

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ МЕТОДІВ ЛОГІСТИКИ ТА РОЗРОБКА

CRM-СИСТЕМИ ДЛЯ ТВАРИННИЦЬКИХ ФЕРМ

DEVELOPMENT OF A CRM SYSTEM FOR LIVESTOCK FARMS, INCLUDING

A COMPREHENSIVE STUDY OF LOGISTIC METHODS

Виконав: студент(ка) 2 курсу, групи КНТ-113М
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ДОЦЕНКО С.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ДОЦЕНКО Степана Сергійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження методів логістики та розробка CRM-системи для тваринницьких ферм. Development of a CRM System for Livestock Farms, Including a Comprehensive Study of Logistic Methods.

керівник проєкту (роботи) к.т.н., ГЛАДКОВА Ольга Миколаївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “07” жовтня 2024 року № 398

2. Строк подання студентом проєкту (роботи) 5 грудня 2024 року

3. Вихідні дані до проєкту (роботи) технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Дослідження клієнт-серверних рішень CRM-систем для тваринницьких ферм. 2. Засоби розробки програмного продукту. 3. Проєктування програмного продукту. 4. Конструювання програмного продукту. 5. Експлуатація та тестування розробленого програмного продукту.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 4 » вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Дослідження предметної області та рішень вирішення завдання.	1 тиждень	Розділ 1
3	Вибір інструментів для розробки програмного продукту.	2 тиждень	Розділ 2
4	Моделювання вимог та проєктування архітектури до програмного продукту.	3 тиждень	Розділ 3
5	Розробка програмного продукту.	4-7 тижні	Розділ 4
6	Тестування та експериментальне дослідження розробленого програмного продукту.	8 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	9-10 тижні	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Степан ДОЦЕНКО
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Ольга ГЛАДКОВА
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
122 с., 6 табл., 43 рис., 3 дод., 32 джерела.

БІЗНЕС-ПРОЦЕС, ВЕБЗАСТОСУНОК, CRM, ЛОГІСТИКА,
ТВАРИННИЦЬКА ФЕРМА, ВЕТЕРИНАРНІ ПОСЛУГИ, PHP, LARAVEL.

Об'єкт дослідження – системи та методи логістики та бізнес-процеси діяльності тваринницьких ферм.

Предмет дослідження – CRM-система для керування бізнес-процесами та вирішення логістичних задач для ветеринарів та власників тваринницьких ферм.

Мета роботи – дослідження і розробка вебзастосунку, що надає функціональні можливості для керування бізнес-процесами ветеринарних лабораторій, клінік та власників тваринницьких ферм.

Матеріали, методи та технічні засоби: текстовий редактор Jetbrains PhpStorm, функціональне програмування, мова розмітки вебдокументів HTML, мова опису стилів CSS, фреймворк Bootstrap 5, мови програмувань JS та PHP, фреймворки до них Vue 3 JS та Laravel 10, бібліотека компонентів інтерфейсу користувача Vuestic UI, бібліотека маршрутизації вебсторінок Vue Router, СКБД MariaDB, та ноутбук з операційною системою Microsoft Windows 11 та WSL(Windows Subsystem for Linux) з процесором Intel Core i7-13700H.

Результати. Розроблено вебзастосунок, який надає функції керування бізнес-процесами, допомагає у вирішанні логістичних задач для переміщення тварин та підвищує ефективність роботи з обслуговування клієнтів для керівників та співробітників ветеринарних лабораторій та клінік.

Висновки. Було виконано проєктування, конструювання та тестування розробленого вебзастосунку для керування бізнес-процесами та вирішення логістичних задач для переміщення тварин.

Галузь використання – ветеринари, ветеринарні клініки, фермери.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 122 pages, 43 figures, 6 tables, 3 appendixes, 32 sources.

BUSINESS PROCESS, WEB APPLICATION, CRM, LOGISTICS, LIVESTOCK FARM, VETERINARY SERVICES, PHP, LARAVEL.

Object of the study – logistics methods, systems and business processes of livestock farms.

Subject of the study – a CRM system for managing business processes and solving logistics tasks for veterinarians and livestock farm owners.

Purpose of the work – to research and develop a web application that provides functional capabilities for managing the business processes of veterinary laboratories, clinics, and livestock farm owners.

Materials, methods, and technical tools: JetBrains PhpStorm text editor, functional programming, HTML markup language, CSS styling language, Bootstrap 5 framework, JS and PHP programming languages, Vue 3 JS and Laravel 10 frameworks, Vuestic UI user interface component library, Vue Router library for webpage routing, MariaDB DBMS, and a laptop running Microsoft Windows 11 and WSL (Windows Subsystem for Linux) with an Intel Core i7-13700H processor.

Results. A study of software solutions providing similar functions was conducted. A web application was developed that offers business process management functions, helps solve logistics tasks for animal transportation, and improves customer service efficiency for managers and staff of veterinary laboratories and clinics.

Conclusions. The design, construction, and testing of the developed web application for managing business processes and solving logistics tasks for animal transportation were completed.

Field of application – veterinarians, veterinary clinics, farmers.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	8
Вступ.....	9
1 Дослідження клієнт-серверних рішень CRM-систем для тваринницьких ферм.	11
1.1 Аналіз та особливості предметної області.....	11
1.2 Бізнес-процеси тваринницької ферми.....	13
1.3 Огляд CRM-систем для тваринницьких ферм	15
1.3.1 Вебплатформа Zoho CRM.....	15
1.3.2 Вебзастосунок AgroWebb	19
1.3.3 Вебзастосунок Farmbrite	21
1.3.4 Порівняння досліджених вебзастосунків	22
1.4 Аналіз логістичних методів	23
1.5 Постановка завдань дипломної магістерської роботи	24
1.6 Висновки до першого розділу	26
2 Засоби розробки програмного продукту.....	27
2.1 Вибір клієнтської та серверної мови програмування.....	27
2.2 Вибір середовища розробки вебзастосунку.....	30
2.3 Вибір засобів розробки користувацької частини вебзастосунку	32
2.3.1 Мова розмітки вебдокументів HTML	32
2.3.2 Мова опису стилів CSS.....	33
2.3.3 Фреймворк Vue 3 JS	33
2.3.4 Бібліотека компонентів Vuestic UI	35
2.3.5 Бібліотека для створення таблиць Vue 3 Easy Data Table.....	36
2.3.6 Бібліотека маршрутизації вебсторінок Vue Router	36
2.3.7 Менеджер керування станами Vuex	38
2.4 Вибір засобів розробки серверної частини вебзастосунку	40
2.4.1 Фреймворк Laravel	40
2.5 Вибір СКБД.....	43

2.6	Висновки до другого розділу	44
3	Проектування програмного продукту.....	46
3.1	Проектування сценаріїв використання вебзастосунку.....	46
3.2	Проектування структури навігації вебзастосунку.....	48
3.3	Проектування архітектури вебзастосунку	49
3.4	Висновки до третього розділу.....	51
4	Конструювання програмного продукту.....	52
4.1	Фізична модель реалізованої бази даних	52
4.2	Алгоритм функціонування розробленого вебзастосунку.....	53
4.3	Розроблені модулі вебзастосунку	56
4.4	Опис розроблених функцій	56
4.5	Висновки до четвертого розділу.....	58
5	Експлуатація та тестування розробленого програмного продукту	59
5.1	Опис призначення програмного продукту.....	59
5.2	Вимоги до програмно-технічних засобів	60
5.3	Характеристики програмного продукту.....	61
5.4	Виконання програмного продукту	61
5.5	Висновки до п'ятого розділу.....	67
	Висновки	69
	Перелік джерел посилання	72
	Додаток А Технічне завдання	75
	Додаток Б Текст програми	80
	Додаток В Слайди презентації.....	112

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

SQL – Structured Query Language;

UI – User Inteface;

URL – User Inteface;

СКБД – система керування базами даних.

ВСТУП

CRM системи та вебзастосунки стали потужними інструментами для підвищення продуктивності та оптимізації бізнес-процесів. Зокрема, CRM дозволяють компаніям зберігати, структурувати та аналізувати величезні обсяги даних про клієнтів, що значно полегшує маркетинг та підтримку продажів. За допомогою таких систем бізнес отримує можливість краще розуміти потреби клієнтів, передбачати їхні запити та швидше реагувати на змінні ринкові умови, що підвищує їхню конкурентоспроможність на ринку [1].

Окрім цього, вебзастосунки спрощують доступ до даних у реальному часі, що є критично важливим для керування бізнесом в умовах сучасного цифрового світу. Використання таких інструментів дає змогу автоматизувати рутинні завдання, такі як обробка запитів, виставлення рахунків, управління замовленнями тощо. Це дозволяє звільнити час працівників для більш важливих завдань, таких як стратегічне планування та підтримка відносин із клієнтами. За словами аналітиків NetSuite, автоматизація бізнес-процесів сприяє зниженню людських помилок і підвищує стабільність процесів, що важливо для великих компаній з розподіленою структурою [2].

Згідно з дослідженням McKinsey, автоматизація та CRM значно змінюють робочі процеси, дозволяючи працівникам швидше адаптуватися до нових задач, а компаніям – залишатися гнучкими в умовах ринкових коливань. Це створює нові можливості для розвитку бізнесу, включно з впровадженням передових методів роботи на основі штучного інтелекту та машинного навчання [3].

Актуальність дипломної магістерської роботи обумовлена у тому, що кожний бізнес-процес включає велику кількість дій, які віднімають безліч робочого часу, і які суттєво збільшують робоче навантаження. В результаті стає важко проконтролювати кожний бізнес-процес та уникнути помилок в обліку. Рішенням цієї проблеми є впровадження програмного рішення для керування бізнес-процесами та вирішення різних логістичних задач, яке полегшить та автоматизує роботу працівників ветеринарних клінік та фермерів.

Об'єктом дослідження є системи, методи логістики та бізнес-процеси

діяльності тваринницьких ферм.

Предметом дослідження є CRM-система для керування бізнес-процесами та вирішення логістичних задач для ветеринарів та власників тваринницьких ферм.

Метою роботи є створення CRM-системи для керування бізнес-процесами та вирішення логістичних задач для ветеринарів та власників тваринницьких ферм.

У дипломній роботі для досягнення поставленої мети роботи необхідно виконати перераховані нижче задачі магістерського дослідження:

- дослідження програмних рішень, які надають аналогічні функції;
- розробка технічного завдання;
- вибір засобів для розробки CRM-системи для тваринницьких ферм;
- моделювання вимог до CRM-системи для тваринницьких ферм;
- проектування структури навігації CRM-системи для тваринницьких ферм;
- проектування архітектури CRM-системи для тваринницьких ферм;
- конструювання CRM-системи для тваринницьких ферм;
- тестування функціональності розробленого вебзастосунку.

1 ДОСЛІДЖЕННЯ КЛІЄНТ-СЕРВЕРНИХ РІШЕНЬ CRM-СИСТЕМ ДЛЯ ТВАРИННИЦЬКИХ ФЕРМ

У розділі проведено дослідження предметної області та існуючих програмних рішень для керування бізнес-процесами на тваринницькій фермі. Наведено порівняння досліджених аналогічних програмних рішень та сформовано завдання, які будуть вирішуватися при виконанні дипломної магістерської роботи.

1.1 Аналіз та особливості предметної області

CRM-система для тваринницьких ферм повинна враховувати особливості цієї галузі, де основними завданнями є управління обліком тварин, планування ветеринарних заходів, моніторинг здоров'я поголів'я та взаємодія з постачальниками і клієнтами. Система повинна забезпечувати ефективне управління щоденними операціями та сприяти підвищенню продуктивності ферми, дозволяючи швидко отримувати доступ до важливої інформації про кожну тварину.

Однією з ключових функцій CRM-системи для тваринницьких ферм є забезпечення контролю за здоров'ям та продуктивністю поголів'я. Система дозволяє фіксувати інформацію про кожну тварину, включаючи дані про ветеринарні огляди, лікування та щеплення, що сприяє ранньому виявленню захворювань і підвищує ефективність управління здоров'ям стада. Це, в свою чергу, допомагає зменшити витрати та оптимізувати виробничі процеси на фермі [2].

Ще однією важливою функцією є планування ветеринарних заходів. CRM-система має забезпечувати можливість фермерам планувати ветеринарні огляди, вакцинації та лікування, автоматично нагадуючи про ці заходи. Ведення журналу лікувань надає зручний доступ до історії здоров'я кожної тварини, що є критичним для підтримання здорового поголів'я.

CRM-системи для тваринницьких ферм також відіграють важливу роль у плануванні й управлінні розмноженням тварин, що є ключовим аспектом для забезпечення стабільного зростання поголів'я. За допомогою CRM фермери можуть відстежувати цикли розмноження, прогнозувати народження потомства, планувати процедури парування та оптимізувати процеси догляду за молодняком. Завдяки цьому система допомагає знижувати ризики, пов'язані з розмноженням, і підвищувати продуктивність ферми, оскільки всі важливі дані доступні в одному місці [3].

Крім того, CRM-системи сприяють покращенню управління запасами кормів та інших ресурсів. Завдяки інтеграції з системами обліку, CRM дозволяє фермерам відстежувати запаси кормів, планувати їх поповнення та контролювати витрати. Це забезпечує безперебійне постачання, оптимізує витрати на корми та дозволяє уникнути надлишків чи дефіциту, що безпосередньо впливає на рентабельність господарства. Також важливим є автоматизований облік інших витрат, що дозволяє фермерам мати чітке уявлення про економічні показники господарства та своєчасно реагувати на фінансові зміни [2].

Завдяки аналітичним інструментам CRM-системи, фермери також отримують можливість глибокого аналізу продуктивності свого господарства. Дані про надої молока, приріст ваги або рівень смертності можна візуалізувати та аналізувати в режимі реального часу, що допомагає приймати обґрунтовані рішення. Система дозволяє швидко виявляти відхилення в показниках та проводити коригуючі дії, спрямовані на підвищення ефективності всієї ферми.

Функції CRM-систем для тваринницьких ферм:

- ведення обліку тварин (ідентифікаційні дані, порода, дата народження);
- планування ветеринарних заходів (огляди, вакцинації, лікування);
- моніторинг процесів розмноження та прогнозування потомства;
- управління запасами кормів та іншими ресурсами;
- аналіз продуктивності тварин та оцінка показників здоров'я.

Загалом, CRM-система для тваринницьких ферм є комплексним інструментом, який об'єднує дані про здоров'я тварин, управління запасами, планування розмноження та аналітику в єдиній платформі. Це дозволяє ефективно керувати всіма процесами на фермі, покращувати взаємодію між різними підрозділами господарства та досягати більшої продуктивності.

CRM-системи для тваринницьких ферм широко застосовуються фермерами та менеджерами господарств, які прагнуть підвищити ефективність управління своїм поголів'ям. За допомогою такого програмного забезпечення фермери можуть стежити за здоров'ям тварин, зберігати дані про кожну особину та своєчасно проводити ветеринарні заходи, що допомагає уникнути багатьох проблем у подальшому [3].

