**НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
**Завідувач кафедри**
Комп'ютерних наук
_____ _Голуб Б.Л.

"____" _____ 2025 р.

# БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
**на тему**

Програмне забезпечення вибору оптимальних сільськогосподарських культур для певних регіонів з використанням алгоритму LIMBO

Спеціальність 121 – «Інженерія програмного забезпечення»

**Гарант освітньої програми**

К.т.н., доцент _____ Вайганг Г.О.

**Керівник бакалаврської кваліфікаційної роботи**

| к.т.н., доцент | _____ | Вайганг Г.О. |
|---|---|---|
| науковий ступінь та вчене звання) | (підпис) | (ПІБ) |

**Виконав** _____ Риженко Д.А.
(підпис) (ПІБ студента)

**КИЇВ – 2025**

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

**ЗАТВЕРДЖУЮ**
**Завідувач кафедри**
Комп'ютерних наук
_____ Голуб Б.Л.

"____" _____ 2025 р.

**ЗАВДАННЯ**
**на виконання бакалаврської кваліфікаційної роботи студенту**

Риженку Дмитру Андрійовичу

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення вибору оптимальних сільськогосподарських культур для певних регіонів з використанням алгоритму LIMBO

затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

Термін подання завершеної роботи на кафедру 2025.
(рік, місяць, число)

Вихідні дані до роботи: опис програмного забезпечення

Перелік питань що розглядаються:
1. Системний аналіз предметної області
2. Проєктування системи вибору культур на основі кластерного аналізу
3. Розробка програмного забезпечення для підтримки аграрних рішень
4. Впровадження системи та рекомендації щодо її експлуатації

**Дата видачі завдання** «_____»_____2025 р.
**Керівник бакалаврської кваліфікаційної роботи**
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

**Завдання прийняв до виконання** _____ Риженко Д.А.
(підпис) (ПІБ студента)

# ЗМІСТ

# ВСТУП

У сучасних умовах зміни клімату, деградації ґрунтів і нестабільності аграрного ринку питання підвищення ефективності агровиробництва набуває особливої актуальності. Успішне ведення сільського господарства дедалі більше залежить не лише від досвіду аграріїв, а й від точності прийнятих ними рішень щодо вибору культур, які найкраще адаптовані до специфічних умов певного регіону [1]. Класичні підходи, що ґрунтуються на статистичних довідниках або експертних оцінках, мають обмежену здатність до адаптації в умовах стрімких змін природного середовища та зростаючих вимог до стійкості й продуктивності агросистем. Вони часто не враховують багатовимірність впливу факторів, таких як тип ґрунту, кислотність, глибина залягання ґрунтових вод, мікрокліматичні особливості, що в сукупності визначають агрокліматичний потенціал території.

У світовій практиці дедалі ширше застосовуються методи інтелектуального аналізу даних, машинного навчання та кластеризації для виявлення закономірностей у великих обсягах аграрної інформації. Проте більшість існуючих рішень, зокрема в системах західного зразка, фокусуються або на макрорівні (країна чи область), або не враховують локальних особливостей вирощування культур. Зокрема, платформи типу CropSyst та DSSAT хоча й дозволяють моделювати врожайність, проте потребують великих обсягів польових даних та часто не враховують адміністративний поділ, характерний для країн пострадянського простору [2]. Окрім того, зазначені інструменти є надто складними у впровадженні в умовах малого та середнього фермерського господарства.

Для України, як аграрної держави з різноманіттям кліматичних зон і ґрунтових типів, особливо важливим є пошук рішень, які дозволяють ефективно адаптувати агровиробництво до умов конкретного регіону. Використання кластерного аналізу, зокрема алгоритму LIMBO, відкриває можливості для виявлення подібних регіонів за сукупністю агроекологічних характеристик і формування рекомендацій щодо культур, які показали найкращі результати в

схожих умовах. LIMBO як інструмент непараметричної кластеризації забезпечує гнучке групування об’єктів за якісними та кількісними ознаками, що є надзвичайно доречним у контексті аграрної статистики, яка часто представлена в категоріальній або нечіткій формі.

Розробка інтелектуального інструменту, здатного на основі історичних даних і агрохарактеристик визначати найкращі культури для конкретного регіону, має вагоме значення не лише з точки зору підвищення врожайності, але й з огляду на продовольчу безпеку, раціональне використання природних ресурсів та зменшення економічних ризиків для сільськогосподарських підприємств. Такий підхід сприяє не лише підвищенню ефективності агробізнесу, а й стимулює впровадження цифрових технологій в агросектор, що є однією з пріоритетних задач сучасної стратегії розвитку України.

Мета роботи полягає у створенні програмного забезпечення для вибору оптимальних сільськогосподарських культур на основі агроекологічних характеристик регіону з використанням алгоритму кластерного аналізу LIMBO.

Для досягнення поставленої мети, необхідно виконати наступні задачі:

– провести аналіз предметної області сільськогосподарського виробництва та сформулювати вимоги до інтелектуальної системи підтримки аграрних рішень;

– розробити логічну модель бази даних для представлення аграрних характеристик регіонів та культур, адаптовану до потреб кластерного аналізу;

– реалізувати програмний модуль кластеризації на основі алгоритму LIMBO та створити інтерфейс користувача для взаємодії з системою;

– провести тестування програмного забезпечення в експериментальному режимі та сформулювати рекомендації щодо його впровадження в агровиробничих умовах.

Для розробки програмного забезпечення для підтримки прийняття рішень у сфері сільськогосподарського виробництва були застосовані сучасні методи інженерії програмного забезпечення, зокрема метод системного аналізу, структурно-логічне моделювання, критичний аналіз наукових джерел та

практичних рішень, а також методи кластерного аналізу для обробки аграрних даних.

У якості основних технологічних засобів реалізації було обрано мову програмування C# та систему управління базами даних Microsoft SQL Server, що забезпечують оптимальний баланс між продуктивністю, гнучкістю та безпекою.

Структура та обсяг роботи. Пояснювальна записка з кваліфікаційної роботи складається з анотації, переліку умовних скорочень, вступу, 4 розділів, висновків, списку використаних джерел та додатків. Загальний обсяг записки становить 88 сторінок. Зокрема пояснювальна записка містить 39 рисунків, 13 таблиць 2 додатків. Список використаних джерел нараховує 22 найменування.

# 1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис предметної області

Сільське господарство є однією з базових і стратегічно важливих галузей економіки, яка забезпечує продовольчу безпеку, формує експортний потенціал та створює робочі місця, особливо в сільських регіонах. Його значення виходить далеко за межі простої аграрної діяльності, охоплюючи також сфери переробки сільськогосподарської продукції, логістики, аграрного машинобудування та агрохімії. У країнах з потужним агропромисловим комплексом, таких як Україна, ця галузь виконує роль одного з основних драйверів економічного зростання.

Проблематика вибору сільськогосподарських культур є однією з найважливіших у сучасному агровиробництві, адже від неї безпосередньо залежить ефективність використання земельних ресурсів, стабільність врожайності та прибутковість господарств [3]. У процесі прийняття рішень щодо вибору культури для вирощування враховується широкий спектр чинників, пов’язаних з агроекологічними умовами конкретного регіону. Для досягнення максимальної продуктивності важливо не лише обрати культуру, яка відповідає кліматичним умовам, але й передбачити її рентабельність та відповідність ринковому попиту. Особливої ваги ця проблема набуває в умовах кліматичних змін і високої конкуренції на ринку, що зумовлює необхідність прийняття рішень на основі комплексного аналізу. Саме тому автоматизовані системи, що враховують багатофакторну природу агровиробничих рішень, є доцільним інструментом для аграрного сектору, зокрема в Україні. Їхня мета полягає у підвищенні точності рекомендацій щодо вибору культур, зменшенні ризиків та покращенні ефективності управління земельними ресурсами.

На рис. 1.1 наведено узагальнену діаграму, що ілюструє ключові фактори, які впливають на процес вибору сільськогосподарських культур. Діаграма

дозволяє візуалізувати структуру проблеми та класифікувати основні напрями аналізу, що мають враховуватись у процесі прийняття агрономічних рішень.


Рис.1.1 Діаграма розподілу проблематики вибору культур

Центральним елементом діаграми виступає поняття «Проблематика вибору культур», яке інтегрує кілька важливих блоків факторів. Першим із них є економічна доцільність, що включає в себе обґрунтування затрат і очікуваного прибутку, а також орієнтацію на ринковий попит, який визначає, наскільки обрана культура буде комерційно вигідною. Наступний важливий блок – кліматичні умови, які охоплюють такі параметри, як температура, кількість опадів, рівень вологості та інсоляція (світловий режим), що безпосередньо впливають на біологічну продуктивність культур. Третій ключовий компонент – ґрунтові характеристики, які розглядаються через такі аспекти, як механічний склад ґрунту, його родючість, кислотність (pH) та дренувальні властивості. Не менш важливим чинником є забезпеченість водними ресурсами, оскільки наявність або нестача зрошення може істотно обмежувати спектр можливих культур для вирощування.

Алгоритм LIMBO (Lossy Independent Measures for Binary Objects) є методом кластерного аналізу, що дозволяє ефективно групувати об’єкти за схожістю ознак, особливо коли дані мають категоріальний або неповний характер [4]. У контексті агровиробництва його застосування стає доцільним для вирішення задачі поділу регіонів на однорідні групи за агроекологічними характеристиками – зокрема, за типами ґрунтів, кліматичними умовами, рівнем

вологості тощо. Такий підхід дозволяє виявити типові регіони, в яких ті чи інші культури історично давали найкращі результати, і на основі цього – формувати рекомендації для інших, подібних за характеристиками місцевостей. LIMBO є нечутливим до шуму в даних, здатним до масштабування і придатним для роботи з великими наборами інформації, що відповідає реаліям сучасного агросектора.

На рис. 1.2 висвітлено узагальнену діаграму використання алгоритму LIMBO в системі вибору оптимальних сільськогосподарських культур. Ця схема візуалізує взаємозв'язок між вхідними характеристиками регіону, принципами кластеризації та кінцевими результатами рекомендаційної системи.

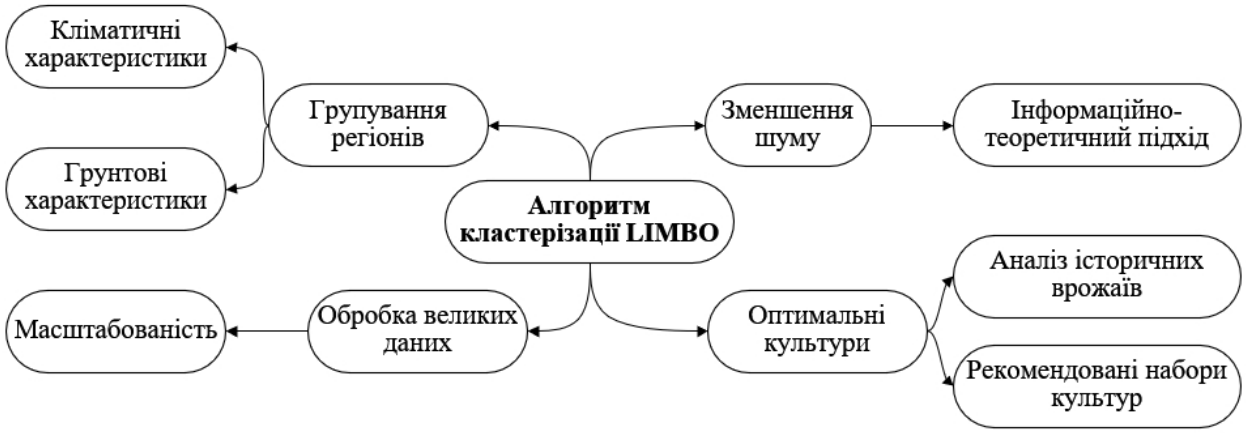


Рис.1.2 Використання алгоритму LIMBO для оптимізації вибору культур

Алгоритм кластеризації LIMBO виступає центральним компонентом процесу аналізу, що забезпечує формування однорідних груп регіонів на основі агроекологічних характеристик. Джерелами вхідної інформації для кластеризації є кліматичні та ґрунтові характеристики, які визначають умови вирощування сільськогосподарських культур. Групування регіонів за цими ознаками дає змогу виявляти схожі за агровиробничими умовами території та використовувати найуспішніші практики між ними. У процесі кластеризації LIMBO використовує інформаційно-теоретичний підхід, що дозволяє оцінювати вагомість кожного параметра, та методи зменшення шуму, що підвищують стійкість алгоритму до неповних або неточних даних. Додатковою перевагою алгоритму є здатність до масштабованої обробки великих масивів аграрної інформації, що є критично

важливим у практиці сільськогосподарського планування. У результаті виконання кластеризації система здійснює аналіз історичних даних про врожайність, визначає оптимальні культури для кожної групи регіонів та формує рекомендації щодо вирощування найбільш придатних культур у конкретних умовах.

Для аналізу придатності регіону до вирощування певних культур використовуються дані кліматичного, ґрунтового, економічного та агрономічного характеру. Вони включають як довгострокові показники (температура, опади, тип ґрунту), так і змінні параметри (врожайність у минулі роки, ринкові ціни, рівень зрошення) [5]. Такі дані надходять із різних джерел – статистичних, агрохімічних, економічних і виробничих – і підлягають уніфікації для подальшої кластеризації регіонів за подібними умовами вирощування. На рис. 1.3 зображено діаграму, яка відображає основні напрями формування інформаційної бази для прийняття аграрних рішень щодо вибору культур.

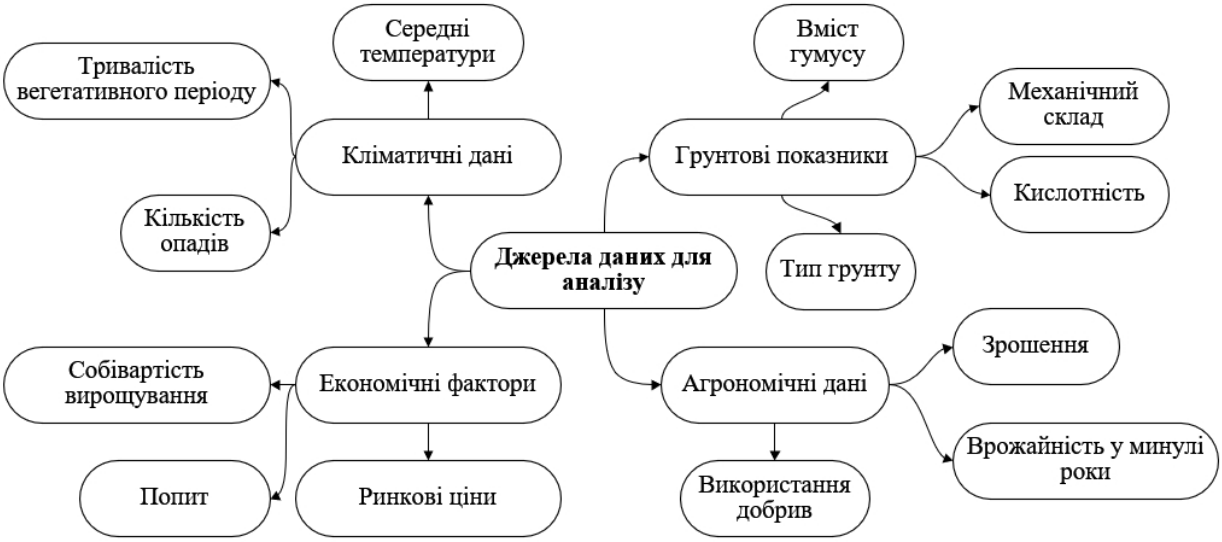


Рис.1.3 Джерела даних для аналізу

Кліматичні дані представлені такими важливими параметрами, як середні температури, кількість опадів і тривалість вегетаційного періоду. Ці показники визначають біокліматичну придатність території для вирощування певних видів культур і суттєво впливають на строки сівби, дозрівання та збирання врожаю.

Ґрунтові характеристики охоплюють тип ґрунту, його механічний склад, кислотність та вміст гумусу. Вони безпосередньо визначають здатність ґрунту забезпечити культуру необхідними поживними речовинами й водою. Окремо виділено економічні чинники, серед яких важливу роль відіграють ринкові ціни, попит на продукцію та витрати, пов’язані з вирощуванням – саме ці параметри формують економічну доцільність вирощування тієї чи іншої культури. Крім того, значну аналітичну вагу мають агрономічні дані, зокрема інформація про використання добрив, рівень зрошення та фактичну врожайність у попередні роки, що дозволяє враховувати практичний досвід господарств у подібних умовах.

Застосування алгоритму LIMBO для вибору оптимальних сільськогосподарських культур є перспективним напрямком у точному землеробстві. Це дозволить об’єднати наукові методи аналізу даних з практичними потребами агровиробників, забезпечуючи ефективні

## 1.2 Визначення вимог до системи прийняття рішень у сільському господарстві

Процес прийняття рішень у сучасному сільському господарстві дедалі більше залежить від комплексного аналізу агроекологічних, кліматичних та економічних факторів, які впливають на вибір оптимальних культур для вирощування в конкретному регіоні. З огляду на динамічні зміни клімату, зростання цін на ресурси та потребу в підвищенні продуктивності з мінімізацією ризиків, інформаційно-аналітична підтримка аграрних рішень набуває критичного значення.

Система, що розробляється, має виконувати функції інтелектуальної підтримки рішень у сфері аграрного планування на основі обробки й кластеризації регіональних характеристик. Тому в процесі проектування було виділено ключові функціональні та нефункціональні вимоги до системи, які наведено в таблиці 1.1.

Таблиця 1.1

Специфікація вимог до системи

| № | Тип вимоги | Вимога | Характеристика вимоги |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 1.1 | Функціональні | збір та інтеграція даних з різних джерел | Система має забезпечувати завантаження та зберігання аграрних характеристик регіонів, включаючи: назву регіону, тип ґрунту, кліматичну зону, кислотність ґрунту, глибину залягання вод, а також культуру, що вирощується в регіоні |
| 1.2 | | кластеризація регіонів за обраним атрибутом | Однією з основних функцій є побудова кластерів регіонів за вказаними ознаками: тип ґрунту, кліматична зона, кислотність, глибина залягання вод або регіон. Користувач самостійно обирає атрибут кластеризації, що дає змогу здійснювати аналіз залежно від контексту використання |
| 1.3 | | генерація звіту з рекомендаціями щодо культур | Система формує текстовий звіт, у якому для кожного кластера зазначаються регіони, які належать до нього, і культура, що є найпоширенішою серед цих регіонів. Це дозволяє ідентифікувати найбільш адаптовану культуру для умов, характерних для кластера |
| 2.1 | Нефункціональні | гнучкість системи | Система повинна мати можливість розширення – зокрема, щодо підтримки нових атрибутів кластеризації, додавання нових типів культур або регіонів без модифікації ядра логіки |
| 2.2 | | зрозумілий та інтуїтивно простий інтерфейс | Інтерфейс має бути орієнтований на користувача з аграрного сектору, що не обов’язково володіє технічними знаннями. Усі дії супроводжуються локалізованими підказками |
| 2.3 | | швидкодія при кластеризації | Оскільки кластеризація здійснюється на основі простого групування, система має забезпечувати обробку сотень записів за частки секунди, що задовольняє потреби середніх і малих господарств |

Таблиця 1.1 (продовження)

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 2.4 | | модульність реалізації | Бізнес-логіка повинна бути винесена в окремий клас, що забезпечує ізольованість обробки даних від інтерфейсу та полегшує тестування |
| 2.5 | | мінімальні апаратні вимоги | Програмне забезпечення повинно функціонувати на стандартних офісних комп'ютерах із 4 ГБ оперативної пам'яті, забезпечуючи прийнятну швидкодію навіть без потужних ресурсів |
| 2.6 | | надійність збереження результатів | Усі результати кластеризації можуть бути збережені у вигляді текстового звіту, що дозволяє агроному або аналітику зберігати аналітичні підсумки для звітності або подальшого аналізу |

Визначені вимоги гарантують, що система прийняття рішень у сільському господарстві буде не лише ефективною з погляду аналітики, а й зручною у використанні для аграріїв, науковців і фахівців з управління регіональним землекористуванням. Гнучка кластеризація на основі даних про клімат, ґрунт і агротехнічні умови дає змогу реалізовувати як адаптаційні стратегії до кліматичних змін, так і підвищувати врожайність на локальному рівні.

## 1.3 Моделювання інформаційної структури

Моделювання інформаційної структури системи є ключовим етапом при розробці програмного забезпечення, орієнтованого на підтримку прийняття аграрних рішень. Цей процес передбачає визначення основних інформаційних сутностей, що відображають предметну область, їхніх властивостей, а також логіки взаємодії між користувачем і функціональними компонентами системи. У нашому випадку інформаційна структура повинна охоплювати об'єкти, пов'язані з аграрними регіонами, кліматичними умовами, типами ґрунтів, сільськогосподарськими культурами та відповідними характеристиками, що впливають на вибір культур. Особливу увагу також приділено ролям

користувачів та можливості розмежування прав доступу відповідно до функціонального призначення системи.

Для формалізації взаємодії між користувачами та системою було визначено перелік варіантів використання, які описують типові сценарії роботи з інформаційними об'єктами та сервісними функціями. У табл. 1.2 наведено детальний опис таких варіантів використання, які охоплюють як адміністративну, так і аналітичну складову функціональності системи.

Таблиця 1.2

Опис варіантів використання системи

| № | Ім’я | Опис |
|---|---|---|
| 1 | 2 | 3 |
| UC1 | Реєстрація облікового запису | Дозволяє новому користувачу зареєструватися в системі шляхом введення персональних даних та облікової інформації |
| UC2 | Авторизація в системі | Надає можливість користувачу увійти до системи з використанням логіна та пароля для доступу до функціоналу відповідно до ролі |
| UC3 | Перегляд користувачів системи | Дає адміністратору можливість отримати повний список зареєстрованих користувачів із доступом до профілю кожного |
| UC4 | Додавання користувача | Дозволяє адміністратору створити новий обліковий запис для іншого користувача без необхідності самореєстрації |
| UC5 | Редагування профілю користувача | Дозволяє змінювати персональні дані або облікову інформацію зареєстрованих користувачів системи |
| UC6 | Перегляд наявних регіонів | Дозволяє отримати список усіх регіонів, що використовуються в аналітичних або кластеризаційних модулях системи |
| UC7 | Додавання регіону | Забезпечує можливість внесення нового регіону до бази з відповідною назвою та описом |
| UC8 | Оновлення інформації про регіон | Дозволяє змінювати назву або опис наявного регіону в системі |
| UC9 | Перегляд типів ґрунтів | Дає змогу отримати перелік ґрунтів, які використовуються для характеристики агрорегіонів |

Таблиці 1.2 (продовження)

| 1 | 2 | 3 |
|---|---|---|
| UC10 | Додавання нового типу ґрунту | Забезпечує введення нового типу ґрунту до бази даних із назвою та описом |
| UC11 | Редагування типу ґрунту | Дозволяє модифікувати інформацію про наявний тип ґрунту |
| UC12 | Перегляд кліматичних зон | Відображає перелік кліматичних зон, які асоціюються з відповідними регіонами |
| UC13 | Додавання кліматичної зони | Дозволяє внести нову кліматичну зону до довідника системи |
| UC14 | Редагування кліматичної зони | Дозволяє змінити назву або опис раніше доданої кліматичної зони |
| UC15 | Перегляд сільськогосподарських культур | Надає можливість переглядати всі культури, що використовуються у системі як об’єкти рекомендації |
| UC16 | Додавання сільськогосподарської культури | Дозволяє внести нову культуру до системи з відповідним описом |
| UC17 | Редагування культури | Забезпечує можливість оновлення назви або опису вже доданої культури |
| UC18 | Перегляд характеристик регіонів | Відображає повну таблицю регіональних характеристик, яка включає ґрунт, клімат, кислотність, глибину води та культуру |
| UC19 | Додавання характеристики регіону | Дозволяє внести повну характеристику конкретного регіону із прив’язкою до ґрунту, клімату та культури |
| UC20 | Редагування характеристики регіону | Дозволяє оновити наявну характеристику для регіону, змінюючи будь-який з параметрів |
| UC21 | Кластеризація та вибір оптимальних культур | Дозволяє провести кластеризацію регіонів за обраним атрибутом та отримати рекомендації щодо найдоцільніших культур |
| UC22 | Перегляд журналу подій | Дозволяє адміністраторам та аналітикам переглядати події, дії користувачів або зміни в системі у вигляді логів |

На основі сформульованих функціональних вимог до інформаційної системи підтримки вибору оптимальних сільськогосподарських культур було розроблено діаграми варіантів використання (use-case діаграми) для двох основних ролей: адміністратора та користувача. Відповідні діаграми наведено на рис. 1.4 та рис. 1.5.

