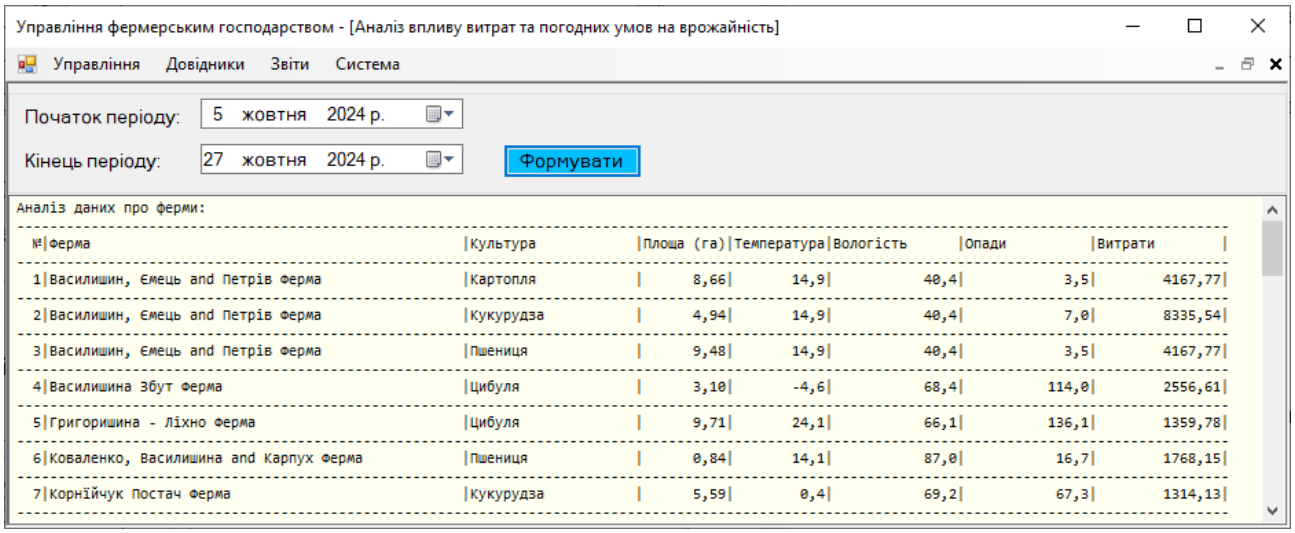

Рис. 3.28 Звітність про аналіз витрат та погодних умов на врожайність

Такий звіт надає можливість виявити, які умови та фінансові вкладення є найбільш сприятливими для вирощування певних культур, а також дає змогу оптимізувати витрати, зосереджуючись на тих факторах, які мають найбільший вплив на врожайність. Аналіз даних сприяє більш ефективному управлінню ресурсами та підвищенню загальної рентабельності фермерського господарства.

Для завершення роботи з програмою користувачу необхідно обрати пункт меню «Управління» та натиснути «Вихід». Слід зазначити, що розроблена система є зручною та інтуїтивно зрозумілою, що значно полегшує її

використання навіть для користувачів із базовими навичками роботи з комп'ютером. Незважаючи на відносну простоту функціоналу, програма ефективно виконує поставлені завдання та забезпечує зручне управління даними фермерського господарства.

3.4 Висновок

У рамках даного розділу було розроблено, протестовано та детально описано програмне забезпечення для управління фермерським господарством, яке дозволяє оптимізувати основні процеси обліку та аналізу даних про ферми, поля, витрати, власників та інші важливі аспекти господарської діяльності. Було реалізовано програмні модулі системи підтримки рішень, розроблено класи та методи для виконання ключових операцій з обробки та зберігання даних. Результати функціонального та модульного тестування, проведеного на основі тестових сценаріїв та автоматизованого тестування з використанням MS Test, підтвердили коректність і стабільність роботи системи, що забезпечує її готовність до експлуатації.

Наведено інструкцію для користувача, яка містить мінімальні вимоги для запуску програмного забезпечення, опис процесу його розгортання та рекомендації з використання для управління фермерським господарством. Завдяки простому і зрозумілому інтерфейсу система є легкою у використанні і доступною для широкого кола користувачів, що дозволяє швидко здійснювати облік ресурсів та аналізувати інформацію для підтримки прийняття рішень. Отримані результати підтверджують, що розроблене програмне забезпечення відповідає поставленим задачам і може ефективно застосовуватися для підтримки фермерської діяльності.

ВИСНОВКИ

Дана кваліфікаційна робота присвячена розробці програмного забезпечення для підтримки прийняття рішень в управлінні фермерським господарством, що дозволяє автоматизувати ключові процеси управління, покращити продуктивність та зменшити витрати шляхом раціонального використання ресурсів та аналізу даних в реальному часі. Створене рішення надає користувачам інструменти для організації обліку, контролю витрат і управління основними виробничими процесами, що сприяє підвищенню ефективності роботи фермерських господарств та полегшує управління господарською діяльністю.

У першому розділі було проведено аналіз особливостей діяльності фермерських господарств та їхніх потреб в автоматизації процесів, а також досліджено поточні інформаційні системи агробізнесу. Це дало змогу виявити їхні сильні та слабкі сторони та визначити основні вимоги до створення нового програмного продукту, що відповідав би специфічним потребам фермерів.

Другий розділ висвітлює вибір технологій розробки та архітектурні рішення, прийняті при створенні програмного забезпечення. Було обрано мову програмування C# та MS SQL Server для створення надійної системи з трирівневою архітектурою, що забезпечує гнучкість та масштабованість додатку. Також було розроблено структуру бази даних, бізнес-логіку та інтерфейс користувача, які відповідають вимогам функціональності та продуктивності.

Третій розділ був присвячений розробці, тестуванню та опису використання програмного продукту. У рамках цього розділу було створено програмні модулі для обробки даних про ферми, поля, витрати та інші важливі аспекти господарської діяльності. Функціональне та модульне тестування, проведене з використанням MS Test, підтвердило стабільність роботи системи та її відповідність заданим вимогам. Крім того, була розроблена інструкція для

користувача, що охоплює мінімальні вимоги до запуску, процес розгортання програмного забезпечення та основні кроки роботи з додатком.

Проведена робота демонструє застосування сучасних технологій для створення інтуїтивно зрозумілого та ефективного інструменту, який дозволяє фермерам автоматизувати облік і аналіз ресурсів, підвищувати ефективність господарських процесів та покращувати управління фінансовими та виробничими показниками. Використання розробленого додатку на практиці дозволяє забезпечити оперативний доступ до інформації, ефективно розподіляти ресурси та приймати обґрунтовані рішення, спрямовані на оптимізацію фермерського господарства.

Результати виконаної роботи можуть бути корисними для подальшого вдосконалення програмного забезпечення шляхом інтеграції додаткових модулів аналітики або адаптації до специфічних вимог окремих типів господарств. Розроблене програмне забезпечення створює основу для більш детального аналізу витрат, оптимізації ресурсів та підвищення ефективності виробничих процесів, що сприятиме автоматизації та модернізації управління фермерськими господарствами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Павлюк, Т., Волонтир Л. Використання сучасних інформаційних технологій в сільському господарстві. Формування ринкової економіки в Україні, 2017, 38: 126 с.