Крім того, CRM-системи активно використовують ветеринари, які працюють із фермерськими господарствами. Завдяки доступу до медичних записів тварин, ветеринари можуть планувати огляди та лікування, спираючись на точні дані про історію кожної особини. Це значно полегшує роботу ветеринарів, а також підвищує якість обслуговування тварин [2].

Великі агрокомпанії також використовують CRM для управління витратами та підвищення рентабельності своїх ферм. Застосування такого програмного забезпечення дозволяє ефективно керувати запасами кормів, моніторити економічні показники та оцінювати продуктивність окремих ферм, що є важливим для великого бізнесу з розподіленими господарствами [3].

1.2 Бізнес-процеси тваринницької ферми

Бізнес-процес – це логічна впорядкована взаємопов'язана послідовність дій або операцій, виконуваних посадовими особами та підрозділами компанії для отримання бажаного кінцевого результату (досягнення мети, вирішення завдання, реалізації програмного продукту, надання послуги). Будь-який процес обов'язково повинен мати вхід і вихід, тобто отримувати будь-яку інформацію або ресурси від зовнішнього об'єкта або іншого процесу, перетворювати їх і

передавати іншому об'єкту або процесу [4].

Для ферми виокремлюють 6 основних видів бізнес- процесів [5]:

- управління стадом;
- виробничий процес;
- управління кормами;
- здоров'я тварин;
- управління персоналом;
- фінансовий облік та планування.

Використання CRM-систем у тваринництві надає фермерам можливість ефективно управляти інформацією про кожну тварину, включаючи дані про здоров'я, продуктивність та розмноження. Це сприяє прийняттю обґрунтованих рішень щодо догляду та управління стадом, підвищуючи загальну ефективність господарства [6].

Для ветеринарів CRM-системи забезпечують доступ до детальної історії здоров'я тварин, що полегшує діагностику та планування лікування. Це дозволяє надавати більш якісні послуги та підвищує довіру з боку фермерів [7].

Крім того, CRM-системи сприяють покращенню комунікації між фермерами та ветеринарами, забезпечуючи централізоване зберігання та обмін інформацією. Це дозволяє швидко реагувати на зміни в стані здоров'я тварин та координувати необхідні заходи [8].

Бізнес-логіка в CRM-системах виконує роль "мозку" системи, який забезпечує зв'язок між даними, бізнес-процесами та користувацьким інтерфейсом. Вона відповідає за обробку даних, які надходять від користувача, і перетворює їх у зрозумілі процеси, що виконуються системою. Бізнес-логіка також встановлює правила, умови та послідовність дій, яких система має дотримуватись для досягнення певної мети, наприклад, автоматизації рутинних завдань, таких як обробка запитів клієнтів або відстеження продажів.

Основною метою бізнес-логіки є забезпечення правильності та ефективності виконання бізнес-процесів. Вона допомагає уникати дублювання даних, забезпечує інтеграцію різних модулів системи та спрощує управління

складними процесами, такими як обробка замовлень, управління запасами або планування ресурсів. За допомогою бізнес-логіки CRM-система може автоматично направляти ліди до відповідних менеджерів, надсилати повідомлення клієнтам про оновлення статусу їхніх замовлень або запускати маркетингові кампанії відповідно до визначених критеріїв [9].

Бізнес-логіка також робить систему гнучкою та адаптованою до потреб конкретного бізнесу. Вона дозволяє налаштовувати процеси відповідно до специфіки діяльності компанії, створювати індивідуальні робочі сценарії та встановлювати пріоритети для різних завдань. У результаті, CRM-система не просто зберігає інформацію, а й активно сприяє досягненню бізнес-цілей, забезпечуючи ефективну взаємодію між різними процесами та підвищуючи продуктивність команди.

Таким чином, бізнес-логіка є фундаментальним компонентом, який забезпечує виконання функцій CRM-системи відповідно до потреб компанії, підтримує якість обслуговування клієнтів, допомагає швидко реагувати на запити та спрощує управління бізнесом у реальному часі [10].

1.3 Огляд CRM-систем для тваринницьких ферм

Нижче наведено результати проведеного дослідження вебзастосунків для керування бізнес-процесами у тваринницьких фермах. Для дослідження було обрано системи Zoho CRM, AgriWebb та Farmbrite.

1.3.1 Вебплатформа Zoho CRM

Zoho CRM – це потужна хмарна система управління взаємовідносинами з клієнтами, яка може бути адаптована для потреб тваринницьких ферм. Використання Zoho CRM у тваринництві дозволяє оптимізувати управління господарством, підвищити ефективність операцій та покращити взаємодію з клієнтами та постачальниками. Інтерфейс зображений на рисунку 1.1.

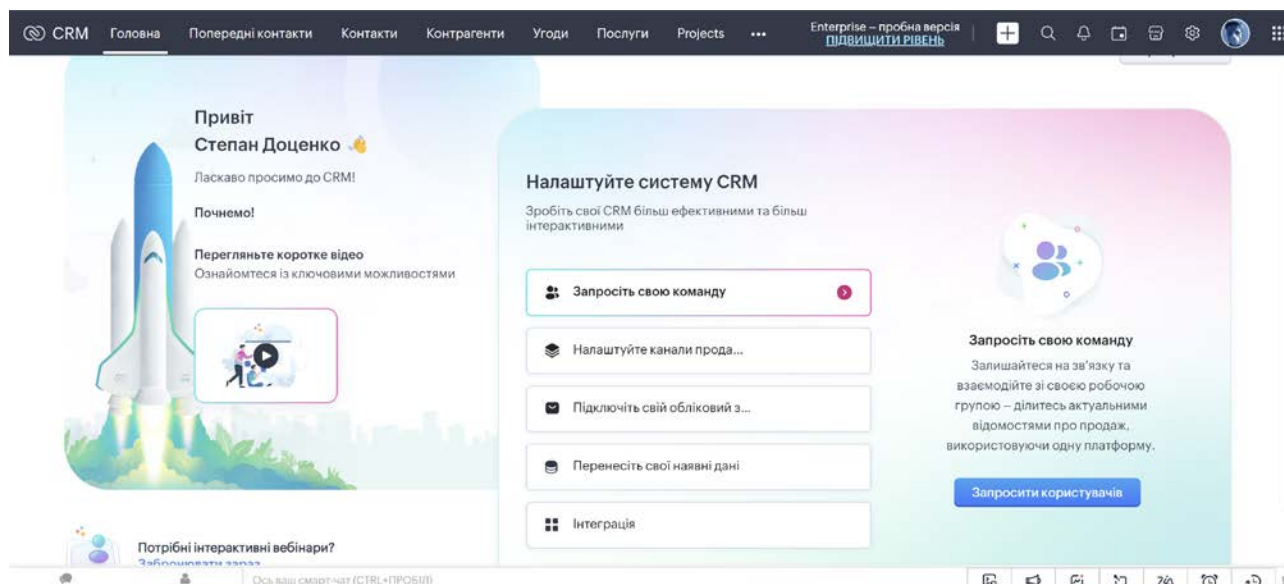


Рисунок 1.1 – Інтерфейс головної сторінки вебзастосунку Zoho CRM

Функціональні можливості вебзастосунку Zoho CRM:

- облік кожної тварини з урахуванням ідентифікаційного номера, породи та віку;
- ведення журналу ветеринарних оглядів і вакцинацій;
- планування та призначення завдань персоналу ферми;
- ведення бази постачальників кормів і ветеринарних препаратів;
- відображення бази потенційних покупців продукції ферми;
- додавання звернень від клієнтів та запитів на продукцію;
- ведення бази послуг, що надаються фермерам, таких як огляди та консультації;
- створення техкарт для розмноження і відгодівлі тварин;
- контроль і поповнення запасів кормів та медикаментів;
- список очікування на послуги ветеринара або консультації [11].

Вебзастосунок Zoho CRM надає можливість отримувати аналітичні звіти, що охоплюють ключові процеси та показники діяльності фермерського господарства, такі як продажі, витрати, управління запасами кормів, ветеринарні процедури та взаємодії з постачальниками. Керівник тваринницької ферми може переглядати основні показники продуктивності, витрат та доходів, що дозволяє здійснювати моніторинг і контроль за діяльністю ферми.

До аналітичних звітів Zoho CRM входять:

- звіт про загальний прибуток та витрати ферми;
- аналітика запасів кормів та інших ресурсів;
- аналіз продажів продукції за напрямками (м'ясо, молочні продукти тощо);
- аналітика видів оплати клієнтів;
- аналіз візитів ветеринарів та інших спеціалістів;
- аналітика клієнтів та постачальників;
- аналіз найбільш затребуваних та найменш популярних товарів;
- оцінка ефективності акцій та знижок;
- аналіз виконання завдань персоналом;
- звіт про кількість запитів та звернень від клієнтів;
- аналіз постачання та витрат на корм;
- оцінка завантаження тижня в контексті візитів ветеринарів та персоналу.

Журнал записів (рис. 1.2) у вебзастосунку Zoho CRM дозволяє вести облік візитів фахівців на ферму, таких як ветеринари, зоотехніки та інші спеціалісти. Функція дозволяє записувати відвідування, контролювати розклад і відстежувати статус наданих послуг. Zoho CRM також дозволяє вести список очікування для клієнтів, які потребують послуг у визначений час.

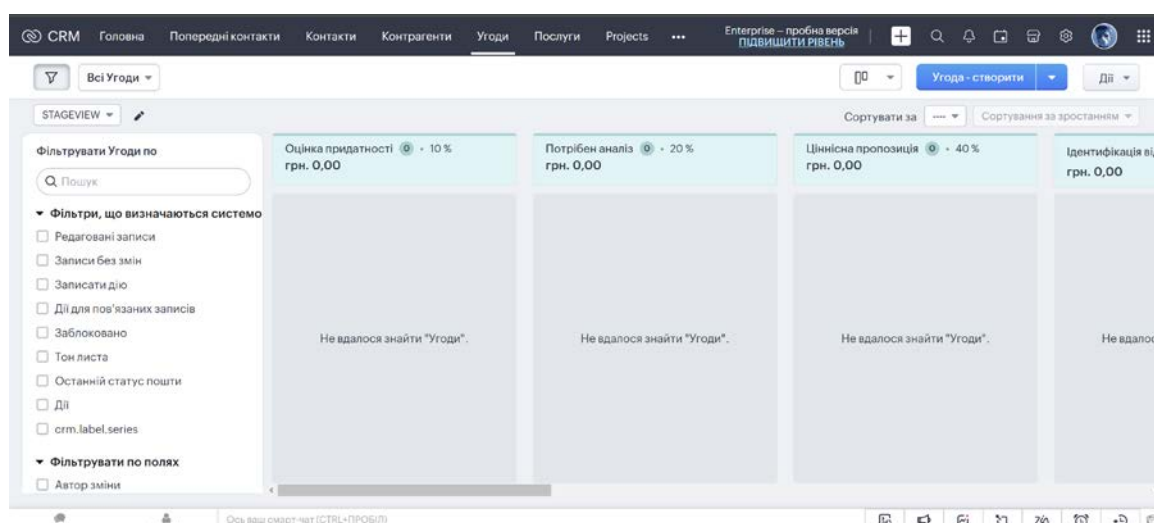


Рисунок 1.2 – Інтерфейс списку записів угод Zoho CRM

Вебзастосунок пропонує модуль для коротких швидких записів і завдань, які працівники однієї зміни можуть залишати для наступної зміни. Це зручно для організації безперервної роботи, де кожен спеціаліст знає свої обов'язки і відомо про поточні справи.

Zoho CRM підтримує функцію редагування візитів, що включає зміну спеціаліста, додавання нових послуг, вибір іншого часу та додавання додаткових товарів або ресурсів. Це дозволяє коригувати графік роботи спеціалістів і задовольняти потреби клієнтів [12].

У вебзастосунку Zoho CRM також є три типи задач:

- автоматичні задачі для нагадування про наступний візит;
- повторювані задачі для регулярних обов'язків;
- звичайні задачі для організації роботи персоналу.

Модуль автозадач дозволяє автоматизувати нагадування про наступні візити для клієнтів, що сприяє підвищенню якості обслуговування та своєчасності надання послуг.

Таблиця клієнтів у Zoho CRM (рис. 1.3) забезпечує зручне зберігання та управління базою клієнтів, надаючи можливість додавати нових клієнтів, редагувати їхні профілі, фільтрувати записи, надсилати повідомлення, завантажувати базу клієнтів і переглядати статистику взаємодії. Функція групової обробки клієнтів у Zoho CRM дозволяє присвоювати групи або теги великій кількості клієнтів одночасно, та керувати їхніми бонусними балансами.

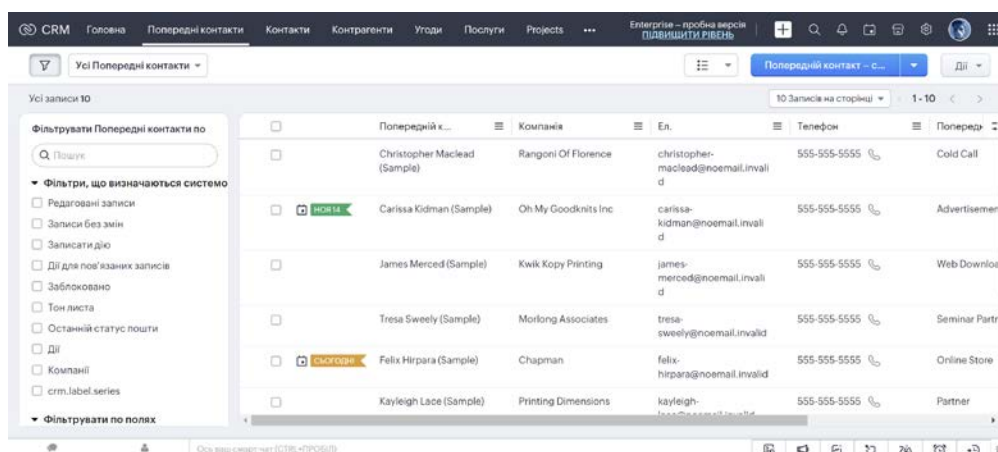


Рисунок 1.3 – Інтерфейс списку клієнтів Zoho CRM

Zoho CRM підтримує управління запасами, облік реклаमाцій, що дозволяє відстежувати невдоволення клієнтів і запобігати подальшим проблемам.

Система також включає модуль управління фінансами, що дозволяє відслідковувати рух грошових коштів, доходи та витрати компанії.

Таким чином, Zoho CRM забезпечує комплексне управління тваринницьким господарством, допомагаючи фермерам ефективно управляти процесами, оптимізувати роботу персоналу та підтримувати високий рівень обслуговування клієнтів [13].