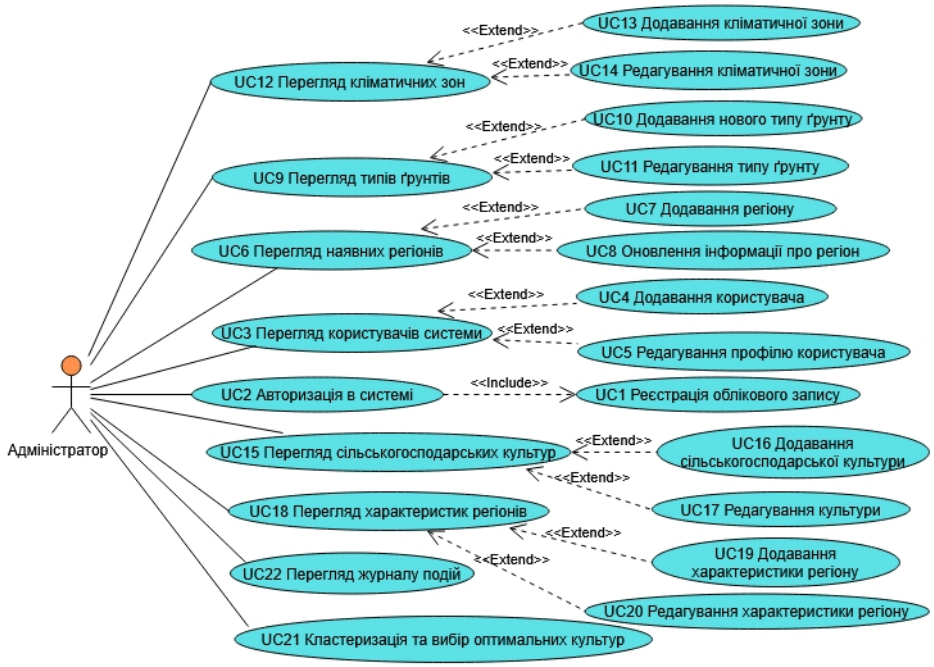


Рис.1.4 Діаграма варіантів використання для ролі адміністратора

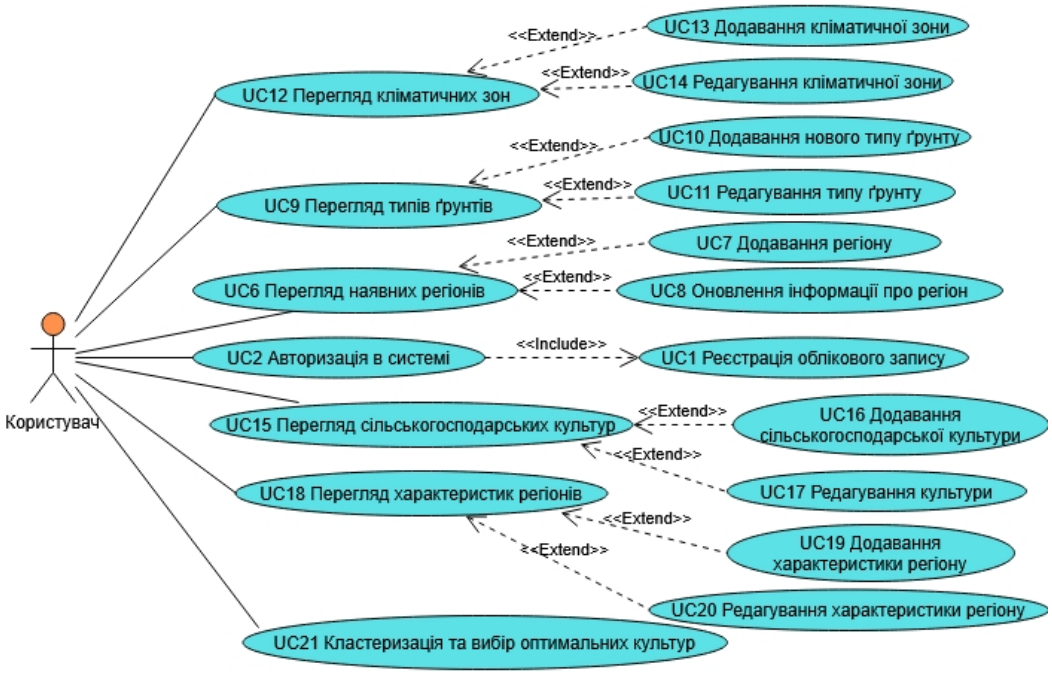


Рис.1.5 Діаграма варіантів використання для ролі користувача

Діаграми варіантів використання відображають основні сценарії взаємодії користувачів із системою вибору культур, з урахуванням їхніх прав доступу. Для адміністратора передбачено розширені можливості управління довідковою інформацією (регіони, типи ґрунтів, кліматичні зони, культури), а також повний контроль над характеристиками регіонів, обліковими записами користувачів та результатами кластеризації. Адміністратор також має доступ до журналу подій, що дозволяє здійснювати моніторинг активності системи.

Зі свого боку, звичайний користувач системи має змогу виконувати базові дії: реєстрацію, авторизацію, перегляд доступної інформації, додавання або редагування культур і характеристик регіонів, а також запуск процесу кластеризації та перегляд отриманих рекомендацій. Вказані варіанти використання охоплюють повний життєвий цикл роботи з даними в системі та забезпечують структурований підхід до підтримки аграрних рішень. Структура діаграм дозволяє чітко відобразити логіку взаємодії між користувачами й функціональними модулями системи, що є основою для подальшої реалізації інтерфейсів і сервісів.

## 1.4 Аналіз існуючих програмних рішень для аграрного прогнозування

Сучасний аграрний сектор дедалі активніше впроваджує цифрові технології для оптимізації виробничих процесів, серед яких особливе місце займають системи аграрного прогнозування. Такі програмні рішення дозволяють автоматизувати процеси прийняття рішень щодо вибору культур, строків посіву, прогнозування врожайності та оцінки агроекологічних ризиків. Їхнє використання дає змогу підвищити ефективність господарської діяльності, скоротити втрати ресурсів і забезпечити адаптацію до змін кліматичних умов.

Проте, попри значну кількість існуючих інструментів, переважна більшість з них орієнтована на велике виробництво, потребує глибоких технічних знань або спеціального обладнання, а також базується на складних

моделей прогнозування з великою кількістю вхідних параметрів. Крім того, багато з них не враховують локальних особливостей ґрунтів і клімату, що знижує точність рекомендацій у межах окремих регіонів. У зв'язку з цим виникає потреба у створенні адаптивного, інтуїтивно зрозумілого та масштабованого рішення, здатного враховувати комбінацію агроекологічних параметрів та історичних даних про врожайність.

Agremo – це хмарне програмне забезпечення для аграрної аналітики, яке спеціалізується на обробці аерофотознімків, отриманих із дронів, з метою оцінювання стану посівів, прогнозування врожайності та виявлення проблем на полях (рис. 1.6). Основною метою системи є надання аграріям, агрономам та компаніям засобів для прийняття обґрунтованих рішень на основі візуальних даних і аналітичних моделей [6].


Рис.1.6 Приклад інтерфейсу системи «Agremo» [7]

Архітектурно Agremo реалізоване як сервісно-орієнтована платформа з модульною структурою, де фронтенд реалізовано у вигляді веб-інтерфейсу з доступом через браузер, а бекенд функціонує на базі хмарних обчислень з використанням інструментів для зберігання, обробки та машинного аналізу зображень. Система інтегрується з платформами дронів, дозволяє завантажувати геопросторові знімки, виконує їх сегментацію, аналіз вегетаційних індексів

(NDVI, NDRE та інші) і формує звіти у вигляді карт розподілу продуктивності, зон з проблемами та очікуваної врожайності. Компоненти взаємодіють через API, що забезпечує масштабованість, гнучке розширення та підтримку різних форматів вхідних даних.

Agremo є потужним та сучасним інструментом для аграрного аналізу на основі зображень з дронів. Він ефективно вирішує задачі моніторингу стану посівів та прогнозування врожайності. Але його застосування доцільне передусім у господарствах із доступом до авіазнімків і належною технічною інфраструктурою. Для систем аграрного прогнозування, орієнтованих на ширший спектр вхідних даних, Agremo може виступати як допоміжний візуалізаційний модуль, але не як універсального рішення.

FieldView – це комплексне цифрове рішення, розроблене компанією Climate Corporation, що призначене для моніторингу, аналізу та оптимізації агровиробничих процесів (рис. 1.7).


Рис.1.7 Приклад інтерфейсу системи «FieldView» [11]

Основне призначення застосунку полягає в інтеграції даних з техніки, сенсорів і супутникових джерел для надання аграріям рекомендацій щодо

управління посівами, внесення добрив, захисту рослин та збору врожаю [10]. Система дозволяє аналізувати продуктивність полів у динаміці, виявляти аномалії та будувати агротехнічні стратегії на основі просторових і часових закономірностей.

Архітектурно FieldView реалізований як хмарна платформа з мобільним і веб-інтерфейсами, що забезпечують доступ до даних у реальному часі. Система має багаторівневу структуру, яка включає локальний збір даних через FieldView Drive – пристрій, що підключається до обладнання на техніці та передає дані в хмарне середовище. Бекенд обробляє просторові карти, сенсорні потоки, погодні дані й аналітику врожайності, інтегруючи їх у єдину аналітичну панель. Інтерфейс користувача дозволяє переглядати карти полів, проводити зонування, порівнювати різні агрономічні сценарії та приймати рішення щодо оптимізації агротехнічних заходів.

FieldView є також потужною платформою аграрного моніторингу, орієнтованою на великих виробників, які використовують сучасну техніку та цифрову інфраструктуру. Застосунок надає глибокий аналітичний функціонал для управління агротехнологіями та прийняття рішень на основі польових даних. Проте для фермерів, які не мають доступу до відповідного обладнання або працюють у специфічних регіональних умовах, його використання може бути частково обмеженим або економічно недоцільним.

Agrian – це багатофункціональна платформа для аграрного документообігу, агрохімічного управління та агрономічного планування, що призначена для підтримки повного циклу ведення господарства – від рекомендацій щодо засобів захисту рослин до складання звітів відповідно до нормативних вимог (рис. 1.8). Система орієнтована як на агрономів-консультантів, так і на виробників, постачальників агрохімікатів і аграрні кооперативи [14]. Основне її призначення – забезпечити точне ведення аграрної документації, облік польових операцій, контроль за агрохімічним навантаженням та ефективну комунікацію між усіма учасниками аграрного ланцюга.

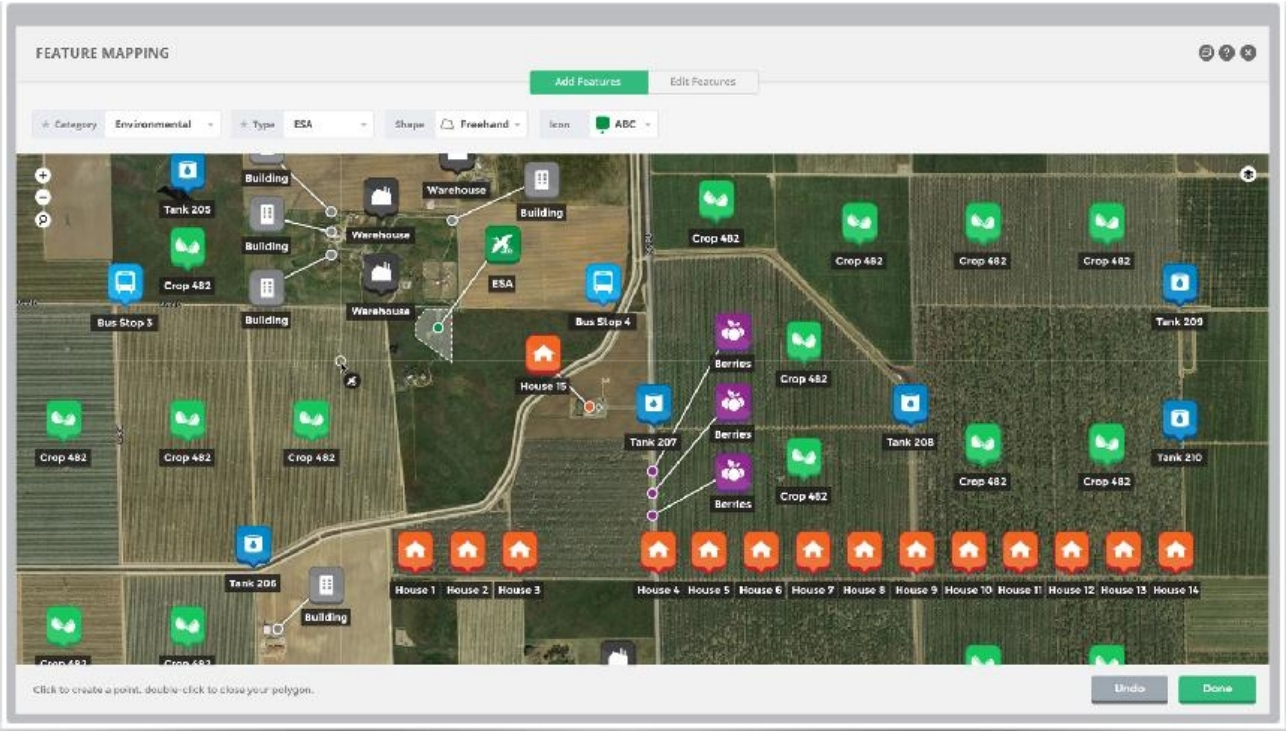


Рис.1.8 Приклад інтерфейсу системи «Agrian» [15]

Архітектура Agrian побудована на основі хмарної моделі з централізованою базою даних, яка синхронізується між десктопними й мобільними додатками. Бекенд системи забезпечує логіку формування агрономічних рекомендацій, перевірку відповідності регуляторним вимогам та створення інтегрованих звітів. Інтерфейс реалізований у вигляді веб-порталу з окремими модулями для агрономічного планування, розрахунків норм внесення ЗЗР, фіксації операцій на полі та складання звітної документації. Завдяки сервісній модульності Agrian легко адаптується до специфіки господарства та підтримує спільну роботу між агрономами, постачальниками та фермерами.

Agrian – це ефективне рішення для централізованого аграрного документообігу та агрохімічного менеджменту, орієнтоване на виробників, які ведуть точний облік усіх агроопераций і дотримуються нормативних вимог. Завдяки широким функціям у сфері агрономічного планування та звітності, система особливо корисна для корпоративного сегменту. Водночас її застосування може бути обмеженим для фермерів, які не потребують глибокого регуляторного контролю або працюють в інших законодавчих юрисдикціях.

Таблиця 1.3

Порівняльний аналіз аналогів системи

| № | Засто-сунки | Переваги | Недоліки |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 1 | Agremo | підтримка аналізу даних із дронів, що дозволяє оперативно оцінювати стан посівів на великих площах [8] | – залежність від наявності якісних зображень з дронів, що обмежує застосування у господарствах без відповідного обладнання |
| | | інтуїтивно зрозумілий веб-інтерфейс, доступний без встановлення на локальний ПК | висока чутливість до погодних умов при зборі даних, особливо в періоди з хмарністю або дощем |
| | | автоматична побудова аналітичних звітів із геопросторовими візуалізаціями та вегетаційними індексами | фокус лише на візуальних параметрах – система не враховує повноцінно агрохімічні чи економічні дані [9] |
| | | можливість інтеграції з платформами управління господарством та іншими зовнішніми сервісами через API | платна основа з підписною моделлю, що може бути дорогою для дрібних фермерських господарств |
| 2 | FieldView | широка інтеграція з технікою та обладнанням через пристрій FieldView Drive, що дозволяє збирати точні польові дані в реальному часі | потребує спеціального обладнання для повноцінного збору даних, що обмежує його застосування без сумісної техніки; |
| | | підтримка аналізу просторових і часових закономірностей для побудови зон управління та оптимізації технологічних операцій [12] | платна підписка з обмеженою функціональністю у безкоштовній версії |
| | | доступність з мобільних і десктопних пристроїв із синхронізацією даних через хмару | менша орієнтація на локальні особливості агровиробництва, особливо в країнах поза США та Канадою [13] |

Таблиця 1.3 (продовження)

| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 2 | FieldView | доступність з мобільних і десктопних пристроїв із синхронізацією даних через хмару | менша орієнтація на локальні особливості агровиробництва, особливо в країнах поза США та Канадою [13] |
| | | можливість порівняльного аналізу полів і сезонів, що сприяє стратегічному плануванню на основі історичних даних | інтерфейс може виявитися перевантаженим для користувачів без досвіду роботи з ГІС та аграрною аналітикою |
| 3 | Agrian | забезпечує повний цикл аграрного документообігу – від агрономічного планування до формування регуляторних звітів [16] | зосередженість на нормативній і хімічній стороні агробізнесу, без глибокої інтеграції біофізичних моделей |
| | | підтримує хмарну синхронізацію між усіма користувачами та пристроями, що спрощує колективну роботу | обмежена підтримка локальних нормативів поза межами США та Канади |
| | | містить актуальну базу даних засобів захисту рослин із автоматичною перевіркою на відповідність нормам | відносно складний інтерфейс для нових користувачів, які не мають досвіду у веденні агрономічної документації |
| | | гнучка модульна архітектура дозволяє адаптувати систему під потреби різних типів господарств | частина функціоналу доступна лише за підпискою, що може бути фінансово невигідно для дрібних фермерів |

Аналіз існуючих програмних рішень показав, що більшість із них орієнтовані на великотоварні господарства, мають складну структуру, потребують спеціального обладнання або передбачають оплату підписки. У багатьох випадках вони не враховують локальних особливостей ґрунтів і клімату, що обмежує їхню практичну цінність для малих і середніх господарств, особливо в умовах України. У зв’язку з цим доцільною є розробка простого, доступного та безкоштовного програмного забезпечення з базовим

функціоналом, який охоплює лише ключові аналітичні можливості – кластеризацію регіонів за агроекологічними характеристиками та формування рекомендацій щодо культур. Таке рішення легко впроваджується, не потребує складної технічної інфраструктури й адаптоване до потреб фермерів, які прагнуть ефективно використовувати доступні ресурси без надмірної складності.

## 1.5 Постановка завдання

У контексті актуалізації цифрових інструментів для аграрного планування виникає потреба в розробці прикладного рішення, що дозволяє обґрунтовано визначати оптимальні сільськогосподарські культури для вирощування з урахуванням реальних агроекологічних умов регіонів. Більшість існуючих систем орієнтовані на складну аналітику або потребують дорогого обладнання, тоді як у малих та середніх господарствах на перший план виходять простота, доступність і практична доцільність рішень. З огляду на це, метою даної роботи є розробка інтелектуального програмного забезпечення, що реалізує процес кластерного аналізу характеристик регіонів для подальшого формування рекомендацій щодо вибору культур. Ключовою особливістю рішення стане застосування алгоритму LIMBO, який дозволяє ефективно працювати з якісними ознаками – такими як тип ґрунту, кліматична зона, кислотність, глибина залягання вод – та об’єднувати регіони в смислові кластери.

У межах поставленої задачі буде реалізовано наступні компоненти:

– ведення довідкової інформації, яка включає створення функціоналу для перегляду, додавання та редагування даних про регіони, типи ґрунтів, кліматичні зони та культури;

– реєстрація та авторизація користувачів, що забезпечує доступу до системи для різних ролей із відповідними правами, а також адміністративне управління обліковими записами;

– формування та обробка характеристик регіонів, яка є однією з ключових функцій системи і забезпечує створення агроекологічного профілю

регіону на основі множини ознак (тип ґрунту, кліматична зона, кислотність ґрунту, глибина залягання ґрунтових вод та культура, що вирощувалась), має інтерфейс введення даних з можливістю вибору з довідників, а також редагування існуючих характеристик, що є базовими об’єктами для аналізу та використовуються в кластеризаційній процедурі. Усі зміни фіксуються, щоб забезпечити прозорість і збереження історії оновлень;

– модуль кластерного аналізу, що використовує алгоритм LIMBO для автоматичного групування регіонів за схожістю ознак є центральним аналітичним компонентом. LIMBO забезпечує обробку якісних і категоріальних даних, виявляючи найбільш інформативні атрибути, що дозволяє будувати змістовні кластери навіть за наявності неповних або нечітких значень. Після виконання кластеризації користувач повинен отримувати набір рекомендованих культур для кожного кластеру на основі аналізу історичних даних. Результати подаються у вигляді узагальненого текстового звіту з можливістю збереження.

– ведення журналу подій забезпечує прозорість і контроль при збереженні інформації про дії користувачів та основні події в системі.

Програмне забезпечення буде реалізоване з використанням мови C# у середовищі Windows Forms із підключенням до реляційної бази даних Microsoft SQL Server, що забезпечить простоту впровадження, масштабованість і стійкість до збоїв. Готова система дозволить аграріям без спеціальних знань у сфері аналітики отримувати обґрунтовані рекомендації щодо вибору культур, підвищуючи ефективність використання земельних ресурсів за мінімальних витрат.

# 2 ПРОЄКТУВАННЯ СИСТЕМИ ВИБОРУ КУЛЬТУР НА ОСНОВІ КЛАСТЕРНОГО АНАЛІЗУ

## 2.1 Розробка логічної моделі аграрних даних

Функціонування системи вибору оптимальних сільськогосподарських культур неможливе без чітко структурованої інформаційної основи, яка забезпечує надійне зберігання, обробку та взаємозв'язок агроекологічних даних. Для досягнення цієї мети необхідно сформувати логічну модель бази даних, яка відображатиме предметну область та підтримає основні бізнес-процеси системи. Логічна модель визначає основні сутності, їх атрибути та відношення між ними, забезпечуючи цілісність даних і оптимізацію доступу до інформації.