2. Кісь О. В., Антощенков Р. В. Комп'ютеризація та інформаційні технології у сільському господарстві. Вісник Харківського національного технічного університету сільського господарства, Вип. 199 «Механізація сільськогосподарського виробництва». 2019. С. 239.

3. Поповиченко Г. С. Особливості маркетингової діяльності фермерських господарств. Вісник Херсонського національного технічного університету, 2022, 2 (81): 126 С.

4. Кропивко М. Особливості інноваційно-інвестиційної діяльності фермерських господарств України. Економічний дискурс, 2018, 1: 110 С.

5. Полятикіна Л. І., Кадацька А.М. Стан та перспективи розвитку фермерських господарств в Україні. Інноваційна економіка, 2016, 1-2: 44 С.

6. Донець, О. В. Організаційні форми господарської діяльності у сільському господарстві .Науковий вісник Ужгородського університету. – Ужгород: Видавництво УжНУ, 2011. – Спецвип. 33. Ч.1. – С. 101.

7. Підопригора І. В.; Харіна І. В.; Кірюшкіна Ю. М. Особливості обліку та шляхи його вдосконалення у фермерських господарствах. Причорноморські економічні студії, 2018, 34: 197 С.

8. Начичко Н. С. Напрями удосконалення обліку виробничої діяльності фермерських господарств. Актуальні аспекти розвитку підприємств аграрної сфери: облік, аудит та фінансування,2015. – 244 С.

9. Довгань О. М., Мандибура Я. В. Органічне виробництво: сутність, об'єктивна необхідність, ефективність. Сталий розвиток економіки, 2013, 1: 206 С.

10. Малік М. Й. Фермерські господарства як чинник розвитку сільських територій. Український журнал прикладної економіки та техніки. 2023. Т. 8.№ 2. С. 201. URL: https://doi.org/10.36887/2415-8453-2023-2-29

11. Жесан Р. В. Автоматизація управління автономним енергопостачанням з використанням відновлюваних джерел в умовах селянського (фермерського) господарства. 2001. –37 С.

12. Юрчук Н. П.; Кіпоренко С. С. Цифровізація сільського господарства: виклики і можливості для фермерських господарств. Агросвіт. 2024.№ 19. С. 53. URL: https://doi.org/10.32702/2306-6792.2024.19.53

13. Prokopyshyn O., Pryimak H., Trushkina H., & Konoval M. (2024). ACCOUNTING AND ANALYTICAL SUPPORT OF THE FARMING DEVELOPMENT MANAGEMENT: THEORETICAL ASPECTS. Bulletin of Lviv National Environmental University. Series "AIC Economics", (31), p. 110.

14. User Guide for Trimble Ag Software | Farmer Core. URL: https://cdn2.hubspot.net/hubfs/2571402/TABS/TABS Training/Farmer Core Onboarding Guide.2019.pdf (дата звернення 24.10.2024).

15. Vegetable Farm Reaps Benefits of Software URL: https://ww2.agriculture.trimble.com/blog/vegetable-farm-reaps-benefits-of-software/(дата звернення 24.10.2024).

16. FarmLogs: The Farm Management App | Chas S. Middleton. URL: https://chassmiddleton.com/blog/farmlogs (дата звернення 24.10.2024).

17. Pros and Cons of FarmLogs 2024 URL: https://www.trustradius.com/products/farmlogs/reviews?qs=pros-and-cons (дата звернення 24.10.2024).

18. AgroOffice - Home - Agro-Office AG. URL: https://www.agro-office.ch/wp/ (дата звернення 24.10.2024).

19. Городенко В.П., Марценюк О.П. Програмування на мові Python. Теорія та практика: Навчальний посібник. - К.: Центр навчальної літератури, 2016. - 480 с.

20. Бондаренко В.В., Коваленко В.В. Розробка програмного забезпечення з використанням Python, Java та C#: Збірник наукових праць. - Одеса: Видавництво "Сонячна Україна", 2018. - 250 с.

21. Карапецький В. П. Побудова графічного контенту додатків з використанням JavaFX і Swing компонентів і даних, взятих із баз даних / В. П. Карапецький. – Науковий вісник НЛТУ. – 2015. – 418 с.

22. Kishori Sharan. Learn JavaFX 8 (The Expert’s Voice in Java): Building User Experience and Interfaces with Java 8. – New York, 2015. – 1173 p.

23. Безменов М.І., Безменова О.М., Калінін Д.В. Основи візуального програмування мовою C#: навч. посіб. для студентів навчально-наукового інституту комп’ютерних наук та інформаційних технологій. – Харків: ФОП Панов А. М., 2023. 648 с.

24. Коноваленко І.В., Марущак П.О. Платформа .NET та мова програмування C# 8.0: навчальний посібник. Тернопіль: ФОП Паляниця В. А., 2020. 320 с.

25.Microsoft Visual Studio Reviews. URL: https://www.softwareadvice.com/game-development/microsoft-visual-studio-profile/reviews/ (дата звернення 26.10.2024).

26. Visual Studio Code Reviews & Product Details. URL: https://www.g2.com/products/visual-studio-code/reviews (дата звернення 26.10.2024).

27. Most Helpful Rider Reviews. URL: https://www.gartner.com/reviews/market/integrated-development-environment-ide-software/vendor/jetbrains/product/rider (дата звернення 26.10.2024).

28. Козловська В. Організація баз даних та знань: конспект лекцій. Одеса, Одеський державний екологічний університет, 2019. – 130с.

29. Голубчик В., Зайцев А. MySQL трюки: оптимізація, розробка, адміністрування. Київ: Видавництво "Вильямс", 2020. 500 с.

30. Іван Петров. Основи Firebird SQL. – Харків, видавництво “Фоліо” 2020. – 350 с. Транслітерація: Ivan Petrov. Fundamentals of Firebird SQL. / Packt Publishing – Birmingham, 2018. – 350 p.

31. Томашевський О. М., Цегелик Г. Г., Вітер М. Б., Дубук В. І. Інформаційні технології та моделювання бізнес-процесів. Київ : «Центр учбової літератури», 2012. 296 с.

32. Трофименко О.Г., Прокоп Ю.В., Логінова Н.І, Копитчук І.М. Організація баз даних. Одеса, 2019 – 322 с.

33. Liu, C., Song, Y., Li, R., Ma, W., Hao, J. L., & Qiang, G. Three-level modular grid system for sustainable construction of industrialized residential buildings: A case study in China. Journal of Cleaner Production, 2023. – 395 p.

