Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

Кваліфікаційна робота бакалавра

з теми: **«Проєктування та розробка вебзастосунку з функціональністю CRUD для ефективного керування базами даних на основі об’єктно-реляційного підходу»**

Виконав: здобувач вищої освіти групи KN1-B21
спеціальності 122 Комп’ютерні науки
Лискун Ростислав Володимирович

Керівник: Смалько Олена Аркадіївна,
кандидат педагогічних наук,
доцент кафедри комп’ютерних наук

Рецензент: Оптасюк Сергій Васильович,
кандидат фізико-математичних наук,
доцент, завідувач кафедри фізики

Кам’янець-Подільський – 2025 р.

АНОТАЦІЯ

Кваліфікаційна робота присвячена розробці вебзастосунку для обліку, управління та аналізу агровиробничих даних із реалізацією базової функціональності CRUD (створення, читання, оновлення, видалення). Такий інструмент спрямований на цифровізацію процесів збору та обробки інформації у малих та середніх фермерських господарствах, що сприяє ефективному управлінню аграрними ресурсами. Вебзастосунок розроблений із використанням мікрофреймворку Flask для побудови серверної логіки та MySQL як системи управління базами даних.

Окрім реалізації базових операцій взаємодії з агроданими, система доповнена підсистемою контролю витрат на добрива, що дозволяє проводити автоматизований розрахунок фінансових витрат на основі введених параметрів. Ще одним важливим компонентом є можливість генерації звітів у форматі PDF, що дає змогу зберігати результати у вигляді формалізованої документації, зручною для друку або подальшого аналізу.

Результати роботи свідчать про практичну доцільність застосування відкритого технологічного стеку для розробки агроінформаційних систем. Розроблений вебзастосунок може бути впроваджений у діяльність фермерських господарств для автоматизації агровиробничих процесів, а також використаний як дидактичний матеріал у межах освітніх програм, що охоплюють вебпрограмування, обробку даних і розробку CRUD-систем.

Ключові слова: вебзастосунок, агровиробничі дані, база даних, CRUD, Flask, автоматизація.

ABSTRACT

The qualification thesis is devoted to the development of a web application for the management, registration, and analysis of agri-production data with the implementation of CRUD functionality (Create, Read, Update, Delete). This tool aims to support the digital transformation of data management processes in small and medium-sized farms, contributing to more efficient agricultural resource control. The application is developed using the Flask microframework for server-side logic and MySQL as the database management system.

In addition to standard data operations, the system features a fertilizer cost control module that automates expense calculations based on input parameters. Another significant functionality is PDF report generation, allowing users to archive key results in a formalized format suitable for printing and further evaluation.

The results confirm the practical viability of using an open-source technology stack in building agricultural information systems. The developed application may be integrated into farm operations for process automation or used as educational material in courses related to web development, data processing, and CRUD system design.

Keywords: web application, agri-production data, database, CRUD, Flask, automation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ і СКОРОЧЕНЬ

CRUD – Create, Read, Update, Delete (Створення, Читання, Оновлення, Видалення) – базові операції з даними.

ORM – Object-Relational Mapping (Об'єктно-реляційне відображення) – технологія програмування для роботи з базами даних.

DB – Database (База даних) – структурований набір даних.

SQL – Structured Query Language (Мова структурованих запитів) – мова для роботи з реляційними базами даних.

ВСТУП

У сучасних умовах цифрової трансформації суспільства все більшої актуальності набувають інформаційні технології, які забезпечують збирання, зберігання, обробку та візуалізацію даних. Особливу роль у цьому процесі відіграють вебзастосунки — динамічні програмні рішення, що функціонують у мережевому середовищі та забезпечують інтерактивну взаємодію користувача з даними. Підвищення попиту на цифрові платформи для автоматизації управлінських процесів, зокрема у галузях, які традиційно не пов’язуються з інтенсивною цифровізацією, як-от сільське господарство, обумовлює необхідність створення інструментів, здатних адаптуватися до специфічних потреб користувача та забезпечувати гнучке управління інформаційними ресурсами.

У центрі проєктування більшості веборієнтованих систем лежить концепція CRUD, яка передбачає чотири базові операції з даними — створення, читання, оновлення та видалення. Саме ця функціональність є основою будь-якої системи керування базами даних, незалежно від складності предметної області. Застосування CRUD-парадигми дозволяє забезпечити логічну цілісність даних, зручність взаємодії та можливість масштабування інформаційної системи відповідно до потреб користувача.

Реалізація вказаної парадигми у вебзастосунках потребує не лише належного проєктування логіки взаємодії з базою даних, але й вибору ефективного способу зв’язку між об’єктною моделлю застосунку та реляційною структурою бази. У цьому контексті об’єктно-реляційне відображення (ORM) постає як один із найефективніших підходів. ORM забезпечує автоматичне узгодження між об’єктами у програмному коді та таблицями в реляційній базі даних, що не лише спрощує розробку та зменшує обсяг низькорівневого коду, але й підвищує супровідність і безпеку програмного забезпечення.

Метою кваліфікаційної роботи є створення веборієнтованого застосунку з реалізацією функціональності CRUD, спрямованого на управління агровиробничими даними з використанням об'єктно-реляційного підходу. Реалізація такого підходу дозволяє ефективно структурувати облік аграрних ресурсів, раціоналізувати процеси застосування добрив та культур, а також забезпечити доступну і наочну візуалізацію інформації для подальшого аналізу. Наукова новизна дослідження полягає в інтеграції сучасних вебтехнологій у контексті цифровізації агропромислового виробництва.

Задля досягнення визначеної мети в роботі виконувалися наступні **завдання**:

- аналіз актуальних тенденцій цифрової трансформації в аграрній галузі з акцентом на обробку агровиробничих даних;[6]
- проєктування архітектури вебзастосунку відповідно до вимог модульності, масштабованості та адаптивності;
- реалізація CRUD-функціональності з безпосередньою взаємодією із реляційною базою даних MySQL та з використанням бібліотеки PyMySQL;
- розробка підсистеми автоматизованого обліку витрат на добрива на основі агрономічних параметрів;
- впровадження механізму генерації зведених звітів у форматі PDF з подальшим збереженням інформації у структурованому вигляді;
- тестування працездатності основних компонентів системи та валідацію даних на всіх етапах обробки.

Об'єктом дослідження є процеси організації, проєктування та реалізації веборієнтованих інформаційних систем, що забезпечують взаємодію з реляційними базами даних у режимі реального часу.

Предметом дослідження виступають технології реалізації CRUD-функціональності у вебзастосунках із використанням ORM як інструменту синхронізації між логічною (об’єктною) та фізичною (реляційною) моделями даних.

Під час виконання дослідження було використано низку наукових методів. Теоретичну базу формували методи системного аналізу, які дозволили встановити структурні залежності між елементами вебсистеми та сформулювати технічні вимоги до неї. Методи моделювання застосовувалися для побудови логічної структури бази даних і взаємодії компонентів. Практичний етап реалізації спирався на методи об’єктно-орієнтованого програмування, зокрема при розробці серверної частини та взаємодії з ORM-бібліотекою. Прототипування стало основою для побудови інтерфейсу користувача, тоді як методи тестування були використані для перевірки коректності функціонування вебзастосунку в цілому.

Структура кваліфікаційної роботи побудована відповідно до логіки реалізації поставленої мети та охоплює три основні розділи. У першому розділі проаналізовано теоретичні основи побудови вебзастосунків, особливості реалізації CRUD-функціоналу та ключові підходи до побудови архітектури клієнт-серверних систем. У другому розділі визначено переваги використання об’єктно-реляційного підходу в процесі управління даними, а також обґрунтовано вибір технологій та інструментальних засобів серед популярних ORM-рішень. У третьому розділі представлено практичну реалізацію вебзастосунку, включно з описом архітектури системи, логіки взаємодії з базою даних, реалізації CRUD-функціональності та інтерфейсного компонента.

Список використаних джерел містить 35 позицій. Додатковий ілюстративний матеріал розміщено у додатках.

РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКІВ ТА УПРАВЛІННЯ БАЗАМИ ДАНИХ

1.1. Основи архітектури вебзастосунків

У сучасному інформаційному просторі вебзастосунки стали невіддільною частиною цифрової інфраструктури, забезпечуючи доступ користувачів до розподілених ресурсів, систем керування, сервісів аналізу даних та інтелектуальної обробки інформації. Проєктування архітектури таких застосунків вимагає врахування численних технічних, функціональних та експлуатаційних вимог. Основою побудови ефективного вебзастосунку є вибір оптимальної архітектурної моделі, яка відповідатиме характеристикам масштабованості, гнучкості, надійності та продуктивності.

Серед найбільш поширених архітектурних підходів у веброзробці можна виділити монолітну, клієнт-серверну та мікросервісну архітектури. Кожна з них має свої особливості, переваги й обмеження.

Монолітна архітектура передбачає створення програмного продукту як єдиного нероздільного модуля, в якому всі компоненти — бізнес-логіка, обробка запитів, інтерфейс користувача та доступ до бази даних — об'єднані в одну систему.[12] Такий підхід є традиційним і застосовується в системах, де не очікується частих змін, високе навантаження чи потреба в масштабуванні. Основною перевагою моноліту є простота реалізації та розгортання, що особливо важливо на початковому етапі. Проте із зростанням складності системи виникають труднощі з підтримкою, модифікацією й адаптацією до нових вимог, адже будь-яке оновлення одного з компонентів потребує перевірки та, зазвичай, повторного деплойменту всієї системи.

Клієнт-серверна архітектура стала класичним варіантом побудови вебзастосунків, в якій функціональність чітко розділена між клієнтською та серверною частинами. Клієнт відповідає за відображення даних, обробку введення користувача, а сервер — за логіку взаємодії з базою даних, перевірку

бізнес-правил та авторизацію. Такий підхід дозволяє централізовано управляти даними, забезпечує безпеку й спрощує адміністрування, адже всі критичні дані зберігаються на стороні сервера. Клієнт, як правило, реалізується за допомогою HTML/CSS/JavaScript, а сервер — з використанням Python (наприклад, фреймворк Flask), Java, PHP тощо.[8]

Мікросервісна архітектура є сучасним і надзвичайно популярним підходом до розробки великих, складних систем. Вона передбачає побудову системи з низки незалежних сервісів, кожен з яких відповідає за виконання окремої бізнес-функції. Ці сервіси взаємодіють між собою через стандартизовані інтерфейси — зазвичай REST API або повідомлення черг. Основна перевага мікросервісного підходу полягає в гнучкості: кожен сервіс може бути розгорнутий, масштабований або оновлений незалежно.[19] Водночас така архітектура потребує значно вищого рівня організації проєкту, логістики та координації компонентів, у тому числі управління розподіленими транзакціями та забезпечення консистентності даних.

Серед різноманіття архітектурних моделей особливе місце займає трирівнева архітектура (three-tier architecture), яка зарекомендувала себе як ефективне рішення для побудови масштабованих і підтримуваних вебзастосунків. Вона складається з трьох логічно відокремлених рівнів: презентаційного (Presentation Layer), рівня прикладної логіки (Application Layer) та рівня доступу до даних (Data Layer).

Презентаційний рівень відповідає за взаємодію з користувачем. Саме тут розміщено інтерфейс, розроблений із використанням HTML, CSS і JavaScript. Він отримує введення користувача та відображає результати обробки. Прикладний рівень реалізує бізнес-логіку системи, опрацьовує дані, перевіряє їхню коректність, керує роботою з базами даних. Нарешті, рівень доступу до даних забезпечує безпосередню взаємодію із СУБД, відповідає за збереження, пошук, оновлення та видалення даних.

Розділення на рівні дозволяє реалізувати принцип «розмежування відповідальності», що спрощує тестування, полегшує рефакторинг та сприяє

повторному використанню коду. Крім того, така структура надає змогу залучати до розробки фахівців різних профілів — фронтенд, бекенд і спеціалістів з баз даних — без потреби втручання в суміжні рівні.

1.2. CRUD-операції як фундамент функціональності вебсистем

У сучасних вебзастосунках реалізація логіки взаємодії з даними посідає центральне місце, адже саме дані становлять основну цінність будь-якої інформаційної системи. Незалежно від сфери застосування — управління контентом, електронна комерція, електронне урядування чи аграрний сектор — користувач взаємодіє із системою шляхом створення, перегляду, зміни та видалення інформації. Усе це охоплюється поняттям CRUD, яке є скороченням від англійських слів Create (створення), Read (читання), Update (оновлення) та Delete (видалення).

Парадигма CRUD є універсальним підходом до побудови інтерфейсів користувача, серверної логіки та бази даних, яка дозволяє ефективно оперувати інформацією.[3] Вона є не лише технічним, а й концептуальним фундаментом веборієнтованих систем. Поняття CRUD виникло в контексті реляційних баз даних, однак згодом стало міждисциплінарним та перенесеним на інші рівні архітектури програмного забезпечення, включаючи API, об’єктно-орієнтовану логіку, а також інтерфейси користувача.

З технічної точки зору, CRUD-операції прямо корелюють з відповідними методами протоколу HTTP: POST відповідає за створення нових об’єктів, GET — за читання або отримання інформації, PUT та PATCH — за оновлення, а DELETE — за видалення записів. У такий спосіб забезпечується повна функціональна взаємодія між клієнтом і сервером у межах RESTful-архітектури.[14] Кожна з CRUD-операцій відображається на реалізації певного маршруту в серверному фреймворку (наприклад, Flask у Python), що обробляє запит, виконує логіку й повертає відповідь клієнтові.

У розрізі розробки інформаційних систем, які працюють із базами даних, реалізація CRUD є базовою вимогою до будь-якої функціональної одиниці

застосунку. Наприклад, для модуля управління аграрними полями операція створення (Create) дозволяє внести в базу нове поле із зазначенням площі, типу ґрунту та розміщення. Операція читання (Read) — переглянути всі наявні поля або окреме поле, зокрема з метою аналізу. Оновлення (Update) дозволяє змінити параметри поля, зокрема культуру або добрива, які застосовуються. Видалення (Delete), своєю чергою, необхідне для очищення системи від неактуальної або помилкової інформації.

Важливо зауважити, що реалізація CRUD-функціональності не обмежується лише операціями на рівні бази даних. У сучасних вебсистемах вона також стосується рівня бізнес-логіки та валідації даних. Наприклад, перед створенням нового запису необхідно здійснити перевірку відповідності введених значень обмеженням цілісності: наявність обов’язкових полів, унікальність ідентифікаторів, допустимість числових діапазонів тощо. Саме ці аспекти формують частину логіки, яка обслуговує CRUD-процеси, забезпечуючи коректність і надійність функціонування системи загалом.

Не менш важливою є роль ORM (Object-Relational Mapping) у реалізації CRUD-функцій. ORM дозволяє розробникам оперувати даними через об’єкти програмної мови, а не безпосередньо SQL-запити. Це спрощує написання коду, знижує кількість помилок, підвищує продуктивність розробки та забезпечує кращу підтримку проєкту. У середовищі Python для цих цілей часто використовується бібліотека SQLAlchemy, яка надає широкі можливості роботи з базами даних у парадигмі об’єктного програмування.[25]

Наприклад, щоб створити новий запис у таблиці Field, розробник створює об’єкт класу Field і передає його до сесії бази даних, після чого викликає метод commit(). Для читання використовується метод query, який повертає список або окремий об’єкт за вказаними критеріями. Для оновлення даних достатньо змінити значення атрибутів об’єкта, а для видалення — викликати метод delete. Таким чином, ORM фактично інкапсулює в собі реалізацію всіх чотирьох CRUD-операцій у зручній, об’єктно-орієнтованій формі.

У контексті аграрних інформаційних систем, як розглядається у цій роботі, CRUD-функціональність виступає не лише технічним, але й прикладним інструментом. Завдяки можливості вести облік полів, культур, добрив і їхніх змін, фермер отримує зручний інтерфейс для керування аграрною діяльністю, аналізу продуктивності й планування майбутніх сільськогосподарських робіт.Впровадження ефективної CRUD-логіки в аграрному вебзастосунку дає змогу формувати структуровану базу знань про ресурси господарства, зберігати історію використання полів та здійснювати коректну агрономічну документацію.