1.3.2 Вебзастосунок AgroWebb

AgriWebb — це спеціалізована CRM-система, яка пропонує фермерам інструменти для комплексного управління тваринницькими господарствами. Вона дозволяє не лише вести облік тварин, але й забезпечує зручне картографування ферми, що включає розташування пасовищ, розміщення тварин та інфраструктури. Це дозволяє легко планувати та оптимізувати використання земельних ресурсів, а також покращує моніторинг усього господарства. Головні функції цього застосунку:

- облік кожної тварини з індивідуальними показниками та історією здоров'я;
- картографування господарства з візуалізацією пасовищ та розташування тварин;
- управління випасом та оптимізація використання пасовищ;
- ведення журналу ветеринарних оглядів та вакцинацій;
- контроль запасів кормів, медикаментів та інших ресурсів;
- створення та моніторинг завдань для співробітників;
- офлайн-доступ до даних з автоматичною синхронізацією при відновленні інтернет з'єднання;
- аналітика продуктивності тварин та ефективності розведення;

- управління добривами та покривними культурами для пасовищ;
- відстеження витрат і аналіз прибутковості господарства.

Однією з ключових функцій AgriWebb є управління індивідуальними та груповими показниками тварин. Система надає можливість відстежувати здоров'я кожної тварини, історію оглядів і вакцинацій, а також продуктивність, що дозволяє фермерам приймати обґрунтовані рішення щодо розведення чи вибракування. Завдяки цим даним фермери можуть значно підвищити ефективність управління стадами.

AgriWebb також пропонує інструменти для управління випасом тварин і моніторингу пасовищ (рис. 1.4). Система надає детальну аналітику використання пасовищ, включаючи дані про добрива та покривні культури. Це дозволяє оптимізувати ресурси, підвищуючи продуктивність господарства. Крім того, система забезпечує ведення журналу ветеринарних оглядів, що дозволяє слідкувати за станом здоров'я стада в режимі реального часу.

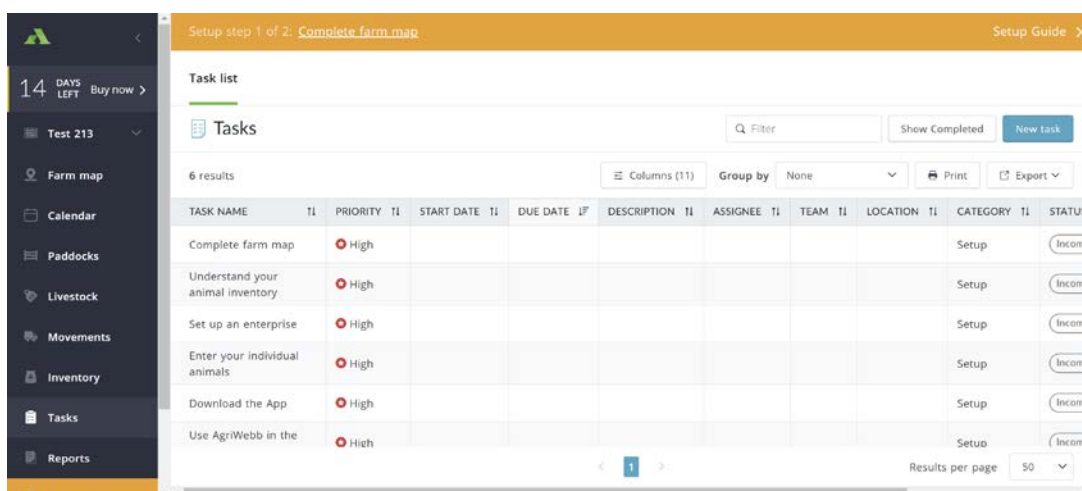


Рисунок 1.4 – Інтерфейс сторінки задач для ферми AgriWebb

Система підтримує комплексне управління запасами, що включає контроль за наявністю кормів, медикаментів та інших необхідних ресурсів. AgriWebb також надає інструменти для створення та моніторингу завдань для співробітників, що полегшує управління фермою та підвищує продуктивність команди. Особливо зручною є функція офлайн-доступу, яка дозволяє

продовжувати роботу навіть за відсутності інтернет-з'єднання з автоматичною синхронізацією після його відновлення.

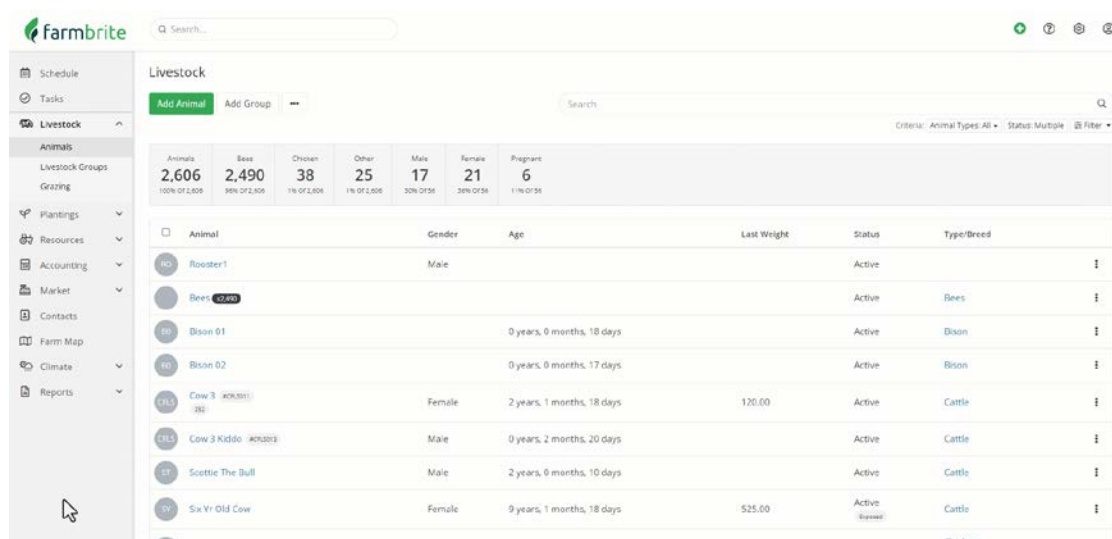
Таким чином, AgriWebb — це CRM, яка значно спрощує управління тваринницьким господарством, надаючи потужні інструменти для обліку, планування та аналізу, що сприяє зниженню витрат і підвищенню рентабельності бізнесу [14].

1.3.3 Вебзастосунок Farmbrite

Farmbrite — це комплексна CRM-система, створена для управління фермерськими господарствами, зокрема тими, що займаються тваринництвом. Вона пропонує широкий спектр функцій, які полегшують роботу фермерів, забезпечуючи зручний облік та моніторинг усіх аспектів ферми.

Система надає інструменти для обліку кожної тварини, включаючи індивідуальні показники, такі як вік, вага та історія вакцинацій (рис. 1.5). Завдяки цьому фермери можуть легко відстежувати стан здоров'я тварин і контролювати ефективність різних підходів до догляду та розведення.

Farmbrite також забезпечує ведення журналу ветеринарних оглядів та вакцинацій, що допомагає запобігати захворюванням і підтримувати поголів'я у доброму стані.



Animal	Gender	Age	Last Weight	Status	Type/Breed
Rooster 1	Male			Active	
Bees				Active	Bees
Bison 01		0 years, 0 months, 18 days		Active	Bison
Bison 02		0 years, 0 months, 17 days		Active	Bison
Cow 3	Female	2 years, 1 months, 18 days	120.00	Active	Cattle
Cow 3 Kid	Male	0 years, 2 months, 20 days		Active	Cattle
Scottie The Bull	Male	2 years, 0 months, 10 days		Active	Cattle
Six Yr Old Cow	Female	9 years, 1 months, 18 days	525.00	Active	Cattle
					Chicken

Рисунок 1.5 – Інтерфейс сторінки списку тварин Farmbrite

Система надає можливість управління випасом, допомагаючи фермерам оптимізувати використання пасовищ та контролювати витрати на корми. Окрім цього, Farmbrite пропонує інструменти для управління добривами та покривними культурами, що сприяє підвищенню продуктивності земельних ресурсів.

Farmbrite також підтримує контроль за запасами ресурсів, таких як корми та медикаменти, що допомагає уникати дефіциту або надмірних витрат.

Система дозволяє створювати завдання для співробітників і контролювати їх виконання. Завдяки офлайн-доступу користувачі можуть працювати з даними без інтернету, а після відновлення з'єднання дані автоматично синхронізуються. Це особливо зручно для ферм, розташованих у віддалених районах з нестабільним інтернет-зв'язком.

Farmbrite також надає аналітичні інструменти для оцінки продуктивності тварин та аналізу прибутковості господарства. Система дозволяє відстежувати витрати, доходи та інші фінансові показники, що дає можливість фермерам приймати обґрунтовані рішення для підвищення ефективності бізнесу [15].

1.3.4 Порівняння досліджених вебзастосунків

Нижче наведено порівняння трьох CRM-систем (таблиця 1.1), що підходять для управління тваринницькими фермами: Zoho CRM, AgriWebb та Farmbrite. Кожна з них має власний набір функцій, що дозволяє оптимізувати різні аспекти фермерського господарства, такі як облік тварин, управління пасовищами, контроль запасів кормів та медикаментів, а також аналітичні звіти для оцінки ефективності бізнесу.

Zoho CRM пропонує широкі можливості інтеграції та управління фінансами, що підходить для фермерських господарств із фокусом на клієнтські взаємодії та фінансовий контроль. AgriWebb спеціалізується на управлінні тваринництвом, включаючи функції для оптимізації випасу та догляду за тваринами. Farmbrite забезпечує комплексне управління фермою з можливістю ведення обліку тварин, фінансів і доступом до даних навіть без інтернету.

Таблиця 1.1 – Порівняння досліджених CRM-систем для тваринницьких ферм.

Критерій порівняння	Zoho CRM	AgriWebb	Farmbrite
Облік тварин	+	+	+
Журнал ветеринарних оглядів та вакцинацій	+	+	+
Управління пасовищами та випасом	—	+	+
Картографування ферми	—	+	+
Контроль запасів кормів і медикаментів	+	+	+
Управління завданнями персоналу	+	+	+
Аналітичні звіти	+	+	+
Офлайн-доступ	—	+	+
Інтеграція з іншими сервісами	+	—	+
Управління фінансами та бюджетом	+	—	+
Управління добривами і покривними культурами	—	+	+
База постачальників ресурсів	+	—	—
Модуль для створення техкарт	—	+	+
Візуалізація та контроль виробничих процесів	+	+	+
Облік прибутків та витрат	+	+	+
Управління клієнтськими базами	+	—	+
Доступ до мобільного додатку	+	—	—

1.4 Аналіз логістичних методів

Аналіз та вирішення логістичних задач вимагають застосування різноманітних методів, які дозволяють оптимізувати процеси управління матеріальними потоками, знижувати витрати та підвищувати ефективність логістичних систем. Одним із ключових підходів є системний аналіз, що передбачає комплексне вивчення логістичних систем як єдиних цілісних

утворень з урахуванням взаємозв'язків між їхніми елементами. Цей метод дозволяє виявити слабкі місця в системі та розробити стратегії для їх усунення.

Іншим важливим методом є дослідження операцій, яке включає застосування математичних моделей та кількісних підходів для прийняття оптимальних рішень у логістиці. Цей метод дозволяє моделювати різні сценарії розвитку подій та обирати найкращі варіанти дій.

Прогностика, або методи прогнозування, також відіграють значну роль у логістиці. Вони дозволяють передбачати майбутні потреби в ресурсах, обсяги продажів та інші показники, що сприяє ефективному плануванню логістичних операцій.

Методи моделювання, зокрема імітаційне моделювання, широко застосовуються для дослідження логістичних систем. Вони дозволяють створювати віртуальні моделі логістичних процесів та аналізувати поведінку в різних умовах, що сприяє прийняттю обґрунтованих управлінських рішень [16].

АВС-аналіз є методом класифікації товарно-матеріальних ресурсів за ступенем їх важливості. Він базується на принципі Парето, згідно з яким 20% найменувань продукції можуть приносити 80% прибутку. Він дозволяє зосередитись на ключових позиціях та оптимізувати управління запасами [17].

Аналіз повної вартості передбачає облік усіх витрат, пов'язаних з логістичними операціями, з метою мінімізації загальних витрат у системі. Цей метод дозволяє виявити приховані витрати та оптимізувати логістичні процеси [18].

Застосування цих методів у комплексі дозволяє ефективно вирішувати логістичні задачі, підвищувати продуктивність логістичних систем та забезпечувати конкурентоспроможність підприємства на ринку.

1.5 Постановка завдань дипломної магістерської роботи

Користувачами вебзастосунку є приватні ветеринари, власники ферм, адміністратори, менеджери та власники застосунку. Відповідно до ролі

користувача керівник або адміністратор при додаванні співробітника вказує розмежування прав доступу. На основі проведеного дослідження аналогічних програмних рішень були сформовані функціональні можливості, які повинен надавати вебзастосунок CRM-система для тваринницьких ферм:

- аутентифікація та авторизація користувача;
- ведення обліку тварин з індивідуальними показниками та історією здоров'я;
- додавання тварин із історії попередніх записів та оглядів;
- адаптивний інтерфейс для зручної роботи на будь-яких пристроях;
- планування ветеринарних оглядів і автоматичне нагадування про події;
- автоматичне створення звітів про тестування тварин;
- створення заяв для перевезення тварин між фермами;
- створення відповідних документів для перевезення тварин;
- родове дерево кожної тварини;
- створення нових ділянок для вже наявних ферм за допомогою інтерактивної мапи та державної бази даних;
- прокладання маршруту перевезення тварин на мапі;
- тестування тварин під час каратнину за допомогою протоколів;
- створення документів з результатами тестування;
- сповіщення шляхом надсилання листів на електронну пошту;
- надання, відповідних ролі користувача, дозволів;
- бот для отримання даних про найближчу ферму або місцевого ветеринара;
- API для отримання даних про ферму або тварину;
- створення заяв на полювання та реєстрації спеціального ВU номера (тільки для ферм, які мають на своїй території буйволів).

1.6 Висновки до першого розділу

У першому розділі досліджено предметну область та аналогічні вебзастосунки для керування бізнес-процесами тваринницької ферми. Було сформовано завдання, які будуть вирішуватися при виконанні дипломної магістерської роботи.

Провівши дослідження аналогічних вебзастосунків можна зробити висновок, що метою кожного вебзастосунку для керування бізнес-процесами на тваринницькій фермі є автоматизація щоденних завдань співробітників ферм, ветеринарів та ветеринарних клінік, спрощення ведення обліку бізнес-процесів.

Користувачами у досліджених вебзастосунків виступають приватні ветеринари, фермери, власники ферм, ветеринарні лабораторії тощо. У ході дослідження аналогічних вебзастосунків було виявлено основні функціональні можливості, які повинен надавати програмний продукт для керування бізнес-процесами на тваринницькій фермі:

- ведення обліку користувачів;
- ведення обліку ферм, лабораторій, приватних ветеринарів;
- ведення обліку тварин та їх характеристик;
- ведення загальної інформації про тестування тварин;
- ведення обліку заяв на перевезення тварин між фермами;
- проводити тестування тварин за протоколами;
- надання інформації про зареєстрованих тварин та ферм.

2 ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

У розділі описано обрані засоби для розробки інтерфейсної та серверної частини вебзастосунку CRM-системи для вирішення логістичних задач тваринницьких ферм, а також засоби взаємодії з базою даних. Наведено порівняння обраної мови програмування з іншими мовами, редакторів коду, засобів для розробки інтерфейсної частини, програмних каркасів для розробки серверної частини та нереляційних документо-орієнтованих СКБД.