У межах розробки моделі бази даних було враховано особливості зберігання даних про регіони, типи ґрунтів, кліматичні зони, сільськогосподарські культури, а також характеристики регіонів із агроекологічними параметрами. Окрему увагу приділено організації даних про користувачів системи та їх активність, що забезпечує реалізацію функціональних вимог щодо реєстрації, авторизації та ведення журналу подій.

Таблиця Region призначена для зберігання інформації про регіони, що використовуються у системі для характеристики агроекологічних умов. Вона містить базову інформацію про найменування регіону та його описову характеристику (табл. 2.1).

Таблиця 2.1

Атрибути сутності «Region»

| № | Найменування | Тип даних | Призначення |
|---|---|---|---|
| 1 | RegionId | INT (PK) | Ідентифікатор регіону |
| 2 | RegionName | NVARCHAR(200) | Назва регіону |
| 3 | Description | NVARCHAR(MAX) | Опис регіону, що може містити додаткову інформацію |

Таблиця SoilType призначена для зберігання інформації про типи ґрунтів, які використовуються при описі агроекологічних характеристик регіонів у системі. Вона включає назви ґрунтів та їх додаткові описи, що допомагають точніше класифікувати умови вирощування культур (табл. 2.2).

Таблиця 2.2

Атрибути сутності «SoilType»

| № | Найменування | Тип даних | Призначення |
|---|---|---|---|
| 1 | SoilTypeId | INT (PK) | Ідентифікатор типу ґрунту |
| 2 | SoilTypeName | NVARCHAR(200) | Назва типу ґрунту |
| 3 | Description | NVARCHAR(MAX) | Додатковий опис типу ґрунту |

Таблиця ClimateZone призначена для зберігання даних про кліматичні зони, що впливають на аграрні умови вирощування сільськогосподарських культур. Вона містить інформацію про найменування зони та її загальні характеристики (табл. 2.3).

Таблиця 2.3

Атрибути сутності «ClimateZone»

| № | Найменування | Тип даних | Призначення |
|---|---|---|---|
| 1 | ClimateZoneId | INT (PK) | Ідентифікатор кліматичної зони |
| 2 | ClimateZoneName | NVARCHAR(200) | Назва кліматичної зони |
| 3 | Description | NVARCHAR(MAX) | Опис особливостей кліматичної зони |

Таблиця Crop призначена для зберігання інформації про сільськогосподарські культури, які можуть бути рекомендовані для вирощування в певних агроекологічних умовах. Вона містить базові дані про назви культур та їх додаткові характеристики (табл. 2.4).

Таблиця 2.4

Атрибути сутності «Crop»

| № | Найменування | Тип даних | Призначення |
|---|---|---|---|
| 1 | CropId | INT (PK) | Ідентифікатор культури |
| 2 | CropName | NVARCHAR(200) | Назва сільськогосподарської культури |

| | | | |
|---|---|---|---|
| 3 | Description | NVARCHAR(MAX) | Опис культури та її особливостей |

Таблиця RegionCharacteristics призначена для зберігання інформації про агроекологічні характеристики регіонів у поєднанні з вирощуваними культурами. Вона пов’язує регіони, типи ґрунтів, кліматичні зони та культури, доповнюючи ці дані такими параметрами, як кислотність ґрунту і глибина залягання вод, що дозволяє проводити комплексний аналіз умов вирощування (табл. 2.5).

Таблиця 2.5

Атрибути сутності «RegionCharacteristics»

| № | Найменування | Тип даних | Призначення |
|---|---|---|---|
| 1 | RegionCharId | INT (PK) | Ідентифікатор характеристики регіону |
| 2 | RegionId | INT (FK) | Посилання на регіон у таблиці Region |
| 3 | SoilTypeId | INT (FK) | Посилання на тип ґрунту в таблиці SoilType |
| 4 | ClimateZoneId | INT (FK) | Посилання на кліматичну зону в таблиці ClimateZone |
| 5 | CropId | INT (FK) | Посилання на сільськогосподарську культуру |
| 6 | SoilAcidity | NVARCHAR(200) | Опис кислотності ґрунту |
| 7 | WaterDepth | NVARCHAR(200) | Опис глибини залягання вод |

Таблиця Users призначена для зберігання даних про користувачів системи, які мають доступ до її функціоналу. Вона містить інформацію про персональні дані користувачів, облікові записи, роль у системі та контактні відомості (табл. 2.6).

Таблиця 2.6

Атрибути сутності «Users»

| № | Найменування | Тип даних | Призначення |
|---|---|---|---|
| 1 | UsersId | INT (PK) | Ідентифікатор користувача |
| 2 | FirstName | NVARCHAR(60) | Ім'я користувача |
| 3 | LastName | NVARCHAR(60) | Прізвище користувача |
| 4 | UsersName | NVARCHAR(60) | Логін користувача для авторизації |
| 5 | UsersPassword | NVARCHAR(250) | Захешований пароль користувача |
| 6 | RoleId | INT | Ідентифікатор ролі користувача в системі |
| 7 | Description | NVARCHAR(1200) | Додаткова інформація або опис користувача |

| | | | |
|---|---|---|---|
| 8 | Email | NVARCHAR(150) | Електронна адреса користувача |

Таблиця Logs призначена для реєстрації важливих подій та дій, що виконуються користувачами в системі. Вона дозволяє здійснювати моніторинг активності та забезпечувати інформаційну безпеку за допомогою журналювання операцій (табл. 2.7).

Таблиця 2.7

Атрибути сутності «Logs»

| № | Найменування | Тип даних | Призначення |
|---|---|---|---|
| 1 | LogsId | INT (PK) | Ідентифікатор запису події |
| 2 | UsersId | INT | Ідентифікатор користувача, який виконав дію |
| 3 | EventNameShow | NVARCHAR(MAX) | Опис події або дії, яка була здійснена |
| 4 | EventDate | DATETIME | Дата та час виконання події |

Для забезпечення цілісності та взаємозв'язку аграрних даних між сутностями системи була розроблена логічна модель бази даних. Побудова моделі здійснювалась на основі проаналізованих таблиць, що охоплюють регіони, типи ґрунтів, кліматичні зони, сільськогосподарські культури, характеристики регіонів, користувачів і журнал подій. Відповідна логічна модель представлена на рис. 2.1 у нотації Чена.

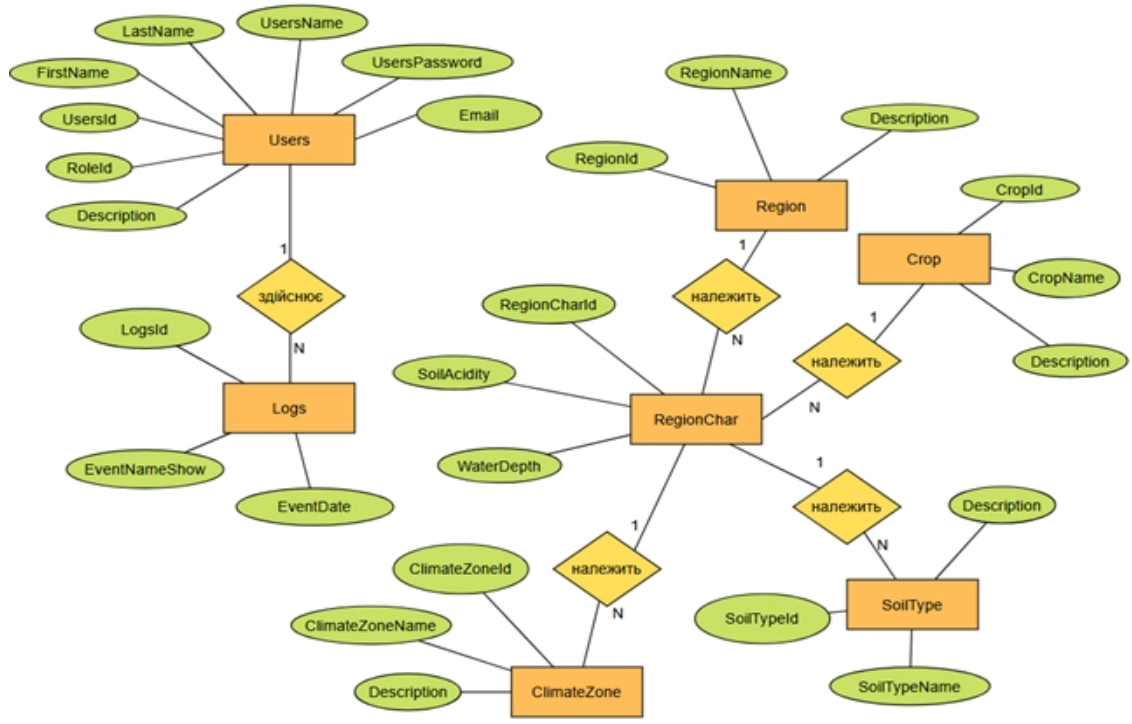

Рис.2.1 Логічна модель бази даних у нотації Чена

Побудована логічна модель чітко визначає основні сутності системи, їх атрибути та зв'язки, що дозволяє забезпечити узгоджене зберігання даних і підтримку бізнес-процесів. Модель є основою для фізичної реалізації бази даних і подальшої розробки програмних компонентів системи.

## 2.2 Діаграма класів та кооперацій

Для забезпечення структурованого опису архітектури програмного забезпечення важливим кроком є побудова діаграми класів та кооперацій. Діаграма класів відображає основні компоненти системи, їхню внутрішню організацію, взаємозв'язки, а також розподіл обов'язків між модулями. На рис. 2.2 подано узагальнену логічну модель взаємодії між основними модулями програмного забезпечення, призначеного для вибору оптимальних сільськогосподарських культур на основі кластерного аналізу.

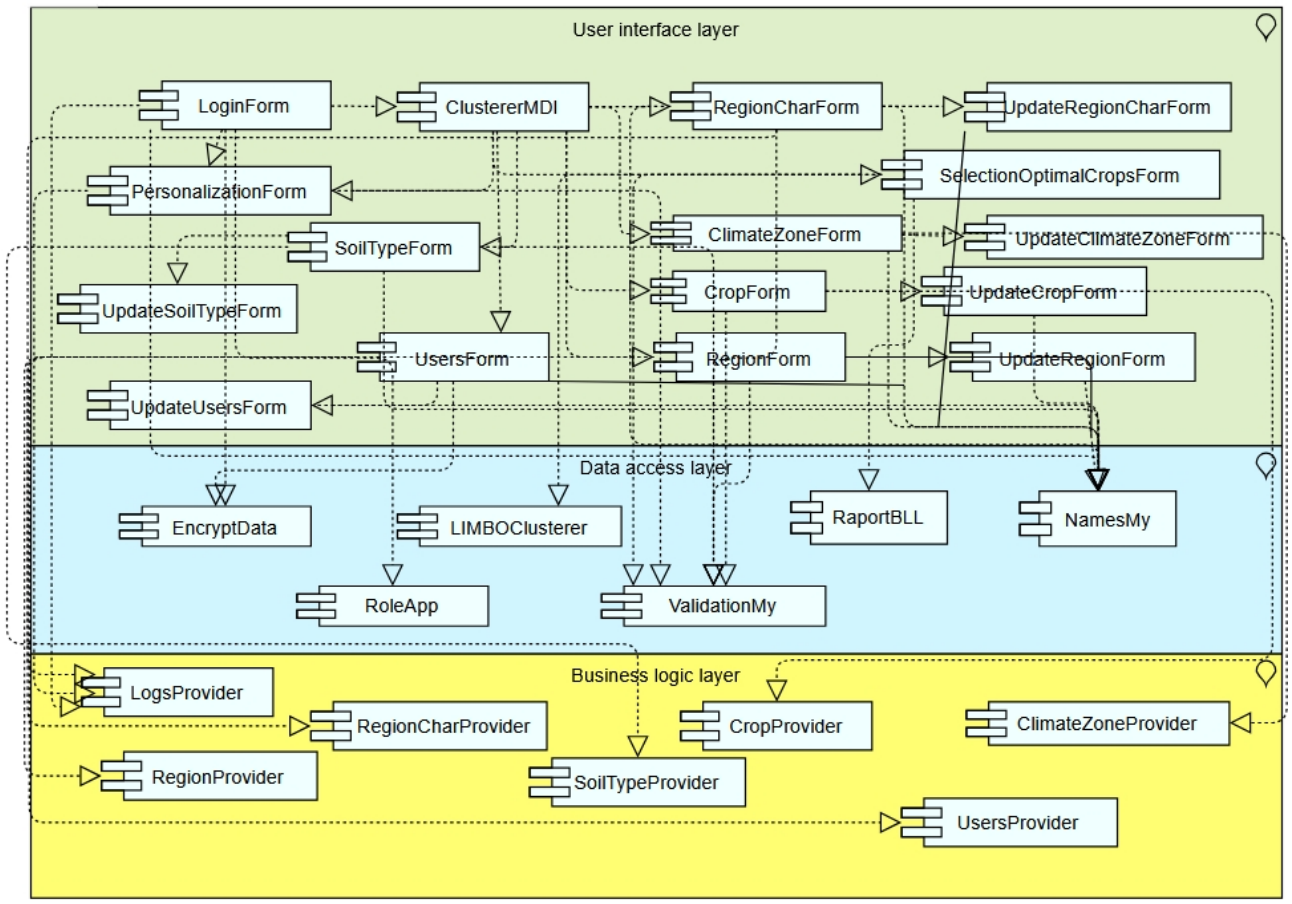

Рис.2.2 Узагальнена схема логічної структури взаємодії між модулями системи

Дана схема відображає трирівневу архітектуру системи, поділену на шар інтерфейсу користувача, шар доступу до даних і шар бізнес-логіки. На рівні інтерфейсу користувача представлені форми для авторизації, персоналізації профілю, перегляду та редагування довідкової інформації про регіони, культури, ґрунти й кліматичні зони, а також форми для кластеризації та вибору оптимальних культур. Усі форми взаємодіють із відповідними сервісами бізнес-логіки через шар доступу до даних.

Шар доступу до даних включає класи шифрування даних, кластеризації LIMBO, валідації введених даних, а також допоміжні модулі для роботи зі звітністю та управління ролями користувачів. Цей шар виступає посередником між інтерфейсом і бізнес-логікою, забезпечуючи цілісність даних і правильність їх обробки.

Шар бізнес-логіки представлений провайдерами, кожен з яких відповідає за окрему сутність системи: регіони, типи ґрунтів, кліматичні зони, культури, характеристики регіонів, користувачів і журнал подій. Провайдери реалізують основні операції із даними, включаючи створення, оновлення, видалення та отримання інформації. Завдяки такій структурі система є легко масштабованою і може бути розширена додатковими функціями без зміни існуючої архітектури.

З метою деталізації реалізації внутрішньої структури програмного забезпечення на рис. 2.3 подано класичну UML-діаграму класів, що охоплює ключові компоненти системи та їх взаємозв’язки.

Діаграма структурована на три логічні рівні: інтерфейс користувача, бізнес-логіка, шар доступу до даних. Вона демонструє об’єкти, їх атрибути, методи та зв’язки між класами.

На рівні інтерфейсу користувача (User Interface Layer) представлено форми керування й введення даних, зокрема SelectionOptimalCropsForm, яка реалізує метод ClusterAnalysisByAttribute() та ініціює звернення до бізнес-логіки.

На рівні бізнес-логіки (Business Logic Layer) клас RegionClusterer відповідає за кластеризацію на основі атрибутів регіональних даних. Клас RaportBLL забезпечує отримання даних з бази.

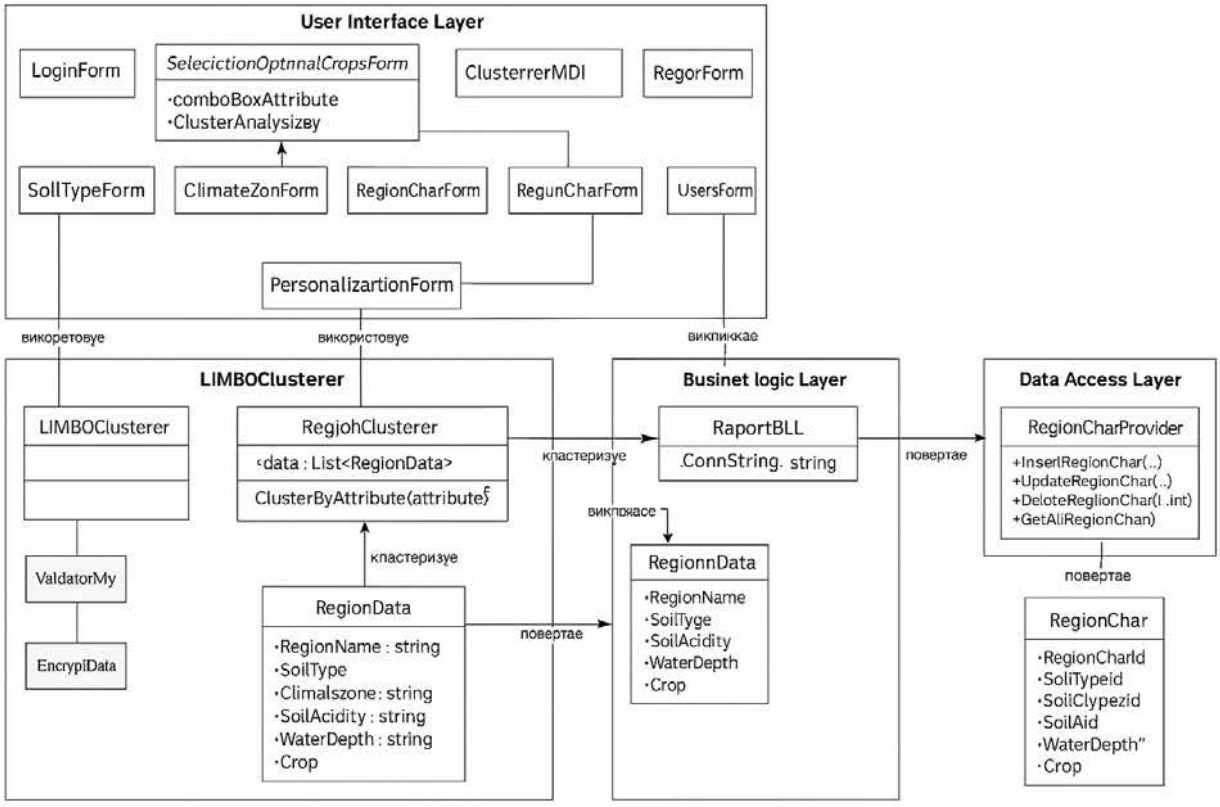


Рис. 2.3 – UML-діаграма класів інформаційної системи вибору оптимальних сільськогосподарських культур

На рівні доступу до даних (Data Access Layer) розміщено клас RegionCharProvider, який виконує операції читання та модифікації базових характеристик регіонів.

Діаграма підкреслює залежності між компонентами, реалізацію інкапсуляції та принципів багаторівневої архітектури.

Одним із важливих інструментів опису архітектури взаємодії між компонентами системи є діаграма кооперацій. Вона слугує для наочного відображення послідовності обміну повідомленнями між об'єктами в рамках реалізації певного сценарію використання. За допомогою діаграми кооперацій можна детально простежити, як саме об'єкти взаємодіють між собою під час

виконання прикладної логіки, які методи викликаються, у якій послідовності відбувається передача даних та яка роль кожного об'єкта у загальному процесі.

У межах розробки програмного забезпечення для вибору оптимальних сільськогосподарських культур діаграма кооперацій застосовується для моделювання взаємодії компонентів під час виконання кластеризації регіонів. Її побудова дозволяє зафіксувати взаємозв'язок між елементами інтерфейсу користувача, модулями обробки даних і кластеризаційним алгоритмом, що в подальшому забезпечує чіткість і передбачуваність роботи системи. Крім того, діаграма допомагає виявити можливі вузькі місця в архітектурі ще на етапі проєктування та підвищити узгодженість розробки всіх компонентів.

На рис. 2.4 представлено діаграму кооперацій для процесу кластеризації регіонів за обраним атрибутом і формування аналітичного звіту.

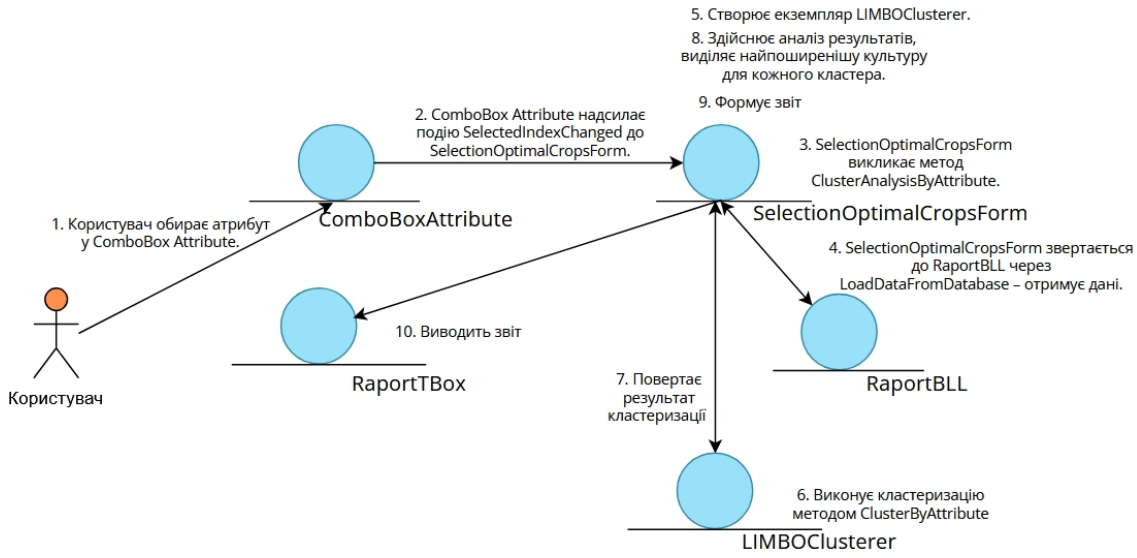


Рис.2.4 Діаграма кооперацій процесу кластеризації за атрибутом

Користувач обирає потрібний атрибут для кластеризації у компоненті ComboBoxAttribute. Після вибору атрибута генерується подія SelectedIndexChanged, яка надсилається до форми SelectionOptimalCropsForm. У відповідь форма викликає метод ClusterAnalysisByAttribute, який організовує послідовність операцій для аналізу даних. Для підготовки інформації SelectionOptimalCropsForm звертається до модуля RaportBLL за допомогою методу LoadDataFromDatabase, отримуючи агроекологічні дані регіонів.

Після завантаження даних створюється екземпляр кластеризатора LIMBOClusterer, який виконує кластеризацію за обраним атрибутом за допомогою методу ClusterByAttribute. Повернуті результати кластеризації обробляються у SelectionOptimalCropsForm, де здійснюється аналіз кожного кластеру та формування узагальненого звіту. Сформований текстовий звіт передається до текстового вікна RaportTBox, де відображається для перегляду користувачем.

## 2.3 Діаграма пакетів

Структуризація програмного забезпечення за пакетами є важливим аспектом проектування системи, що забезпечує логічний поділ функціональності, покращує читабельність коду та полегшує супровід проєкту. Діаграма пакетів використовується для відображення модульної організації системи, показуючи, як основні компоненти групуються за функціональним призначенням та як відбувається взаємодія між ними. На рис. 2.5 наведено діаграму пакетів розробленої системи вибору оптимальних сільськогосподарських культур.