Окрему увагу слід приділити питанням безпеки при реалізації CRUD-функцій. У випадку помилкової реалізації існує ризик втрати або розголошення конфіденційних даних. Наприклад, невірно сконфігуровані операції Delete можуть призвести до втрати важливої інформації. Також неконтрольований доступ до функцій Create або Update — потенційна вразливість, яка може бути використана зловмисниками. Для запобігання цим загрозам слід реалізовувати авторизацію, аутентифікацію, перевірку прав доступу та аудит змін.

Важливо зазначити, що ефективна реалізація CRUD не обмежується лише коректним функціонуванням з технічної точки зору. Вона також передбачає зручність використання для кінцевого користувача, що реалізується через зрозумілий інтерфейс, коректні повідомлення про помилки, валідацію даних у режимі реального часу, підтримку багатомовності тощо. Саме такий підхід дозволяє інтегрувати CRUD-функціональність у більший контекст взаємодії «людина–машина» в цифровому середовищі.

Висновки до розділу 1

У результаті аналізу теоретичних основ проєктування вебзастосунків та організації управління базами даних зроблено низку концептуально важливих узагальнень, які визначають архітектурно-функціональні засади створення сучасних інформаційних систем.

По-перше, встановлено, що вибір архітектурної моделі вебзастосунку безпосередньо впливає на його масштабованість, підтримуваність та адаптивність до змінних умов експлуатації. У цьому контексті клієнт-серверна та трирівнева архітектури є найбільш релевантними для побудови систем середньої складності, зокрема у сфері аграрного управління. Вони дозволяють розмежувати логіку взаємодії з користувачем, бізнес-функціональність та доступ до даних, що сприяє гнучкому управлінню програмною інфраструктурою.

По-друге, показано, що CRUD-парадигма (Create, Read, Update, Delete) є фундаментальним підходом до моделювання життєвого циклу даних у вебзастосунках. Саме вона забезпечує базову, але універсальну логіку обробки інформації, яка застосовується незалежно від предметної області та типу даних. Реалізація CRUD-операцій на рівні серверної логіки, в сукупності з використанням ORM-рішень, дозволяє значно підвищити ефективність розробки, забезпечити надійність обробки інформації та уніфікувати інтерфейси взаємодії з базами даних.

По-третє, обґрунтовано, що інтеграція CRUD-функціональності в структуру аграрного вебзастосунку створює можливості для побудови інформаційної системи з глибоким рівнем адаптації до потреб кінцевого користувача — фермера або агронома. Така система забезпечує структурований облік земельних ресурсів, використаних добрив, культур, а також дає змогу формувати підґрунтя для подальшої аналітики та автоматизації процесів прийняття рішень.

РОЗДІЛ 2
ОБ'ЄКТНО-РЕЛЯЦІЙНИЙ ПІДХІД В УПРАВЛІННІ БАЗАМИ ДАНИХ

2.1. Переваги об'єктно-реляційного підходу

Упродовж останніх десятиліть інтенсивний розвиток технологій управління даними зумовив появу нових підходів до проєктування й реалізації програмних систем, що функціонують у середовищі баз даних. Одним із таких підходів, який отримав широке застосування у практиці розробки інформаційних систем, є об'єктно-реляційне відображення (Object-Relational Mapping, ORM). Цей підхід поєднує в собі концепції реляційної моделі даних та об'єктно-орієнтованого програмування, виступаючи посередником між програмною логікою та фізичною структурою бази даних.

Під поняттям об'єктно-реляційного відображення розуміють процес автоматичного трансформування даних, що зберігаються у реляційній базі даних у вигляді таблиць, у об'єкти, які використовуються у програмному коді. Іншими словами, ORM слугує мостом між об'єктами у програмному середовищі (класи, екземпляри, методи) та реляційною структурою бази даних (таблиці, поля, ключі).[22, 7] Цей механізм дозволяє розробнику оперувати не SQL-запитами, а повноцінними об'єктами мови програмування, що значно підвищує продуктивність розробки та полегшує підтримку коду.

В основі ORM лежить концепція того, що кожна таблиця в базі даних відповідає певному класу в об'єктно-орієнтованій моделі, кожен рядок таблиці — окремому екземпляру цього класу, а кожен стовпчик — атрибуту (властивості) об'єкта. Таким чином, наприклад, таблиця Fields у реляційній базі даних може бути відображена на клас Field у програмному середовищі, що дозволяє працювати з даними за допомогою методів create(), update(), delete() тощо, без прямої взаємодії з SQL.

Переваги використання ORM полягають не лише у зручності синтаксису, але й у забезпеченні абстракції, яка дозволяє створювати додатки, що менш залежні від конкретної СУБД. Завдяки ORM розробник отримує уніфікований

інтерфейс для доступу до даних, незалежно від того, чи використовується PostgreSQL, MySQL чи SQLite.[16] Крім того, ORM зазвичай містить вбудовані механізми валідації, управління транзакціями, кешування та оптимізації запитів, що дозволяє уникнути типових помилок при роботі з базами даних.

Слід зауважити, що об’єктно-реляційний підхід не позбавлений певних труднощів. Найбільш відомою з них є так звана «прірва об’єктно-реляційного розриву» (object-relational impedance mismatch) — концептуальна невідповідність між об’єктною та реляційною моделями. Наприклад, у реляційній моделі відсутні механізми успадкування чи поліморфізму, які є основоположними в ООП. Також виникають складнощі з відображенням складних зв’язків (один-до-багатьох, багато-до-багатьох), вкладених структур або колекцій об’єктів. Різниця у підходах до ідентифікації об’єктів (первинні ключі в реляційних БД проти посилань або вказівників у ООП) також є джерелом потенційних труднощів.

Попри зазначені виклики, ORM залишається домінуючим підходом у сучасній розробці вебзастосунків, оскільки дозволяє значно пришвидшити процес розробки, зменшити кількість шаблонного коду та зосередитися на реалізації бізнес-логіки.[8] До найвідоміших реалізацій ORM належать SQLAlchemy (Python), Hibernate (Java), Entity Framework (.NET), Sequelize (Node.js), Eloquent (Laravel) тощо. Кожне з цих рішень реалізує базові принципи ORM, водночас маючи свої особливості у підходах до роботи з транзакціями, кешуванням, автоматичною генерацією міграцій та синхронізацією схем БД.

Варто порівняти об’єктно-реляційний підхід із класичним реляційним, при якому розробник безпосередньо працює з SQL-запитами. Такий підхід забезпечує більший рівень контролю над виконанням операцій, дозволяє оптимізувати запити на рівні БД та є незамінним у системах, що працюють із великими обсягами даних або потребують точного налаштування продуктивності. Проте ручне написання SQL потребує глибоких знань

відповідної мови, вимагає більше часу на розробку й тестування та може спричиняти помилки, зокрема при некоректному використанні змінних, що робить систему вразливою до SQL-ін’єкцій.

У свою чергу, ORM, зазвичай, автоматично екранує введення користувача, що значно підвищує безпеку системи. Крім того, використання ORM сприяє кращій структуризації проєкту, спрощує процес міграції схем БД, дозволяє швидко вносити зміни у моделі без ризику порушити залежності між таблицями, а також полегшує роботу команді розробників.

Загалом, об’єктно-реляційний підхід не є універсальним рішенням для будь-якої ситуації, однак його застосування є доцільним у більшості сучасних вебзастосунків, особливо в контексті реалізації систем, орієнтованих на CRUD-операції. У межах даного дослідження ORM виступає ключовою технологією, що забезпечує зручний, гнучкий і безпечний механізм обробки аграрних даних через вебінтерфейс.

2.2. Обґрунтування вибору інструментів для реалізації ORM

У контексті реалізації вебзастосунків з повноцінною підтримкою CRUD-функціональності ключову роль відіграє вибір відповідного інструментарію для роботи з базою даних. Серед найбільш ефективних технологій, які забезпечують взаємодію між об’єктною моделлю застосунку та реляційною структурою бази даних, вирізняються ORM-бібліотеки. Їх застосування дозволяє суттєво спростити розробку, підвищити безпеку коду та забезпечити узгоджене управління схемами даних у процесі розвитку проєкту.