2.1 Вибір клієнтської та серверної мови програмування

Для розробки клієнтської та серверної частин вебзастосунку було обрано мову програмування PHP, оскільки вона здатна вирішувати різноманітні завдання та надає широкий спектр фреймворків для створення вебзастосунків.

Кожен вебзастосунок складається з клієнтської та серверної частин. Клієнтська частина (frontend) — це частина вебзастосунку, яка завантажується з сервера та виконується на клієнтському пристрої (веббраузері). Серверна частина (backend) працює на сервері, обробляючи запити від клієнта та надаючи необхідні дані.

PHP (Hypertext Preprocessor) — це популярна мова програмування загального призначення, особливо придатна для веброзробки та вбудована в HTML. Вона широко використовується для створення динамічних вебсторінок і серверних застосунків [19].

PHP є мовою зі слабкою динамічною типізацією, що означає, що тип змінної визначається під час виконання програми, а не під час компіляції. Це дозволяє розробникам писати код швидше, але вимагає уважності для уникнення помилок, пов'язаних із типами даних.

Мова PHP є інтерпретованою, тобто код виконується безпосередньо сервером без попередньої компіляції. Це спрощує процес розробки та налагодження, оскільки зміни в коді можуть бути негайно протестовані.

PHP є реєстрозалежною мовою програмування, тобто розрізняє великі та малі літери в іменах змінних, функцій та інших ідентифікаторів. Це вимагає від розробників уважності при написанні коду, щоб уникнути помилок, пов'язаних із неправильним використанням регістру.

Існують стандарти кодування, які регламентують оформлення коду, використання синтаксичних конструкцій та інші аспекти програмування на PHP. Дотримання цих стандартів покращує читабельність коду та спрощує командну розробку.

PHP використовується для роботи з базами даних, обробки форм, управління сесіями, генерації динамічного вмісту та багатьох інших завдань у веброботі [19].

JavaScript — це динамічна, об'єктно-орієнтована мова програмування, яка широко використовується для створення інтерактивних вебсторінок та вебзастосунків. Вона є реалізацією стандарту ECMAScript і підтримує різні парадигми програмування, включаючи імперативну, функціональну та прототипно-орієнтовану. JS є мовою з динамічною типізацією, що означає, що тип змінної визначається під час виконання програми, а не на етапі компіляції. Це надає гнучкість у написанні коду, але вимагає уважності від розробника для уникнення помилок, пов'язаних з типами даних [20].

Основні сфери застосування JavaScript включають розробку інтерфейсної частини вебзастосунків (frontend): JS використовується для створення динамічних та інтерактивних елементів на вебсторінках, таких як анімації, валідація форм та обробка подій користувача. Також JavaScript застосовується для розробки серверної частини (backend) за допомогою платформи Node.js, що забезпечує можливість використовувати одну мову програмування як на клієнтській, так і на серверній стороні. JavaScript підходить для розробки мобільних застосунків завдяки фреймворкам, як-от React Native, які дозволяють створювати кросплатформні мобільні додатки. Використовуючи такі фреймворки, як Electron, розробники можуть створювати десктопні застосунки за допомогою вебтехнологій.

Серед переваг JavaScript можна виділити кросплатформеність: JS підтримується всіма сучасними веббраузерами. Важливою перевагою також є велика спільнота та розвинена екосистема: існує безліч бібліотек та фреймворків, які спрощують розробку та розширюють можливості мови. JS відзначається високою швидкістю виконання завдяки сучасним рушіям, як-от V8. Мова підтримує асинхронне програмування, що дозволяє ефективно обробляти операції вводу-виводу без блокування основного потоку виконання.

Недоліки JavaScript включають можливі проблеми з безпекою: через свою відкритість і широке використання, JS може бути вразливим до атак, таких як XSS (Cross-Site Scripting). Також динамічна типізація може призводити до помилок, пов'язаних з типами даних, які важко виявити на етапі розробки. Крім того, існують відмінності у реалізації JavaScript у різних браузерах, що може вимагати додаткового тестування та адаптації коду [20].

В таблиці 2.1 наведено підсумкове порівняння мов програмування для розробки вебзастосунку.

Таблиця 2.1 – Порівняння мов програмування для розробки вебзастосунків

Критерій порівняння	JS	C#	Python	PHP
Рішення різних завдань	++	++	++	++
Мультипарадигмальна мова	+	++	+	+
Інтерпретована мова	+	—	+	+
Динамічна типізація	+	—	+	+
Зручний синтаксис	+	+	+—	++
Реєстрозалежна мова	+	+	+	+
Високорівнева мова	+	+	+	+
Наявність менеджера пакетів	+	+	+	+
Прототип-орієнтований стиль	+	—	—	—
Велика кількість програмних каркасів для розробки вебзастосунків	++	+	++	++
Відносно менші витрати часу	+	+—	+	+—
Висока швидкість виконання	+	+	—	+

2.2 Вибір середовища розробки вебзастосунку

Середовищем для розробки вебзастосунку було обрано редактор коду JetBrains PhpStorm, оскільки обране середовище розробки є дуже потужним та має безліч інструментів саме для розробки на PHP.

Вибір відповідного середовища розробки є критичним етапом у створенні ефективного та продуктивного вебзастосунку. Інтегроване середовище розробки (IDE) забезпечує розробників інструментами для написання, тестування та відлагодження коду, що сприяє підвищенню якості та швидкості розробки.

Серед популярних IDE для веброзробки варто виділити:

- Visual Studio Code (VS Code) - це безкоштовний редактор коду від Microsoft, відомий своєю розширюваністю та підтримкою численних плагінів;
- Sublime Text легкий та швидкий редактор коду з підтримкою багатьох мов програмування та можливістю налаштування;
- IntelliJ IDEA потужне середовище розробки від JetBrains, яке підтримує широкий спектр мов та технологій.

Для розробки даного проєкту було обрано PhpStorm — спеціалізоване IDE для PHP від компанії JetBrains. PhpStorm пропонує інтелектуальний редактор коду з підсвіткою синтаксису, автодоповненням, інструментами для відлагодження та тестування, а також інтеграцію з системами контролю версій. Це середовище розробки забезпечує високу продуктивність та зручність роботи з PHP-кодом, що робить його оптимальним вибором для реалізації проєкту.

PhpStorm має потужні інструменти для налагодження та тестування, включаючи інтеграцію з Xdebug та PHPUnit. Це дозволяє розробникам швидко виявляти та виправляти помилки, тестуючи код у реальному часі. Інтеграція з системами керування версіями (Git, SVN, Mercurial) робить процес спільної розробки зручнішим і контрольованішим. Крім того, IDE підтримує роботу з базами даних, дозволяючи виконувати SQL-запити безпосередньо з середовища розробки [21].

В таблиці 2.2 наведено порівняння текстових редакторів для розробки вебзастосунку на пові PHP.

Таблиця 2.2 – Порівняння текстових редакторів для розробки вебзастосунку

Критерій порівняння	Visual Studio Code	PHP Storm
Кодування багатьма мовами	+	+
Перевірка правильності написання коду	+	+
Автоматична генерація каркаса проєкту або окремих фрагментів програмного коду	+	+
Вбудована підтримка Git	+	+
Вбудований налагоджувач для PHP	—	+
Вбудований інтерфейс бази даних	—	+
Встановлення розширень	—	+
Файловий менеджер	+	+
Діагностика помилок	+	+
Карта програмного коду	+	+
Розділення робочої області	+	+
Підтримка декількох курсорів	+	+
Вбудований інтерфейс командного рядка	+	+
Підтримка Emmet	+	+
Можливість роботи з базами даних	—	+
Високий рівень безпеки	—	+
Можливість роботи з декількома проєктами одночасно	+	+
Наявність командного рядка для швидкого виконання різних команд	+	+
Рефакторинг	+	+
Інтегрований термінал	+	+
Можливість тестування	+	+

PhpStorm пропонує високий рівень адаптивності. Розробники можуть налаштувати інтерфейс, гарячі клавіші та інші аспекти IDE відповідно до своїх потреб. Середовище також забезпечує інтеграцію з популярними інструментами розробки, такими як Composer, Docker і REST-клієнти, що спрощує управління проєктами та підвищує продуктивність.

Завдяки своїм можливостям PhpStorm дозволяє зосередитися на розробці, а не на технічних аспектах, що робить його ідеальним вибором для створення складних і масштабованих вебзастосунків.

Це середовище розробки було обране для реалізації цього проєкту через його гнучкість, потужність і широкий спектр інструментів, які відповідають потребам сучасної веброботи [21].

2.3 Вибір засобів розробки користувацької частини вебзастосунку

Для розробки інтерфейсної частини вебзастосунку було обрано використовувати наступні засоби:

- мову розмітки елементів вебдокументів HTML;
- мову опису стилів оформлення елементів вебдокументів CSS;
- фреймворк Vue 3 JS;
- бібліотеку компонентів інтерфейсу користувача Vuestic UI;
- бібліотеку маршрутизації вебсторінок Vue Router;
- бібліотеку керування станами вебзастосунку Vuex;
- фреймворк Bootstrap 5;
- бібліотека для створення таблиць Vue3 Easy Data Table.

2.3.1 Мова розмітки вебдокументів HTML

HTML (мова опису гіпертекстових вебдокументів, HyperText Markup Language) – це мова, що визначає структуру та вміст сторінки. Мова розмітки містить набір правил для структурування (розмітки) текстової інформації, додавання зображень, створення таблиць, форм, списків тощо.

Основою мови HTML є тег, який виступає елементом гіпертекстової розмітки. Деякі теги використовуються для створення структури HTML документа, а інші для форматування та створення елементів на вебсторінці. Документ, який розмічений за допомогою HTML, інтерпретується браузером, в

результаті чого користувач баче не вихідний код з елементами розмітки, а остаточний результат обробки, тобто вебсторінку [22].

2.3.2 Мова опису стилів CSS

CSS (Cascading Style Sheets) – це мова визначення правил стилізації елементів до вебдокументів HTML. CSS використовується для форматування вмісту вебсторінок та містить загальні властивості зовнішнього вигляду документа, тобто розмір шрифту, колір тексту, інтервал між рядками, розміщення зображень, окремих блоків та інших елементів. Формат розширення файлу .css призначений для того, щоб розділити опис логічної структури вебсторінки з розміткою від її зовнішнього вигляду. CSS дозволяє значно спростити розробку та підтримку вебсайтів завдяки своїм основним характеристикам [23].

2.3.3 Фреймворк Vue 3 JS

Vue 3 — це прогресивний фреймворк для створення користувацьких інтерфейсів, який відрізняється простотою та гнучкістю. Він заснований на компонентній архітектурі, що дозволяє розбивати додаток на незалежні, багаторазові блоки коду — компоненти. Кожен компонент у Vue 3 є ізольованим модулем, який може містити свою власну логіку, шаблон і стилі. Це робить Vue 3 ідеальним для побудови як невеликих інтерактивних елементів, так і масштабованих вебзастосунків [24].

Компоненти у Vue 3 зазвичай складаються з трьох основних частин: шаблон, логіка та стилі. Шаблон визначає, як компонент буде виглядати у DOM, використовуючи синтаксис, схожий на HTML. Важливою особливістю є можливість використання директив, таких як v-if, v-for, v-bind, які дозволяють додавати динамічність до шаблону. Шаблон завжди повинен бути вкладеним у тег <template>.

Логіка компонента описується у секції `<script>`. Тут визначаються дані, методи, властивості, життєві цикли і реактивність. У Vue 3 використовується композиційний API, який дозволяє організувати логіку компонента в більш структурований та гнучкий спосіб. Замість використання великих об'єктів, як у Vue 2, у Vue 3 логіка компонента організована за допомогою функції `setup`, яка дозволяє працювати з реактивними змінними, методами та іншими функціями.

Стили компонента визначаються у секції `<style>`, де можна додати як глобальні, так і локальні стилі. Локальні стилі досягаються за допомогою атрибута `scoped`, який обмежує застосування стилів лише до поточного компонента. Це дозволяє уникнути конфліктів стилів між різними компонентами.

Щоб створити компонент, необхідно зареєструвати його у батьківському компоненті або глобально. У Vue 3 є можливість автоматичної реєстрації компонентів, що спрощує роботу з великими проєктами. Компоненти можуть взаємодіяти між собою через властивості (`props`), які передають дані від батьківського компонента до дочірнього, і події (`events`), які дозволяють дочірньому компоненту повідомляти батьківський про зміну стану [24].

Однією з важливих новацій у Vue 3 є використання Composition API, який дозволяє працювати з логікою та станом компонента в більш гнучкий спосіб. Це особливо корисно для масштабних проєктів, де потрібно повторно використовувати логіку в різних компонентах. Крім того, Vue 3 пропонує оптимізовану продуктивність завдяки покращенню механізму відстеження змін у DOM і більш ефективній роботі з реактивністю.

Vue 3 також підтримує реактивні об'єкти та реактивні змінні через функції `reactive` та `ref`. Ці функції дозволяють створювати реактивні дані, які автоматично оновлюють шаблон при зміні значення. Наприклад, `ref` використовується для створення простих реактивних змінних, а `reactive` — для роботи зі складними об'єктами.

Компоненти Vue 3 можуть використовувати слоти, які дозволяють передавати довільний контент у шаблон компонента. Це корисно для створення

гнучких і багаторазових компонентів, які можуть приймати вміст від батьківського компонента. Крім того, Vue 3 підтримує динамічні компоненти, які дозволяють перемикаати компоненти на основі змінних або умов.

Vue 3 продовжує розвивати можливості для роботи з великими проєктами, пропонуючи підтримку модульної структури, TypeScript і інструментів для тестування. Завдяки своїй простоті, гнучкості та потужності Vue 3 став одним із найпопулярніших фреймворків для сучасної веброзробки [25].

Порівняння інструментів розробки інтерфейсу користувача наведено в таблиці 2.3.

Таблиця 2.3 – Порівняння інструментів для розробки інтерфейсу користувача

Критерій порівняння	React	Vue	Angular
Мова програмування JS	+	+	—
Бібліотека	+	—	—
Віртуальна модель DOM	+	+	—
Двостороннє зв'язування даних	—	+	+
Швидке освоєння інструменту	+	++	—
Компонентний підхід розробки	+	+	+
Наявність вбудованого інструменту для маршрутизації вебсторінок	—	+	+
Наявність вбудованого інструменту для керування станами вебзастосунку	—	+	+
Розробка середніх вебпроєктів	+	+—	++
Наявність документації	+	+	+
Вбудовані стилі компонентів	+	—	—
Робота з REST API	+	+	+
Висока швидкість кодування	+	++	—
Розширена екосистема	+	+	+

2.3.4 Бібліотека компонентів Vuestic UI

Vuestic UI — це безкоштовна та відкрита бібліотека компонентів для Vue

3, розроблена компанією Eristax. Вона пропонує понад 50 готових до використання компонентів, які легко налаштовуються та адаптуються до різних дизайнів. Серед ключових особливостей Vuestic UI — підтримка темної теми, глобальна конфігурація, адаптивний дизайн та інтеграція з i18n для спрощення локалізації додатків. Бібліотека розповсюджується за ліцензією MIT, що дозволяє вільно використовувати її у різних проєктах.