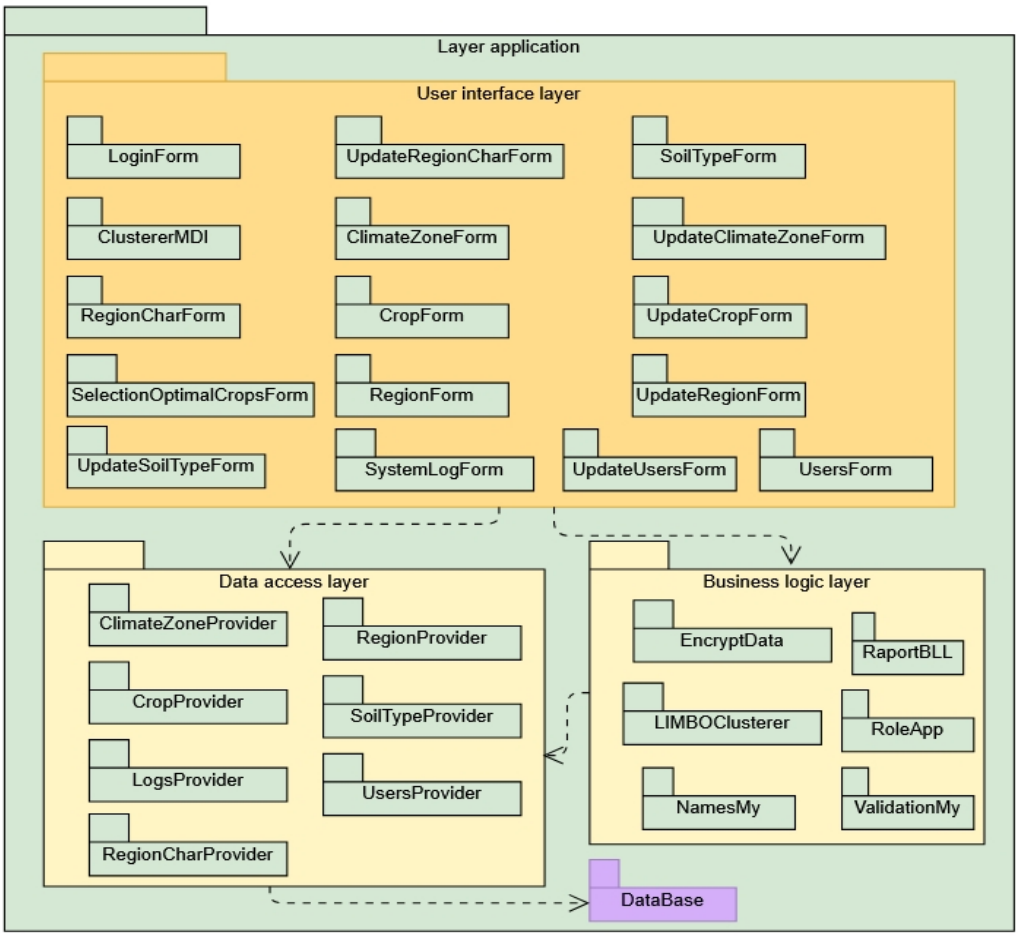


Рис.2.5 Діаграма пакетів системи

Діаграма відображає багаторівневу архітектуру системи, поділену на три основні пакети: User Interface Layer, Data Access Layer та Business Logic Layer, кожен із яких виконує окремі функції. Пакет User Interface Layer об'єднує форми користувацького інтерфейсу, що відповідають за реєстрацію та авторизацію користувачів, управління довідковою інформацією про регіони, типи ґрунтів, кліматичні зони, культури, перегляд журналу подій, а також за кластеризацію та вибір оптимальних культур. Кожна форма орієнтована на виконання окремої задачі, що забезпечує модульність інтерфейсу та зручність у використанні.

Пакет Data Access Layer містить провайдери для роботи з базою даних, які забезпечують отримання, оновлення та видалення даних із таблиць. Серед провайдерів виділяються ClimateZoneProvider, CropProvider, LogsProvider, RegionProvider, RegionCharProvider, SoilTypeProvider та UsersProvider, кожен з яких працює з відповідною сутністю.

Пакет Business Logic Layer включає модулі бізнес-логіки, такі як EncryptData для шифрування даних, RaportBLL для формування звітів, LIMBOClusterer для виконання кластеризації, а також RoleApp та ValidationMy для управління ролями і перевірки правильності введення інформації.

Усі пакети взаємодіють із базою даних, при цьому зв'язки між шарами чітко визначені: інтерфейсний шар взаємодіє лише із шаром доступу до даних і бізнес-логікою, що забезпечує дотримання принципів багаторівневої архітектури. Така організація структури системи сприяє її надійності, гнучкості та можливості подальшого розширення без істотних змін у вже реалізованих компонентах.

## 2.4 Діаграма компонентів

Під час проектування архітектури програмного забезпечення важливо не лише визначити логічні зв'язки між класами та модулями, а й формалізувати фізичну структуру системи у вигляді взаємозв'язку програмних компонентів. Діаграма компонентів слугує для відображення того, як окремі частини застосунку організовані на рівні вихідних файлів, бібліотек і зовнішніх ресурсів, а також як вони взаємодіють між собою під час виконання. На рис. 2.5 наведено діаграму компонентів розробленої системи для підтримки вибору оптимальних сільськогосподарських культур.

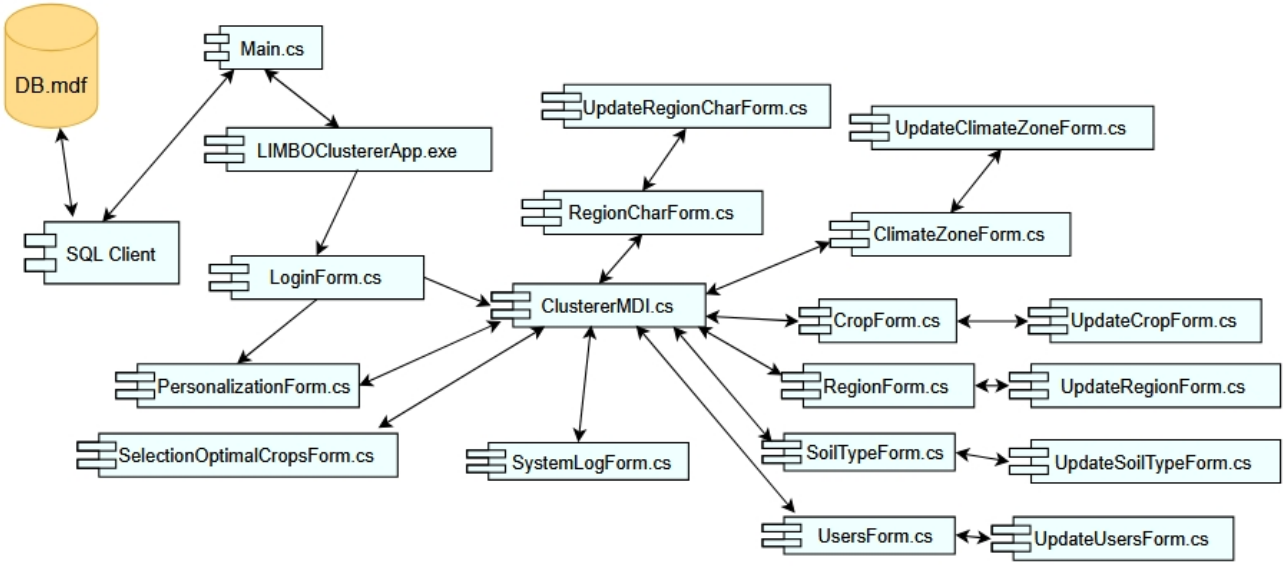

Рис.2.5 Діаграма компонентів

Діаграма демонструє ключові компоненти системи та їхню взаємодію. Центральним елементом є форма ClustererMDI.cs, яка виступає головним вікном застосунку та забезпечує навігацію між основними підформами. Вона безпосередньо пов'язана із такими компонентами, як RegionForm.cs, CropForm.cs, SoilTypeForm.cs, ClimateZoneForm.cs, UsersForm.cs, RegionCharForm.cs, SelectionOptimalCropsForm.cs, SystemLogForm.cs та PersonalizationForm.cs, кожен з яких відповідає за окремий розділ функціональності.

Кожна форма редагування даних, така як UpdateRegionForm.cs, UpdateCropForm.cs, UpdateSoilTypeForm.cs, UpdateClimateZoneForm.cs, UpdateRegionCharForm.cs та UpdateUsersForm.cs, створена для реалізації механізму оновлення записів відповідної сутності в системі. Ці форми забезпечують користувачу можливість змінювати атрибути об'єктів через інтуїтивно зрозумілий інтерфейс, що мінімізує ризик помилок при внесенні даних. Після редагування інформації відповідні форми обміну даними інтегруються зі своїми базовими формами перегляду, оновлюючи відображення змінених записів та підтримуючи актуальність інформації в інтерфейсі.

Компонент LoginForm.cs відповідає за авторизацію користувача та ініціалізацію доступу до основної форми ClustererMDI.cs. Вихідний файл Main.cs виконує початкове завантаження програми та керує запуском головного застосунку. Окремо винесено компонент LIMBOClustererApp.exe, який реалізує функціональність кластеризації даних і працює у зв'язці із SQL-клієнтом для доступу до бази даних DB.mdf.

Отже, діаграма компонентів ілюструє фізичну організацію програмних частин системи, показуючи, як окремі форми, модулі обробки та зовнішні ресурси об'єднуються у єдиний цілісний застосунок для вирішення поставлених завдань.

# 3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ПІДТРИМКИ АГРАРНИХ РІШЕНЬ

## 3.1 Система управління інформаційною базою

У сучасних умовах цифровізації аграрного сектору особливе значення набуває ефективна організація інформаційної бази, яка забезпечує зберігання, цілісність і оперативний доступ до даних, що використовуються в аналітичних системах підтримки аграрних рішень. Зокрема, для програмного забезпечення, яке виконує кластерний аналіз агроекологічних характеристик регіонів, критично важливим є вибір надійної та продуктивної системи управління інформаційною базою (СУІБ). Така система повинна не лише зберігати довідкові та аналітичні дані, а й забезпечувати швидкий обмін інформацією між модулями, підтримку одночасної роботи кількох користувачів, гнучкість у масштабуванні та безпеку даних.

У рамках цієї розробки СУІБ виконує функцію посередника між прикладною логікою та фізичним рівнем зберігання інформації. Вона обслуговує запити з інтерфейсу користувача, постачає дані для кластеризаційного модуля та підтримує актуальність інформації під час її перегляду, редагування чи аналітичного використання. При виборі СУІБ особливу увагу приділялося її здатності ефективно працювати з якісними й категоріальними даними, а також забезпечувати високу стабільність при виконанні типових CRUD-операцій (створення, читання, оновлення, видалення).

Серед найбільш поширених СУІБ, які використовуються для побудови аналітичних рішень у сфері сільського господарства, слід відзначити Microsoft SQL Server, MySQL та Oracle. Нижче коротко розглянуто їх з погляду придатності до використання в системі аграрного кластерного аналізу.

Microsoft SQL Server є потужною комерційною реляційною СУІБ, розробленою корпорацією Microsoft, яка характеризується високим рівнем надійності, масштабованості та зручності в адмініструванні [17]. Вона надає

широкі можливості для реалізації складних аналітичних запитів, зокрема підтримку функцій агрегації, сортування, фільтрації, транзакцій і процедур. Інтеграція з платформою .NET робить SQL Server особливо зручним для розробки програмного забезпечення на C#, що було ключовим чинником вибору цієї СУІБ у межах даного проєкту. Крім того, система має розвинуті засоби безпеки, зокрема контроль доступу, шифрування та аудит змін, що є важливим при збереженні персональних даних користувачів і результатів аналітики.

MySQL – це відкрита реляційна СУІБ, яка відзначається своєю простотою, швидкодією та широкою підтримкою спільноти. Вона активно використовується у веб-застосунках, проте також придатна для побудови настільних систем невеликої та середньої складності [18]. Серед її переваг – невисокі вимоги до ресурсів, швидке розгортання та підтримка базових механізмів транзакцій. Проте у контексті проєктування системи, орієнтованої на активну роботу з великими наборами структурованих даних у середовищі Microsoft, MySQL поступається SQL Server за рівнем інтеграції з мовою C# та інструментами бізнес-логіки на базі .NET.

Oracle Database – одна з найпотужніших корпоративних СУІБ, що підтримує розподілену архітектуру, обробку великих обсягів даних, складні аналітичні задачі та високий рівень безпеки [19]. Вона є оптимальним рішенням для масштабованих систем з високими навантаженнями, але її використання вимагає значних обчислювальних ресурсів, професійного супроводу та ліцензійних витрат. З огляду на це, Oracle зазвичай використовується у великих інституційних або промислових агроплатформах і менш доцільна для настільних систем з обмеженим функціоналом.

З метою обґрунтування вибору платформи управління даними, доцільним є порівняльний аналіз основних систем керування базами даних, розглянутих вище. Для об'єктивного зіставлення враховано низку ключових характеристик, що мають вирішальне значення у розробці настільних аналітичних систем аграрного спрямування. Зокрема, порівнюються такі аспекти, як рівень інтеграції з мовою програмування C#, підтримка складної аналітики, продуктивність при

роботі з категоріальними даними, можливості масштабування, безпека та загальна придатність для кластеризаційних задач. Узагальнене порівняння СУІБ подано в табл. 3.1.

Таблиця 3.1

Порівняння систем управління базами даних за основними критеріями

| № | Характеристика | MS SQL Server | MySQL | Oracle Database |
|---|---|---|---|---|
| 1 | Інтеграція з C# / .NET | Відмінна | Обмежена | Середня |
| 2 | Підтримка аналітичних запитів | Висока | Базова | Висока |
| 3 | Продуктивність при кластеризації категорій | Висока | Середня | Висока |
| 4 | Масштабованість для середніх проєктів | Висока | Обмежена | Висока |
| 5 | Безпека та контроль доступу | Розширена | Базова | Розширена |
| 6 | Легкість налаштування в середовищі Windows | Висока | Середня | Низька |
| 7 | Вартість використання | Безкоштовна версія (Express) | Безкоштовна | Комерційна |
| 8 | Придатність для аграрних аналітичних систем | Оптимальна | Задовільна | Надлишкова за потужністю |

Вибір MS SQL Server для розробки системи зумовлений його високим рівнем сумісності з мовою програмування C#, що забезпечує стабільну та ефективну взаємодію між програмним інтерфейсом і базою даних. Завдяки підтримці складних аналітичних запитів і високій продуктивності при роботі з категоріальними даними, дана СУІБ ідеально підходить для реалізації кластеризаційного аналізу у сільському господарстві. Також перевагою є наявність безкоштовної версії (SQL Server Express), яка повністю задовольняє потреби проєкту на початкових етапах. Розширені механізми безпеки та простота розгортання у середовищі Windows роблять MS SQL Server оптимальним вибором для створення настільного аналітичного застосунку.

## 3.2 Розробка інформаційної бази

Розробка інформаційної бази для програмного забезпечення, що реалізує вибір оптимальних сільськогосподарських культур на основі кластеризаційного підходу, є одним із найважливіших етапів проєктування системи. Саме від якості структурування даних та організації взаємозв'язків між сутностями залежить ефективність зберігання, обробки та подальшого аналізу аграрної інформації. Як платформу для реалізації бази даних було обрано систему керування базами даних MS SQL Server, що забезпечує високу продуктивність, захист інформації та зручність у масштабуванні в разі розширення функціоналу.

Процес побудови інформаційної структури базувався на кількох ключових кроках:

– визначення сутностей, що відображають основні логічні компоненти предметної області. У межах даної системи такими сутностями стали регіони, типи ґрунтів, кліматичні зони, культури, характеристики регіонів, користувачі та журнали подій;

– визначення атрибутів для кожної сутності, які описують її властивості. Наприклад, сутність «Кліматична зона» включає такі атрибути, як унікальний ідентифікатор, назва та опис, а таблиця «RegionChar» фіксує кислотність ґрунту, глибину залягання води, пов'язані регіон, ґрунт, клімат та культуру;

– установлення зв'язків між сутностями, що дозволяє відображати залежності, необхідні для коректного формування аграрних кластерів. Зокрема, таблиця RegionChar має зовнішні ключі до таблиць Region, SoilType, ClimateZone і Crop, об'єднуючи ці характеристики в єдину модель;

– створення таблиць у СУБД та індексація ключових полів. Усі таблиці мають первинні ключі, які гарантують унікальність записів. Для поліпшення продуктивності під час виконання запитів застосовано зовнішні ключі та індекси на полях, що часто використовуються в обчисленнях або фільтрації.

Такий підхід дозволив сформувати цілісну та логічно зв'язану інформаційну модель, яка є основою для алгоритмічної обробки вхідних

аграрних даних і подальшого формування рекомендацій. Побудована структура забезпечує швидкий доступ до даних, гнучкість у розширенні, а також узгодженість між усіма підсистемами розробленого ПЗ.

На рис. 3.1 представлено фізичну модель бази даних, побудовану у середовищі MS SQL Server на основі розроблених таблиць та зв’язків між ними.

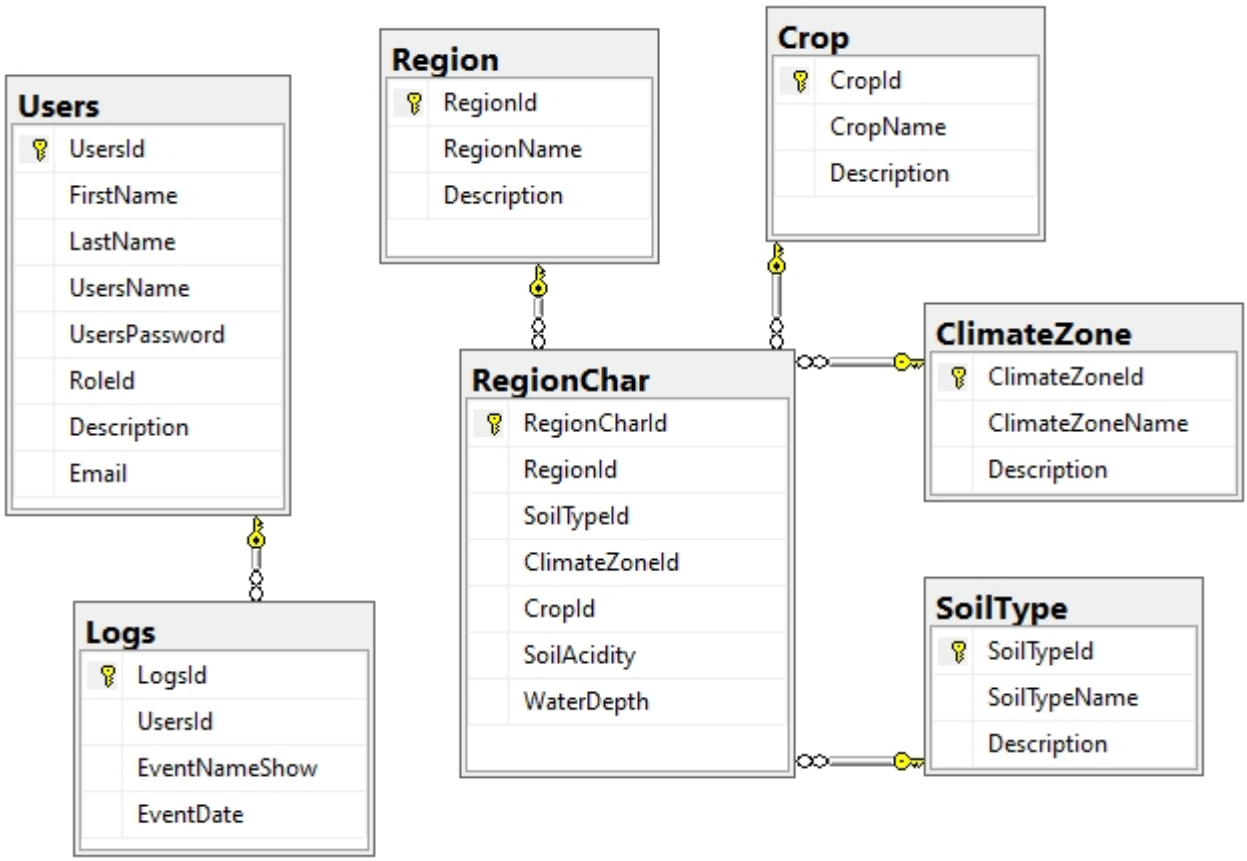


Рис.3.1 Модель бази даних системи вибору оптимальних культур

Після завершення формування структури було проведено перевірку коректності взаємозв’язків, тестування додавання, оновлення та видалення записів, а також симуляцію навантаження для оцінки швидкодії. Результати підтвердили відповідність бази даних усім технічним вимогам, що дозволяє системі стабільно функціонувати навіть за умов збільшення обсягів агроекологічної інформації.

## 3.3 Вибір інструментарію для створення прикладного програмного забезпечення

У процесі створення сучасного прикладного програмного забезпечення важливим етапом є вибір мови програмування та середовища розробки, які найкраще відповідають функціональним вимогам, архітектурі системи та особливостям предметної області. У контексті побудови настільного аналітичного застосунку для аграрної кластеризації критичне значення має підтримка роботи з базами даних, зручність побудови графічного інтерфейсу, а також сумісність із бібліотеками машинного навчання та алгоритмами обробки категоріальних даних. Тому було проаналізовано можливості трьох популярних мов програмування – Python, Java та C# – для визначення найпридатнішої платформи реалізації.

Python є інтерпретованою мовою з відкритим кодом, яка широко використовується в наукових обчисленнях, аналітиці даних та розробці прототипів систем штучного інтелекту [20]. Вона має потужну екосистему бібліотек для машинного навчання (Scikit-learn, TensorFlow, Pandas), проте менш зручна для створення настільних графічних інтерфейсів. У сфері розробки локальних бізнес-застосунків Python здебільшого використовується для серверної логіки або автоматизації обчислень.

Java – це мова з багаторічною історією, що вирізняється високою кросплатформеністю, стабільністю та багатим набором бібліотек для розробки корпоративного програмного забезпечення [21]. Вона активно використовується у великих проєктах, що потребують масштабування та обслуговування багатьох користувачів. Однак створення настільних програм у Java вимагає більше ресурсів, а побудова інтерфейсу є менш гнучкою порівняно з іншими технологіями.

C# – об’єктно-орієнтована мова програмування, яка розроблена корпорацією Microsoft та ідеально підходить для створення десктопних застосунків у середовищі Windows [22]. Завдяки глибокій інтеграції з

платформою .NET, C# забезпечує зручні засоби для роботи з базами даних (через ADO.NET, Entity Framework), створення інтерфейсів (WinForms, WPF) та реалізації модульної архітектури. Крім того, C# підтримує взаємодію з ML.NET – фреймворком машинного навчання, що дозволяє вбудовувати аналітичні алгоритми, включно з кластеризацією, безпосередньо у програму.

З метою обґрунтування вибору оптимальної мови програмування для реалізації прикладного програмного забезпечення було проведено порівняльний аналіз розглянутих мов. У табл. 3.2 узагальнено результати оцінки за як кількісними, так і якісними характеристиками.