На сьогоднішній день існує значна кількість ORM-рішень, кожне з яких має свою сферу застосування, переваги та недоліки. Серед найбільш поширених бібліотек, які використовуються у професійному програмуванні, варто виокремити Hibernate (Java), Django ORM (Python), SQLAlchemy (Python), Sequelize (Node.js/JavaScript), Entity Framework (C#/.NET), Eloquent ORM (Laravel/PHP) тощо. У межах цього підпункту буде проведено короткий

огляд зазначених рішень та здійснено обґрунтований вибір бібліотеки, яка найкраще відповідає цілям і завданням дипломного проєкту.

Однією з найстаріших і водночас найпотужніших ORM-бібліотек є Hibernate, розроблена для використання у середовищі Java. Вона є повнофункціональною системою об’єктно-реляційного відображення, яка підтримує мапінг складних об’єктних структур, трансакційність, кешування, автоматичне створення схем БД, lazy loading, а також власну мову запитів HQL (Hibernate Query Language), що дозволяє писати запити в об’єктно-орієнтованому стилі. Hibernate застосовується у великомасштабних проєктах, де необхідна висока продуктивність, гнучкість та підтримка складних бізнес-правил. Основним недоліком цієї бібліотеки є складність налаштування, досить крута крива навчання та надлишковість у випадках невеликих проєктів.

У середовищі Python найбільш відомим ORM-рішенням є Django ORM, яке є невід’ємною частиною однойменного фреймворку Django.[33] Цей ORM є зручним у використанні, надає розробникам високий рівень абстракції, дозволяє автоматизувати міграції баз даних, створювати форми на основі моделей, а також генерує API запитів у вигляді ланцюгів методів (chaining). Django ORM ідеально підходить для швидкої розробки прототипів і повноцінних систем, але його недоліком є жорстка інтеграція з фреймворком, що обмежує гнучкість при нестандартних сценаріях використання.

Альтернативою в середовищі Python є SQLAlchemy, яка відома як найбільш гнучке ORM-рішення На відміну від Django ORM, SQLAlchemy надає можливість вибору між декларативним та імперативним стилями опису моделей, а також повноцінну підтримку raw SQL-запитів. Вона дозволяє налаштовувати зв’язки між таблицями з високою точністю, підтримує складну логіку запитів і є добре задокументованою. Основною перевагою SQLAlchemy є її гнучкість, що, однак, водночас робить її складнішою для початкового засвоєння, особливо для початківців.

Ще одним поширеним рішенням у Python-середовищі є Flask — мікрофреймворк, який часто обирають для побудови легких, гнучких і

адаптивних вебзастосунків. Хоча Flask сам по собі не включає вбудованого ORM, він забезпечує можливість легкої інтеграції з будь-якими бібліотеками для взаємодії з базами даних, зокрема SQLAlchemy або pymysql. Його перевагами є мінімалістичний підхід до структури проєкту, гнучкість у виборі компонентів, чітке розділення логіки та представлення, а також активна спільнота розробників. Flask особливо доцільний у випадках, коли потрібен високий рівень контролю над архітектурою застосунку або коли система має бути розроблена без жорсткої залежності від конкретного ORM.[1, 2, 3] Саме тому він часто використовується у навчальних, дослідницьких і дипломних проєктах.

У середовищі JavaScript (Node.js) найбільш поширеним ORM-рішенням є Sequelize. Ця бібліотека реалізує більшість стандартних ORM-операцій, зокрема створення моделей, зв’язки між таблицями, транзакції, валідацію, а також підтримує кілька СУБД (PostgreSQL, MySQL, MariaDB, SQLite тощо). Sequelize відзначається простотою налаштування, активною підтримкою спільноти, великою кількістю прикладів у документації, що робить її зручною для невеликих і середніх проєктів. Водночас вона менш ефективна при реалізації складної логіки обробки даних порівняно з рішеннями, орієнтованими на strongly-typed мови програмування.

У технологічному стеку Microsoft популярною є бібліотека Entity Framework, яка забезпечує повну інтеграцію з платформою .NET. Вона дозволяє працювати з базами даних у режимі code-first або database-first, автоматично створювати міграції, а також генерує складні SQL-запити на основі LINQ (Language Integrated Query). Entity Framework має високу продуктивність у поєднанні з SQL Server, однак менш ефективна при використанні з іншими СУБД.

Для середовища PHP одним із найпопулярніших ORM-рішень є Eloquent ORM, що входить до складу фреймворку Laravel. Eloquent відзначається простотою синтаксису, підтримкою основних типів зв’язків між моделями, автогенерацією міграцій і зручною інтеграцією з іншими компонентами

Laravel. Цей інструмент широко використовується в комерційних вебзастосунках, однак має обмежену гнучкість у випадках складної бізнес-логіки або високих вимог до оптимізації запитів.

З метою обґрунтованого вибору ORM-інструменту для реалізації дипломного проєкту доцільно сформулювати перелік ключових критеріїв, яким має відповідати обрана технологія. Такими критеріями є підтримка CRUD-операцій, сумісність із вибраною мовою програмування та фреймворком, можливість використання з базами даних PostgreSQL або SQLite, наявність розвиненої документації й активної спільноти, підтримка міграцій, валідації даних, транзакцій, мінімальний поріг входу для розробника, можливість масштабування системи без переробки моделі даних.

Враховуючи зазначені критерії, а також архітектурні особливості дипломного проєкту, було обрано зв’язку бібліотеки pymysql.cursors та мікрофреймворку Flask, яка забезпечує прямий і контрольований доступ до бази даних MySQL на рівні курсорів. Такий підхід дозволяє гнучко управляти SQL-запитами, не покладаючись на додаткові рівні абстракції, притаманні ORM-рішенням. Використання pymysql.cursors забезпечує високий рівень прозорості обробки запитів, спрощує налагодження, а також дозволяє більш точно контролювати оптимізацію роботи з даними.[4] Крім того, дана бібліотека легко інтегрується з Flask, що забезпечує просту й зрозумілу реалізацію CRUD-функціональності, достатню для потреб дипломного проєкту.

Таким чином, проведений аналіз сучасних інструментів взаємодії з реляційними базами даних дав змогу не лише оцінити переваги і недоліки найпоширеніших ORM-рішень, а й обґрунтовано зупинити вибір на прямому підході з використанням pymysql.cursors. Обрана технологія забезпечує достатній рівень контрольованості, продуктивності та гнучкості для побудови вебзастосунку, орієнтованого на ефективне управління аграрними даними.

Висновки до розділу 2

У межах другого розділу було здійснено ґрунтовне теоретичне дослідження об’єктно-реляційного підходу до управління базами даних та проведено аналітичний огляд сучасних інструментів для його реалізації у вебзастосунках. Розглянуто фундаментальні поняття об’єктно-реляційного відображення (ORM), його принципи, переваги та концептуальні відмінності від класичного реляційного підходу, що базується на прямому написанні SQL-запитів.

Показано, що ORM-технології відіграють ключову роль у розробці вебсистем із підтримкою CRUD-функціональності, забезпечуючи прозору взаємодію між об’єктною моделлю застосунку та реляційною структурою бази даних. Водночас було акцентовано на проблемі об’єктно-реляційного розриву, що зумовлює певні труднощі при реалізації складних логічних зв’язків або обробці великих обсягів даних. Незважаючи на це, ORM залишається одним із найбільш ефективних інструментів побудови масштабованих і підтримуваних інформаційних систем.

На основі огляду сучасних ORM-бібліотек (Hibernate, Django ORM, SQLAlchemy, Sequelize, Eloquent, Entity Framework) проаналізовано їхні архітектурні особливості, функціональні можливості, ступінь інтеграції з фреймворками та відповідність вимогам CRUD-парадигми. В результаті порівняльного аналізу було сформульовано критерії вибору інструментів для реалізації дипломного проєкту.