2.3.5 Бібліотека для створення таблиць Vue 3 Easy Data Table

Vue 3 Easy Data Table — це компонент для Vue.js 3.x, розроблений для відображення та управління даними у вигляді таблиць. Він забезпечує простоту інтеграції та підтримує широкий спектр функцій, включаючи сортування, пагінацію, пошук і налаштування через слоти. Компонент дозволяє розробникам створювати адаптивні таблиці з можливістю кастомізації вигляду окремих комірок або рядків. Також підтримується серверна пагінація та сортування, що дозволяє працювати з великими обсягами даних.

Easy Data Table пропонує зручні опції для стилізації таблиць, кастомізації порожніх станів та елементів завантаження. Для початку роботи необхідно встановити пакет через `npm` або `yarn`, після чого компонент можна легко підключити до проєкту та використовувати у шаблоні. Завдяки простоті використання та багатофункціональності, Vue 3 Easy Data Table ідеально підходить для створення зручних і ефективних інтерфейсів для управління даними.

2.3.6 Бібліотека маршрутизації вебсторінок Vue Router

Vue Router 4 — офіційний маршрутизатор для Vue.js 3, створений для роботи з односторінковими додатками (SPA). Він дозволяє розробникам пов'язувати URL з компонентами, забезпечуючи плавну навігацію між

частинами додатка без перезавантаження сторінки. Це робить Vue Router ключовим інструментом у створенні сучасних вебзастосунків [24].

Основою роботи Vue Router є маршрути, які мапують URL на компоненти. Кожен маршрут визначається об'єктом, що містить шлях (path), ім'я (name) і відповідний компонент (component). Це дозволяє легко структурувати додаток, організовувати навігацію і динамічно змінювати контент залежно від URL.

Vue Router підтримує динамічні маршрути, які дозволяють використовувати змінні у шляху, наприклад, /users/:id. Це корисно для створення сторінок із персоналізованим контентом, як-от профілі користувачів або картки товарів. Маршрути можуть бути вкладеними, що полегшує побудову складних ієрархій компонентів.

Для забезпечення безпеки та контролю доступу Vue Router пропонує навігаційні охоронці. Вони дозволяють перевіряти умови перед переходом на маршрут, наприклад, перевірку автентифікації. Це корисно для захисту конфіденційного контенту.

Програмна навігація в Vue Router реалізується через методи `router.push()` та `router.replace()`. Вони дозволяють динамічно змінювати маршрути залежно від дій користувача чи стану додатка. Це дає змогу створювати інтерактивні та динамічні інтерфейси [25].

Vue Router підтримує кілька режимів історії: хеш-навігацію та історію HTML5. Хеш-навігація додає символ # до URL, що підходить для проєктів без серверної підтримки маршрутизації. Режим HTML5 дозволяє використовувати "чисті" URL, які виглядають професійніше, але вимагають налаштування на сервері.

Інтеграція Vue Router у проєкт відбувається шляхом встановлення пакету через `npm` або `yarn`. Після цього потрібно створити екземпляр маршрутизатора з конфігурацією маршрутів і передати його Vue-додатку. Це забезпечує централізоване управління всією навігацією додатка.

Vue Router також дозволяє використовувати слоти для кастомізації посилань, наприклад, компонент `<router-link>`. Це дає змогу створювати індивідуальний дизайн посилань і зберігати зручність навігації. Завдяки такій гнучкості Vue Router підходить для будь-якого проєкту.

Цей маршрутизатор підтримує серверний рендеринг, що дозволяє покращити SEO для SPA-додатків. Серверний рендеринг також покращує швидкість завантаження сторінок, що позитивно впливає на досвід користувачів і оптимізацію для пошукових систем [25].

2.3.7 Менеджер керування станами Vuex

Vuex — це офіційна бібліотека для управління станом у додатках, створених на основі Vue.js. Вона забезпечує централізоване сховище для всіх компонентів додатка, з чіткими правилами, що гарантують передбачувані зміни стану [25].

Основна концепція Vuex полягає в тому, що стан додатка зберігається в одному місці — сховищі (store). Це дозволяє будь-якому компоненту отримувати доступ до необхідних даних без потреби передавати їх через вкладеність компонентів. Такий підхід спрощує управління станом у великих додатках та забезпечує кращу підтримку коду.

Сховище Vuex складається з кількох ключових елементів [24]:

- State (стан): об'єкт, що містить дані додатка;
- Getters (гетери): функції, які обчислюють похідні дані на основі стану;
- Mutations (мутації): єдиний спосіб змінити стан. Мутації завжди синхронні;
- Actions (дії): функції, що можуть містити асинхронний код і викликають мутації;
- Modules (модулі): дозволяють розбивати сховище на менші частини для кращої організації коду.

Використання Vuex особливо корисне в додатках зі складною логікою та великою кількістю компонентів, де управління станом може стати викликом. Однак, для невеликих проєктів, де обмін даними між компонентами мінімальний, використання Vuex може бути зайвим.

Варто зазначити, що з появою Vue 3 офіційна команда Vue.js рекомендує використовувати Pinia як новий стандарт для управління станом. Pinia пропонує схожий або покращений API порівняно з Vuex 5 і працює як з Vue 2.x, так і з Vue 3.x. Хоча Vuex 3 та 4 все ще підтримуються, нові функції додаються переважно в Pinia.

Таким чином, Vuex залишається потужним інструментом для управління станом у Vue-додатках, але розробникам рекомендується розглянути можливість переходу на Pinia для нових проєктів або поступової міграції існуючих додатків.

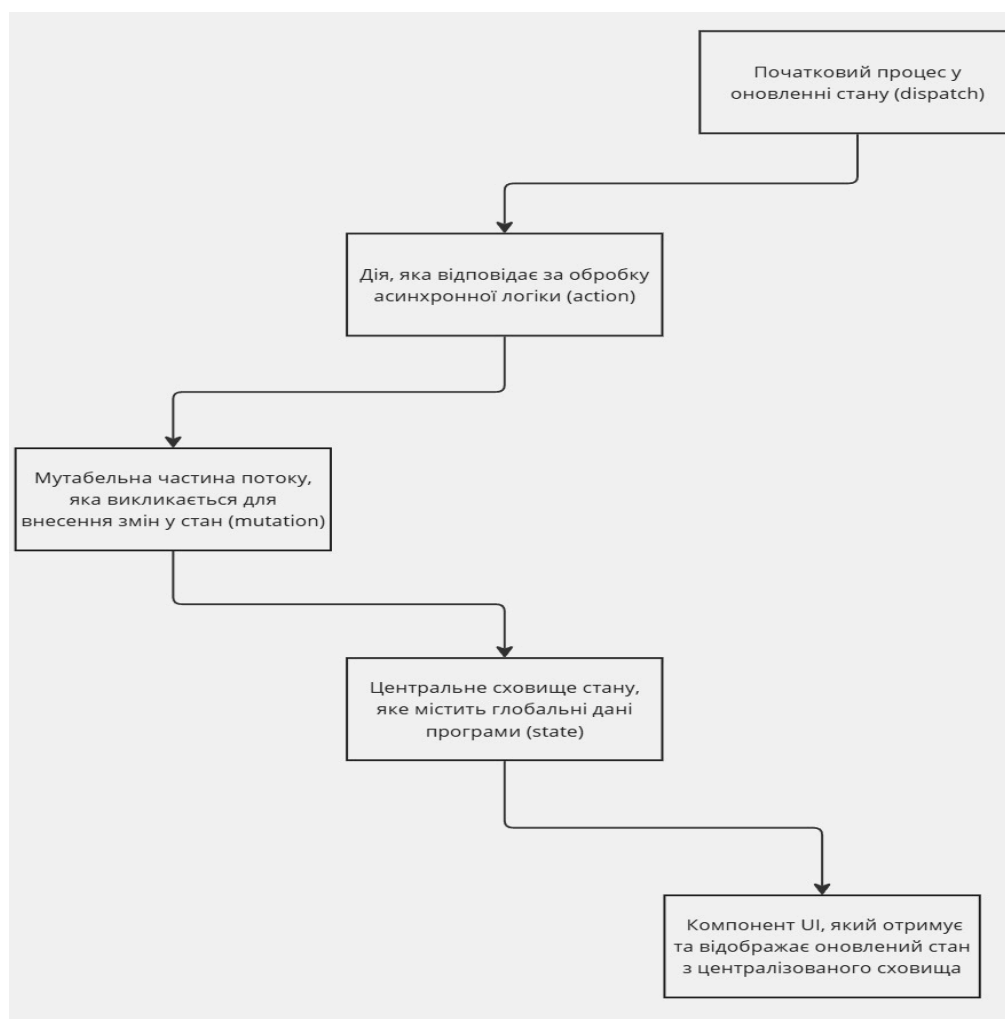


Рисунок 2.1 – Потоки даних в менеджері станів Vuex

2.4 Вибір засобів розробки серверної частини вебзастосунку

Для розробки серверної частини вебзастосунку було обрано використовувати наступні інструменти:

- фреймворк Laravel 10;
- сервер Apache 2.4;
- СКБД MariaDB.

Використання REST API дозволяє взаємодіяти клієнту з сервером використовуючи HTTP методи, тобто CRUD операції. До CRUD операцій належать чотири методи для взаємодії з ресурсами REST API, такі як створення нового ресурсу, отримання певного ресурсу або набору ресурсів, оновлення ресурсу та видалення ресурсу.

2.4.1 Фреймворк Laravel

Laravel — це популярний PHP-фреймворк з відкритим кодом, призначений для розробки веб-додатків за архітектурним шаблоном Model-View-Controller (MVC). Він надає розробникам інструменти для швидкої та ефективної побудови вебзастосунків, забезпечуючи високу продуктивність, безпеку та масштабованість [26].

Однією з ключових особливостей Laravel є його зручна структура проєкту, яка дотримується принципів MVC. Це дозволяє чітко розділити бізнес-логіку, інтерфейс користувача та взаємодію з базою даних, що спрощує підтримку та розширення додатка.

Фреймворк пропонує потужну систему маршрутизації, яка дозволяє визначати прості та складні маршрути для обробки HTTP-запитів. Це забезпечує гнучкість у побудові RESTful API та спрощує інтеграцію з іншими сервісами.

Eloquent ORM, вбудований в Laravel, забезпечує зручний спосіб взаємодії з базами даних за допомогою PHP-об'єктів. Він абстрагує деталі запитів SQL,

дозволяючи працювати з даними на більш високому рівні абстракції, що прискорює розробку та зменшує кількість помилок.

Шаблонізатор Blade, інтегрований у Laravel, надає зручний та виразний синтаксис для створення динамічних сторінок. Він дозволяє легко вставляти PHP-код безпосередньо в HTML-розмітку, використовуючи спеціальні директиви, що спрощує розробку інтерфейсу користувача.

Laravel також пропонує вбудовані інструменти для аутентифікації та авторизації користувачів, що забезпечує високий рівень безпеки додатків. Це особливо важливо при розробці CRM-систем, де необхідно контролювати доступ до конфіденційної інформації.

Фреймворк підтримує планування завдань за допомогою вбудованого планувальника, що дозволяє автоматизувати різні процеси, такі як відправка нагадувань клієнтам або створення регулярних звітів. Це підвищує ефективність роботи CRM-системи та зменшує навантаження на персонал.

Завдяки підтримці реального часу, Laravel дозволяє реалізувати функції, такі як оновлення інформації на дашбордах або повідомлення у чатах, що покращує взаємодію з користувачами та підвищує їх задоволеність.

Використання Laravel для розробки CRM-систем забезпечує високу продуктивність, безпеку та масштабованість, що є критично важливим для успішного управління відносинами з клієнтами та ефективної роботи бізнесу [26].

Laravel пропонує можливості для побудови інтерактивних панелей управління та дашбордів, що є основою сучасних CRM-систем. Завдяки функціоналу Blade та підтримці таких бібліотек, як Chart.js або D3.js, розробники можуть створювати інтерактивні графіки, таблиці та звіти. Це дозволяє користувачам аналізувати ключові показники бізнесу в режимі реального часу, приймаючи обґрунтовані рішення.

Ще однією перевагою Laravel є його підтримка модульного підходу. Розробники можуть розділяти CRM-систему на функціональні модулі, наприклад, управління клієнтами, продажами чи фінансами. Це забезпечує

легкість додавання нових функцій, зменшує складність коду та покращує його підтримку. Такий підхід також сприяє масштабуванню додатка з розвитком бізнесу.

Підтримка командної роботи є важливим аспектом у розробці складних систем, таких як CRM. Laravel інтегрується з Git та іншими системами контролю версій, що дозволяє ефективно керувати спільною роботою розробників. Інструменти, такі як Forge і Envoyer, допомагають автоматизувати процеси розгортання, оновлення та резервного копіювання системи.

Laravel надає можливість легко розробляти мультитенантні CRM-системи, які підтримують кілька організацій або відділів у межах однієї інфраструктури. Завдяки потужній системі конфігурації та ізоляції даних кожен клієнт отримує персоналізований досвід роботи. Це особливо важливо для SaaS-рішень, де кожен користувач має свій ізольований набір даних і функціонал.

Що стосується продуктивності, Laravel підтримує масштабовані архітектури, які дозволяють додавати нові сервери для обробки зростаючого навантаження. Використання черг, кешування та оптимізованої обробки запитів допомагає забезпечити високу швидкість роботи навіть при обробці великих обсягів даних. Це робить Laravel придатним для розробки CRM, які обслуговують тисячі або навіть мільйони користувачів [27].

Одним із ключових аспектів Laravel є його розширювана екосистема. Завдяки великій кількості пакетів і бібліотек, доступних через Composer, розробники можуть інтегрувати додатковий функціонал у свої проєкти. Наприклад, пакети для роботи з календарями, інтерфейсами API або генерації PDF дозволяють значно розширити можливості CRM-системи без значних витрат часу.

Laravel також забезпечує високий рівень автоматизованого тестування. Його вбудовані інструменти дозволяють проводити як юніт-, так і функціональне тестування, що знижує ризики помилок при внесенні змін до системи. Це особливо важливо для CRM, де кожен збій може негативно вплинути на бізнес-процеси клієнтів [26].

Підтримка хмарних сервісів є ще однією перевагою Laravel. Завдяки простій інтеграції з Amazon S3, Google Cloud або Azure, CRM-системи, створені на Laravel, можуть зберігати дані у хмарі, забезпечуючи високу доступність і надійність.

Таким чином, Laravel є не лише ефективним фреймворком для створення CRM-систем, але й інструментом, який дозволяє будувати сучасні, масштабовані та багатофункціональні рішення. Його інструменти, архітектура та гнучкість роблять його ідеальним вибором для розробників, які прагнуть створити надійний продукт для управління бізнесом [27].

2.5 Вибір СКБД

MariaDB — це реляційна система керування базами даних (РСКБД) з відкритим вихідним кодом, створена як форк MySQL у 2009 році. Її розробка розпочалася через занепокоєння спільноти щодо майбутнього MySQL після придбання компанією Oracle. MariaDB поширюється під ліцензією GNU General Public License (GPL) і підтримується спільнотою розробників та організацією MariaDB Foundation.