ДОДАТОК А

ЛІСТИНГИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Сторінок 25

CropProvider

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using FarmManagementApp.AppCode;

namespace FarmManagementApp.Providers {
  internal class CropProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertCrop(string CropName, string Season, string Description) {
      string SqlString = "INSERT INTO Crop (CropName, Season, Description)
Values(@CropName, @Season, @Description)";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@CropName", CropName);
          cmd.Parameters.AddWithValue("@Season", Season);
          cmd.Parameters.AddWithValue("@Description", Description);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Crop> GetAllCrop() {
      int num = 0;
      string SqlString = "SELECT * FROM Crop ORDER BY CropName";
      List<Crop> listAllCrop = new List<Crop>();

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          conn.Open();
          using (SqlDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Crop oneCrop = new Crop();
              oneCrop.Number = ++num;
              oneCrop.CropId = Convert.ToInt32(reader["CropId"]);
              oneCrop.CropName = reader["CropName"].ToString();
              oneCrop.Season = reader["Season"].ToString();
              oneCrop.Description = reader["Description"].ToString();
              listAllCrop.Add(oneCrop);
            }
          }
```

```
        conn.Close();
      }
    }

    if (listAllCrop.Count == 0) {
      Crop noCrop = new Crop();
      noCrop.CropId = 0;
      noCrop.Message = NamesMy.NoDataNames.NoDataInCrop; // Повідомлення при
відсутності даних
      listAllCrop.Add(noCrop);
    }

    return listAllCrop;
  }


  public Crop SelectedCropByCropId(int CropId) {
    string SqlString = "SELECT * FROM Crop WHERE CropId=@CropId";

    Crop oneCrop = new Crop();
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.Parameters.AddWithValue("@CropId", CropId);
        conn.Open();
        using (SqlDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneCrop.CropId = Convert.ToInt32(reader["CropId"]);
            oneCrop.CropName = reader["CropName"].ToString();
            oneCrop.Season = reader["Season"].ToString();
            oneCrop.Description = reader["Description"].ToString();
          }
        }
        conn.Close();
      }
    }
    return oneCrop;
  }

  public void UpdateCrop(string CropName, string Season, string Description, int CropId) {
    string SqlString = "UPDATE Crop SET CropName=@CropName, Season=@Season,
Description=@Description WHERE CropId=@CropId";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@CropName", CropName);
        cmd.Parameters.AddWithValue("@Season", Season);
        cmd.Parameters.AddWithValue("@Description", Description);
        cmd.Parameters.AddWithValue("@CropId", CropId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
```

```
        }
      }
    }

    public void DeleteCropByCropId(int CropId) {
      string SqlString = "DELETE FROM Crop WHERE CropId=@CropId";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.Parameters.AddWithValue("@CropId", CropId);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }



  }
}

public class Crop {
  private int _Number;
  private int _CropId;
  private string _CropName;
  private string _Season;
  private string _Description;
  private string _Message;

  // Конструктор за замовчуванням
  public Crop() {
    _Number = 0;
    _CropId = 0;
    _CropName = string.Empty;
    _Season = string.Empty;
    _Description = string.Empty;
    _Message = string.Empty;
  }

  // Властивості

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }


  // Унікальний ідентифікатор культури
  public int CropId {
    set { _CropId = value; }
    get { return _CropId; }
```

```
  }

  // Назва культури
  public string CropName {
    set { _CropName = value; }
    get { return _CropName; }
  }

  // Сезон вирощування (весна, літо тощо)
  public string Season {
    set { _Season = value; }
    get { return _Season; }
  }

  // Опис
  public string Description {
    set { _Description = value; }
    get { return _Description; }
  }

  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }

}
```

Лістинг 2. Код класу «RaportBLL»

```
using FarmManagementApp.Providers;
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using static System.Windows.Forms.VisualStyles.VisualStyleElement.ListView;

namespace FarmManagementApp.AppCode {
  internal class RaportBLL {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public List<CropYieldReport> GetCropYieldReport() {
      var cropYieldReports = new List<CropYieldReport>();
      string query = @"
        SELECT f.FarmName AS FarmName, cr.CropName AS CropName, SUM(ff.FieldSize) AS
TotalArea
        FROM Farm f
        JOIN FarmField ff ON f.FarmId = ff.FarmId
        JOIN Crop cr ON ff.CropId = cr.CropId
        GROUP BY f.FarmName, cr.CropName
        ORDER BY f.FarmName, cr.CropName";
```

```
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      SqlCommand command = new SqlCommand(query, conn);
      conn.Open();

      using (SqlDataReader reader = command.ExecuteReader()) {
        while (reader.Read()) {
          string farmName = reader["FarmName"].ToString();
          string cropName = reader["CropName"].ToString();
          double totalArea = Convert.ToDouble(reader["TotalArea"]);

          cropYieldReports.Add(new CropYieldReport(farmName, cropName, totalArea));
        }
      }
    }
    return cropYieldReports;
  }


  public List<WeatherReport> GetWeatherReport(DateTime startDate, DateTime endDate, int
farmId) {
    var weatherReports = new List<WeatherReport>();

    string query = @"
      SELECT f.FarmName AS FarmName, wd.WeatherDate,
        wd.Temperature, wd.Humidity, wd.Precipitation
      FROM WeatherData wd
      JOIN Farm f ON wd.FarmId = f.FarmId
      WHERE wd.WeatherDate BETWEEN @StartDate AND @EndDate AND wd.FarmId =
@FarmId
      ORDER BY f.FarmName, wd.WeatherDate";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      SqlCommand command = new SqlCommand(query, conn);
      command.Parameters.AddWithValue("@StartDate", startDate);
      command.Parameters.AddWithValue("@EndDate", endDate);
      command.Parameters.AddWithValue("@FarmId", farmId);

      conn.Open();

      using (SqlDataReader reader = command.ExecuteReader()) {
        while (reader.Read()) {
          string farmName = reader["FarmName"].ToString();
          DateTime weatherDate = Convert.ToDateTime(reader["WeatherDate"]);
          float temperature = Convert.ToSingle(reader["Temperature"]);
          float humidity = Convert.ToSingle(reader["Humidity"]);
          float precipitation = Convert.ToSingle(reader["Precipitation"]);

          weatherReports.Add(new WeatherReport(farmName, weatherDate, temperature, humidity,
precipitation));
        }
      }
    }
```

```
    return weatherReports;
  }

  public List<ExpenseReport> GetExpenseReport(DateTime startDate, DateTime endDate, int
farmId) {
    var expenseReports = new List<ExpenseReport>();

    string query = @"
        SELECT f.FarmName AS FarmName, e.ExpenseDate, e.Description, e.Amount
        FROM Expense e
        JOIN Farm f ON e.FarmId = f.FarmId
        WHERE e.ExpenseDate BETWEEN @StartDate AND @EndDate AND e.FarmId =
@FarmId
        ORDER BY e.ExpenseDate";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      SqlCommand command = new SqlCommand(query, conn);
      command.Parameters.AddWithValue("@StartDate", startDate);
      command.Parameters.AddWithValue("@EndDate", endDate);
      command.Parameters.AddWithValue("@FarmId", farmId);

      conn.Open();

      using (SqlDataReader reader = command.ExecuteReader()) {
        while (reader.Read()) {
          string farmName = reader["FarmName"].ToString();
          DateTime expenseDate = Convert.ToDateTime(reader["ExpenseDate"]);
          string description = reader["Description"].ToString();
          double amount = Convert.ToDouble(reader["Amount"]);

          expenseReports.Add(new ExpenseReport(farmName, expenseDate, description, amount));
        }
      }
    }

    return expenseReports;
  }

  public List<MaintenanceReport> GetMaintenanceReport(DateTime startDate, DateTime
endDate) {
    var maintenanceReports = new List<MaintenanceReport>();

    string query = @"
        SELECT f.FarmName AS FarmName, e.EquipmentName, e.EquipmentType,
e.LastServiceDate
        FROM Equipment e
        JOIN Farm f ON e.FarmId = f.FarmId
        WHERE e.LastServiceDate BETWEEN @StartDate AND @EndDate
        ORDER BY f.FarmName, e.EquipmentName";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
```