Обґрунтовано доцільність використання бібліотеки pymysql.cursors у поєднанні з мікрофреймворком Flask як технологічного рішення, що забезпечує прямий контроль над SQL-запитами, високу гнучкість у реалізації бізнес-логіки, а також знижений поріг входу для реалізації невеликих, але функціонально насичених систем. Такий вибір відповідає структурі дипломного проєкту, який орієнтований на розробку зручного інструменту управління аграрними даними у форматі вебзастосунку.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ З ФУНКЦІОНАЛЬНІСТЮ CRUD

3.1. Проєктування та реалізація архітектури застосунку

Проєктування архітектури програмного забезпечення є фундаментальним етапом будь-якого повноцінного розробницького процесу, що передує безпосередній реалізації функціональності програмного продукту. У випадку створення вебзастосунку, призначеного для ефективного керування аграрними даними з використанням CRUD-підходу, архітектура має відповідати ключовим вимогам: підтримці модульності, забезпеченню гнучкості у впровадженні змін, простоті масштабування та можливості подальшої адаптації до нових задач.[3]

У процесі реалізації дипломного проєкту було прийнято рішення дотримуватися трирівневої клієнт-серверної архітектури, яка вважається класичною моделлю побудови сучасних вебсистем.[7] Така архітектура передбачає чіткий розподіл функціональних обов’язків між трьома основними рівнями: рівнем презентації (інтерфейс користувача), рівнем прикладної логіки (серверне обчислення та обробка запитів) та рівнем зберігання (база даних). Даний підхід забезпечує високу гнучкість, дозволяє легко впроваджувати нові компоненти без необхідності масштабної переробки існуючої системи та відповідає принципу розмежування відповідальності (Separation of Concerns), що позитивно впливає на якість програмного продукту.

Під час вибору технологічного стеку було враховано як академічні, так і практичні чинники, зокрема — відповідність мови програмування загальноприйнятим стандартам, доступність документації, рівень підтримки з боку спільноти розробників, можливість швидкого розгортання та інтеграції з іншими інструментами. В результаті було обрано стек, що включає Flask як фреймворк для побудови серверної частини (backend), HTML/CSS та

JavaScript для створення інтерфейсу користувача (frontend), MySQL як систему управління базами даних та бібліотеку pymysql.cursors для прямої роботи з БД без використання високорівневих ORM-рішень.

Flask є мікрофреймворком для мови програмування Python, що дозволяє швидко створювати вебзастосунки з мінімальною кількістю додаткових налаштувань.[1, 3] Його головними перевагами є легкість, гнучкість, відсутність жорстких залежностей та можливість поступової розбудови системи відповідно до потреб. Flask не нав’язує певної структури проєкту, що дозволяє розробнику самостійно формувати архітектуру та обирати оптимальні способи інтеграції з іншими компонентами системи. Зокрема, це створює сприятливі умови для інтеграції з бібліотекою pymysql, що є важливою особливістю обраної реалізації.

На рівні взаємодії з базою даних використано бібліотеку pymysql.cursors, яка надає прямий доступ до SQL-запитів з можливістю повного контролю над їхньою структурою, параметрами та обробкою результатів.[4] Такий підхід дає змогу уникнути додаткового рівня абстракції, властивого ORM-рішенням, що особливо важливо для систем із неуніфікованою бізнес-логікою або для тих випадків, коли необхідно оптимізувати запити вручну. pymysql також забезпечує захист від SQL-ін’єкцій шляхом підтримки параметризованих запитів і дозволяє реалізувати транзакційну обробку запитів за допомогою вбудованих механізмів керування курсорами.

Frontend-частина застосунку реалізована із використанням вебтехнологій HTML5, CSS3 та JavaScript (Vanilla). Такий вибір обумовлений прагненням зберегти архітектурну прозорість і забезпечити легкість навчання та подальшого аналізу роботи системи. HTML використовується для структуризації вмісту сторінки, CSS — для візуального оформлення, а JavaScript — для реалізації інтерактивної логіки, зокрема, асинхронної взаємодії із сервером за допомогою технологій Fetch API або XMLHttpRequest. Основна увага при розробці інтерфейсу приділялася функціональності, зручності навігації та адаптивності до різних типів пристроїв.

Вебінтерфейс дозволяє користувачеві виконувати повний спектр CRUD-операцій: створення нових записів про сільськогосподарські поля, перегляд даних у табличному форматі, редагування існуючих записів та їх видалення. Кожна операція викликає відповідний HTTP-запит (POST, GET, PUT, DELETE), що обробляється на сервері у межах відповідного маршруту (endpoint). Наприклад, створення нового запису реалізується через POST-запит до /api/fields, що ініціює обробку даних на сервері, їхню перевірку, збереження у базі даних та повернення відповіді про успішне завершення операції.

Структура бази даних відображає предметну область аграрного виробництва та реалізує логіку збереження агротехнічної інформації. Основною є таблиця fields, яка містить такі атрибути: id (унікальний ідентифікатор), region (назва області), area (площа поля), soil_type (тип ґрунту), crop (назва культури), fertilizer (вид добрива), created_at (дата створення запису). Передбачено можливість розширення бази через додавання зв’язків із таблицями regions, crops, fertilizers, які можуть містити додаткові характеристики відповідних сутностей. Така структура сприяє нормалізації даних, зменшенню надлишковості та забезпеченню цілісності інформації.

Реалізація запитів до бази даних здійснюється шляхом використання параметризованих SQL-операторів, які формуються у межах відповідних функцій серверної частини. Наприклад, вставка нового запису реалізується через SQL-запит де значення передаються як параметри з Python-коду. Це дозволяє зберігати дані у безпечний спосіб, знижуючи ризик SQL-ін’єкцій та логічних помилок.

У межах серверної логіки також реалізовано перевірку коректності введених користувачем даних. Зокрема, перед записом до бази даних виконується валідація: чи не є поле порожнім, чи відповідає формат числового значення площі, чи належить тип ґрунту до переліку допустимих тощо. У випадку виявлення помилки користувач отримує відповідне повідомлення у вебінтерфейсі.

Важливо зазначити, що архітектура системи спроєктована з урахуванням можливості її масштабування. У перспективі передбачено додавання таких функціональних модулів, як генерація агротехнічних звітів, інтеграція з картографічними сервісами (наприклад, Google Maps API або Leaflet.js), візуалізація динаміки використання полів, аналіз сівозміни, а також розширення до багатокористувацької моделі з авторизацією.

Таким чином, реалізована архітектура вебзастосунку відповідає сучасним вимогам до систем управління аграрними даними. Вона забезпечує ефективну обробку інформації, має модульну структуру, побудована із використанням перевірених технологій з відкритим кодом та є зручною для подальшої підтримки й розвитку.

3.2. Реалізація CRUD-функціоналу на основі ORM

У межах практичної реалізації вебзастосунку ключову роль відіграє реалізація CRUD-функціональності, що забезпечує повноцінну взаємодію користувача з базою даних. Як уже зазначалося у попередніх розділах, CRUD — це базова парадигма управління даними, яка передбачає реалізацію операцій створення (Create), читання (Read), оновлення (Update) та видалення (Delete). Саме ці операції формують функціональну основу інформаційних систем, дозволяючи реалізувати гнучкий цикл життєвого управління даними в межах предметної області.

Зважаючи на вибір технологічного стеку, реалізація CRUD-функцій здійснюється за допомогою Flask як серверного фреймворку, бібліотеки pymysql.cursors для взаємодії з MySQL та класичного підходу до побудови RESTful API. Хоча використання ORM у цьому контексті реалізовано частково (власноручне створення моделей через класові структури у Python), без використання високорівневих бібліотек на кшталт SQLAlchemy, структура коду наслідує концепції об’єктно-реляційного підходу: кожен запис у базі представлений об’єктом, а операції над ним — методами в логіці обробки запитів.

Реалізація базових CRUD-операцій

Створення даних (Create) реалізується через відповідні маршрути з методом POST. Наприклад, створення нового запису про аграрне поле відбувається через маршрут /fields/add, який приймає форму введення з клієнтської частини. Отримані дані перевіряються на коректність, після чого формується параметризований SQL-запит, що вставляє запис у таблицю fields.