Однією з ключових особливостей MariaDB є її сумісність з MySQL. Більшість команд, API та протоколів MariaDB відповідають MySQL, що дозволяє використовувати її як заміну без значних змін у коді додатків. Проте MariaDB пропонує додаткові функції та покращення, такі як нові сховища даних (Aria, ColumnStore, MyRocks), покращена продуктивність та розширені можливості реплікації [28].

MariaDB активно розвивається та підтримується спільнотою. Вона регулярно випускає оновлення, що включають нові функції, виправлення помилок та покращення безпеки. Це забезпечує стабільність та надійність системи, що є важливим для критично важливих застосунків [29].

Завдяки своїй відкритості та активній підтримці, MariaDB стала популярною серед розробників та компаній, які шукають надійну та

масштабовану РСКБД. Вона використовується в різних галузях, від невеликих проєктів до великих корпоративних систем, забезпечуючи ефективне управління даними та високу продуктивність [29].

Порівняння СКБД для розробки CRM системи для тваринницької ферми наведено в таблиці 2.5.

Таблиця 2.5 – Порівняння СКБД

Критерій порівняння	MariaDB	PostgreSQL	MySQL
Сумісність з MySQL	++	—	++
Підтримка JSON	+	++	+
Реплікація	++	++	++
Масштабованість	++	++	+
Агрегація даних	++	++	+
Підтримка складних запитів (CTE)	+	++	—
Розробка високонавантажених систем	++	++	+
Швидкість виконання запитів	++	++	++
Мультиплатформність	+	+	+
Простота використання	++	+	++
Широкий вибір механізмів зберігання	++	+	+
Розширені можливості безпеки	+	++	+
Розширюваність через плагіни	+	++	+
Підтримка геопросторових даних (GIS)	++	++	+
Ліцензування	Open Source	Open Source	Open Source
Активність спільноти розробників	++	++	++

2.6 Висновки до другого розділу

Провівши огляд особливостей обраних інструментів для програмної реалізації вебзастосунку та порівнявши з іншими існуючими засобами, було обрано використовувати при розробці наступні засоби:

- клієнтську мову програмування JS;

- серверну мову програмування PHP;
- редактор коду PHP Storm;
- мову розмітки елементів вебсайтів HTML;
- мову опису стилів оформлення елементів вебсайтів CSS;
- фреймворк Vue 3 JS;
- бібліотеку компонентів інтерфейсу користувача Vuestic UI;
- бібліотеку маршрутизації вебсторінок Vue Router;
- бібліотеку керування станами вебзастосунку Vuex;
- фреймворк Laravel 10;
- реляційну СКБД MongoDB.

У пунктах розділу було описано особливості та переваги обраних засобів для розробки інтерфейсної та серверної частини вебзастосунку CRM-системи для тваринницьких ферм та вирішення логістичних завдань.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ПРОДУКТУ

У розділі наведені діаграми варіантів використання вебзастосунку, які моделюють бізнес-процеси, тобто описують функціональні вимоги до вебзастосунку. Наведено структуру навігації вебзастосунку та схему обраної програмної архітектури. Також у розділі розглянуто основні архітектурні моменти та описано переваги обраного архітектурного рішення для реалізації вебзастосунку.

3.1 Проєктування сценаріїв використання вебзастосунку

Провівши дослідження аналогічних вебзастосунків для керування бізнес-процесами у CRM-системі для тваринницьких ферм було сформовано функціональні вимоги, які повинен надавати розроблювальний вебзастосунок. Користувачами у розроблювальному вебзастосунку є власник ферми, ветеринар, лаборант ветеринарної клініки, або інституту, адміністратор та менеджер системи. Нижче перераховані сформовані функціональні вимоги для користувачів вебзастосунку:

- аутентифікація та авторизація користувача за допомогою Google OAuth 2.0;
- погодження з умовами користування та політикою конфіденційності;
- отримання ролі;
- створення компанії або приватної особи для реєстрації ферми;
- додавання тварин на ферму;
- подання заявки на перевезення тварин;
- подання заявки на отримання номеру, який дає можливість додавати на ферму буйволів;
- додавання результатів тестування своїх тварин для заповнення історичних даних;

- збереження файлів, які містять результати тестування та загальну інформацію про ізоляцію;
- отримання повного списку своїх тварин та спеціальних тегів цих тварин;
- пошук по списку тварин;
- пошук по списку ізоляцій, які доступні;
- пошук по списку заявок на перевезення тварин;
- реєстрація нових ділянок та ферм за допомогою мапи;
- додавання учасників компанії, лабораторії;
- створення ролей для учасників компанії, лабораторії;
- додавання дозволів для компаній, лабораторій;
- створення ролей для користувачів;
- список усіх користувачів;
- бан користувачів;
- додавання ролей до користувачів;
- редагування ролей та додавання дозволів;
- список усіх тварин;
- пошук по всім тваринам;
- родове дерево для тварин;
- історія власників тварин;
- API для отримання даних про ферму або тварину;
- створення місцевих ветеринарів;
- додавання зон відповідальності до ветеринарів;
- створення сафарі тегів для створення фізичних пломб з кодом;
- створення параметрів для тварин (вага, зріст і тп.);
- створення медикаментів для вакцинування тварин;
- створення протоколів для тестування тварин на хвороби;
- створення заявок Червоного Хреста на дозвіл перевезення тварин.

Діаграму взаємодії спеціаліста із вебзастосунком для Схема взаємодії спеціаліста з вебзастосунком для тваринництва зображено на рисунку 3.1.

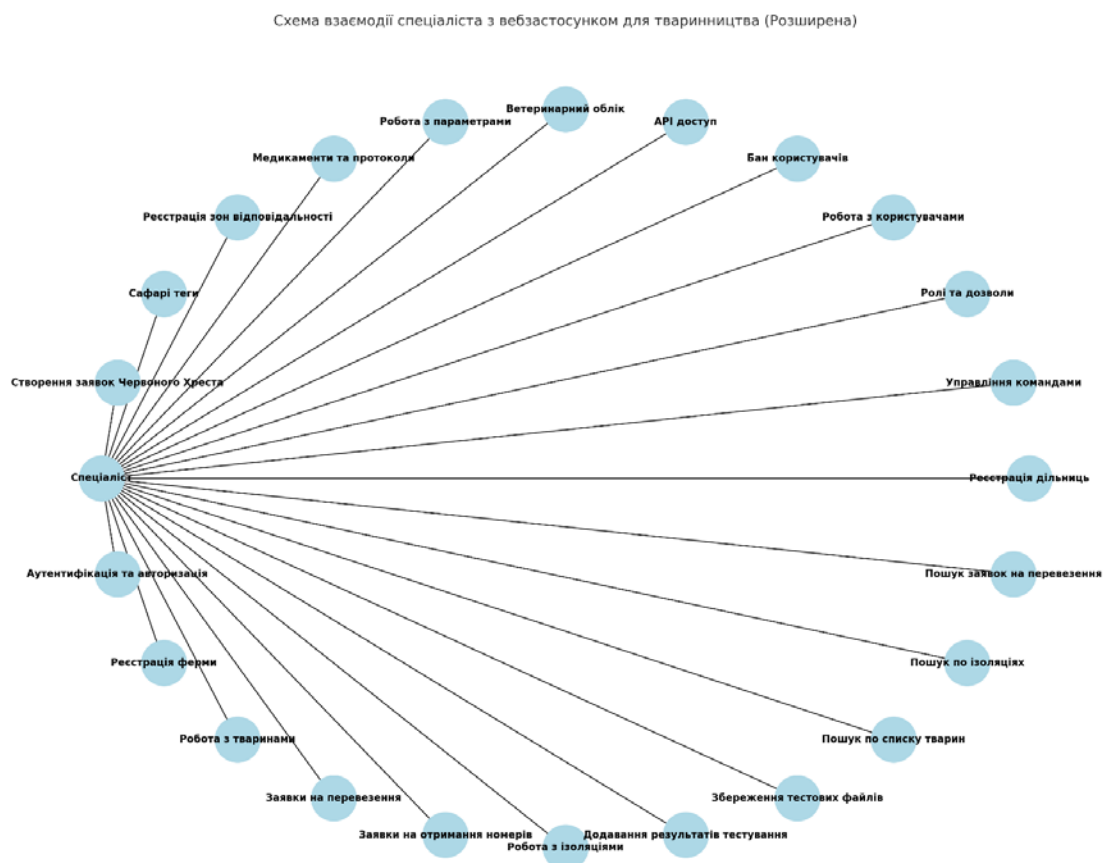


Рисунок 3.1 – Схема взаємодії спеціаліста з вебзастосунком для тваринництва

3.2 Проектування структури навігації вебзастосунку

У CRM-системі для тваринницьких ферм буде розроблене навігаційне меню, яке дозволить користувачам переходити між розділами та обирати необхідну вебсторінку. CRM-система для тваринницьких ферм буде мати такі розділи як «Інструменти адміністратора», «Налаштування», «Компанії», «Вет послуги», «Ферми», «Лабораторії», «Статистика», «Хвороби», «Протоколи хвороб», «Ізоляції», «Перевезення», «Тварини», «Теги», «Сафарі Теги», «Мапа ферм», «Ролі», «Користувачі» та «Дозволи Червоного Хреста».

Спроектована структура навігації вебзастосунку зображена на рисунку 3.2.

Схема взаємодії користувачів з CRM-системою для тваринницьких ферм

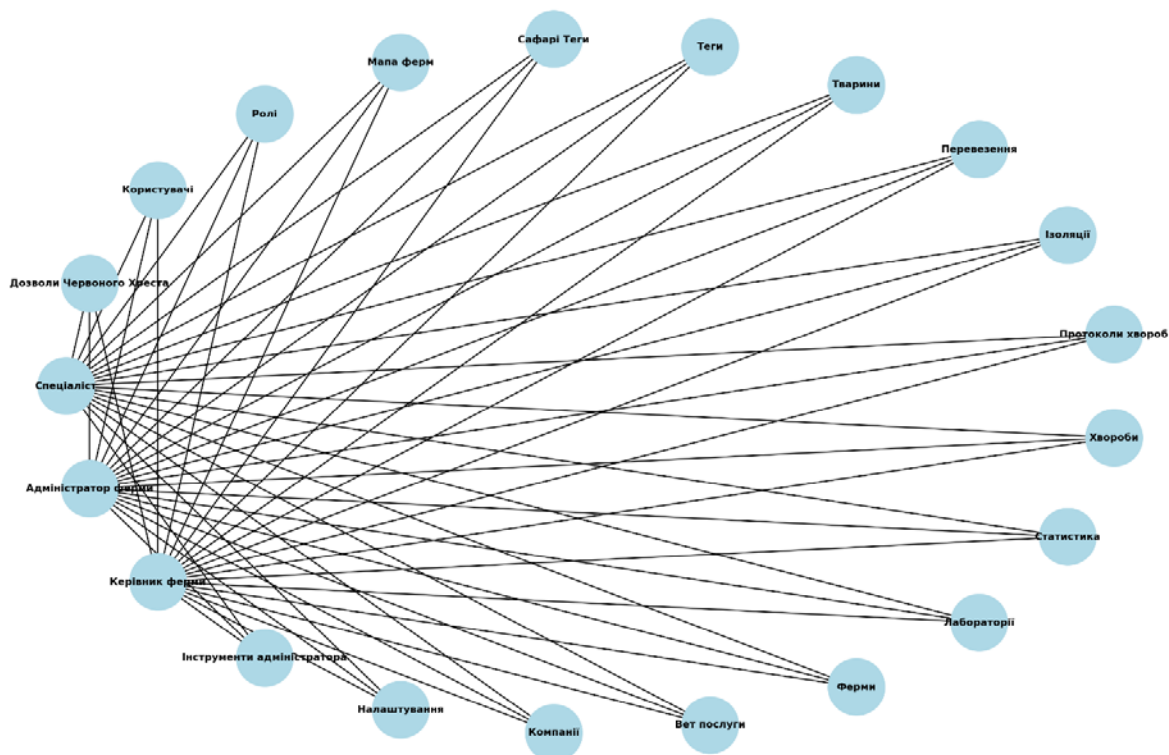


Рисунок 3.2 – Структура навігації вебзастосунку

3.3 Проєктування архітектури вебзастосунку

Архітектурним рішенням для розробки вебзастосунку для керування бізнес-процесами у CRM-системі було обрано використовувати патерн MVC та підхід Сервери та Репозиторії. Архітектурне рішення MVC дозволяє ділити вебзастосунок на три рівні [30]:

- контролер (сервер);
- представлення (інтерфейс);
- модель (завдання пов'язані з базою даних).

На рисунку 3.5 наведено схему архітектурного рішення MVC в розробленому застосунку.

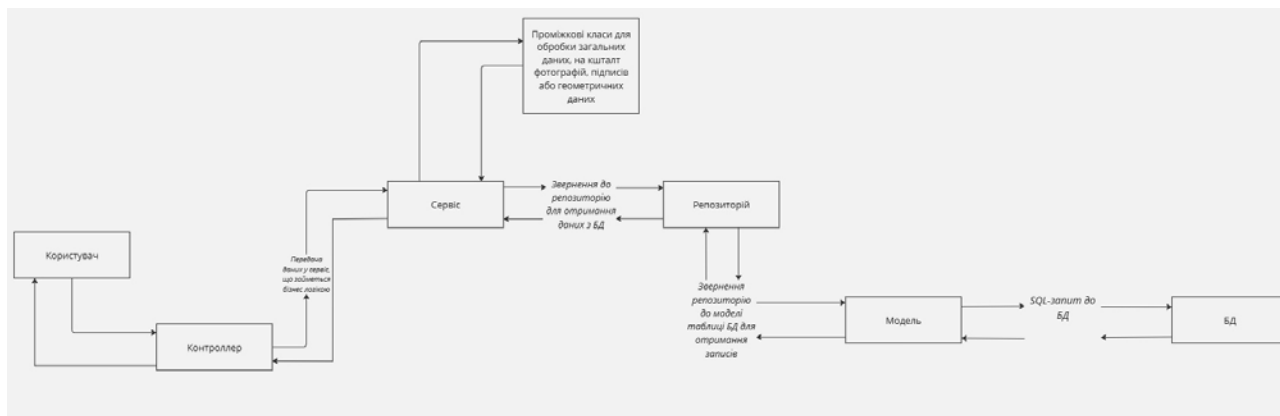


Рисунок 3.3 – Архітектурний шаблон MVC розробленого застосунку

Архітектурний патерн Model-View-Controller (MVC) широко використовується в розробці програмного забезпечення для розділення додатка на три основні компоненти: Модель, Представлення та Контролер. Цей підхід сприяє кращій організації коду, полегшує його підтримку та тестування.

Модель (Model) відповідає за управління даними та бізнес-логікою додатка. Вона взаємодіє з базою даних або іншими джерелами даних, обробляє інформацію та надає її іншим компонентам.

Представлення (View) відповідає за відображення даних користувачу. Воно отримує інформацію від Моделі та відображає її у зручному для користувача форматі, забезпечуючи інтерфейс взаємодії.