```
        SqlCommand command = new SqlCommand(query, conn);
        command.Parameters.AddWithValue("@StartDate", startDate);
        command.Parameters.AddWithValue("@EndDate", endDate);

        conn.Open();

        using (SqlDataReader reader = command.ExecuteReader()) {
          while (reader.Read()) {
            string farmName = reader["FarmName"].ToString();
            string equipmentName = reader["EquipmentName"].ToString();
            string equipmentType = reader["EquipmentType"].ToString();
            DateTime lastServiceDate = Convert.ToDateTime(reader["LastServiceDate"]);

            maintenanceReports.Add(new MaintenanceReport(farmName, equipmentName,
equipmentType, lastServiceDate));
          }
        }
      }

      return maintenanceReports;
    }

    public List<FarmAnalysisReport> GetFarmAnalysisReport(DateTime startDate, DateTime
endDate) {
      var analysisReports = new List<FarmAnalysisReport>();

      string query = @"
        SELECT
          f.FarmName AS FarmName,
          cr.CropName AS CropName,
          SUM(ff.FieldSize) AS TotalFieldArea,
          AVG(wd.Temperature) AS AvgTemperature,
          AVG(wd.Humidity) AS AvgHumidity,
          SUM(wd.Precipitation) AS TotalPrecipitation,
          SUM(e.Amount) AS TotalExpenses,
          COUNT(DISTINCT e.ExpenseDate) AS NumberOfExpenseDays
        FROM
          dbo.Farm f
        JOIN
          dbo.FarmField ff ON f.FarmId = ff.FarmId
        JOIN
          dbo.Crop cr ON ff.CropId = cr.CropId
        JOIN
          dbo.WeatherData wd ON wd.FarmId = f.FarmId
        JOIN
          dbo.Expense e ON e.FarmId = f.FarmId
        WHERE
          wd.WeatherDate BETWEEN @StartDate AND @EndDate
          AND e.ExpenseDate BETWEEN @StartDate AND @EndDate
        GROUP BY
          f.FarmName, cr.CropName
        HAVING
```

```
            SUM(ff.FieldSize) > 0
            AND SUM(e.Amount) > 0
        ORDER BY
            f.FarmName, cr.CropName";

        using (SqlConnection conn = new SqlConnection(_ConnString)) {
          SqlCommand command = new SqlCommand(query, conn);
          command.Parameters.AddWithValue("@StartDate", startDate);
          command.Parameters.AddWithValue("@EndDate", endDate);

          conn.Open();

          using (SqlDataReader reader = command.ExecuteReader()) {
            while (reader.Read()) {
              string farmName = reader["FarmName"].ToString();
              string cropName = reader["CropName"].ToString();
              double totalFieldArea = Convert.ToDouble(reader["TotalFieldArea"]);
              double avgTemperature = Convert.ToDouble(reader["AvgTemperature"]);
              double avgHumidity = Convert.ToDouble(reader["AvgHumidity"]);
              double totalPrecipitation = Convert.ToDouble(reader["TotalPrecipitation"]);
              double totalExpenses = Convert.ToDouble(reader["TotalExpenses"]);
              int numberOfExpenseDays = Convert.ToInt32(reader["NumberOfExpenseDays"]);

              analysisReports.Add(new FarmAnalysisReport(farmName, cropName, totalFieldArea,
avgTemperature, avgHumidity,
                                                totalPrecipitation, totalExpenses, numberOfExpenseDays));
            }
          }
        }

        return analysisReports;
      }



    }
}

public class CropYieldReport {
  public string FarmName { get; set; }
  public string CropName { get; set; }
  public double TotalArea { get; set; }

  public CropYieldReport(string farmName, string cropName, double totalArea) {
    FarmName = farmName;
    CropName = cropName;
    TotalArea = totalArea;
  }
}

public class WeatherReport {
  public string FarmName { get; set; }
```

```
  public DateTime WeatherDate { get; set; }
  public float Temperature { get; set; }
  public float Humidity { get; set; }
  public float Precipitation { get; set; }

  public WeatherReport(string farmName, DateTime weatherDate, float temperature, float humidity,
float precipitation) {
    FarmName = farmName;
    WeatherDate = weatherDate;
    Temperature = temperature;
    Humidity = humidity;
    Precipitation = precipitation;
  }
}

public class ExpenseReport {
  public string FarmName { get; set; }
  public DateTime ExpenseDate { get; set; }
  public string Description { get; set; }
  public double Amount { get; set; }

  public ExpenseReport(string farmName, DateTime expenseDate, string description, double
amount) {
    FarmName = farmName;
    ExpenseDate = expenseDate;
    Description = description;
    Amount = amount;
  }
}

public class MaintenanceReport {
  public string FarmName { get; set; }
  public string EquipmentName { get; set; }
  public string EquipmentType { get; set; }
  public DateTime LastServiceDate { get; set; }

  public MaintenanceReport(string farmName, string equipmentName, string equipmentType,
DateTime lastServiceDate) {
    FarmName = farmName;
    EquipmentName = equipmentName;
    EquipmentType = equipmentType;
    LastServiceDate = lastServiceDate;
  }
}

public class FarmAnalysisReport {
  public string FarmName { get; set; }
  public string CropName { get; set; }
  public double TotalFieldArea { get; set; }
  public double AvgTemperature { get; set; }
  public double AvgHumidity { get; set; }
  public double TotalPrecipitation { get; set; }
```

```
  public double TotalExpenses { get; set; }
  public int NumberOfExpenseDays { get; set; }