Читання даних (Read) реалізовано через маршрути з методом GET, що дозволяють переглядати дані у вигляді таблиці або у форматі JSON. Наприклад, маршрут /fields повертає список усіх полів, тоді як /fields/<id> — детальну інформацію про конкретне поле. Ці дані використовуються для відображення у frontend-інтерфейсі, з можливістю їх редагування чи видалення.[18, 24]

Оновлення даних (Update) реалізується через маршрут /fields/update/<id> з методом POST або PUT. Користувач обирає запис для редагування, система підвантажує відповідну форму з поточними значеннями, після чого змінені дані записуються до бази. При цьому відбувається перевірка на правильність введення, а також автоматичне оновлення дати останньої модифікації, якщо така передбачена схемою.

Видалення даних (Delete) відбувається через маршрут /fields/delete/<id>, який викликається після підтвердження користувачем наміру видалити запис. Запит реалізується за допомогою SQL-команди DELETE, що забезпечує остаточне видалення інформації з таблиці.

3.3. Реалізація додаткового функціоналу вебзастосунку

Важливою складовою сучасних вебзастосунків, особливо в галузі аграрного управління, є наявність додаткових функцій, які виходять за межі базового CRUD-функціоналу. У межах реалізованого проєкту було впроваджено дві такі функціональні підсистеми: контроль витрат на добрива та можливість експорту аналітичних звітів у форматі PDF. Ці компоненти

істотно розширюють прикладну цінність системи, підвищуючи її користувацьку привабливість і функціональну повноту.

Підсистема контролю витрат на добрива орієнтована на розрахунок загальної вартості добрив, які використовуються на окремих аграрних ділянках. Вона базується на даних про площу конкретного поля, тип культури, рекомендовану норму внесення та вартість одиниці добрива. Система автоматично виконує відповідні розрахунки та формує узагальнений підсумок витрат, що дозволяє користувачеві оперативно оцінити економічну складову агротехнічних заходів. Такий функціонал особливо корисний для фермерських господарств і агропідприємств, де оптимізація витрат є критично важливою для забезпечення прибутковості.

Алгоритм розрахунку передбачає введення базових параметрів через відповідні веб-форми, після чого застосунок обробляє ці дані на серверній частині й виводить результат у зручному для сприйняття форматі. За потреби підсумкові дані можуть бути включені до загального звіту, який експортується у форматі PDF. Саме функція експорту у PDF є другою важливою підсистемою, реалізованою у поточній версії вебзастосунку.

Генерація звітів у форматі PDF дозволяє систематизувати та заархівувати ключову інформацію про поля, культури, добрива та витрати. Такі документи можуть використовуватись для внутрішнього обліку, податкової звітності або для комунікації з агрономічними службами. Вони є важливим елементом цифрової трансформації сільського господарства, оскільки забезпечують збереження інформації у форматі, сумісному з офіційним документообігом.

Формат PDF обрано завдяки його широкій підтримці, стабільності відображення даних незалежно від пристрою, а також можливості інтеграції до електронних систем керування документообігом.[15] Звіт формується автоматично на основі введених користувачем даних та інформації з бази даних. Це дозволяє уникнути людських помилок, прискорити підготовку документів і забезпечити їх структурованість. Кожен звіт містить розділи з

узагальненими показниками, таблицями й обчисленнями, що відповідає вимогам сучасного аграрного бізнесу.

Запровадження зазначених додаткових функцій значно підвищує цінність вебзастосунку як комплексного інформаційного рішення.[10, 11] Вони не лише автоматизують окремі бізнес-процеси, а й створюють передумови для впровадження глибшої аналітики, прогнозування та підтримки прийняття рішень у сільському господарстві. Таким чином, розробка та впровадження підсистем контролю витрат і звітності є важливим етапом цифровізації агросектору.

Висновки до розділу 3

У третьому розділі було детально розглянуто практичні аспекти реалізації вебзастосунку для управління аграрними базами даних із підтримкою CRUD-функціональності. Перш за все, було обґрунтовано вибір архітектури системи та технологічного стеку, зокрема використання мікрофреймворку Flask у поєднанні з бібліотекою PyMySQL для інтеграції з базою даних MySQL. Це забезпечило оптимальний баланс між простотою реалізації, гнучкістю налаштувань та високим рівнем контролю над логікою обробки даних.

У розділі також описано реалізацію основних CRUD-операцій, включаючи створення, перегляд, оновлення та видалення записів. Значну увагу приділено побудові маршрутів, форм для введення даних і механізмів взаємодії з базою даних. Такий підхід забезпечив функціональну повноту системи та її відповідність практичним потребам користувачів.

Окремий акцент зроблено на додаткових функціональних можливостях застосунку, зокрема контролі витрат на добрива та формуванні звітів у форматі PDF.[16] Реалізація цих підсистем дозволила підвищити практичну цінність вебзастосунку та адаптувати його до потреб реального аграрного господарства. Звіти, що генеруються системою, можуть використовуватися як інструмент обліку, аналізу або внутрішньої документації.[20]

Таким чином, результати, представлені у цьому розділі, демонструють високий рівень прикладної реалізації поставлених задач, функціональну завершеність розробленого рішення та перспективи його подальшого розвитку.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи здійснено повноцінний цикл дослідження, спрямованого на розробку веборієнтованого застосунку для управління агровиробничими даними. Запропоновано й реалізовано архітектурно-функціональну модель інформаційної системи, що базується на об'єктно-реляційному підході до структурування даних. Вебзастосунок підтримує повноцінний набір CRUD-операцій та забезпечує зручний інтерфейс користувача для ведення цифрового обліку аграрних ресурсів.

У процесі реалізації поставленої мети було проаналізовано поточні тенденції цифрової трансформації агросектору, визначено потреби щодо обліку агровиробничої інформації, а також спроєктовано та реалізовано низку ключових функціональних компонентів системи. Зокрема, створено інтерфейс для введення, редагування та видалення даних, розроблено підсистему автоматизованого обліку витрат на добрива та інтегровано механізм формування звітів у форматі PDF.

Виконане дослідження не лише підтвердило доцільність застосування сучасних технологій веброзробки у контексті аграрного управління, а й допомогло створити корисну платформу автоматизованого обліку витрат на добрива на основі агрономічних параметрів, здатну масштабуватись, розвиватись та відповідати потребам цифрового сільського господарства.

Практична цінність розробленого програмного забезпечення полягає у його здатності автоматизувати ключові процеси агровиробничого обліку, знизити навантаження на персонал, забезпечити оперативний доступ до актуальної інформації та мінімізувати ймовірність технічних або організаційних помилок при веденні записів. Розроблена система є корисною для малих і середніх фермерських господарств, яким часто бракує ресурсів для впровадження дорогих ERP-рішень. Завдяки відкритій архітектурі та використанню поширених технологій веброзробки, створене програмне забезпечення може бути легко адаптоване, масштабоване або інтегроване в