Таблиця 3.2

Порівняння мов програмування за основними критеріями

| № | Характеристика | Python | Java | C# |
|---|---|---|---|---|
| 1 | Складність побудови GUI | Висока (Tkinter, PyQt) | Середня (Swing, JavaFX) | Низька (WinForms, WPF) |
| 2 | Час виконання програми (умовна шкала, сек) | 2.5 | 1.8 | 1.2 |
| 3 | Споживання пам’яті (MB) | 140 | 180 | 110 |
| 4 | Інтеграція з СУБД | Через бібліотеки | Через JDBC | Пряма (ADO.NET, EF) |
| 5 | Наявність ML-фреймворків | Дуже висока | Середня | Висока (ML.NET) |
| 6 | Підтримка кросплатформеності | Висока (через Python VM) | Висока | Висока (.NET Core) |
| 7 | Документація та підтримка спільноти | Широка | Широка | Широка |
| 8 | Розповсюдження ліцензії | Вільна | Вільна | Вільна |
| 9 | Простота навчання (1–10) | 8 | 6 | 7 |
| 10 | Сумісність з Windows API | Обмежена | Обмежена | Повна |

Вибір C# як основної мови програмування зумовлений її високою сумісністю з платформою Windows, зручною підтримкою побудови графічного інтерфейсу через WinForms, а також нативною інтеграцією з Microsoft SQL Server через ADO.NET та Entity Framework. У порівнянні з Python і Java, C#

забезпечує вищу продуктивність виконання настільних застосунків, менше споживання пам’яті та повну підтримку роботи з API операційної системи. Крім того, використання ML.NET дозволяє реалізувати кластеризаційний аналіз без необхідності залучення сторонніх бібліотек чи середовищ. Сукупність цих чинників робить C# оптимальним вибором для розробки системи аграрного аналізу з акцентом на надійність, швидкодію та простоту впровадження.

## 3.4 Алгоритмізація та програмування програмних модулів

Розробка прикладного програмного забезпечення для підтримки аграрного кластерного аналізу передбачає реалізацію ряду функціональних модулів, що взаємодіють між собою відповідно до заданої логіки. Щоб забезпечити надійність роботи системи та зрозуміле управління даними, усі етапи взаємодії користувача з програмою формалізуються у вигляді алгоритмів. Один із ключових процесів – авторизація, яка забезпечує перевірку прав доступу до функціональних частин системи, – реалізується на початковому етапі взаємодії з додатком.

На рис. 3.2 наведено блок-схему алгоритму авторизації користувача.

Алгоритм розпочинається з введення логіну та пароля користувачем. Після цього відбувається натискання кнопки підтвердження, в результаті чого облікові дані передаються на перевірку до бази даних. Система формує запит і очікує на відповідь, у якій міститься результат аутентифікації. Якщо введені дані є некоректними, користувач отримує повідомлення про помилку, після чого має можливість повторити спробу. У випадку правильного введення даних, система дозволяє доступ до головного вікна програми, після чого процедура авторизації завершується. Такий підхід забезпечує базовий рівень безпеки та контроль доступу до інформаційної системи.

На рис. 3.3 зображено алгоритм додавання нової кліматичної зони до бази даних у межах функціоналу редагування довідкової інформації.

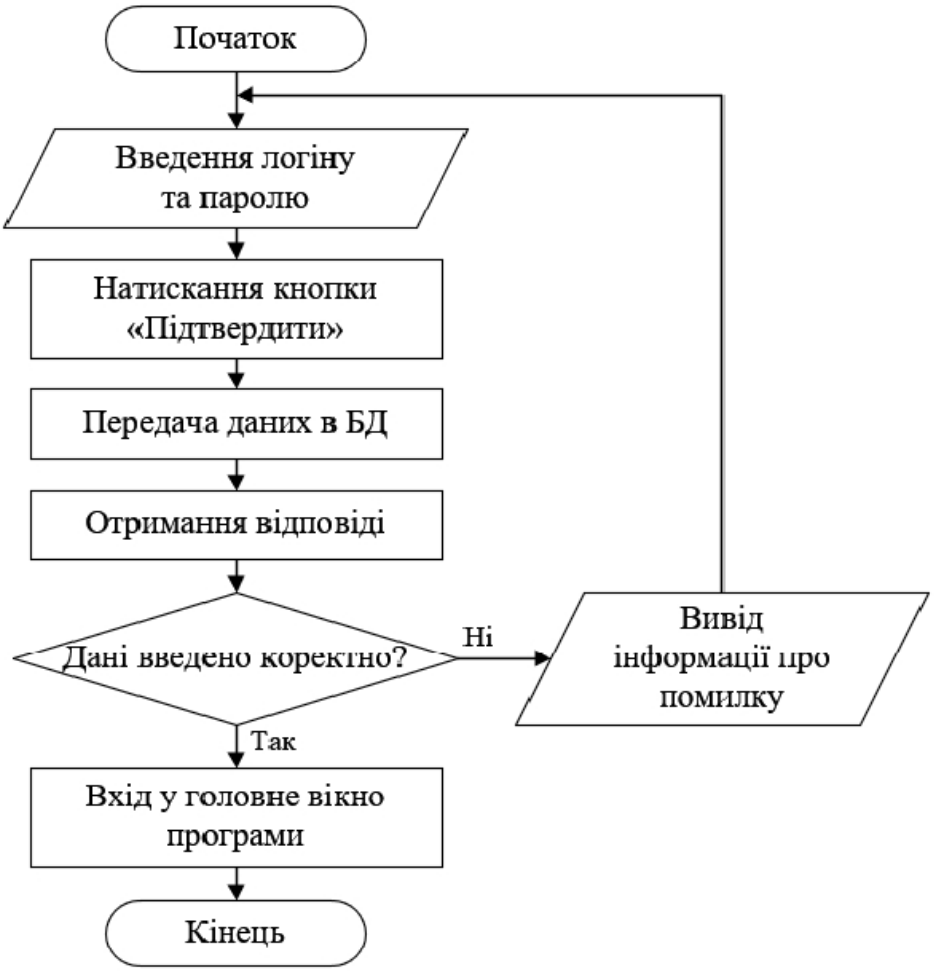


Рис.3.2 Алгоритм перевірки даних авторизації користувача

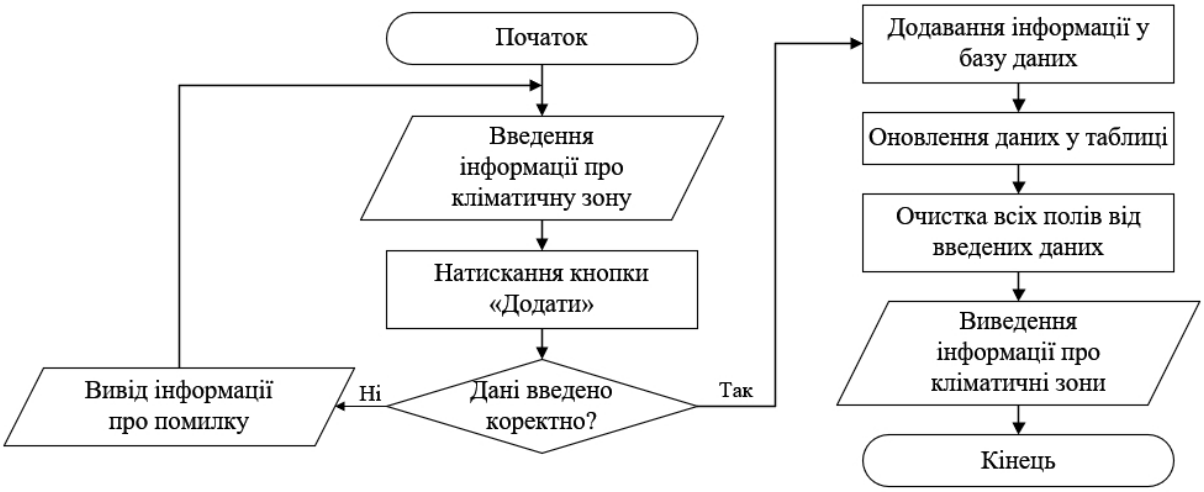


Рис.3.3 Алгоритм обробки введення нової кліматичної зони

Процес починається з введення користувачем назви кліматичної зони та її опису у відповідні поля введення. Після натискання кнопки «Додати» система виконує перевірку на коректність заповнення даних. У разі виявлення помилки з’являється відповідне повідомлення, і користувач має можливість виправити введену інформацію. Якщо всі поля заповнені правильно, система додає новий запис до бази даних, оновлює відповідну таблицю у вікні перегляду, очищує форми введення та виводить оновлений список кліматичних зон. Таким чином, алгоритм забезпечує повний цикл додавання довідкової інформації з контролем введення та автоматичним оновленням інтерфейсу.

Рис. 3.4 відображає алгоритм редагування інформації про регіон, що реалізується у відповідному функціональному модулі системи.

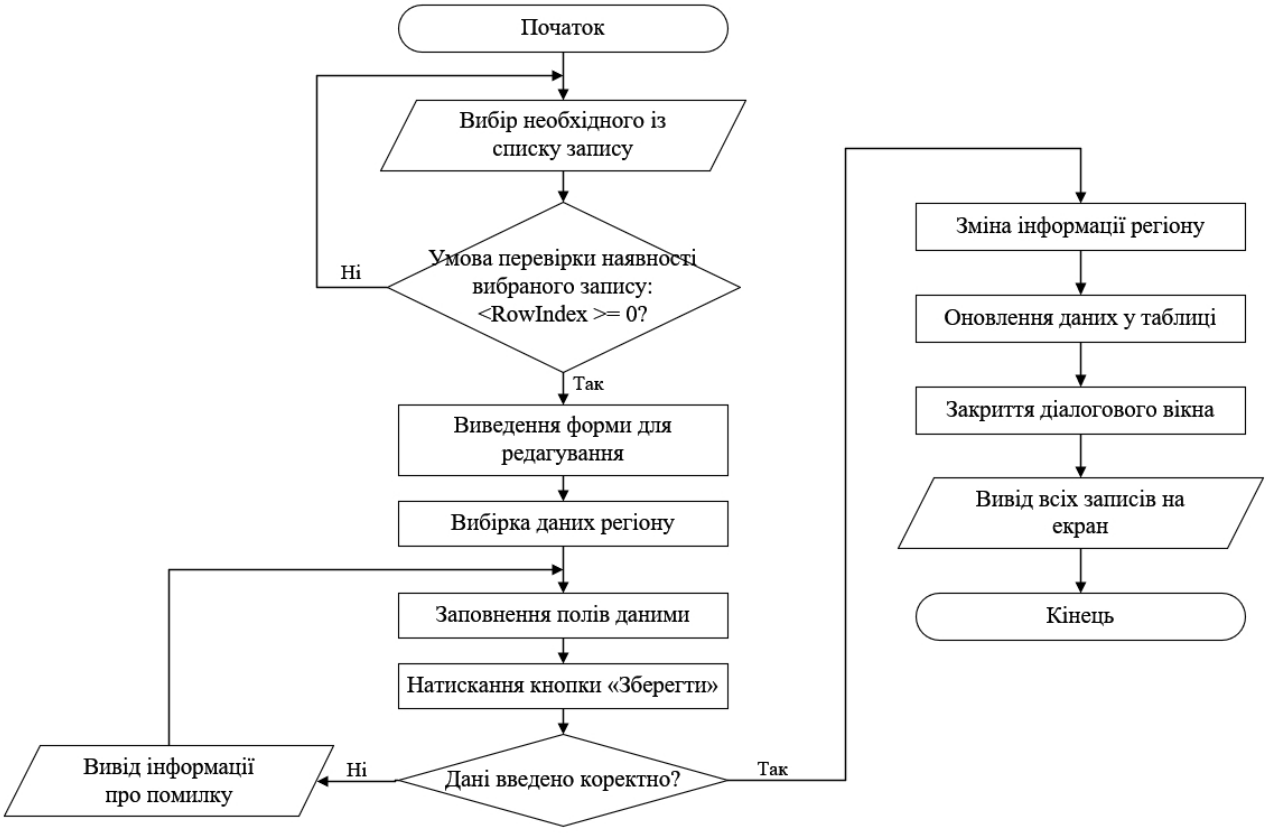


Рис.3.4 Алгоритм редагування даних про регіон

Процес починається з вибору користувачем необхідного запису зі списку регіонів, після чого система перевіряє, чи обрано коректний індекс рядка в таблиці. У разі, якщо запис не було обрано, відображається повідомлення про

помилку. Якщо індекс рядка дійсний, відкривається форма редагування, у яку автоматично підтягуються дані обраного регіону. Користувач заповнює або змінює значення в полях і натискає кнопку «Зберегти». Після перевірки правильності введених даних система оновлює запис у базі, оновлює вміст таблиці, закриває вікно редагування та виводить на екран оновлений список усіх регіонів. Така логіка забезпечує цілісність даних та інтуїтивну взаємодію користувача з системою.

На рис. 3.5 зображено алгоритм автоматизованого вибору оптимальних сільськогосподарських культур, реалізований у модулі кластеризації на основі алгоритму LIMBO.

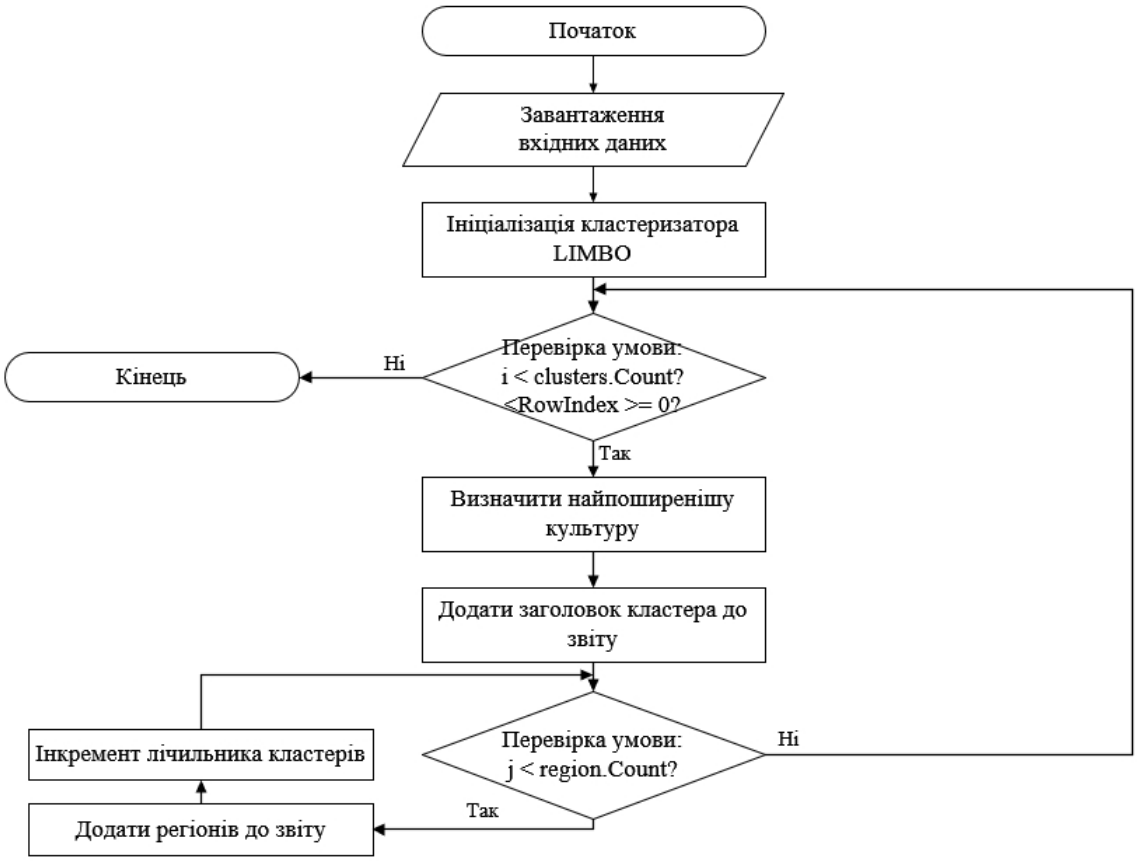


Рис.3.5 Алгоритм вибору оптимальних сільськогосподарських культур

Процес починається із завантаження вхідних даних, що містять сукупність агроекологічних характеристик регіонів. Далі виконується ініціалізація кластеризатора LIMBO, після чого система починає поетапну обробку кожного з кластерів. Для кожного кластеру визначається найпоширеніша культура серед

регіонів, які до нього належать, і ця інформація додається до заголовка звіту. Наступним кроком є перебір усіх регіонів усередині поточного кластеру – їх дані поступово додаються до текстового звіту. Після завершення обробки регіонів лічильник переходить до наступного кластеру, і процедура повторюється. Завершення роботи настає після обробки всіх кластерів, у результаті чого користувач отримує звіт із рекомендаціями щодо культур, релевантних кожному з агроекологічних кластерів.

Після реалізації алгоритмічної логіки та користувацького інтерфейсу важливим етапом стала розробка рівня доступу до даних, який забезпечує зв’язок між прикладними модулями програми та фізичним сховищем інформації. Цей рівень відповідає за зчитування, запис, оновлення та видалення інформації з бази даних відповідно до дій користувача або вимог бізнес-логіки. Надійна конфігурація з’єднання із СУБД є критичною для коректного функціонування системи загалом.

На рис. 3.6 представлено конфігурацію підключення системи до бази даних, яка задається у вигляді параметра в конфігураційному файлі застосунку.

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

Рис.3.6 Конфігурації з’єднання з базою даних

З’єднання із базою даних реалізовано через локальний екземпляр SQL Server Express (LocalDB), який дозволяє прикріпити файл бази даних напряму з робочої директорії застосунку. У параметрах підключення вказано шлях до файлу .mdf, який зберігається у каталозі програми, а також застосовано інтегровану безпеку Windows (Integrated Security=True), що забезпечує зручність доступу без необхідності окремого логіну й пароля. Такий підхід є оптимальним для настільного застосунку, що працює на локальній машині користувача, та гарантує простоту розгортання системи без складного налаштування зовнішніх серверів.

У наведеному на рис. 3.7 методі реалізовано механізм додавання нової характеристики регіону до таблиці бази даних. Метод приймає як вхідні параметри ідентифікатори регіону, типу ґрунту, кліматичної зони та культури, а також текстові значення кислотності ґрунту та глибини залягання води. Створюється SQL-запит на вставку даних у таблицю RegionChar, де передбачено шість відповідних полів.

```
public void InsertRegionChar(int regionId, int soilTypeId, int climateZoneId, int cropId,
  string soilAcidity, string waterDepth) {
  string SqlString = @"INSERT INTO RegionChar
(RegionId, SoilTypeId, ClimateZoneId, CropId, SoilAcidity, WaterDepth)
VALUES (@RegionId, @SoilTypeId, @ClimateZoneId, @CropId, @SoilAcidity, @WaterDepth)";

  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("@RegionId", regionId);
      cmd.Parameters.AddWithValue("@SoilTypeId", soilTypeId);
      cmd.Parameters.AddWithValue("@ClimateZoneId", climateZoneId);
      cmd.Parameters.AddWithValue("@CropId", cropId);
      cmd.Parameters.AddWithValue("@SoilAcidity", soilAcidity);
      cmd.Parameters.AddWithValue("@WaterDepth", waterDepth);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рис.3.7 Код методу «InsertRegionChar»

Для уникнення SQL-ін’єкцій значення параметрів передаються за допомогою прив’язки змінних через AddWithValue. Після ініціалізації з’єднання з базою даних виконується команда ExecuteNonQuery, яка здійснює фізичне додавання запису. Завершення роботи відбувається через закриття з’єднання, що забезпечує правильне вивільнення ресурсів. Такий підхід є безпечним, структурованим і добре інтегрується в загальну архітектуру рівня доступу до даних.

У фрагменті коду на рис. 3.8 реалізовано метод для отримання повного переліку агроекологічних характеристик регіонів разом із супутньою довідковою інформацією. Він формує SQL-запит, який об’єднує таблицю RegionChar з таблицями Region, SoilType, ClimateZone і Crop через оператори внутрішнього з’єднання.

```
public List<RegionChar> GetAllRegionChar() {
    int num = 0;
    string SqlString = @"
                SELECT rc.*,
                       r.RegionName,
                       s.SoilTypeName,
                       cz.ClimateZoneName,
                       c.CropName
                FROM RegionChar rc
                INNER JOIN Region r ON rc.RegionId = r.RegionId
                INNER JOIN SoilType s ON rc.SoilTypeId = s.SoilTypeId
                INNER JOIN ClimateZone cz ON rc.ClimateZoneId = cz.ClimateZoneId
                INNER JOIN Crop c ON rc.CropId = c.CropId
                ORDER BY rc.RegionCharId";
```

Рис.3.8 Код методу «GetAllRegionChar»

Вибірка результатів впорядковується за первинним ключем RegionCharId, а отримані дані записуються до списку об’єктів, який надалі можна передати в інші модулі системи. Метод реалізує повноцінну агрегацію логічно пов’язаних полів, забезпечуючи гнучкість у формуванні звітів і відображенні регіональних характеристик у зручному для аналізу вигляді.

Метод на рис. 3.9 призначений для оновлення вже існуючих записів у таблиці регіональних характеристик. У його межах формується SQL-запит типу UPDATE, який змінює всі ключові поля відповідного запису на основі переданих параметрів.

```
public void UpdateRegionChar(int regionId, int soilTypeId, int climateZoneId, int cropId,
  string soilAcidity, string waterDepth, int regionCharId) {
  string SqlString = @"UPDATE RegionChar
SET RegionId = @RegionId,
    SoilTypeId = @SoilTypeId,
    ClimateZoneId = @ClimateZoneId,
    CropId = @CropId,
    SoilAcidity = @SoilAcidity,
    WaterDepth = @WaterDepth
WHERE RegionCharId = @RegionCharId";

  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("@RegionId", regionId);
      cmd.Parameters.AddWithValue("@SoilTypeId", soilTypeId);
      cmd.Parameters.AddWithValue("@ClimateZoneId", climateZoneId);
      cmd.Parameters.AddWithValue("@CropId", cropId);
      cmd.Parameters.AddWithValue("@SoilAcidity", soilAcidity);
      cmd.Parameters.AddWithValue("@WaterDepth", waterDepth);
      cmd.Parameters.AddWithValue("@RegionCharId", regionCharId);
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рис.3.9 Код методу «UpdateRegionChar»

В оновленні беруть участь такі поля, як ідентифікатори регіону, ґрунту, кліматичної зони та культури, а також текстові значення кислотності ґрунту й глибини залягання води. Операція виконується лише для одного запису, що визначається унікальним ідентифікатором RegionCharId, переданим як аргумент. Такий механізм дозволяє швидко редагувати характеристики регіону без створення нового запису, зберігаючи узгодженість даних у базі.

Розробка рівня інтерфейсу користувача є важливою складовою проєктування прикладного програмного забезпечення, оскільки саме цей рівень забезпечує зручну взаємодію користувача із функціональністю системи. Інтерфейс має бути інтуїтивно зрозумілим, візуально впорядкованим і адаптованим під потреби цільової аудиторії – агрономів, аналітиків або інших користувачів, що не мають технічної підготовки. При створенні форм використовуються стандартні компоненти середовища WinForms, що дозволяє забезпечити стабільну роботу програми на настільних системах під керуванням Windows.