  public FarmAnalysisReport(string farmName, string cropName, double totalFieldArea, double
avgTemperature, double avgHumidity,
                    double totalPrecipitation, double totalExpenses, int numberOfExpenseDays) {
    FarmName = farmName;
    CropName = cropName;
    TotalFieldArea = totalFieldArea;
    AvgTemperature = avgTemperature;
    AvgHumidity = avgHumidity;
    TotalPrecipitation = totalPrecipitation;
    TotalExpenses = totalExpenses;
    NumberOfExpenseDays = numberOfExpenseDays;
  }
}
```

Лістинг 3. Код класу «DecisionSupportProvider»

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using FarmManagementApp.AppCode;

namespace FarmManagementApp.Providers {
  internal class DecisionSupportProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertDecisionSupport(int FarmFieldId, DateTime DecisionSupportDate, string
Recommendation, string ActionStatus) {
      string SqlString = "INSERT INTO DecisionSupport (FarmFieldId, DecisionSupportDate,
Recommendation, ActionStatus) Values(@FarmFieldId, @DecisionSupportDate,
@Recommendation, @ActionStatus)";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@FarmFieldId", FarmFieldId);
          cmd.Parameters.AddWithValue("@DecisionSupportDate", DecisionSupportDate);
          cmd.Parameters.AddWithValue("@Recommendation", Recommendation);
          cmd.Parameters.AddWithValue("@ActionStatus", ActionStatus);

          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }
```

```
    public List<DecisionSupport> GetAllDecisionSupport() {
      string SqlString = @"
        SELECT ds.SupportId, ds.FarmFieldId, ds.DecisionSupportDate, ds.Recommendation,
ds.ActionStatus,
               ff.FarmFieldName
        FROM DecisionSupport ds
        LEFT JOIN FarmField ff ON ds.FarmFieldId = ff.FarmFieldId
        ORDER BY ds.DecisionSupportDate";

      List<DecisionSupport> listAllDecisionSupport = new List<DecisionSupport>();
      int i = 0;

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          conn.Open();
          using (SqlDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              DecisionSupport oneDecisionSupport = new DecisionSupport();
              oneDecisionSupport.Number = ++i;
              oneDecisionSupport.SupportId = Convert.ToInt32(reader["SupportId"]);
              oneDecisionSupport.FarmFieldId = Convert.ToInt32(reader["FarmFieldId"]);
              oneDecisionSupport.DecisionSupportDate =
Convert.ToDateTime(reader["DecisionSupportDate"]);
              oneDecisionSupport.Recommendation = reader["Recommendation"].ToString();
              oneDecisionSupport.ActionStatus = reader["ActionStatus"].ToString();
              oneDecisionSupport.FarmFieldName = reader["FarmFieldName"].ToString();

              listAllDecisionSupport.Add(oneDecisionSupport);
            }
          }
          conn.Close();
        }
      }

      if (listAllDecisionSupport.Count == 0) {
        DecisionSupport noDecisionSupport = new DecisionSupport();
        noDecisionSupport.SupportId = 0;
        noDecisionSupport.Message = NamesMy.NoDataNames.NoDataInDecisionSupport; //
Повідомлення при відсутності даних
        listAllDecisionSupport.Add(noDecisionSupport);
      }

      return listAllDecisionSupport;
    }


    public DecisionSupport SelectedDecisionSupportBySupportId(int SupportId) {
      string SqlString = "SELECT * FROM DecisionSupport WHERE SupportId=@SupportId";

      DecisionSupport oneDecisionSupport = new DecisionSupport();
      using (SqlConnection conn = new SqlConnection(_ConnString)) {
```

```
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.Parameters.AddWithValue("@SupportId", SupportId);
        conn.Open();
        using (SqlDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneDecisionSupport.SupportId = Convert.ToInt32(reader["SupportId"]);
            oneDecisionSupport.FarmFieldId = Convert.ToInt32(reader["FarmFieldId"]);
            oneDecisionSupport.DecisionSupportDate =
Convert.ToDateTime(reader["DecisionSupportDate"]);
            oneDecisionSupport.Recommendation = reader["Recommendation"].ToString();
            oneDecisionSupport.ActionStatus = reader["ActionStatus"].ToString();
          }
        }
        conn.Close();
      }
    }
    return oneDecisionSupport;
  }

  public void UpdateDecisionSupport(int FarmFieldId, DateTime DecisionSupportDate, string
Recommendation, string ActionStatus, int SupportId) {
    string SqlString = "UPDATE DecisionSupport SET FarmFieldId=@FarmFieldId,
DecisionSupportDate=@DecisionSupportDate, " +
      "Recommendation=@Recommendation, ActionStatus=@ActionStatus WHERE
SupportId=@SupportId";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@FarmFieldId", FarmFieldId);
        cmd.Parameters.AddWithValue("@DecisionSupportDate", DecisionSupportDate);
        cmd.Parameters.AddWithValue("@Recommendation", Recommendation);
        cmd.Parameters.AddWithValue("@ActionStatus", ActionStatus);
        cmd.Parameters.AddWithValue("@SupportId", SupportId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }


  public void DeleteDecisionSupportBySupportId(int SupportId) {
    string SqlString = "DELETE FROM DecisionSupport WHERE SupportId=@SupportId";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.Parameters.AddWithValue("@SupportId", SupportId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
```

```
    }
   }



  }
}

public class DecisionSupport {
  private int _Number;
  private int _SupportId;
  private int _FarmFieldId;
  private string _FarmFieldName;
  private DateTime _DecisionSupportDate;
  private string _Recommendation;
  private string _ActionStatus;
  private string _Message;

  // Конструктор за замовчуванням
  public DecisionSupport() {
    _Number = 0;
    _SupportId = 0;
    _FarmFieldId = 0;
    _FarmFieldName = string.Empty;
    _DecisionSupportDate = DateTime.Now;
    _Recommendation = string.Empty;
    _ActionStatus = string.Empty;
    _Message = string.Empty;
  }

  // Властивості

  // Номер запису
  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }

  // Унікальний ідентифікатор рекомендації
  public int SupportId {
    set { _SupportId = value; }
    get { return _SupportId; }
  }

  // Ідентифікатор поля (посилання на таблицю "Field")
  public int FarmFieldId {
    set { _FarmFieldId = value; }
    get { return _FarmFieldId; }
  }
  public string FarmFieldName {
    set { _FarmFieldName = value; }
    get { return _FarmFieldName; }
```

```
  }
  // Дата видачі рекомендації
  public DateTime DecisionSupportDate {
    set { _DecisionSupportDate = value; }
    get { return _DecisionSupportDate; }
  }

  // Текст рекомендації
  public string Recommendation {
    set { _Recommendation = value; }
    get { return _Recommendation; }
  }

  // Статус виконання дії (заплановано, виконано тощо)
  public string ActionStatus {
    set { _ActionStatus = value; }
    get { return _ActionStatus; }
  }

  // Повідомлення або додаткова інформація
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 4. Код класу «DataGen»

```
using Bogus;
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FarmManagementApp.Providers {
  internal class DataGen {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertSampleCrops() {
      // Список реальних культур
      var crops = new List<(string CropName, string Season, string Description)>
      {
        ("Пшениця", "весна", "Основна зернова культура, широко вирощується для виробництва
борошна."),
        ("Кукурудза", "літо", "Популярна зернова та кормова культура, багата на вуглеводи."),
        ("Соя", "весна", "Бобова культура, багата на білок, використовується для виготовлення
масла та кормів."),
        ("Овес", "осінь", "Зернова культура, використовувана для корму та виробництва
вівсянки."),
```

```
        ("Соняшник", "літо", "Основна олійна культура для виробництва соняшникової олії."),
        ("Ячмінь", "весна", "Зернова культура, використовується для корму та виробництва пива."),
        ("Картопля", "осінь", "Ключовий коренеплід, основний продукт харчування у багатьох країнах."),
        ("Ріпак", "весна", "Олійна культура, використовується для виробництва біопалива та масла."),
        ("Морква", "літо", "Коренеплідна овочева культура, багата на вітаміни та мінерали."),
        ("Цибуля", "осінь", "Овочева культура, використовується в кулінарії та має лікувальні властивості.")
    };