Контролер (Controller) обробляє вхідні запити від користувача, взаємодіє з Моделлю для отримання або оновлення даних та визначає, яке Представлення слід відобразити. Таким чином, Контролер виступає посередником між Моделлю та Представленням [30].

У контексті MVC часто використовуються додаткові патерни, такі як Сервіси (Services) та Репозиторії (Repositories), для покращення структури та розділення відповідальностей у додатку.

Сервіси реалізують бізнес-логіку додатка. Вони містять методи, які виконують специфічні операції, пов'язані з бізнес-процесами, та можуть викликатися Контролерами або іншими компонентами. Використання сервісів дозволяє ізолювати бізнес-логіку від деталей реалізації, що сприяє кращій підтримуваності та тестованості коду [31].

Репозиторії відповідають за доступ до даних. Вони надають абстракцію над джерелами даних, такими як бази даних, та забезпечують методи для отримання, збереження та оновлення інформації. Використання репозиторіїв дозволяє ізолювати логіку доступу до даних від бізнес-логіки, що полегшує зміну джерел даних без впливу на інші компоненти додатка [32].

Поєднання MVC з патернами Сервісів та Репозиторіїв сприяє створенню модульних, масштабованих та легко підтримуваних додатків. Цей підхід дозволяє чітко розділити відповідальності між компонентами, що спрощує розробку, тестування та подальшу еволюцію програмного забезпечення.

3.4 Висновки до третього розділу

У пунктах даного розділу було описано функціональні вимоги для кожного користувача та приведено набір діаграм сценаріїв використання вебзастосунку для керування бізнес-процесами у CRM-системі для тавринницьких ферм. Наведені діаграми відображають бізнес-процеси у розроблювальному вебзастосунку.

Спроектовано структуру навігації розроблювального вебзастосунку, яка дозволить користувачам переходити між розділами вебзастосунку. Також було оглянуто особливості обраного архітектурного рішення та його переваги.

4 КОНСТРУЮВАННЯ ПРОГРАМНОГО ПРОДУКТУ

У розділі приведено фізичну модель розробленої бази даних у СКБД MariaDB та наведено сутності розроблених колекцій. Наведено розроблений алгоритм функціонування вебзастосунку для керування CRM-системою для тваринницьких ферм. Описано розроблені модулі, функції розроблених частин вебзастосунку та наведено опис структуризації файлової системи, тобто розроблених директорій вебзастосунку.

4.1 Фізична модель реалізованої бази даних

Дані CRM-системи для тваринницьких ферм зберігаються у СКБД MariaDB. Виконавши дослідження існуючих вебзастосунків було виокремлено сутності для розробки колекцій:

- користувачі (users);
- персони (persons);
- ролі (roles);
- ферми (farms);
- ізоляції (isolations);
- тварини (animals);
- протоколи (testing protocols);
- перевезення (movements);
- країни (countries);
- адреси (addresses);
- медикаменти (medicines);
- підвиди (species);
- історія тварин (animals_history);
- родове дерево (family tree);
- тести (tests);
- хвороби (diseases);

- користувачі телеграм (telegram users).

Схему кількох таблиць MariaDB зображено на рисунку 4.1.

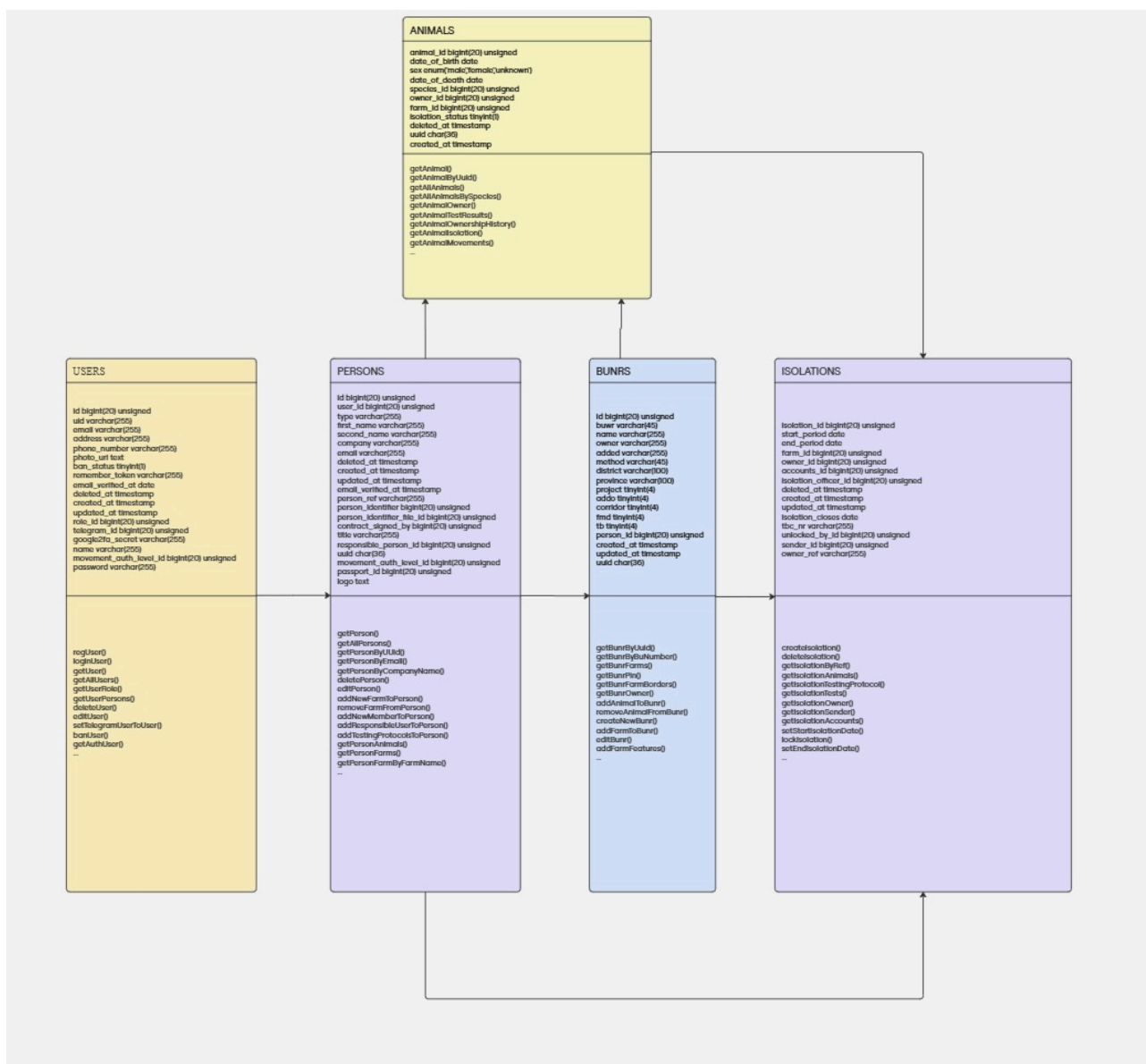


Рисунок 4.1 – Фізична модель відношень між користувачем, компанією, тваринами, фермами та ізоляціями

4.2 Алгоритм функціонування розробленого вебзастосунку

Алгоритм функціонування CRM-системи для тваринницьких ферм складається з входу користувача у систему та в залежності, хто цей користувач, будуть відбуватись такі кроки:

- реєстрація приватної особи, компанії чи лабораторії;

- додавання ферм;
- додавання тварин;
- створювання ізоляцій, або робота з вже існуючими;
- додавання відповідних до протоколу тестування тестів до ізоляції;
- заповнення ізоляції результатами та додавання файлів-висновків;
- створення заявки на перевезення тварин;
- створення дозволу Червоного Хреста.

Алгоритм виконання перевезення тварин зображений на рисунку 4.2. Весь ланцюг алгоритму вебзастосунку наведений на рисунку 4.3.

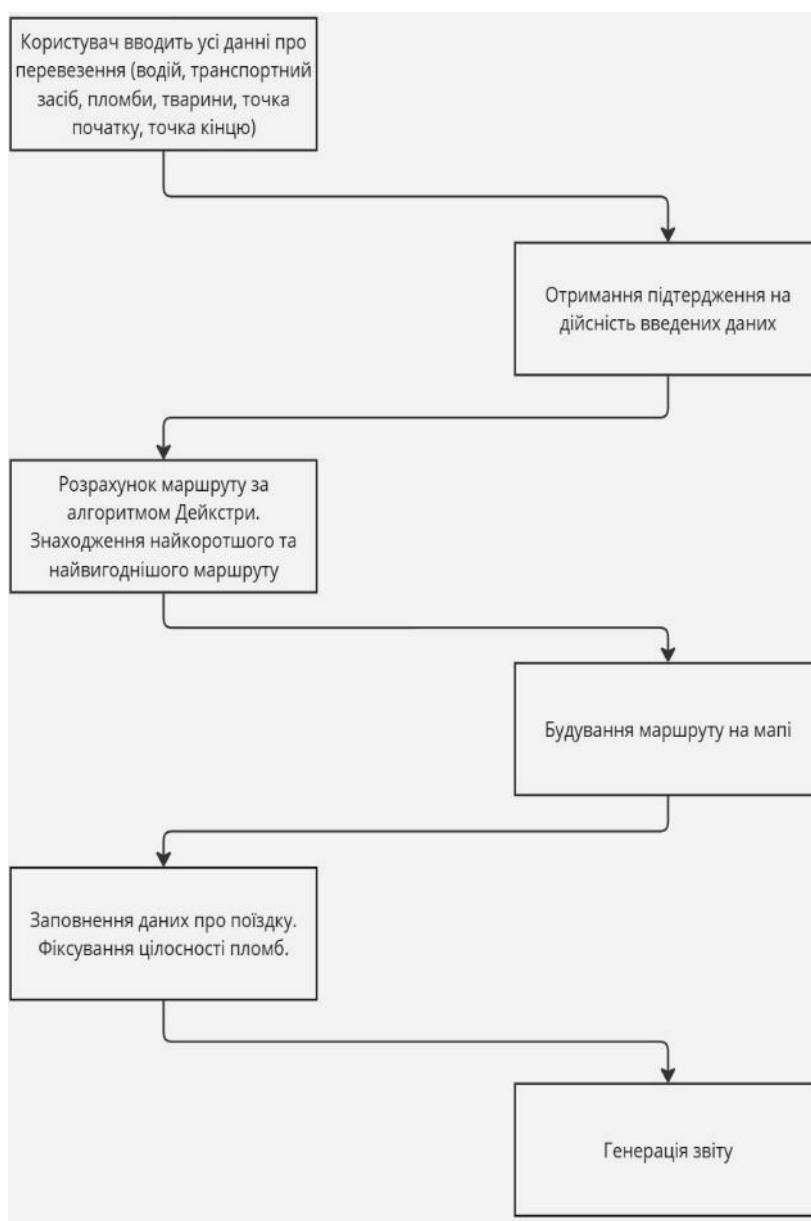


Рисунок 4.2 – Алгоритм виконання перевезення тварин

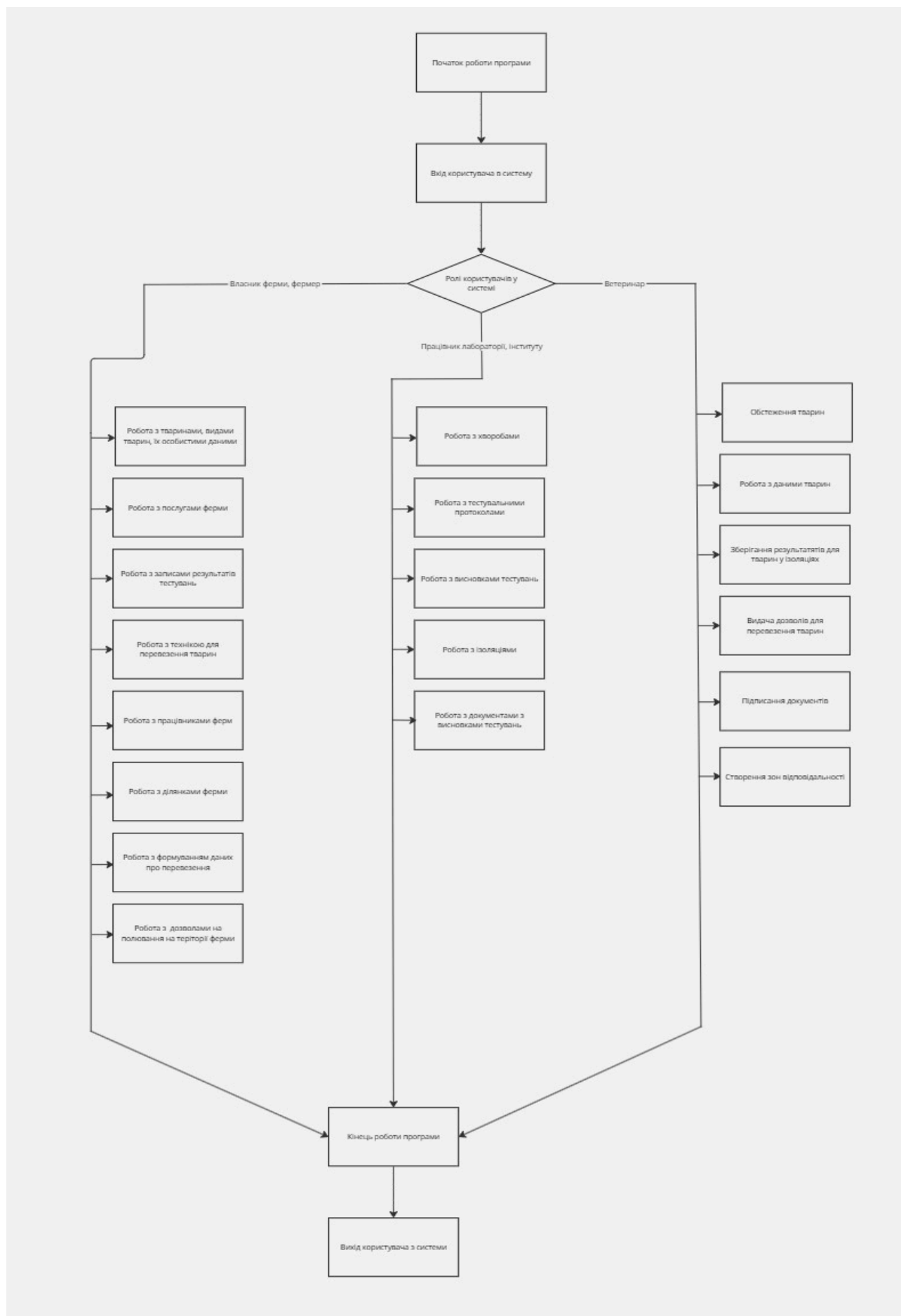


Рисунок 4.3 – Розроблений алгоритм функціонування вебзастосунку

4.3 Розроблені модулі вебзастосунку

Були розроблені наступні модулі у CRM-системі для тваринницьких ферм:

- модуль аутентифікації та авторизації;
- модуль відображення та роботи зі списком клієнтів;
- модуль відображення та роботи з тваринами;
- модуль відображення та роботи зі хворобами;
- модуль відображення та роботи зі тестами;
- модуль відображення та роботи з протоколами тестування;
- модуль відображення та роботи з мікрочіпами тварин;
- модуль налаштування переліку можливостей користувача;
- модуль відображення та роботи з ізоляціями тварин;
- модуль відображення та роботи з заявками на перевезення тварин;
- модуль відображення та роботи з дозволами Червоного Хреста;
- модуль відображення та роботи з дозволами на полювання;
- модуль відображення та роботи з дозволами на реєстрацію номера ферми з буйволами;
- модуль відображення та роботи з сафарі тегами;
- модуль відображення та роботи з родовим деревом тварини;
- модуль відображення та роботи зі списком знижок для клієнтів;
- модуль відображення та роботи зі списком медикаментів;
- модуль відображення та роботи зі списком компаній, ферм, лабораторій;
- модуль відображення та роботи зі списком усіх отриманих файлів.