На рис. 3.10 зображено приклад реалізованої форми для обліку характеристик регіонів та вирощуваних культур.

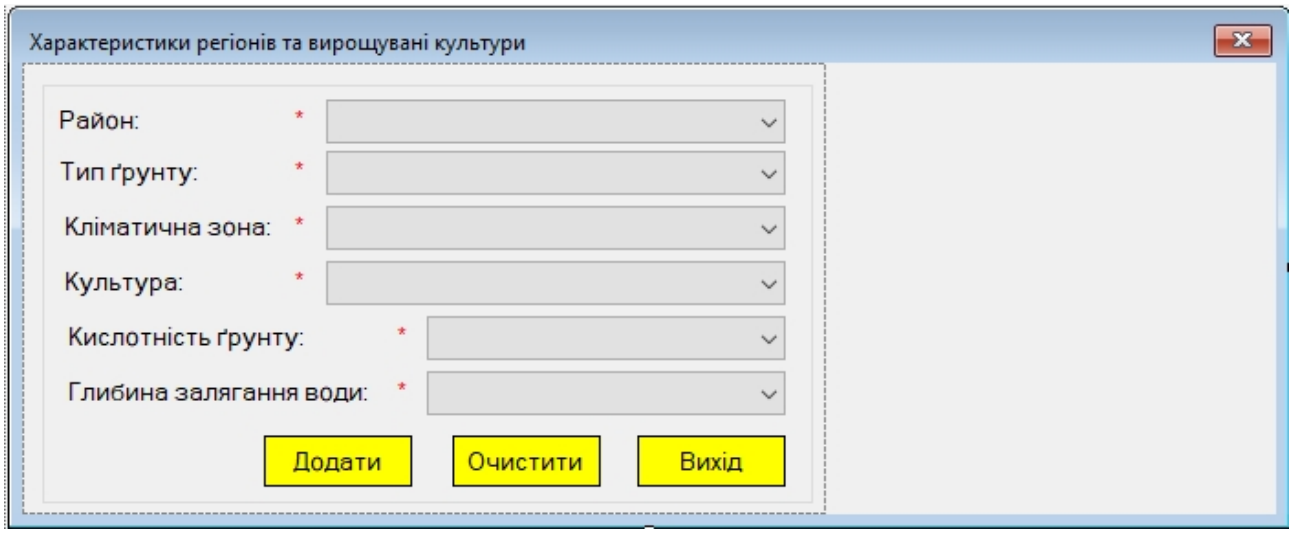


Рис.3.10 Форма обліку характеристик регіонів та культур

Форма містить набір елементів керування для вибору регіону, типу ґрунту, кліматичної зони, культури, а також введення кислотності ґрунту та глибини залягання води. Усі поля, обов’язкові для заповнення, позначені символом «*»,

що підвищує користувацьку інформативність і зменшує ймовірність помилок введення. Нижня частина інтерфейсу містить три функціональні кнопки: «Додати» – для збереження нової характеристики, «Очистити» – для скидання введених даних, та «Вихід» – для закриття форми. Така структура дозволяє швидко вносити регіональні параметри для подальшого аналізу й кластеризації.

У методі на .рис. 3.11 реалізовано початкове завантаження даних для заповнення випадаючих списків у формі, яка відповідає за введення регіональних характеристик. Після виклику цього методу ініціалізуються локальні списки, у які завантажуються всі наявні записи з відповідних довідників – кліматичних зон, культур, регіонів та типів ґрунтів.

```
private void LoadAllDate() {
  SoilAcidityCBox.SelectedIndex = 0;
  _ClimateZoneList = _ClimateZoneProvider.GetAllClimateZone();
  ClimateZoneCBox.DataSource = _ClimateZoneList;
  ClimateZoneCBox.ValueMember = "ClimateZoneId";
  ClimateZoneCBox.DisplayMember = "ClimateZoneName";

  _CropList = _CropProvider.GetAllCrop();
  CropCBox.DataSource = _CropList;
  CropCBox.ValueMember = "CropId";
  CropCBox.DisplayMember = "CropName";

  _RegionList = _RegionProvider.GetAllRegion();
  RegionCBox.DataSource = _RegionList;
  RegionCBox.ValueMember = "RegionId";
  RegionCBox.DisplayMember = "RegionName";

  _SoilTypeList = _SoilTypeProvider.GetAllSoilType();
  SoilTypeCBox.DataSource = _SoilTypeList;
  SoilTypeCBox.ValueMember = "SoilTypeId";
  SoilTypeCBox.DisplayMember = "SoilTypeName";

  SoilAcidityCBox.SelectedIndex = 0;
  WaterDepthCBox.SelectedIndex = 0;
}
```

Рис.3.11 Завантаження даних для заповнення елементів форми

Кожен з елементів управління типу ComboBox зв’язується з відповідним джерелом даних, де чітко вказуються як значення для збереження (ValueMember), так і текстове відображення для користувача (DisplayMember). Додатково, для полів кислотності ґрунту та глибини залягання води встановлюються початкові значення індексів, що дозволяє уникнути незаповнених полів при першому відкритті форми. Метод викликається

автоматично під час створення форми, забезпечуючи її повну готовність до взаємодії з користувачем без додаткових дій.

На рис. 3.12 наведено метод, що виконується під час натискання кнопки додавання та відповідає за обробку введених користувачем даних у формі введення аграрних характеристик.

```
private void AddBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    _RegionCharProvider.InsertRegionChar(Convert.ToInt32(RegionCBox.SelectedValue),
      Convert.ToInt32(SoilTypeCBox.SelectedValue), Convert.ToInt32(ClimateZoneCBox.SelectedValue),
      Convert.ToInt32(CropCBox.SelectedValue), SoilAcidityCBox.Text, WaterDepthCBox.Text);
    DataLoad();
    LoadAllDate();
    ClearAllControls();
  }
}
```

Рис.3.12 Завантаження даних для заповнення елементів форми

Насамперед відбувається перевірка правильності заповнення полів за допомогою допоміжного методу, який забезпечує валідацію. Якщо всі дані визнані коректними, здійснюється виклик функції вставки нового запису до таблиці RegionChar, де використовуються значення, вибрані з комбінованих списків, а також текстові поля. Після додавання система оновлює відображення інформації у таблиці, знову завантажує всі довідкові дані для синхронізації інтерфейсу, а також очищує поля введення, підготовлюючи форму до наступного циклу взаємодії.

У методі на рис. 3.13 реалізовано обробку події кліку по комірці таблиці, що викликає форму для редагування обраної характеристики регіону.

```
private void RegionCharGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
  if (e.RowIndex >= 0 && RegionCharGridView[0, e.RowIndex].Value.ToString() != _ClimateZoneList[0].Message) {
    _selectedRowIndex = e.RowIndex;
    UpdateRegionCharForm updateRegionCharForm =
      new UpdateRegionCharForm(Convert.ToInt32(RegionCharGridView[0, e.RowIndex].Value.ToString()));
    updateRegionCharForm.ShowDialog();
    DataLoad();
  }
}
```

Рис.3.13 Виклик форми редагування обраної характеристики регіону

Після перевірки, що натискання відбулося по дійсному рядку, а не заголовку чи порожній області, виконується додаткова умова, що захищає від

відкриття форм редагування у випадках, коли запис не відповідає очікуваній структурі. У разі виконання умов зберігається індекс обраного рядка, після чого створюється новий екземпляр форми редагування, який ініціалізується ідентифікатором запису з першої колонки таблиці. Форма відкривається у модальному режимі, що блокує взаємодію з основним вікном до завершення редагування. Після закриття форми автоматично оновлюється відображення таблиці, що гарантує відображення актуального стану даних без потреби в ручному перезапуску інтерфейсу.

Розробка рівня бізнес-логіки є важливою частиною архітектури програмного забезпечення, оскільки саме цей рівень відповідає за опрацювання прикладних сценаріїв, реалізацію функціональних правил та підготовку даних для подальшого використання в інтерфейсі або аналітичних модулях. На відміну від рівня доступу до даних, бізнес-логіка не лише отримує або зберігає інформацію, а й перетворює її в зручну для обробки форму, агрегує, фільтрує або структурує відповідно до вимог системи.

У наведеному фрагменті на рис. 3.14 реалізується метод завантаження регіональних даних із бази, які вже приведені до узагальненої форми для подальшого використання у кластеризаційному модулі або побудові звітів. SQL-запит об'єднує кілька таблиць через внутрішні зв’язки та вибирає основні характеристики: назву регіону, тип ґрунту, кліматичну зону, кислотність, глибину води й культуру.

```
public List<RegionData> LoadDataFromDatabase() {
  string SqlString = @"
SELECT
  r.RegionName,
  s.SoilTypeName AS SoilType,
  cz.ClimateZoneName AS ClimateZone,
  rc.SoilAcidity,
  rc.WaterDepth,
  c.CropName AS Crop
FROM RegionChar rc
INNER JOIN Region r ON rc.RegionId = r.RegionId
INNER JOIN SoilType s ON rc.SoilTypeId = s.SoilTypeId
INNER JOIN ClimateZone cz ON rc.ClimateZoneId = cz.ClimateZoneId
INNER JOIN Crop c ON rc.CropId = c.CropId
ORDER BY r.RegionName";
```

Рис.3.14 Фрагмент методу «LoadDataFromDatabase»

Імена деяких полів перейменовуються для зручності подальшої обробки, а результат впорядковується за назвою регіону. Дані, отримані з бази, перетворюються у список об’єктів RegionData, що можуть безпосередньо передаватися до алгоритмів кластеризації або відображатися у таблицях. Такий метод дозволяє зосередити логіку формування вхідних даних у єдиному місці, підвищуючи структурованість і повторне використання коду.

Розробка окремого класу для реалізації кластеризаційного алгоритму LIMBO дозволяє винести всю аналітичну логіку в ізольований функціональний модуль, незалежний від інтерфейсу та структури зберігання даних. Такий підхід відповідає принципам модульності та розділення обов’язків, що особливо важливо у випадку роботи з алгоритмами машинного навчання або інтелектуального аналізу даних. Клас LIMBOClusterer призначений для виконання групування регіональних записів за подібністю агроекологічних характеристик, підготовлених у форматі RegionData.

У наведеному на рис. 3.15 фрагменті коду визначено основну структуру класу LIMBOClusterer, в якому зберігається список вхідних даних _data.

```
internal class LIMBOClusterer {
  private List<RegionData> _data;

  1 reference
  public LIMBOClusterer(List<RegionData> data) {
    _data = data;
  }
```

Рис.3.15 Фрагмент методу «LoadDataFromDatabase»

У наведеному фрагменті коду визначено основну структуру класу LIMBOClusterer, в якому зберігається список вхідних даних _data. Передача цього здійснюється через конструктор, що дозволяє ініціалізувати кластеризатор безпосередньо під час створення об'єкта. Така реалізація забезпечує простоту виклику алгоритму в інших частинах системи, а також дозволяє застосовувати повторне кластеризування з різними наборами даних без зміни основної структури класу. У подальшому клас буде доповнений методами аналізу, обчислення евристик та формування кластерів на основі заданих атрибутів.

У наведеному на рис. 3.16 методі реалізовано обчислення ентропії Шеннона для набору аграрних даних на основі частот розподілу культур. Метод приймає колекцію об'єктів типу RegionData, у якій кожен елемент має значення культури, що вирощується в певному регіоні. Спочатку визначається загальна кількість записів у вибірці, після чого дані групуються за назвою культури. Для кожної групи обчислюється ймовірність її появи у вибірці, яка використовується для формування значення ентропії згідно з формулою інформаційної невизначеності.

```
private double CalculateEntropy(List<RegionData> data) {
  var total = data.Count;
  var cropGroups = data.GroupBy(x => x.Crop);
  double entropy = 0;
  foreach (var group in cropGroups) {
    double p = (double)group.Count() / total;
    entropy -= p * Math.Log(p, 2);
  }
  return entropy;
}
```

Рис.3.16 Код методу «CalculateEntropy»

Підсумкове значення відображає рівень різноманітності культур у сукупності даних: чим вища ентропія, тим більша неоднорідність у вирощуваних культурах. Такий показник є ключовим для алгоритму LIMBO, який використовує ентропійну міру для побудови ефективних кластерів.

Метод на рис. 3.17 реалізує обчислення приросту інформації, який використовується для визначення інформативності певного атрибута в процесі кластеризації. На початку визначається початкова ентропія всієї вибірки, яка відображає загальну невизначеність у розподілі культур. Далі за допомогою функції-селектора набір даних розбивається на підгрупи відповідно до значень обраного атрибута, наприклад, типу ґрунту чи кліматичної зони. Для кожної підгрупи обчислюється її частка в загальній вибірці, а також локальна ентропія, після чого ці значення сумуються з урахуванням ваги.

```
private double CalculateInformationGain(List<RegionData> data,
  Func<RegionData, string> selector) {
  double entropyBefore = CalculateEntropy(data);
  int total = data.Count;
  var partitions = data.GroupBy(selector);
  double weightedEntropy = 0;
  foreach (var partition in partitions) {
    double p = (double)partition.Count() / total;
    weightedEntropy += p * CalculateEntropy(partition.ToList());
  }
  return entropyBefore - weightedEntropy;
}
```

Рис.3.17 Код методу «CalculateEntropy»

Отриманий показник – зважена ентропія після розбиття – порівнюється з початковою ентропією, і на виході метод повертає величину виграшу інформації. Таким чином, можна оцінити, наскільки ефективно певний атрибут розділяє дані, що є ключовим критерієм для побудови релевантних кластерів у алгоритмі LIMBO.

У методі на рис. 3.18 реалізовано логіку автоматизованого визначення найінформативнішого атрибута для подальшого групування регіонів за схожими агроекологічними характеристиками. Спочатку обчислюється приріст інформації для кожного з потенційних атрибутів – типу ґрунту, кліматичної зони, кислотності ґрунту та глибини залягання води.

```
public Dictionary<string, List<RegionData>> ClusterByRegion() {
  var gains = new Dictionary<string, double> {
    { "SoilType", CalculateInformationGain(_data, r => r.SoilType) },
    { "ClimateZone", CalculateInformationGain(_data, r => r.ClimateZone) },
    { "SoilAcidity", CalculateInformationGain(_data, r => r.SoilAcidity) },
    { "WaterDepth", CalculateInformationGain(_data, r => r.WaterDepth) }
  };
  foreach (var g in gains)
    Console.WriteLine($"Gain({g.Key}) = {g.Value:F4}");
  var bestAttr = gains.Aggregate((a, b) => a.Value > b.Value ? a : b).Key;
  Func<RegionData, string> selector;
  if (bestAttr == "SoilType")
    selector = r => r.SoilType;
  else if (bestAttr == "ClimateZone")
    selector = r => r.ClimateZone;
  else if (bestAttr == "SoilAcidity")
    selector = r => r.SoilAcidity;
  else if (bestAttr == "WaterDepth")
    selector = r => r.WaterDepth;
  else
    selector = r => r.SoilType;
  return _data.GroupBy(r => r.RegionName)
  .ToDictionary(g => g.Key, g => g.ToList());
}
```

Рис.3.18 Код методу «ClusterByRegion»

Отримані значення зберігаються у словнику, який також використовується для виводу діагностичних повідомлень у консоль з метою контролю точності розрахунків. На основі максимального приросту інформації визначається атрибут, що найкраще розділяє дані, і на його основі задається відповідна функція-селектор. Проте, незалежно від обраного атрибута, фінальне групування даних у цьому фрагменті реалізується за назвою регіону, і результатом методу є словник, у якому ключем виступає назва регіону, а значенням – список об'єктів RegionData, що належать до нього. Такий підхід забезпечує гнучкість у виборі стратегії кластеризації та дозволяє легко адаптувати метод до подальших вдосконалень, зокрема включення більш глибокої ієрархічної структури кластерів.

На рис. 3.19 наведено метод, що реалізує механізм кластеризації даних за вказаною ознакою, яка передається як текстовий параметр. У залежності від значення аргументу attributeName, виконується групування колекції об’єктів RegionData за одним із ключових агроекологічних атрибутів: регіоном, типом ґрунту, кліматичною зоною, кислотністю ґрунту або глибиною залягання води.

```
public Dictionary<string, List<RegionData>> ClusterByAttribute(string attributeName) {
  Dictionary<string, List<RegionData>> result = new Dictionary<string, List<RegionData>>();
  switch (attributeName) {
    case "Region":
      result = _data.GroupBy(d => d.RegionName).ToDictionary(g => g.Key, g => g.ToList());
      break;
    case "SoilType":
      result = _data.GroupBy(d => d.SoilType).ToDictionary(g => g.Key, g => g.ToList());
      break;
    case "ClimateZone":
      result = _data.GroupBy(d => d.ClimateZone).ToDictionary(g => g.Key, g => g.ToList());
      break;
    case "SoilAcidity":
      result = _data.GroupBy(d => d.SoilAcidity).ToDictionary(g => g.Key, g => g.ToList());
      break;
    case "WaterDepth":
      result = _data.GroupBy(d => d.WaterDepth).ToDictionary(g => g.Key, g => g.ToList());
      break;
    default:
      throw new ArgumentException("Невідомий атрибут кластеризації: " + attributeName);
  }

  return result;
}
```

Рис.3.19 Код методу «ClusterByAttribute»

Групування здійснюється за допомогою методу GroupBy, а результати трансформуються у словник, де ключем є унікальне значення обраного атрибута, а значенням – список усіх записів, що йому відповідають. У випадку передання невідомої назви поля, викликається виняток з поясненням помилки, що дозволяє запобігти некоректному використанню методу. Така реалізація є зручною для побудови кластерів за конкретною ознакою, яка може бути обрана користувачем або системою динамічно.

# 4 ВПРОВАДЖЕННЯ СИСТЕМИ ТА РЕКОМЕНДАЦІЇ ЩОДО ЇЇ ЕКСПЛУАТАЦІЇ

## 4.1 Тестування програмного рішення

Розробка інформаційної системи для підтримки прийняття рішень у сільському господарстві базується на актуальній проблемі оптимального вибору культур для вирощування в умовах кліматичних змін, деградації ґрунтів та зростаючих економічних ризиків. Сільське господарство є однією з провідних галузей економіки України, тому впровадження ІТ-рішень у цей сектор дозволяє значно підвищити точність планування та ефективність використання природних ресурсів. З огляду на складність агроекологічних систем та багатофакторність впливу на урожайність, виникає необхідність у створенні програмного забезпечення, яке поєднує в собі аналітичні алгоритми кластеризації та дружній до користувача інтерфейс.

Щоб забезпечити надійність роботи такої системи, важливим етапом є верифікація функціональності всіх її компонентів через формалізовані тестові сценарії. Вони дозволяють підтвердити відповідність поведінки програми очікуваному результату в різних ситуаціях взаємодії. Одним з перших і базових сценаріїв є перевірка роботи механізму авторизації користувача.

*Тест-кейс №1*. Перевірка авторизації користувача у системі.

*Вхідні дані*: логін (admin), пароль (admin123).

*Послідовність виконання дій*:

– запустити програмне забезпечення для вибору оптимальних сільськогосподарських культур;

– дочекатися появи вікна авторизації;

– ввести у відповідне поле значення логіна: admin;

– у полі пароля ввести: admin123;

– натиснути кнопку «Увійти» або натиснути клавішу Enter;

– дочекатися реакції системи та оцінити результат.

*Очікуваний результат*: користувач успішно входить у систему, відображається головне вікно програми з відповідними доступними розділами згідно з роллю користувача. У разі некоректного введення даних система виводить повідомлення про помилку авторизації.

*Отриманий результат*: вхід до системи здійснено успішно; користувач перенаправлений до головного інтерфейсу програми (рис. 4.1).

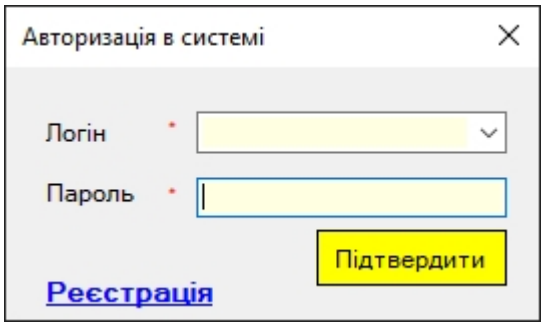


Рис.4.1 Результат тестування сценарію авторизації користувача

*Тест-кейс №2*. Додавання нового регіону до інформаційної бази

*Вхідні дані*: назва регіону, опис.

*Послідовність виконання дій*:

– запустити програму та перейти до розділу «Інформаційна база» → «Регіони»;

– у полі «Назва» ввести значення «Івано-Франківська»;

– у полі «Опис» зазначити коротку характеристику регіону: «Область з переважно гірським рельєфом та помірно-континентальним кліматом»;

– перевірити, що поле «Назва» заповнене (відмічене символом *);

– натиснути кнопку «Додати», розташовану у нижній частині форми;

– переконатися, що запис з новим регіоном з’явився у правій частині в таблиці;

– за потреби натиснути «Очистити» для скидання полів або «Вихід» для закриття форми.

*Очікуваний результат*: відомості про новий регіон успішно збережені в базі даних, а відповідний запис відображається у таблиці праворуч без помилок. Значення індексу № автоматично оновлюється з урахуванням нового запису.

*Отриманий результат*: регіон «Івано-Франківська» додано до системи; запис з'явився у списку на 11-й позиції з відповідною інформацією (рис. 4.2).

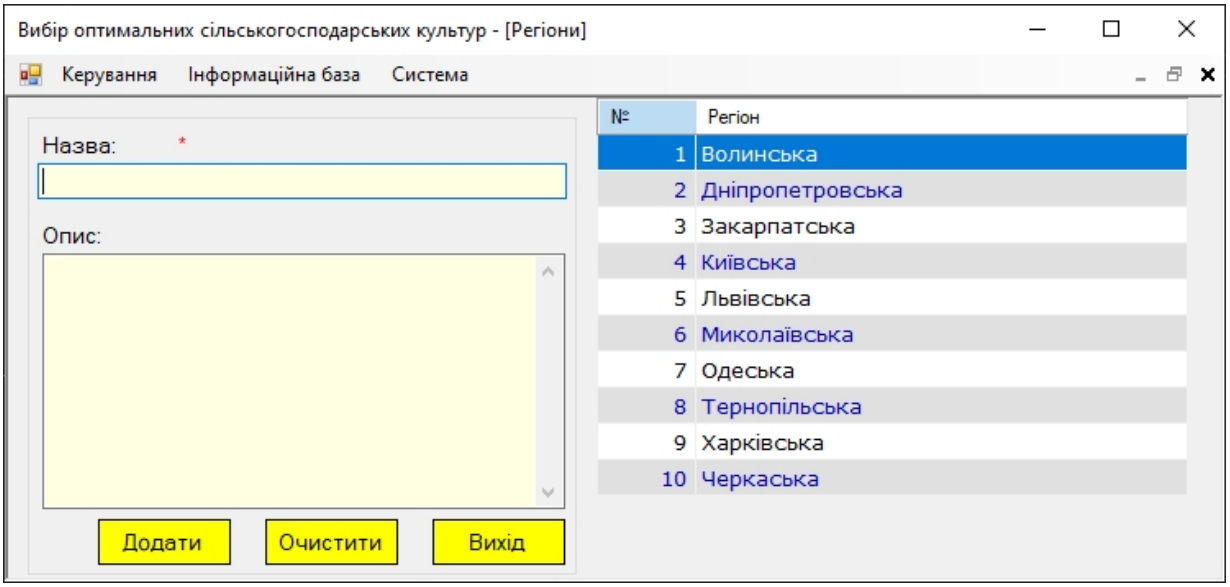


Рис.4.2 Результат тестування сценарію додавання нового регіону

*Тест-кейс №3*. Додавання нового типу ґрунту до системи

*Вхідні дані*: назва типу ґрунту, опис.