     // Додавання кожного запису в базу даних
     foreach (var crop in crops) {
       InsertCrop(crop.CropName, crop.Season, crop.Description);
     }
   }

   private void InsertCrop(string CropName, string Season, string Description) {
     string SqlString = "INSERT INTO Crop (CropName, Season, Description) VALUES (@CropName, @Season, @Description)";

     using (SqlConnection conn = new SqlConnection(_ConnString)) {
       using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
         cmd.CommandType = CommandType.Text;
         cmd.Parameters.AddWithValue("@CropName", CropName);
         cmd.Parameters.AddWithValue("@Season", Season);
         cmd.Parameters.AddWithValue("@Description", Description);
         conn.Open();
         cmd.ExecuteNonQuery();
         conn.Close();
       }
     }
   }

   public void InsertOwners(int count) {
     // Вказуємо тип об'єкта для Faker
     var faker = new Faker<Owner>("uk")
         .RuleFor(o => o.LastName, f => f.Name.LastName())
         .RuleFor(o => o.FirstName, f => f.Name.FirstName())
         .RuleFor(o => o.Phone, f => f.Phone.PhoneNumber("+38 (###) ###-##-##"))
         .RuleFor(o => o.Address, f => f.Address.FullAddress())
         .RuleFor(o => o.Email, f => f.Internet.Email());

     for (int i = 0; i < count; i++) {
       var fakeOwner = faker.Generate();
       InsertOwner(fakeOwner.LastName, fakeOwner.FirstName, fakeOwner.Phone, fakeOwner.Address, fakeOwner.Email);
     }
   }
```

```
    private void InsertOwner(string LastName, string FirstName, string Phone, string Address, string
Email) {
      string SqlString = "INSERT INTO Owner (LastName, FirstName, Phone, Address, Email)
VALUES (@LastName, @FirstName, @Phone, @Address, @Email)";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@LastName", LastName);
          cmd.Parameters.AddWithValue("@FirstName", FirstName);
          cmd.Parameters.AddWithValue("@Phone", Phone);
          cmd.Parameters.AddWithValue("@Address", Address);
          cmd.Parameters.AddWithValue("@Email", Email);
          conn.Open();
          cmd.ExecuteNonQuery();
        }
      }

    }

    public void InsertFarms(List<Owner> owners) {
      // Встановлюємо генератор з українською локалізацією
      var faker = new Faker("uk");

      foreach (var owner in owners) {
        // Генеруємо випадкову кількість ферм для кожного власника (від 1 до 3)
        int farmCount = faker.Random.Int(1, 3);

        for (int i = 0; i < farmCount; i++) {
          // Генеруємо дані для кожної ферми
          string farmName = faker.Company.CompanyName() + " Ферма";
          string location = faker.Address.City() + ", " + faker.Address.StreetAddress();
          string description = faker.Lorem.Sentence();

          // Додаємо ферму в базу даних
          InsertFarm(farmName, location, description, owner.OwnerId);
        }
      }
    }

    private void InsertFarm(string FarmName, string Location, string Description, int OwnerId) {
      string SqlString = "INSERT INTO Farm (FarmName, Location, Description, OwnerId)
VALUES (@FarmName, @Location, @Description, @OwnerId)";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@FarmName", FarmName);
          cmd.Parameters.AddWithValue("@Location", Location);
          cmd.Parameters.AddWithValue("@Description", Description);
          cmd.Parameters.AddWithValue("@OwnerId", OwnerId);
```

```
            conn.Open();
            cmd.ExecuteNonQuery();
            conn.Close();
        }
    }
}


public void InsertFarmFields(List<Crop> crops, List<Farm> farms) {
    var faker = new Faker("uk");

    foreach (var farm in farms) {
        // Випадковий вибір кількості полів для кожної ферми (наприклад, від 1 до 5 полів)
        int fieldCount = faker.Random.Int(1, 5);

        for (int i = 0; i < fieldCount; i++) {
            // Вибір випадкової культури для кожного поля
            var randomCrop = faker.PickRandom(crops);

            // Генеруємо унікальне ім'я для кожного поля
            string fieldType = faker.PickRandom(new[] { "Поле", "Ділянка", "Участок" });
            string fieldIdentifier = faker.Random.String2(5,
"ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789");
            string farmFieldName = $"{fieldType} {fieldIdentifier}";

            // Інші дані для поля
            double fieldSize = faker.Random.Double(0.5, 10.0); // Розмір поля від 0.5 до 10 гектарів
            string description = faker.Lorem.Sentence();

            // Додаємо поле в базу даних
            InsertFarmField(farmFieldName, fieldSize, description, randomCrop.CropId, farm.FarmId);
        }
    }
}

private void InsertFarmField(string FarmFieldName, double FieldSize, string Description, int
CropId, int FarmId) {
    string SqlString = "INSERT INTO FarmField (FarmFieldName, FieldSize, Description, CropId,
FarmId) VALUES (@FarmFieldName, @FieldSize, @Description, @CropId, @FarmId)";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
            cmd.CommandType = CommandType.Text;
            cmd.Parameters.AddWithValue("@FarmFieldName", FarmFieldName);
            cmd.Parameters.AddWithValue("@FieldSize", FieldSize);
            cmd.Parameters.AddWithValue("@Description", Description);
            cmd.Parameters.AddWithValue("@CropId", CropId);
            cmd.Parameters.AddWithValue("@FarmId", FarmId);

            conn.Open();
            cmd.ExecuteNonQuery();
            conn.Close();
```

```
      }
    }
  }

  public void InsertWeatherDataForFarms(List<Farm> farms) {
    var faker = new Faker("uk");

    foreach (var farm in farms) {
      DateTime currentDate = DateTime.Now;

      // Генерація даних на кожний місяць за останній рік
      for (int i = 0; i < 12; i++) {
        // Встановлюємо дату на початок кожного місяця
        DateTime weatherDate = currentDate.AddMonths(-i);

        // Генерація випадкових значень для погоди
        double temperature = faker.Random.Double(-20, 35); // Температура від -20 до +35
градусів
        double humidity = faker.Random.Double(20, 100); // Вологість від 20% до 100%
        double precipitation = faker.Random.Double(0, 200); // Опади від 0 до 200 мм

        // Додавання запису про погоду в базу даних
        InsertWeatherData(weatherDate, temperature, humidity, precipitation, farm.FarmId);
      }
    }
  }

  private void InsertWeatherData(DateTime WeatherDate, double Temperature, double Humidity,
double Precipitation, int FarmId) {
    string SqlString = "INSERT INTO WeatherData (WeatherDate, Temperature, Humidity,
Precipitation, FarmId) VALUES (@WeatherDate, @Temperature, @Humidity, @Precipitation,
@FarmId)";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@WeatherDate", WeatherDate);
        cmd.Parameters.AddWithValue("@Temperature", Temperature);
        cmd.Parameters.AddWithValue("@Humidity", Humidity);
        cmd.Parameters.AddWithValue("@Precipitation", Precipitation);
        cmd.Parameters.AddWithValue("@FarmId", FarmId);

        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

  public void InsertEquipmentForFarms(List<Farm> farms) {
    var faker = new Faker("uk");
```

```
    // Визначення типів обладнання
    var equipmentTypes = new[]
    {
      "Борона", "Зерносушарка", "Комбайн", "Культиватор", "Оприскувач",
      "Плуг", "Посівний агрегат", "Пресс-підбирач", "Система крапельного зрошення",
"Трактор"
   };

    foreach (var farm in farms) {
      // Генеруємо випадкову кількість обладнання для кожної ферми (від 1 до 8)
      int equipmentCount = faker.Random.Int(1, 8);

      for (int i = 0; i < equipmentCount; i++) {
        // Генерація випадкового типу обладнання
        string equipmentType = faker.PickRandom(equipmentTypes);

        // Генерація випадкового імені обладнання
        string equipmentName = $"{equipmentType} {faker.Random.String2(5,
"ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789")}";