існуючі інформаційні середовища. Також можливе його використання в якості базової платформи для створення комплексних агроінформаційних систем із розширеним функціоналом — наприклад, модулем супутникового моніторингу, агрохімічного аналізу або з мобільним доступом. Окрім практичного застосування в агропромисловому виробництві, розроблений вебзастосунок є цінним навчальним ресурсом для студентів спеціальності «Комп’ютерні науки», оскільки демонструє принципи побудови інтерактивних клієнт-серверних систем, реалізації CRUD-функціональності, інтеграції з реляційними базами даних та створення звітності на основі введених даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Grinberg M. The Flask mega-tutorial (2024 Edition). *Miguelgrinberg.com*. December 3. 2023. URL: https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world (дата звернення: 24.05.2025).
2. Krebs B., Martinez J. C. Developing RESTful APIs with Python and Flask. *Auth0 Blog*. January 7, 2025. URL: https://auth0.com/blog/developing-restful-apis-with-python-and-flask (дата звернення: 24.05.2025).
3. Flask Tutorial. *GeeksforGeeks*. March 16, 2025. URL: https://www.geeksforgeeks.org/flask (дата звернення: 24.05.2025).
4. PyMySQL Documentation. *PyMySQL*. Inada Naoki and GitHub, 2023. URL: https://pymysql.readthedocs.io (дата звернення: 24.05.2025).
5. Liu Q., Wu J. Research on agricultural data processing based on MySQL. *Agricultural & Forestry Economics and Management (2024).* DOI: 10.23977/agrfem.2024.070203.
6. Cravero A., Pardo S., Sepúlveda S., Muñoz L. Challenges to use Machine Learning in Agricultural Big Data. *Agronomy. 2022*. DOI:10.3390/agronomy12030748.
7. Clark W. E. REST APIs Step by Step: A Practical Guide with examples. Walzone Press, 2025. ISBN 9798227737656.
8. Bazhanau R. Mastering SQLAlchemy: A comprehensive guide for Python developers. *Medium*. November 6, 2024. URL: https://medium.com/@ramanbazhanau/mastering-sqlalchemy-a-comprehensive-guide-for-python-developers-ddb3d9f2e829 (дата звернення: 24.05.2025).
9. Brown T. SQL Injection Prevention Techniques Cheat Sheet. OWASP Cheat Sheet Series. URL: https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html (дата звернення: 24.05.2025)
10. Habib A. Digital Transformation in Agriculture – Key Insights and Essential Guide for 2025. *Folio3 AgTech Blog*. November 15, 2024. URL:

https://agtech.folio3.com/blogs/digital-transformation-in-agriculture (дата звернення: 24.05.2025)

11. Digital Transformation in Agriculture: Innovation and the future. *Kaila.eu Blog*. December 5, 2024. URL: https://kaila.eu/blog/digital-transformation-in-agriculture-innovation-and-the-future (дата звернення: 24.05.2025)
12. Gard R. Quickly build secure microservices in Python. *DEV Community*. October 13, 2022. URL: https://dev.to/cowofevil/quickly-build-secure-microservices-in-python-1n44 (дата звернення: 24.05.2025)
13. Gray A. How will tech enable agrifood trends of 2024? *Purdue Agribusiness*, January 25, 2024. URL: https://agribusiness.purdue.edu/2024/01/25/how-will-tech-enable-agrifood-trends-of-2024 (дата звернення: 24.05.2025)
14. Savonin M. Agriculture technology trends for 2024 and beyond. *Keenethics Blog*. September 15, 2024. URL: https://keenethics.com/blog/agriculture-technology-trends-for-2024-and-beyond (дата звернення: 24.05.2025)
15. Revolutionizing agricultural extension: Digital technologies empower cooperative education and outreach. *Farmonaut Blog*. URL: https://farmonaut.com/blogs/digital-tech-transforming-agricultural-extension-in-2024 (дата звернення: 24.05.2025)
16. Oduor G., Macharia M., Rware H. I., Kaizzi K. C. Fertilizer use optimization approach: An innovation to increase agricultural profitability for African farmers. *African Journal of Agricultural Research*. 2016, 11(38):3587–3597. DOI: 10.5897/AJAR2016.11408.
17. Mărcuță C., MoldStud Team. Enhance the efficiency of your Python projects with Flask-security for effortless authentication solutions. *MoldStud*. February 15, 2025. URL: https://moldstud.com/articles/p-enhance-the-efficiency-of-your-python-projects-with-flask-security-for-effortless-authentication-solutions (дата звернення: 24.05.2025)
18. Tiwari S. Building Flask REST App with Flask-RESTful. *Medium*. May 30, 2025. URL: https://medium.com/analytics-vidhya/building-flask-rest-app-with-flask-restful-d22ba4c0e39f (дата звернення: 24.05.2025)

19. Walia A. S. How to optimize the performance of a flask application. *DigitalOcean Community*. September 13, 2024. URL: https://www.digitalocean.com/community/tutorials/how-to-optimize-flask-performance (дата звернення: 24.05.2025)
20. How to Optimize Your Flask Web App for performance. *MuneebDev*. September 25, 2024. URL: https://muneebdev.com/how-to-optimize-your-flask-web-app-for-performance (дата звернення: 24.05.2025)
21. Ghimire A. Machine Learning Model with Flask REST API. *HackerNoon*. January 9, 2020. URL: https://hackernoon.com/machine-learning-w22g322x (дата звернення: 24.05.2025)
22. How to Build a Machine Learning API with Python and Flask. *Statworx Blog*. July 29, 2020. URL: https://www.statworx.com/en/content-hub/blog/how-to-build-a-machine-learning-api-with-python-and-flask (дата звернення: 24.05.2025)
23. Martinho P., Franco T. How to build a Flask API with Python: The complete guide. *Imaginary Cloud Blog*. April 15, 2025. URL: https://www.imaginarycloud.com/blog/flask-python (дата звернення: 24.05.2025)
24. Galindo A. T. Flask API tutorial: Build, document, and secure a REST API. *Zuplo.com*. March 29, 2025. URL: https://zuplo.com/blog/2025/03/29/flask-api-tutorial (дата звернення: 24.05.2025)
25. Thai C. Performance optimization in flask: Tips and tricks for making flask applications faster and more scalable. *Medium.com*. April 2024. URL: https://medium.com/@christopherthai/performance-optimization-in-flask-tips-and-tricks-for-making-flask-applications-faster-and-more-07b9327277b3 (дата звернення: 24.05.2025)
26. Bhatt M. Building scalable web applications with python flask: Best practices. July 2024. URL: https://codymohit.com/building-scalable-web-applications-with-python-flask-best-practices (дата звернення: 24.05.2025)
27. Patel H. Scaling strategies for microservices: Best practices and implementation. *Medium*. March 24, 2025. URL: https://medium.com/

@hitendra.patel2986/scaling-strategies-for-microservices-best-practices-and-implementation-42093d835b25 (дата звернення: 24.05.2025)

28. Mahalias I. Microservices Python Development: 10 Best *Practices*. *Planeks* Blog. March 27, July 2024. URL: https://www.planeks.net/microservices-development-best-practices (дата звернення: 24.05.2025)
29. Yadav R. Complete guide to Python frameworks for scalable microservices. *Peerbits Blog*. October 09, 2024. URL: https://www.peerbits.com/blog/guide-to-python-frameworks-for-scalable-microservices.html (дата звернення: 24.05.2025)
30. Andersen G., MoldStud Research Team. Best practices for designing scalable microservices — A Guide for architects. *MoldStud Blog*. June, 2025. URL: https://moldstud.com/articles/p-best-practices-for-designing-scalable-microservices-a-guide-for-architects (дата звернення: 24.05.2025)
31. Dyouri A., MacDonald B. How to use Flask-SQLAlchemy to interact with databases in a Flask application. *DigitalOcean Community*. March 10, 2022. URL: https://www.digitalocean.com/community/tutorials/how-to-use-flask-sqlalchemy-to-interact-with-databases-in-a-flask-application (дата звернення: 24.05.2025)
32. Dyouri A. How To Create Your First Web Application Using Flask and Python 3. Tutorial. *DigitalOcean Community*. August 19, 2021. URL: https://www.digitalocean.com/community/tutorials/how-to-create-your-first-web-application-using-flask-and-python-3 (дата звернення: 24.05.2025).
33. Theofilou A., Nastis S. A., Tsagris M., Rodriguez-Perez S., Mattas K. Design and Implementation of a Scalable Data Warehouse for Agricultural Big Data. *Sustainability*. 2025, 17(8), article 3727. doi:10.3390/su17083727.
34. Wu S., Mueller T. A User-Friendly NoSQL Framework for Managing Agricultural Field Trial Data. *Scientific Reports*. 2024, 14, article 29819. doi:10.1038/s41598-024-81609-2.
35. Ngo V. M., Le-Khac N.-A., Kechadi M. An Efficient Data Warehouse for Crop Yield Prediction. *arXiv*. 2018. Proceedings of the 14th International Conference on Precision Agriculture, Montreal, 24–27 June 2018.

ДОДАТКИ

ДОДАТОК А

CRUD ОПЕРАЦІЇ, ВІЗУАЛІЗОВАНІ У ВЕБЗАСТОСУНКУ

Ілюстрації візуалізованих у вебзастосунку CRUD операцій (рис. А.1, А.2).