*Послідовність виконання дій*:

– перейти до розділу «Інформаційна база» → «Типи ґрунтів»;

– у полі «Назва» ввести значення «Карбонатний»;

– у текстове поле «Опис» внести відповідну характеристику: «Ґрунт з високим вмістом карбонатів, характерний для посушливих регіонів»;

– перевірити правильність заповнення обов'язкових полів;

– натиснути кнопку «Додати» для збереження даних у базі;

– переконатися, що новий тип ґрунту з'явився в таблиці праворуч;

– натиснути «Очистити» для скидання введених значень або «Вихід» для завершення роботи з формою.

*Очікуваний результат*: система успішно додає новий запис до бази даних, оновлює таблицю на екрані, новий тип ґрунту з’являється наприкінці списку без помилок.

*Отриманий результат*: тип ґрунту «Карбонатний» успішно збережено та відображено в списку як одинадцятий елемент (рис. 4.3).

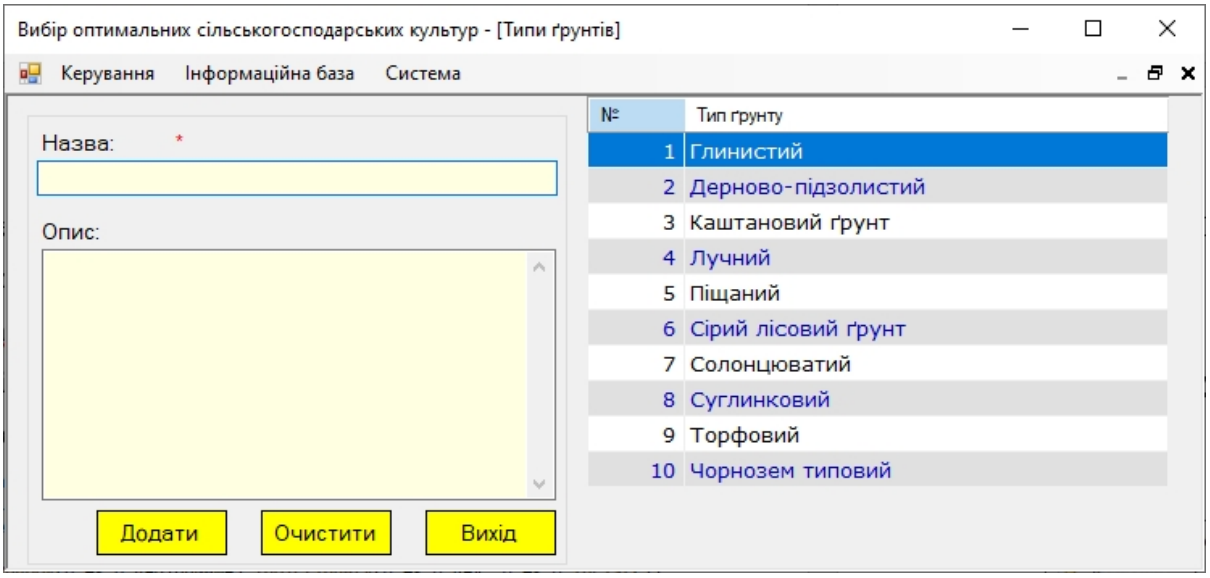


Рис.4.3 Результат тестування сценарію додавання типу ґрунту

*Тест-кейс №4.* Додавання нової характеристики регіону до інформаційної бази

*Вхідні дані*: район, тип ґрунту, кліматична зона, культура, кислотність ґрунту, глибина залягання води.

*Послідовність виконання дій*:

– відкрити розділ «Інформаційна база» → «Характеристики регіонів та вирощувані культури»;

– у комбінованому полі «Район» вибрати значення «Волинська»;

– послідовно вибрати з відповідних полів: «Глинистий» як тип ґрунту, «Гірська» як кліматичну зону, «Буряк цукровий» як культуру;

– в полі «Кислотність ґрунту» встановити значення «Кисла»;

– в полі «Глибина залягання води» вибрати «Глибока»;

– перевірити коректність введених значень;

– натиснути кнопку «Додати» для збереження інформації;

– переконатися, що нова характеристика регіону з’явилася у таблиці праворуч.

*Очікуваний результат*: система успішно додає характеристику регіону до бази даних. Новий запис відображається в таблиці з усіма вказаними параметрами без помилок, у відповідному форматі.

*Отриманий результат*: у таблиці з’явився запис із заданими значеннями параметрів, який відображено у нижній частині списку (рис. 4.4).


Рис.4.4 Результат тестування сценарію додавання характеристики регіону

*Тест-кейс №5*. Виконання кластеризації для вибору оптимальних сільськогосподарських культур

*Вхідні дані*: атрибут для кластеризації.

*Послідовність виконання дій*:

– запустити програму та перейти до розділу «Система» → «Вибір оптимальних культур»;

– у випадаючому списку «Атрибут» обрати значення «Region»;

– натиснути кнопку для запуску кластеризації (автоматично або через подію SelectedIndexChanged);

– дочекатися відображення результатів кластерного аналізу в текстовому полі;

– ознайомитися з розподілом регіонів по кластерах, кожен з яких містить список культур і рекомендацію щодо найдоцільнішої культури для вирощування.

*Очікуваний результат*: система коректно виконує кластеризацію за обраним атрибутом, виводить структурований звіт із розподілом регіонів по кластерах, зазначенням рекомендованих культур та фактичного складу кожного кластеру.

*Отриманий результат*: на формі відображено результати кластеризації за атрибутом «Region», де для кожного кластера вказано перелік регіонів та культуру, що найчастіше зустрічається серед них. Інтерфейс чітко і читабельно подає підсумки обробки (рис. 4.5).

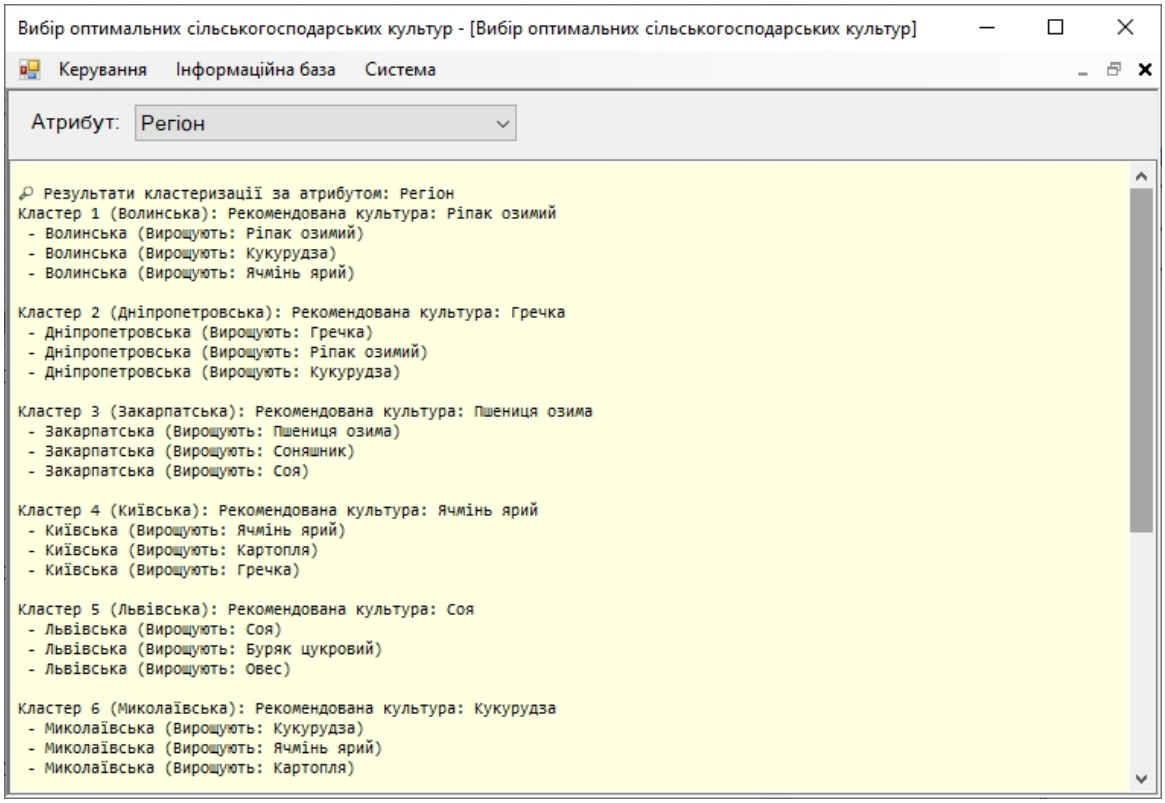


Рис.4.5 Результат тестування сценарію кластеризації для вибору оптимальних культур

Модульне тестування є невід’ємною складовою сучасного процесу розробки якісного та надійного програмного забезпечення. Його головна мета полягає у верифікації коректності роботи окремих логічно завершених компонентів до їх об’єднання в загальну архітектуру. Такий підхід дозволяє виявити помилки на ранніх етапах розробки, що суттєво знижує витрати на виправлення та спрощує відстеження джерела проблеми. Крім того, модульне

тестування сприяє підвищенню стабільності та прогнозованості системи в цілому, оскільки гарантує правильну поведінку кожного окремого модуля незалежно від інших частин програми. Особливо актуально це у випадках роботи з аналітичними модулями, де обробка аграрних даних і формування кластерів є критично важливими для точності кінцевих рекомендацій. Забезпечення тестування таких методів, як обчислення ентропії, інформаційного приросту або класифікація за агроекологічними параметрами, є обов'язковою умовою надійності системи. Модульні тести також створюють фундамент для впровадження безпечних оновлень і швидкої перевірки працездатності системи після змін у кодовій базі.

На рис. 4.6 наведено результати модульного тестування основних функціональних компонентів системи кластеризації сільськогосподарських культур.

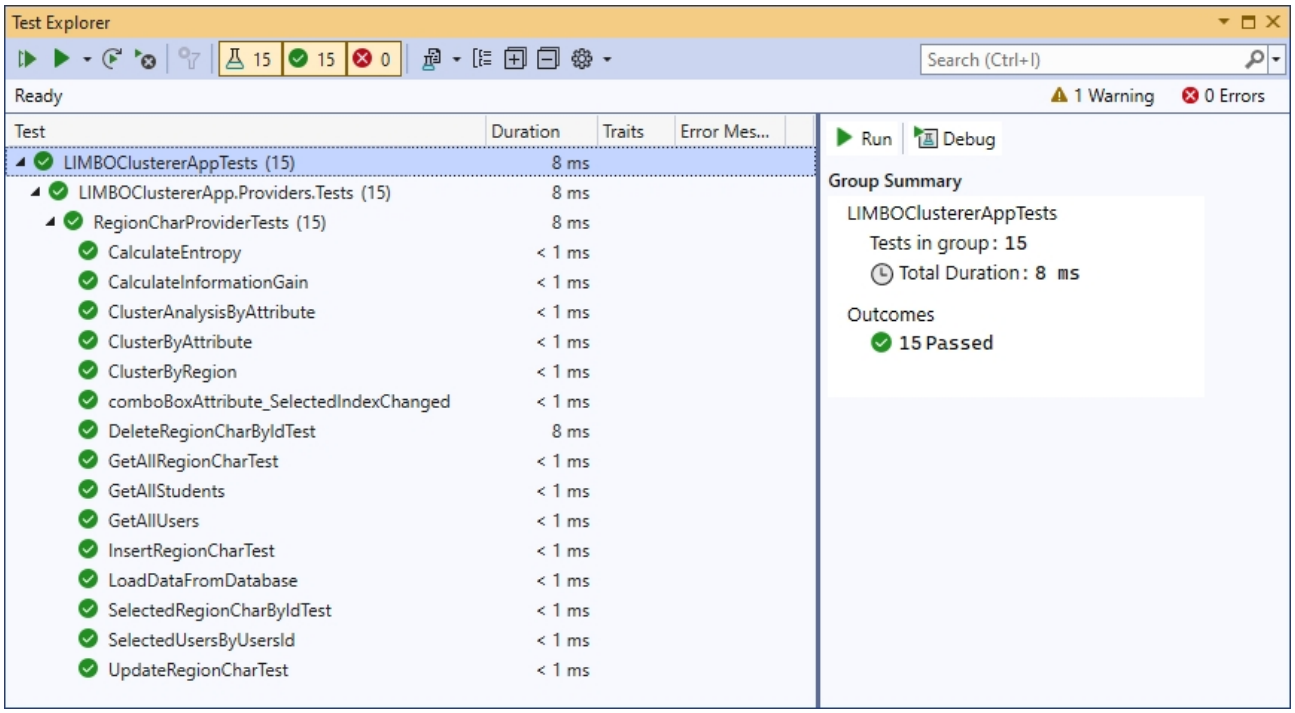


Рис.4.6 Результати модульного тестування системи

Загалом було проведено 15 тестів, усі з яких завершилися успішно. Серед протестованих методів – обчислення ентропії, приросту інформації, кластеризація за різними атрибутами, а також операції роботи з даними в таблиці характеристик регіонів. Всі тести виконано менше ніж за мілісекунду, що

свідчить про високу оптимізованість алгоритмів та ефективність реалізації окремих модулів системи. Успішне проходження тестів підтверджує коректність основних функціональних механізмів розробленого програмного забезпечення.

## 4.2 Визначення технічних вимог до використання системи

Для забезпечення стабільної та ефективної роботи програмного забезпечення, призначеного для вибору оптимальних сільськогосподарських культур на основі кластеризаційного алгоритму LIMBO, необхідно визначити мінімальні та рекомендовані технічні вимоги до апаратного та програмного середовища. Такі вимоги є критично важливими для забезпечення безперебійної роботи всіх модулів системи, зокрема аналітичних підсистем, механізмів обробки даних і формування звітів.

З огляду на характер задач, які вирішує система, а також обсяг інформації, що обробляється (таблиці з регіональними характеристиками, аграрні атрибути, результати кластеризації), можна вважати навантаження середнім. Це дозволяє впроваджувати систему на персональних комп’ютерах сільськогосподарських підприємств, навчальних закладів або локальних адміністрацій, не вимагаючи високопродуктивного серверного обладнання.

Мінімальні технічні вимоги:

- операційна система: Windows 10 або новіша версія;
- процесор: Intel Core i3 або AMD Ryzen 3;
- оперативна пам’ять (RAM): не менше 4 ГБ;
- місце на диску: щонайменше 250 МБ для встановлення та зберігання бази даних;
- роздільна здатність екрану: 1366×768 пікселів;
- програмне забезпечення: .NET Framework 4.7.2 або вище, Microsoft SQL Server LocalDB.

Рекомендовані технічні вимоги:

- операційна система: Windows 11 Pro;

– процесор: Intel Core i5 або AMD Ryzen 5;

– оперативна пам’ять (RAM): 8 ГБ і більше;

– накопичувач: SSD-диск з вільним обсягом щонайменше 500 МБ для зберігання історичних агроданих;

– інше ПЗ: SQL Server Management Studio (SSMS) для адміністрування БД, бібліотеки ML.NET (за потреби розширення функцій).

Усі інтерфейсні компоненти системи адаптовані до роботи у середовищі Windows Forms, що забезпечує коректне відображення форм та елементів керування на більшості робочих станцій. Підключення до локальної бази даних виконується через вбудований SQL-клієнт, що не потребує окремої серверної інфраструктури. Завдяки оптимізації алгоритмів кластеризації, навіть на мінімальній конфігурації користувач може оперативно отримати результати аналізу.

З метою забезпечення стабільного функціонування програмного забезпечення було спроєктовано просту, але гнучку архітектуру розгортання, яка дозволяє ефективно працювати як в умовах ізольованої одиночної машини, так і в локальній мережі. На рівні фізичної реалізації передбачено розміщення основного клієнтського застосунку на персональному комп’ютері користувача, тоді як база даних, що зберігає всі агроекологічні характеристики, може бути інстальована локально або на віддаленому сервері з доступом через TCP/IP.

На рис. 4.7 наведено діаграму розгортання системи, яка демонструє взаємозв’язок між ключовими компонентами розробленого рішення.

У структурі розгортання виділено два основних вузли: ПК користувача, де виконується клієнтський застосунок LIMBOClustererApp, та сервер бази даних, що відповідає за зберігання, обробку та видачу інформації. Взаємодія між ними реалізується через стандартні протоколи з’єднання (наприклад, TCP/IP), що дозволяє гнучко налаштовувати середовище як для автономного, так і для багатокористувацького режиму роботи.

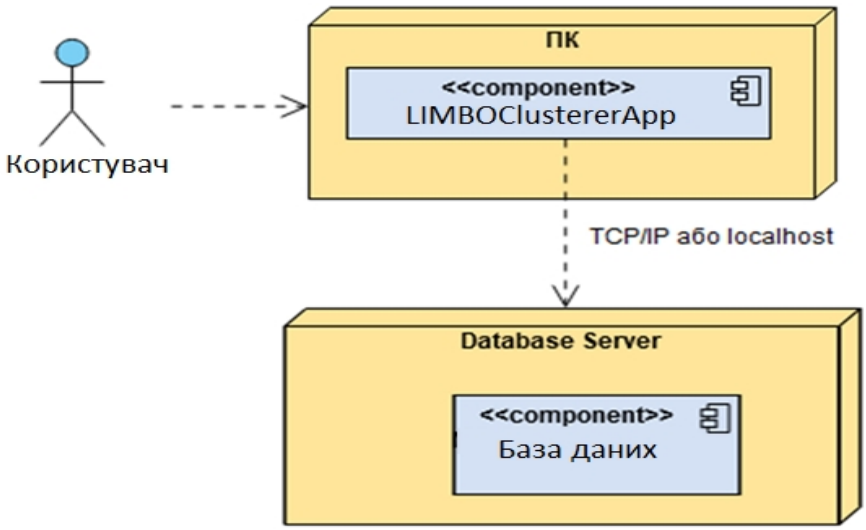


Рис.4.7 Діаграма розгортання системи

Така конфігурація забезпечує ізольованість бізнес-логіки в межах застосунку та надійність збереження аграрних даних, що критично важливо для агроаналітичних систем.

## 4.3 Склад інсталяційного пакету

Для забезпечення зручності розгортання та використання програмного забезпечення, яке розроблено з метою підтримки прийняття рішень щодо вибору оптимальних сільськогосподарських культур з використанням кластерного аналізу за алгоритмом LIMBO, сформовано інсталяційний пакет, що всі необхідні компоненти для запуску та функціонування системи на локальному комп’ютері користувача.

У таблиці 4.1 представлено склад інсталяційного пакету із зазначенням призначення кожного з його елементів.

Пакет забезпечує повністю автономне встановлення системи без потреби у додаткових сторонніх інструментах або спеціальних навичках з боку користувача. Після копіювання інсталяційної папки на робочу машину достатньо лише запустити виконуваний файл – система автоматично під’єднається до бази, зчитавши налаштування з конфігураційного файлу.

Таблиця 4.1

Структура інсталяційного пакету програмного забезпечення

| № | Назва компонента | Формат файлу / розширення | Призначення |
|---|---|---|---|
| 1 | LIMBOClustererApp | .exe | Головний виконуваний файл застосунку. Користувач отримує доступ до функціоналу програми, зокрема до інтерфейсів обліку аграрних характеристик, запуску кластеризації та генерації звітів |
| 2 | DB | .mdf | Локальна база даних із таблицями агроекологічної інформації. Містить усі таблиці, необхідні для роботи системи. База даних працює у режимі локального підключення (LocalDB) без потреби використання окремого серверного середовища |
| 3 | App.config | .config | Файл налаштувань з'єднання та параметрів запуску. Містить параметри з’єднання з базою даних, а також допоміжні налаштування, що впливають на взаємодію між елементами системи. |
| 4 | Збірки бібліотек | .dll | Модулі обробки даних, кластеризації та доступу до БД. Реалізують доступ до даних, обробку аналітичної логіки та виконання алгоритму кластеризації LIMBO. |
| 5 | Форми інтерфейсу | .cs | Компоненти графічного інтерфейсу користувача (WinForms), які відповідають за візуальне відображення, введення та обробку даних. |

Такий підхід значно спрощує процес впровадження системи в аграрних підприємствах, навчальних закладах або в умовах індивідуального використання.

# ВИСНОВКИ

Дана кваліфікаційна робота присвячена розробці програмного забезпечення, що забезпечує вибір оптимальних сільськогосподарських культур для певних регіонів на основі кластеризації із використанням алгоритму LIMBO. Основною метою роботи стало створення зручної та ефективної системи, здатної аналізувати сукупність агроекологічних характеристик та на їх основі формувати обґрунтовані рекомендації щодо культур, які є найдоцільнішими для вирощування в умовах конкретного регіону.

У першому розділі була детально проаналізована предметна область, зокрема розглянуто ключові фактори, що впливають на вибір культур — кліматичні умови, властивості ґрунтів, доступність водних ресурсів, ринковий попит та економічна доцільність. На основі цього побудовано діаграму проблематики вибору культур, охарактеризовано джерела даних, що використовуються для аграрного аналізу, та обґрунтовано вибір алгоритму LIMBO як оптимального для задач кластеризації з дискретними атрибутами. Крім того, сформульовано функціональні та нефункціональні вимоги до системи, побудовано діаграми варіантів використання для адміністратора й користувача, а також проведено огляд трьох програмних аналогів (Agremo, FieldView, Agrian), що дозволило обґрунтувати необхідність створення власного, більш адаптованого рішення.

Другий розділ висвітлює процес проєктування інформаційної структури системи. Описано таблиці бази даних, побудовано її логічну модель у нотації Чена, розроблено діаграми класів та кооперацій, які відображають архітектуру модулів і процес взаємодії між об'єктами при виконанні кластерного аналізу. Структуровано компоненти системи у вигляді діаграми пакетів, а також описано розгортання функціональних елементів через діаграму компонентів.

Третій розділ був присвячений безпосередній реалізації програмного забезпечення. Розглянуто декілька варіантів систем управління базами даних, після чого обґрунтовано вибір MS SQL Server як найбільш відповідного

середовища для зберігання аграрної інформації. Створено фізичну модель бази даних на основі логічної структури, реалізовано таблиці та описано ключові сутності. Для створення прикладної частини системи було обрано C# як найбільш ефективний засіб для побудови Windows-додатку з графічним інтерфейсом і вбудованою логікою кластеризації. Реалізовано набір модулів: обробка авторизації користувача, управління довідниками, формування характеристик регіонів, кластеризація за алгоритмом LIMBO з використанням інформаційної ентропії, а також формування звітів. Описано ключові класи й методи, зокрема механізми обчислення інформаційного виграшу, аналізу атрибутів та групування даних за обраним критерієм.