        // Генерація випадкової дати останнього обслуговування (максимум 2,5 роки - мінімум
2 місяці назад)
        DateTime lastServiceDate = faker.Date.PastOffset(2, DateTime.Now.AddMonths(-
2)).DateTime;

        // Додавання запису про обладнання в базу даних
        InsertEquipment(equipmentName, equipmentType, farm.FarmId, lastServiceDate);
      }
    }
   }

   private void InsertEquipment(string EquipmentName, string EquipmentType, int FarmId,
DateTime LastServiceDate) {
    string SqlString = "INSERT INTO Equipment (EquipmentName, EquipmentType, FarmId,
LastServiceDate) VALUES (@EquipmentName, @EquipmentType, @FarmId,
@LastServiceDate)";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@EquipmentName", EquipmentName);
        cmd.Parameters.AddWithValue("@EquipmentType", EquipmentType);
        cmd.Parameters.AddWithValue("@FarmId", FarmId);
        cmd.Parameters.AddWithValue("@LastServiceDate", LastServiceDate);

        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
   }
```

```
  public void InsertDecisionSupportForFields(List<FarmField> farmFields) {
    var faker = new Faker("uk");

    // Визначення можливих статусів дій
    var actionStatuses = new[] { "заплановано", "виконано" };

    foreach (var field in farmFields) {
      // Генеруємо випадкову кількість записів для кожного поля (від 1 до 3)
      int decisionSupportCount = faker.Random.Int(1, 3);

      for (int i = 0; i < decisionSupportCount; i++) {
        // Генерація випадкової дати підтримки рішень, останні 6 місяців
        DateTime decisionSupportDate = faker.Date.Past(1);

        // Генерація випадкового тексту рекомендації
        string recommendation = faker.Lorem.Sentence();

        // Вибір випадкового статусу дії
        string actionStatus = faker.PickRandom(actionStatuses);

        // Додавання запису до бази даних
        InsertDecisionSupport(field.FarmFieldId, decisionSupportDate, recommendation,
actionStatus);
      }
    }
  }

  private void InsertDecisionSupport(int FarmFieldId, DateTime DecisionSupportDate, string
Recommendation, string ActionStatus) {
    string SqlString = "INSERT INTO DecisionSupport (FarmFieldId, DecisionSupportDate,
Recommendation, ActionStatus) VALUES (@FarmFieldId, @DecisionSupportDate,
@Recommendation, @ActionStatus)";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("@FarmFieldId", FarmFieldId);
        cmd.Parameters.AddWithValue("@DecisionSupportDate", DecisionSupportDate);
        cmd.Parameters.AddWithValue("@Recommendation", Recommendation);
        cmd.Parameters.AddWithValue("@ActionStatus", ActionStatus);

        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }


  public void InsertExpensesForFarms(List<Farm> farms) {
    var faker = new Faker("uk");
```

```
      foreach (var farm in farms) {
        // Генеруємо випадкову кількість витрат для кожної ферми (від 1 до 5)
        int expenseCount = faker.Random.Int(1, 5);

        for (int i = 0; i < expenseCount; i++) {
          // Генерація випадкового опису витрат
          string description = faker.Commerce.ProductName();

          // Генерація випадкової суми витрат від 100 до 5000
          double amount = Math.Round(faker.Random.Double(100, 5000), 2);

          // Генерація випадкової дати витрат за останній рік
          DateTime expenseDate = faker.Date.Past(1);

          // Додавання запису про витрати до бази даних
          InsertExpense(description, amount, expenseDate, farm.FarmId);
        }
      }
    }

    private void InsertExpense(string Description, double Amount, DateTime ExpenseDate, int
FarmId) {
      string SqlString = "INSERT INTO Expense (Description, Amount, ExpenseDate, FarmId)
VALUES (@Description, @Amount, @ExpenseDate, @FarmId)";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@Description", Description);
          cmd.Parameters.AddWithValue("@Amount", Amount);
          cmd.Parameters.AddWithValue("@ExpenseDate", ExpenseDate);
          cmd.Parameters.AddWithValue("@FarmId", FarmId);

          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

  }
}
```

Лістинг 5. Код класу «OwnerProvider»

```
using FarmManagementApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace FarmManagementApp.Providers {
  internal class OwnerProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertOwner(string LastName, string FirstName, string Phone, string Address, string
Email) {
      string SqlString = "INSERT INTO Owner (LastName, FirstName, Phone, Address, Email)
VALUES (@LastName, @FirstName, @Phone, @Address, @Email)";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@LastName", LastName);
          cmd.Parameters.AddWithValue("@FirstName", FirstName);
          cmd.Parameters.AddWithValue("@Phone", Phone);
          cmd.Parameters.AddWithValue("@Address", Address);
          cmd.Parameters.AddWithValue("@Email", Email);
          conn.Open();
          cmd.ExecuteNonQuery();
        }
      }
    }

    public List<Owner> GetAllOwner() {
      int i = 0;
      string SqlString = "SELECT * FROM Owner";

      List<Owner> listAllOwner = new List<Owner>();
      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          conn.Open();
          using (SqlDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Owner oneOwner = new Owner();
              oneOwner.Number = ++i;
              oneOwner.OwnerId = Convert.ToInt32(reader["OwnerId"]);
              oneOwner.FirstName = reader["FirstName"].ToString();
              oneOwner.LastName = reader["LastName"].ToString();
              oneOwner.FIO = oneOwner.LastName + " " + oneOwner.FirstName;
              oneOwner.Phone = reader["Phone"].ToString();
              oneOwner.Address = reader["Address"].ToString();
              oneOwner.Email = reader["Email"].ToString();

              listAllOwner.Add(oneOwner);
            }
          }
        }
      }

      if (listAllOwner.Count == 0) {
```

```
      Owner noOwner = new Owner();
      noOwner.OwnerId = 0;
      noOwner.Message = NamesMy.NoDataNames.NoDataInOwner;
      listAllOwner.Add(noOwner);
    }
    return listAllOwner;
  }

  public Owner SelectedOwnerByOwnerId(int OwnerId) {
    string SqlString = "SELECT * FROM Owner WHERE OwnerId = @OwnerId";

    Owner oneOwner = new Owner();
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.Parameters.AddWithValue("@OwnerId", OwnerId);
        conn.Open();
        using (SqlDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            oneOwner.OwnerId = Convert.ToInt32(reader["OwnerId"]);
            oneOwner.FirstName = reader["FirstName"].ToString();
            oneOwner.LastName = reader["LastName"].ToString();
            oneOwner.FIO = oneOwner.LastName + " " + oneOwner.FirstName;
            oneOwner.Phone = reader["Phone"].ToString();
            oneOwner.Address = reader["Address"].ToString();
            oneOwner.Email = reader["Email"].ToString();
          }
        }
      }
    }
    return oneOwner;
  }

  public void UpdateOwner(string LastName, string FirstName, string Phone, string Address,
string Email, int OwnerId) {
    string SqlString = "UPDATE Owner SET FirstName = @FirstName, LastName = @LastName,
Phone = @Phone, Address = @Address, Email = @Email WHERE OwnerId = @OwnerId";