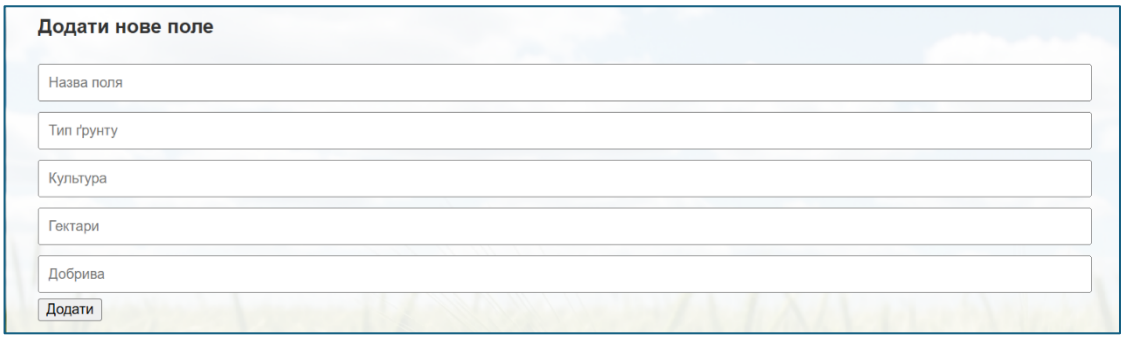

Рис. А.1 – Введення аграрних параметрів для розрахунку витрат на добрива

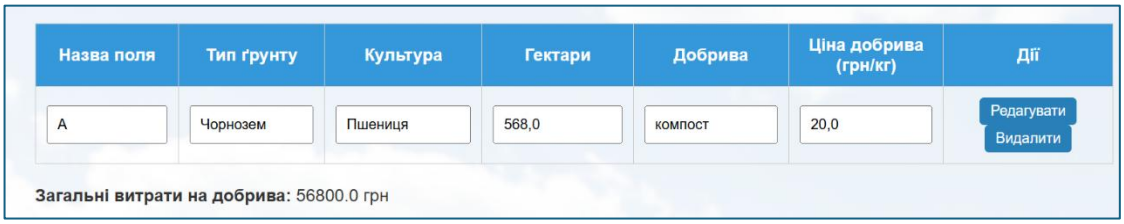

Рис. А.2 – Таблиця з агроданими і операції редагування та видалення

ДОДАТОК Б

МАРШРУТИ ОБРОБКИ CRUD-ОПЕРАЦІЙ У FLASK

Фрагменти коду обробки маршрутів для візуальної частини вебзастосунку (рис. Б.1–Б.3)

```python
@app.route('/edit_field/<int:field_id>', methods=['POST'])
def edit_field(field_id):
    field_name = request.form['field_name']
    soil_type = request.form['soil_type']
    crop = request.form['crop']
    hectares = float(request.form['hectares'])
    fertilizer = request.form['fertilizer']
    price_per_unit = float(request.form['fertilizer_price'])
    region_id = request.form['region_id']

    rate = 5
    fertilizer_cost = hectares * rate * price_per_unit

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        UPDATE fields SET field_name=%s, soil_type=%s, crop=%s, hectares=%s, fertilizer=%s, fertilizer_cost=%s
        WHERE id = %s
    """, (field_name, soil_type, crop, hectares, fertilizer, fertilizer_cost, field_id))
    conn.commit()
    cursor.close()
    conn.close()
    return redirect(url_for('region_fields', region_id=region_id))
```

Рис. Б.1 – Обробка маршруту створення нового запису (Create)

```python
@app.route('/delete_field/<int:field_id>', methods=['POST'])
def delete_field(field_id):
    region_id = request.form['region_id']
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM fields WHERE id = %s", (field_id,))
    conn.commit()
    cursor.close()
    conn.close()
    return redirect(url_for('region_fields', region_id=region_id))
```

Рис. Б.2 – Обробка маршруту редагування запису (Update)

```python
@app.route('/add_field/<int:region_id>', methods=['POST'])
def add_field(region_id):
    field_name = request.form['field_name']
    soil_type = request.form['soil_type']
    crop = request.form['crop']
    hectares = float(request.form['hectares'])
    fertilizer = request.form['fertilizer']

    rate = 5
    price_per_unit = 20
    fertilizer_cost = hectares * rate * price_per_unit

    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO fields (region_id, field_name, soil_type, crop, hectares, fertilizer, fertilizer_cost)
        VALUES (%s, %s, %s, %s, %s, %s, %s)
    """, (region_id, field_name, soil_type, crop, hectares, fertilizer, fertilizer_cost))
    conn.commit()
    cursor.close()
    conn.close()
    return redirect(url_for('region_fields', region_id=region_id))
```

Рис. Б.3 – Обробка маршруту видалення запису (Delete)

ДОДАТОК В

РОЗРАХУНОК ВАРТОСТІ ДОБРИВ І ПРИКЛАД ЗГЕНЕРОВАНОГО ЗВІТУ

На основі таблиці з агроданими (рис. В.1) у застосунку обчислюються важливі агротехнічні дані. Фрагмент коду з обчисленням вартості добрив наведено на рис. В.2.

Всі необхідні дані виводяться у pdf-форматі (рис. В.3).

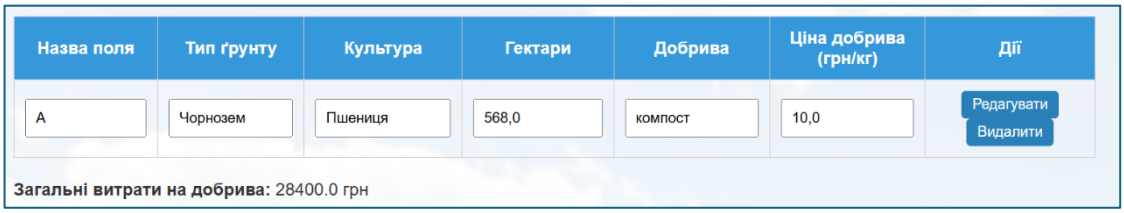

Рис. В.1 – Таблиця з агроданими та розрахунком загальної вартості добрив

```
rate = 5
price_per_unit = 20
fertilizer_cost = hectares * rate * price_per_unit
```

Рис. В.2 – Формула для обчислення вартості добрив у коді

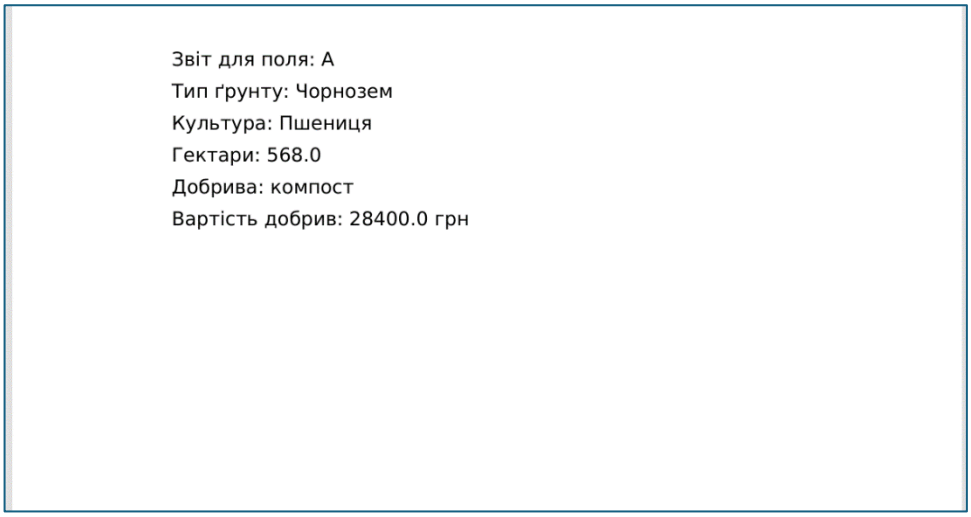

Рис. В.3 – Згенерований звіт у форматі PDF для конкретного поля