У четвертому розділі проведено експериментальне тестування розробленого програмного забезпечення. Побудовано тестові сценарії для перевірки роботи основних форм системи, зокрема авторизації, додавання нових регіонів, типів ґрунтів та характеристик, а також запуску кластеризації з отриманням рекомендованих культур. Результати модульного тестування підтвердили коректність функціонування ключових методів, зокрема обробки аграрних даних та кластерного аналізу. Визначено мінімальні та рекомендовані технічні вимоги до середовища використання системи. Наведено діаграму розгортання, яка ілюструє взаємодію клієнтського застосунку з базою даних. Також описано склад інсталяційного пакету, що дозволяє швидко встановити та налаштувати ПЗ на локальній машині без потреби у сторонніх компонентах.

Проведена робота демонструє застосування кластеризаційного підходу на основі інформаційно-теоретичних принципів у задачах аграрного прогнозування, зокрема у виборі культур відповідно до локальних особливостей регіонів. Результати дослідження засвідчують, що розроблена система дозволяє аналізувати сукупність агроекологічних факторів, виявляти закономірності та формувати рекомендації щодо вирощування культур із найбільшим потенціалом. Це створює перспективу для практичного застосування системи в малих і середніх фермерських господарствах, а також для подальшого розширення її функціоналу в рамках адаптивного землеробства.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Karimi V., Karami E., Karami S., Keshavarz M. Adaptation to climate change through agricultural paradigm shift. Environment, Development and Sustainability. 2021. Vol. 23, No 2, pp. 465-85.

2. Hossain A., Krupnik T., Timsina J., Mahboob M., Chaki A., Farooq M., Hasanuzzaman M. Agricultural land degradation: processes and problems undermining future food security. In Environment, climate, plant and vegetation growth. Cham: Springer International Publishing. 2020. Vol. 1, pp. 17-61.

3. Castrignano A., Buttafuoco G., Khosla R., Mouazen A., Moshou D., Naud O. (Eds.) Agricultural Internet of Things and Decision Support for Precision Smart Farming. Academic Press, 2020. 470 p.

4. Teymourian N., Izadkhah H., Isazadeh, A. A fast clustering algorithm for modularization of large-scale software systems. IEEE Transactions on Software Engineering. 2020. Vol. 48, No 4, pp. 451-462.

5. Manos B., Matsatsinis N., Paparrizos K. Decision Support Systems in Agriculture, Food and the Environment: Trends, Applications and Advances. IGI Global, 2020. 556 p.

6. About Agremo. URL: https://sourceforge.net/software/product/Agremo/ (дата звернення 22.04.2025).

7. Plant Count & Size: Guide - Agremo. URL: https://www.agremo.com/documentation/plant-count-and-size/ (дата звернення 20.04.2025).

8. Top 10 Agremo Alternatives & Competitors in 2025. URL: https://www.g2.com/products/agremo/competitors/alternatives (дата звернення 24.04.2025).

9. Compare Agremo vs. FarmQA. URL: https://slashdot.org/software/comparison/Agremo-vs-FarmQA/ (дата звернення 12.04.2025).

10. Reviews of Viewpoint Field View. URL: https://www.capterra.com/p/240802/Viewpoint-FieldView/reviews/ (дата звернення 10.04.2025).

11. FieldView 201. URL: https://www.climatefieldview.co.za/getting-started/fieldview-201/ (дата звернення 04.05.2025).

12. Viewpoint Field View. URL: https://www.softwareadvice.com/construction/viewpoint-field-view-profile/ (дата звернення 28.04.2025).

13. Climate FieldView Reviews 2025: Details, Pricing, & Features. URL: https://www.g2.com/products/climate-fieldview/reviews (дата звернення 20.04.2025).

14. Agrian Reviews: What Is It Like to Work. URL: https://www.glassdoor.com/Reviews/Agrian-Reviews-E1427005.htm (дата звернення 24.04.2025).

15. Agrian Releases Updated Ag Data Management Platform - Global Ag Tech Initiative. URL: https://www.globalagtechinitiative.com/digital-farming/data-management/agrian-releases-updated-ag-data-management-platform/ (дата звернення 14.04.2025).

16. Working at Agrian. URL: https://www.glassdoor.com/Overview/Working-at-Agrian-EI_IE1427005.11,17.htm (дата звернення 24.04.2025).

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

18. Newman J., Zaitsev C., Zawodny J. High Performance MySQL: Optimization, Backups, and Replication. 4th ed. Sebastopol: O'Reilly Media, 2022. 830 с.

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

20. Балабанов О.І., Павленко А.П. PyCharm для розробників Python: Навчальний посібник. - Львів: Видавництво Львівської політехніки, 2018. – 300 с.

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

22. Price M. J. C# 11 and . NET 7 - Modern Cross-Platform Development Fundamentals: Start Building Websites and Services with ASP. NET Core 7, Blazor, and EF Core 7, 7th Edition. Packt Publishing, Limited, 2022. – 681 p.

# Додаток А. Скрипти створення бази даних

```
USE [master]
GO
/****** Object:  Database [LIMBO]    Script Date: 26.04.2025 10:57:01 ******/
CREATE DATABASE [LIMBO]
 CONTAINMENT = NONE
 ON  PRIMARY
( NAME = N'LIMBO', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\LIMBO.mdf' , SIZE = 8192KB ,
MAXSIZE = UNLIMITED, FILEGROWTH = 65536KB )
 LOG ON
( NAME = N'LIMBO_log', FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\LIMBO_log.ldf' , SIZE = 8192KB ,
MAXSIZE = 2048GB , FILEGROWTH = 65536KB )
 WITH CATALOG_COLLATION = DATABASE_DEFAULT, LEDGER = OFF
USE [LIMBO]
GO
/****** Object:  Table [dbo].[ClimateZone]    Script Date: 26.04.2025 10:57:01 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[ClimateZone](
	[ClimateZoneId] [int] IDENTITY(1,1) NOT NULL,
	[ClimateZoneName] [nvarchar](200) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[ClimateZoneId] 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].[Crop]    Script Date: 26.04.2025 10:57:01 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Crop](
	[CropId] [int] IDENTITY(1,1) NOT NULL,
	[CropName] [nvarchar](200) 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].[Logs]    Script Date: 26.04.2025 10:57:01 ******/
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].[Region]    Script Date: 26.04.2025 10:57:01 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Region](
	[RegionId] [int] IDENTITY(1,1) NOT NULL,
	[RegionName] [nvarchar](200) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[RegionId] 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].[RegionChar]    Script Date: 26.04.2025 10:57:01 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[RegionChar](
	[RegionCharId] [int] IDENTITY(1,1) NOT NULL,
	[RegionId] [int] NULL,
	[SoilTypeId] [int] NULL,
	[ClimateZoneId] [int] NULL,
	[CropId] [int] NULL,
	[SoilAcidity] [nvarchar](200) NULL,
	[WaterDepth] [nvarchar](200) NULL,
PRIMARY KEY CLUSTERED
(
	[RegionCharId] 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].[SoilType]    Script Date: 26.04.2025 10:57:01 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[SoilType](
	[SoilTypeId] [int] IDENTITY(1,1) NOT NULL,
	[SoilTypeName] [nvarchar](200) NULL,
	[Description] [nvarchar](max) NULL,
PRIMARY KEY CLUSTERED
(
	[SoilTypeId] 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: 26.04.2025 10:57:01 ******/
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
ALTER TABLE [dbo].[Logs]  WITH CHECK ADD  CONSTRAINT [FK_Logs_Users]
FOREIGN KEY([UsersId])
REFERENCES [dbo].[Users] ([UsersId])
GO
ALTER TABLE [dbo].[Logs] CHECK CONSTRAINT [FK_Logs_Users]
GO
ALTER TABLE [dbo].[RegionChar]  WITH CHECK ADD  CONSTRAINT
[FK_RegionChar_ClimateZone] FOREIGN KEY([ClimateZoneId])
REFERENCES [dbo].[ClimateZone] ([ClimateZoneId])
```

```
GO
ALTER TABLE [dbo].[RegionChar] CHECK CONSTRAINT [FK_RegionChar_ClimateZone]
GO
ALTER TABLE [dbo].[RegionChar]  WITH CHECK ADD  CONSTRAINT
[FK_RegionChar_Crop] FOREIGN KEY([CropId])
REFERENCES [dbo].[Crop] ([CropId])
GO
ALTER TABLE [dbo].[RegionChar] CHECK CONSTRAINT [FK_RegionChar_Crop]
GO
ALTER TABLE [dbo].[RegionChar]  WITH CHECK ADD  CONSTRAINT
[FK_RegionChar_Region] FOREIGN KEY([RegionId])
REFERENCES [dbo].[Region] ([RegionId])
GO
ALTER TABLE [dbo].[RegionChar] CHECK CONSTRAINT [FK_RegionChar_Region]
GO
ALTER TABLE [dbo].[RegionChar]  WITH CHECK ADD  CONSTRAINT
[FK_RegionChar_SoilType] FOREIGN KEY([SoilTypeId])
REFERENCES [dbo].[SoilType] ([SoilTypeId])
GO
ALTER TABLE [dbo].[RegionChar] CHECK CONSTRAINT [FK_RegionChar_SoilType]
GO
USE [master]
GO
ALTER DATABASE [LIMBO] SET  READ_WRITE
GO
```

# Додаток Б. Лістинги розробленої програми

## Лістинг Б.1 Код класу «SelectionOptimalCropsForm»

```
using LIMBOClustererApp.AppCode;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace LIMBOClustererApp.Forms.Controls {
  public partial class SelectionOptimalCropsForm : Form {
    RaportBLL _RaportBLL = new RaportBLL();
    private Dictionary<string, string> _attributeMap = new Dictionary<string, string> {
      { "Регіон", "Region" },
      { "Тип ґрунту", "SoilType" },
      { "Кліматична зона", "ClimateZone" },
      { "Кислотність ґрунту", "SoilAcidity" },
      { "Глибина залягання води", "WaterDepth" }
    };

    public SelectionOptimalCropsForm() {
      InitializeComponent();
      comboBoxAttribute.Items.AddRange(_attributeMap.Keys.ToArray());
      comboBoxAttribute.SelectedIndex = 0;
      ClusterAnalysisByAttribute();
    }

    private void ClusterAnalysisByAttribute() {
      var data = _RaportBLL.LoadDataFromDatabase();
      var clusterer = new LIMBOClusterer(data);

      // Отримуємо обране українське значення
      string selectedUkrAttr = comboBoxAttribute.SelectedItem.ToString();

      // Отримуємо англійське значення для кластеризації
      string internalAttr = _attributeMap[selectedUkrAttr];

      var clusters = clusterer.ClusterByAttribute(internalAttr);

      RaportTBox.Text = $"\r\n🔎 Результати кластеризації за атрибутом:
{selectedUkrAttr}";

      int clusterNum = 1;
      foreach (var cluster in clusters) {
```

```
        RaportTBox.Text += $"\r\nКластер {clusterNum} ({cluster.Key}):";

        var recommendedCrop = cluster.Value
           .GroupBy(r => r.Crop)
           .OrderByDescending(g => g.Count())
           .First().Key;

        RaportTBox.Text += $" Рекомендована культура: {recommendedCrop}\r\n";

        foreach (var region in cluster.Value)
          RaportTBox.Text += $" - {region.RegionName} (Вирощують: {region.Crop})\r\n";

        clusterNum++;
      }
    }

    private void comboBoxAttribute_SelectedIndexChanged(object sender, EventArgs e) {
      ClusterAnalysisByAttribute();
    }
  }
}
```

### Лістинг Б.2 Код класу «RegionCharProvider»

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

namespace LIMBOClustererApp.Providers {
  public class RegionCharProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertRegionChar(int regionId, int soilTypeId, int climateZoneId, int cropId,
      string soilAcidity, string waterDepth) {
      string SqlString = @"INSERT INTO RegionChar
    (RegionId, SoilTypeId, ClimateZoneId, CropId, SoilAcidity, WaterDepth)
    VALUES (@RegionId, @SoilTypeId, @ClimateZoneId, @CropId, @SoilAcidity,
@WaterDepth)";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@RegionId", regionId);
          cmd.Parameters.AddWithValue("@SoilTypeId", soilTypeId);
          cmd.Parameters.AddWithValue("@ClimateZoneId", climateZoneId);
          cmd.Parameters.AddWithValue("@CropId", cropId);
          cmd.Parameters.AddWithValue("@SoilAcidity", soilAcidity);
          cmd.Parameters.AddWithValue("@WaterDepth", waterDepth);
```

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

public List<RegionChar> GetAllRegionChar() {
  int num = 0;
  string SqlString = @"
            SELECT rc.*,
                r.RegionName,
                s.SoilTypeName,
                cz.ClimateZoneName,
                c.CropName
            FROM RegionChar rc
            INNER JOIN Region r ON rc.RegionId = r.RegionId
            INNER JOIN SoilType s ON rc.SoilTypeId = s.SoilTypeId
            INNER JOIN ClimateZone cz ON rc.ClimateZoneId = cz.ClimateZoneId
            INNER JOIN Crop c ON rc.CropId = c.CropId
            ORDER BY rc.RegionCharId";

  List<RegionChar> listAllRegionChar = new List<RegionChar>();

  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
      conn.Open();
      using (SqlDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          RegionChar oneChar = new RegionChar();
          oneChar.Number = ++num;
          oneChar.RegionCharId = Convert.ToInt32(reader["RegionCharId"]);
          oneChar.RegionId = Convert.ToInt32(reader["RegionId"]);
          oneChar.SoilTypeId = Convert.ToInt32(reader["SoilTypeId"]);
          oneChar.ClimateZoneId = Convert.ToInt32(reader["ClimateZoneId"]);
          oneChar.CropId = Convert.ToInt32(reader["CropId"]);
          oneChar.SoilAcidity = reader["SoilAcidity"].ToString();
          oneChar.WaterDepth = reader["WaterDepth"].ToString();

          oneChar.RegionName = reader["RegionName"].ToString();
          oneChar.SoilTypeName = reader["SoilTypeName"].ToString();
          oneChar.ClimateZoneName = reader["ClimateZoneName"].ToString();
          oneChar.CropName = reader["CropName"].ToString();

          listAllRegionChar.Add(oneChar);
        }
      }
      conn.Close();
    }
  }

  return listAllRegionChar;
}
```

```
    public RegionChar SelectedRegionCharById(int regionCharId) {
      string SqlString = "SELECT * FROM RegionChar WHERE
RegionCharId=@RegionCharId";

      RegionChar oneChar = new RegionChar();
      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.Parameters.AddWithValue("@RegionCharId", regionCharId);
          conn.Open();
          using (SqlDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              oneChar.RegionCharId = Convert.ToInt32(reader["RegionCharId"]);
              oneChar.RegionId = Convert.ToInt32(reader["RegionId"]);
              oneChar.SoilTypeId = Convert.ToInt32(reader["SoilTypeId"]);
              oneChar.ClimateZoneId = Convert.ToInt32(reader["ClimateZoneId"]);
              oneChar.CropId = Convert.ToInt32(reader["CropId"]);
              oneChar.SoilAcidity = reader["SoilAcidity"].ToString();
              oneChar.WaterDepth = reader["WaterDepth"].ToString();
            }
          }
          conn.Close();
        }
      }
      return oneChar;
    }

    public void UpdateRegionChar(int regionId, int soilTypeId, int climateZoneId, int cropId,
      string soilAcidity, string waterDepth, int regionCharId) {
      string SqlString = @"UPDATE RegionChar
    SET RegionId = @RegionId,
        SoilTypeId = @SoilTypeId,
        ClimateZoneId = @ClimateZoneId,
        CropId = @CropId,
        SoilAcidity = @SoilAcidity,
        WaterDepth = @WaterDepth
    WHERE RegionCharId = @RegionCharId";

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("@RegionId", regionId);
          cmd.Parameters.AddWithValue("@SoilTypeId", soilTypeId);
          cmd.Parameters.AddWithValue("@ClimateZoneId", climateZoneId);
          cmd.Parameters.AddWithValue("@CropId", cropId);
          cmd.Parameters.AddWithValue("@SoilAcidity", soilAcidity);
          cmd.Parameters.AddWithValue("@WaterDepth", waterDepth);
          cmd.Parameters.AddWithValue("@RegionCharId", regionCharId);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
```

```
    }

    public void DeleteRegionCharById(int regionCharId) {
      string SqlString = "DELETE FROM RegionChar WHERE RegionCharId =
@RegionCharId";

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

  }
}

public class RegionChar {
  // Порядковий номер для виведення у звітах чи таблицях
  private int _Number;

  // Унікальний ідентифікатор характеристики регіону
  private int _RegionCharId;

  // Ідентифікатор регіону (зовнішній ключ на Regions)
  private int _RegionId;

  // Ідентифікатор типу ґрунту (зовнішній ключ на SoilTypes)
  private int _SoilTypeId;

  // Ідентифікатор кліматичної зони (зовнішній ключ на ClimateZones)
  private int _ClimateZoneId;

  // Ідентифікатор культури (зовнішній ключ на Crops)
  private int _CropId;

  // Кислотність ґрунту (наприклад: "Кисла", "Нейтральна")
  private string _SoilAcidity;

  // Глибина залягання вод (наприклад: "Глибока", "Середня")
  private string _WaterDepth;

  // Повідомлення або додатковий коментар
  private string _Message;

  public RegionChar() {
    _Number = 0;
    _RegionCharId = 0;
    _RegionId = 0;
    _SoilTypeId = 0;
    _ClimateZoneId = 0;
```

```
  _CropId = 0;
  _SoilAcidity = string.Empty;
  _WaterDepth = string.Empty;
  _Message = string.Empty;
}

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

public int RegionCharId {
  get { return _RegionCharId; }
  set { _RegionCharId = value; }
}

public int RegionId {
  get { return _RegionId; }
  set { _RegionId = value; }
}

public int SoilTypeId {
  get { return _SoilTypeId; }
  set { _SoilTypeId = value; }
}

public int ClimateZoneId {
  get { return _ClimateZoneId; }
  set { _ClimateZoneId = value; }
}

public int CropId {
  get { return _CropId; }
  set { _CropId = value; }
}

public string SoilAcidity {
  get { return _SoilAcidity; }
  set { _SoilAcidity = value; }
}

public string WaterDepth {
  get { return _WaterDepth; }
  set { _WaterDepth = value; }
}

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

public string ClimateZoneName { get; set; }
public string CropName { get; set; }
```

```
  public string RegionName { get; set; }
  public string SoilTypeName { get; set; }

}
```

**Лістинг 3. Код класу «RaportBLL»**

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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

    public List<RegionData> LoadDataFromDatabase() {
      string SqlString = @"
    SELECT
      r.RegionName,
      s.SoilTypeName AS SoilType,
      cz.ClimateZoneName AS ClimateZone,
      rc.SoilAcidity,
      rc.WaterDepth,
      c.CropName AS Crop
    FROM RegionChar rc
    INNER JOIN Region r ON rc.RegionId = r.RegionId
    INNER JOIN SoilType s ON rc.SoilTypeId = s.SoilTypeId
    INNER JOIN ClimateZone cz ON rc.ClimateZoneId = cz.ClimateZoneId
    INNER JOIN Crop c ON rc.CropId = c.CropId
    ORDER BY r.RegionName";

      List<RegionData> regionDataList = new List<RegionData>();

      using (SqlConnection conn = new SqlConnection(_ConnString)) {
        using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
          conn.Open();
          using (SqlDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              RegionData data = new RegionData {
                RegionName = reader["RegionName"].ToString(),
                SoilType = reader["SoilType"].ToString(),
                ClimateZone = reader["ClimateZone"].ToString(),
                SoilAcidity = reader["SoilAcidity"].ToString(),
                WaterDepth = reader["WaterDepth"].ToString(),
                Crop = reader["Crop"].ToString()
              };
              regionDataList.Add(data);
            }
```

```
        }
        conn.Close();
      }
    }

    return regionDataList;
  }

 }
}

public class RegionData {
  public string RegionName { get; set; }
  public string SoilType { get; set; }
  public string ClimateZone { get; set; }
  public string SoilAcidity { get; set; }
  public string WaterDepth { get; set; }
  public string Crop { get; set; }
}
```

**Лістинг 4. Код класу «LIMBOClusterer»**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LIMBOClustererApp.AppCode {
  internal class LIMBOClusterer {
    private List<RegionData> _data;

    public LIMBOClusterer(List<RegionData> data) {
      _data = data;
    }

    private double CalculateEntropy(List<RegionData> data) {
      var total = data.Count;
      var cropGroups = data.GroupBy(x => x.Crop);
      double entropy = 0;
      foreach (var group in cropGroups) {
        double p = (double)group.Count() / total;
        entropy -= p * Math.Log(p, 2);
      }
      return entropy;
    }

    private double CalculateInformationGain(List<RegionData> data,
      Func<RegionData, string> selector) {
      double entropyBefore = CalculateEntropy(data);
      int total = data.Count;
      var partitions = data.GroupBy(selector);
      double weightedEntropy = 0;
```

```
    foreach (var partition in partitions) {
      double p = (double)partition.Count() / total;
      weightedEntropy += p * CalculateEntropy(partition.ToList());
    }
    return entropyBefore - weightedEntropy;
  }

  public Dictionary<string, List<RegionData>> ClusterByRegion() {
    var gains = new Dictionary<string, double> {
      { "SoilType", CalculateInformationGain(_data, r => r.SoilType) },
      { "ClimateZone", CalculateInformationGain(_data, r => r.ClimateZone) },
      { "SoilAcidity", CalculateInformationGain(_data, r => r.SoilAcidity) },
      { "WaterDepth", CalculateInformationGain(_data, r => r.WaterDepth) }
    };
    foreach (var g in gains)
      Console.WriteLine($"Gain({g.Key}) = {g.Value:F4}");
    var bestAttr = gains.Aggregate((a, b) => a.Value > b.Value ? a : b).Key;
    Func<RegionData, string> selector;
    if (bestAttr == "SoilType")
      selector = r => r.SoilType;
    else if (bestAttr == "ClimateZone")
      selector = r => r.ClimateZone;
    else if (bestAttr == "SoilAcidity")
      selector = r => r.SoilAcidity;
    else if (bestAttr == "WaterDepth")
      selector = r => r.WaterDepth;
    else
      selector = r => r.SoilType;

    //return _data.GroupBy(selector)
    //            .ToDictionary(g => g.Key, g => g.ToList());

    return _data.GroupBy(r => r.RegionName)
    .ToDictionary(g => g.Key, g => g.ToList());
  }

  public Dictionary<string, List<RegionData>> ClusterByAttribute(string attributeName) {
    Dictionary<string, List<RegionData>> result = new Dictionary<string,
List<RegionData>>();
    switch (attributeName) {
      case "Region":
        result = _data.GroupBy(d => d.RegionName).ToDictionary(g => g.Key, g =>
g.ToList());
        break;
      case "SoilType":
        result = _data.GroupBy(d => d.SoilType).ToDictionary(g => g.Key, g => g.ToList());
        break;
      case "ClimateZone":
        result = _data.GroupBy(d => d.ClimateZone).ToDictionary(g => g.Key, g =>
g.ToList());
        break;
      case "SoilAcidity":
        result = _data.GroupBy(d => d.SoilAcidity).ToDictionary(g => g.Key, g => g.ToList());
```

```
            break;
        case "WaterDepth":
            result = _data.GroupBy(d => d.WaterDepth).ToDictionary(g => g.Key, g =>
g.ToList());
            break;
        default:
            throw new ArgumentException("Невідомий атрибут кластеризації: " +
attributeName);
      }

      return result;
    }
  }
}
```