    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.Parameters.AddWithValue("@FirstName", FirstName);
        cmd.Parameters.AddWithValue("@LastName", LastName);
        cmd.Parameters.AddWithValue("@Phone", Phone);
        cmd.Parameters.AddWithValue("@Address", Address);
        cmd.Parameters.AddWithValue("@Email", Email);
        cmd.Parameters.AddWithValue("@OwnerId", OwnerId);
        conn.Open();
        cmd.ExecuteNonQuery();
      }
    }
  }

  public void DeleteOwnerByOwnerId(int OwnerId) {
```

```
      string SqlString = "DELETE FROM Owner WHERE OwnerId = @OwnerId";
      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.Parameters.AddWithValue("@OwnerId", OwnerId);
          conn.Open();
          cmd.ExecuteNonQuery();
        }
      }
    }

  }
}


public class Owner {
  private int _Number;
  private int _OwnerId;
  private string _FirstName;
  private string _LastName;
  private string _Phone;
  private string _Address;
  private string _Email;
  private string _FIO;
  private string _Message;

  public Owner() {
    _Number = 0;
    _OwnerId = 0;
    _FirstName = String.Empty;
    _LastName = String.Empty;
    _Phone = String.Empty;
    _Address = String.Empty;
    _Email = String.Empty;
    _FIO = String.Empty;
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int OwnerId {
    set { _OwnerId = value; }
    get { return _OwnerId; }
  }
  public string FirstName {
    set { _FirstName = value; }
    get { return _FirstName; }
  }
  public string LastName {
    set { _LastName = value; }
    get { return _LastName; }
```

```
  }
  public string Phone {
    set { _Phone = value; }
    get { return _Phone; }
  }
  public string Address {
    set { _Address = value; }
    get { return _Address; }
  }
  public string Email {
    set { _Email = value; }
    get { return _Email; }
  }
  public string FIO {
    set { _FIO = value; }
    get { return _FIO; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

ДОДАТОК Б

СКРИПТИ БАЗИ ДАНИХ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Сторінок 4

```
USE [master]
GO
/****** Object:  Database [FarmManagement]    Script Date: 26.10.2024 11:13:02 ******/
CREATE DATABASE [FarmManagement]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'FarmManagement', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\FarmManagement.mdf' , SIZE = 8192KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB )
 LOG ON
( NAME = N'FarmManagement_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\FarmManagement_log.ldf' , SIZE = 8192KB ,
MAXSIZE = 2048GB , FILEGROWTH = 65536KB )
 WITH CATALOG_COLLATION = DATABASE_DEFAULT, LEDGER = OFF
GO
ALTER DATABASE [FarmManagement] SET COMPATIBILITY_LEVEL = 160
GO
IF (1 = FULLTEXTSERVICEPROPERTY('IsFullTextInstalled'))
begin
EXEC [FarmManagement].[dbo].[sp_fulltext_database] @action = 'enable'
end
GO
GO
EXEC sys.sp_db_vardecimal_storage_format N'FarmManagement', N'ON'
GO
ALTER DATABASE [FarmManagement] SET QUERY_STORE = ON
GO
ALTER DATABASE [FarmManagement] SET QUERY_STORE (OPERATION_MODE =
READ_WRITE, CLEANUP_POLICY = (STALE_QUERY_THRESHOLD_DAYS = 30),
DATA_FLUSH_INTERVAL_SECONDS = 900, INTERVAL_LENGTH_MINUTES = 60,
MAX_STORAGE_SIZE_MB = 1000, QUERY_CAPTURE_MODE = AUTO,
SIZE_BASED_CLEANUP_MODE = AUTO, MAX_PLANS_PER_QUERY = 200,
WAIT_STATS_CAPTURE_MODE = ON)
GO
USE [FarmManagement]
GO
/****** Object:  Table [dbo].[Crop]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Crop](
	[CropId] [int] IDENTITY(1,1) NOT NULL,
	[CropName] [nvarchar](250) NULL,
	[Season] [nvarchar](50) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[CropId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

```
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[DecisionSupport]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[DecisionSupport](
	[SupportId] [int] IDENTITY(1,1) NOT NULL,
	[FarmFieldId] [int] NULL,
	[DecisionSupportDate] [datetime] NULL,
	[Recommendation] [nvarchar](max) NULL,
	[ActionStatus] [nvarchar](50) NULL,
PRIMARY KEY CLUSTERED
(
	[SupportId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Equipment]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Equipment](
	[EquipmentId] [int] IDENTITY(1,1) NOT NULL,
	[EquipmentName] [nvarchar](250) NULL,
	[EquipmentType] [nvarchar](100) NULL,
	[FarmId] [int] NULL,
	[LastServiceDate] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[EquipmentId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Expense]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Expense](
	[ExpenseId] [int] IDENTITY(1,1) NOT NULL,
	[Description] [nvarchar](max) NULL,
	[Amount] [float] NULL,
	[ExpenseDate] [datetime] NULL,
	[FarmId] [int] NULL,
PRIMARY KEY CLUSTERED
```

```
(
	[ExpenseId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Farm]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Farm](
	[FarmId] [int] IDENTITY(1,1) NOT NULL,
	[FarmName] [nvarchar](250) NULL,
	[Location] [nvarchar](250) NULL,
	[Description] [nvarchar](max) NULL,
	[OwnerId] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[FarmId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[FarmField]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[FarmField](
	[FarmFieldId] [int] IDENTITY(1,1) NOT NULL,
	[FarmFieldName] [nvarchar](250) NULL,
	[FieldSize] [float] NULL,
	[Description] [nvarchar](max) NULL,
	[CropId] [int] NULL,
	[FarmId] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[FarmFieldId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Logs]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Logs](
```

```
	[LogsId] [int] IDENTITY(1,1) NOT NULL,
	[UsersId] [int] NULL,
	[EventNameShow] [nvarchar](max) NULL,
	[EventDate] [datetime] NULL,
PRIMARY KEY CLUSTERED
(
	[LogsId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Owner]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Owner](
	[OwnerId] [int] IDENTITY(1,1) NOT NULL,
	[FirstName] [nvarchar](50) NULL,
	[LastName] [nvarchar](50) NULL,
	[Phone] [nvarchar](20) NULL,
	[Address] [nvarchar](max) NULL,
	[Email] [nvarchar](120) NULL,
PRIMARY KEY CLUSTERED
(
	[OwnerId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
/****** Object:  Table [dbo].[Users]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Users](
	[UsersId] [int] IDENTITY(1,1) NOT NULL,
	[FirstName] [nvarchar](60) NULL,
	[LastName] [nvarchar](60) NULL,
	[UsersName] [nvarchar](60) NULL,
	[UsersPassword] [nvarchar](250) NULL,
	[RoleId] [int] NULL,
	[Description] [nvarchar](1200) NULL,
	[Email] [nvarchar](150) NULL,
PRIMARY KEY CLUSTERED
(
	[UsersId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

```
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[WeatherData]    Script Date: 26.10.2024 11:13:02 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[WeatherData](
	[WeatherDataId] [int] IDENTITY(1,1) NOT NULL,
	[WeatherDate] [datetime] NULL,
	[Temperature] [float] NULL,
	[Humidity] [float] NULL,
	[Precipitation] [float] NULL,
	[FarmId] [int] NULL,
PRIMARY KEY CLUSTERED
(
	[WeatherDataId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
USE [master]
GO
ALTER DATABASE [FarmManagement] SET  READ_WRITE
GO
```