4.4 Опис розроблених функцій

Розроблені функції інтерфейсної та серверної частин вебзастосунку для керування бізнес-процесами у CRM-системі тваринницьких ферм наведені у

таблиці 4.1.

Таблиця 4.1 – функції інтерфейсної та серверної частин вебзастосунку для керування бізнес-процесами у CRM-системі тваринницьких ферм

Назва функції	Опис
authUser()	Функція, яка авторизує користувача та зберігає його дані у сесії
agreeWithContract()	Функція, що зберігає файл та оновлює запис людини, якщо вона погодилась з користувацьким контрактом, що представлений на сайті
saveSignature()	Зберігає підпис користувача у БД
createRole()	Створює роль для користувача
editRole()	Змінює дозволи, або назву ролі
deleteRole()	Видаляє роль
banUser()	Банить користувача
editUser()	Змінює дані користувача
getUser()	Повертає дані про користувача
getAllUsers()	Повертає усіх користувачів
regUser()	Реєструє користувача
loginUser()	Логінить користувача у системі
linkWithTelegram()	Створює зв'язок з користувачем телеграму та користувачем сайту
createPerson()	Створює сутність компанія, лабораторія, або приватний ветеринар
getPerson()	Повертає сутності сутність компанія, лабораторія, або приватний ветеринар
deletePerson()	Видаляє сутність компанія, лабораторія, або приватний ветеринар
verifPerson()	Верифікує електронну адресу сутності
addUserToPerson()	Додає до сутності користувачів (робітників)
createAnimalTag()	Створює мікрочіп для тварини
createSafariTag()	Створює сафарі тегу
getFarm()	Повертає ферму
getFarmPin()	Повертає центру ферми за координатами
getFarmPortions()	Повертає окремої ділянки ферми
createFarmApplication()	Створює заявки на реєстрацію ферми
createWarthogApplication()	Створює заявки на дозвіл полювання на кабанів
createDisease()	Створює хворобу
createDiseaseProtocol()	Створює протокол хвороб для тестування
createTest()	Створює тест
createTestMethod()	Створює метод тестування
setTestToProtocol()	Додає тест до протоколу тестування

Продовження таблиці 4.1

Назва функції	Опис
createIsolation()	Створює ізоляцію для тварин
setProtocolToIsolation()	Додає протокол до ізоляції
addAnimalToIsolation()	Додає тварину до ізоляції
saveAnimalResult()	Зберігає результат тестування тварини
saveCITTAnimalResult()	Зберігає результат тестування тварини на туберкульоз
createNewFarmPortion()	Створює нову ділянку для ферми
uploadIsolationFile()	Зберігає файл з результатами ізоляції
uploadAnimalImages()	Зберігає фотографії тварин
createMedicine()	Створює ліки та препарати
deleteMedicine()	Видаляє ліки
editMedicine()	Змінює назву, строк придатності, або виробника ліків
setMedicineToIsolation()	Зберігає ліки для ізоляції

4.5 Висновки до четвертого розділу

У розділі було наведено структуру розроблених колекцій у СКБД MariaDB та розроблений алгоритм функціонування CRM-системи для тваринницьких ферм. Описано розроблені модулі, функції у розроблених частинах вебпроєкту для керування бізнес-процесами CRM-системи для тваринницьких ферм.

5 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ПРОДУКТУ

У розділі описано призначення вебзастосунку для керування бізнес-процесами у CRM-системі для тваринницьких ферм. Описано вимоги до апаратної та програмної частини для функціонування вебзастосунку, перелік файлів програми, виконання програми, повідомлення користувачам при роботі з вебзастосунком та результати тестування функцій розроблених модулів у веббраузерах.

5.1 Опис призначення програмного продукту

Вебзастосунок BUFFALO.VET призначений для автоматизації щоденних завдань ветеринарів, ветеринарних клінік, власників ферм та інших користувачів з галузі тваринництва, збереження даних про тварин, про ферми, результати тестування, історія перевезення, історія хвороб тварин. Видача дозволів на перевезення, полювання, отримання спеціального номеру для ферми. Вебзастосунок BUFFALO.VET надає користувачам наступні функції:

- реєстрація та авторизація користувачів;
- відновлення паролю через email або телефон;
- редагування профілю ветеринара або фермера;
- завантаження професійних сертифікатів і медичних документів;
- планування візитів до ферм через інтегрований календар;
- сповіщення про заплановані заходи та візити;
- перегляд історії медичних процедур та лікування тварин;
- додавання діагнозів, лікувань, вакцин і нотаток до профілю тварини;
- внесення та перегляд інформації про ферму (назва, номер, локація);
- реєстрація тварин і перегляд їхніх профілів;
- формування звітів для транспортування тварин;
- перевірка дотримання законодавчих вимог щодо переміщення тварин;

- завантаження та зберігання ветеринарних висновків;
- інтеграція з геолокацією для відстеження маршрутів;
- пошук і перегляд записів по тваринах, фермах або діагнозах;
- можливість підписання документів цифровим підписом;
- генерація звітів для різних типів користувачів (ветеринарів, фермерів, державних органів);
- сповіщення про важливі події, наприклад, вакцинацію або закінчення дозволів;
- розмежування ролей користувачів (адміністратор, ветеринар, фермер);
- налаштування доступу до даних залежно від ролі;
- інтеграція з платіжними системами для оплати послуг (за потреби);
- тематичний інтерфейс із налаштуванням мов;
- логування активності користувачів для забезпечення прозорості.

5.2 Вимоги до програмно-технічних засобів

Вимоги до апаратного забезпечення вебзастосунку BUFFALO.VET включають наступні параметри до апаратних засобів:

- процесор Intel Core i5 (або еквівалентний) починаючи з 5-го покоління;
- оперативна пам'ять не менше 8 ГБ для забезпечення стабільної роботи;
- мережеве підключення: стабільний інтернет зі швидкістю не менше 10 Мбіт/с;
- операційна система Windows 10/11, macOS 10.15 або новіші версії, підтримка сучасних веббраузерів.

Вимоги до програмного забезпечення включають:

- Apache/Nginx, PHP 8.0+, MySQL 8.0+/PostgreSQL 13+;
- сучасний веббраузер (Google Chrome, Mozilla Firefox, Microsoft Edge,

Safari) з підтримкою JavaScript і HTML5.

5.3 Характеристики програмного продукту

Найменування вебзастосунку – BUFFALO.VET. Перелік файлів клієнтської частини вебзастосунку:

- app.js – файл, який містить визначені основні компоненти вебзастосунку;
- app.blade.php – файл, що кореневим файлом застосунку;
- App.vue – файл, який виконує розгортання усього вебзастосунку;
- Index.vue – файл, який містить структуру усіх сторінок;
- vuex.js – зберігає усі модулі для Vuex;
- routes.js – зберігає усі шляхи до користувацької частини застосунку.

Перелік файлів серверної частини вебзастосунку:

- bootstrap.php – файл, що запускає проект;
- index.php – файл, що містить початкові інструкції при обробці запиту;
- config.php – файл, що зберігає налаштування серверної частини застосунку;
- .env – файл, що зберігає налаштування логінів, паролів, API ключів, підключення до БД.

5.4 Виконання програмного продукту

На початку роботи вебзастосунком BUFFALO.VET користувачу потрібно пройти аутентифікацію облікового запису за допомогою Google, або логіну та паролем (рис. 5.1). У разі відсутності профіля користувача можна обрати функцію реєстрації нового користувача (рис. 5.2). Діаграма послідовності авторизації та реєстрації користувача зображена на рисунку 5.3.



Welcome Back
Please enter your details below to sign in

Email *

Password *

[Forgot Password?](#)

Log In

Don't have an account? [Sign Up](#)

or

 **Continue with Google**

By submitting you agree to our [Terms of Service](#) and [Privacy Policy](#).

Рисунок 5.1 – Вебсторінка аутентифікації вебзастосунку BUFFALO.VET



Sign Up to Buffalo Analytics
Create an account to proceed

First Name *

Surname *

Email *

Password *

Confirm Password *

Signup

Already have an account? [Log In](#)

or

 **Continue with Google**

By submitting you agree to our [Terms of Service](#) and [Privacy Policy](#).

Рисунок 5.2 – Вебсторінка реєстрації вебзастосунку BUFFALO.VET

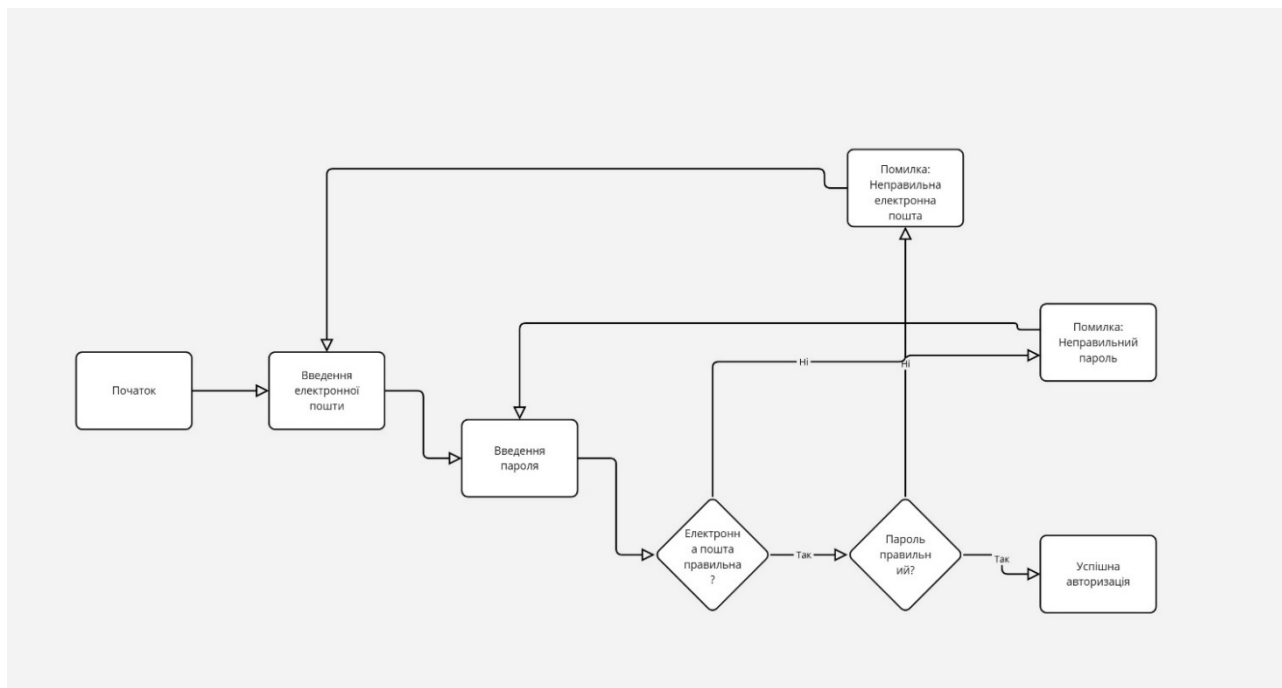


Рисунок 5.3 – Діаграма послідовності авторизації та реєстрації користувача

Адміністратор може видавати користувачам ролі, від яких будуть залежати доступні клієнту дії, такі як меню (рис. 5.4).

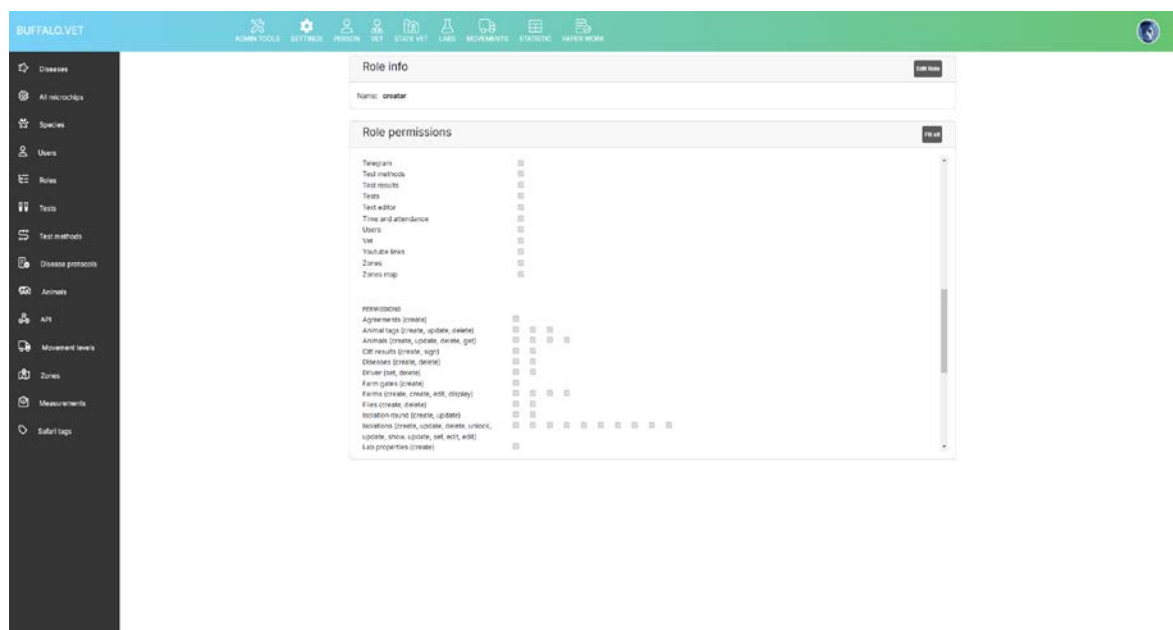


Рисунок 5.4 – Вебсторінка налаштування ролі «Засновник»

Після призначення ролі користувачу, користувач може створити компанію, або приватну особу (рис. 5.5), від лиця якої будуть проводитись подальші дії

(створення ферм, тварин, ізоляцій тощо). До створених компанії, або приватних осіб, користувач може додавати співробітників та назначати їм ролі. Процес створення сутності Особа та присвоєння до неї ферм зображаєна на рисунку 5.6.

Рисунок 5.5 – Сторінка створення компанії, або приватного підприємця

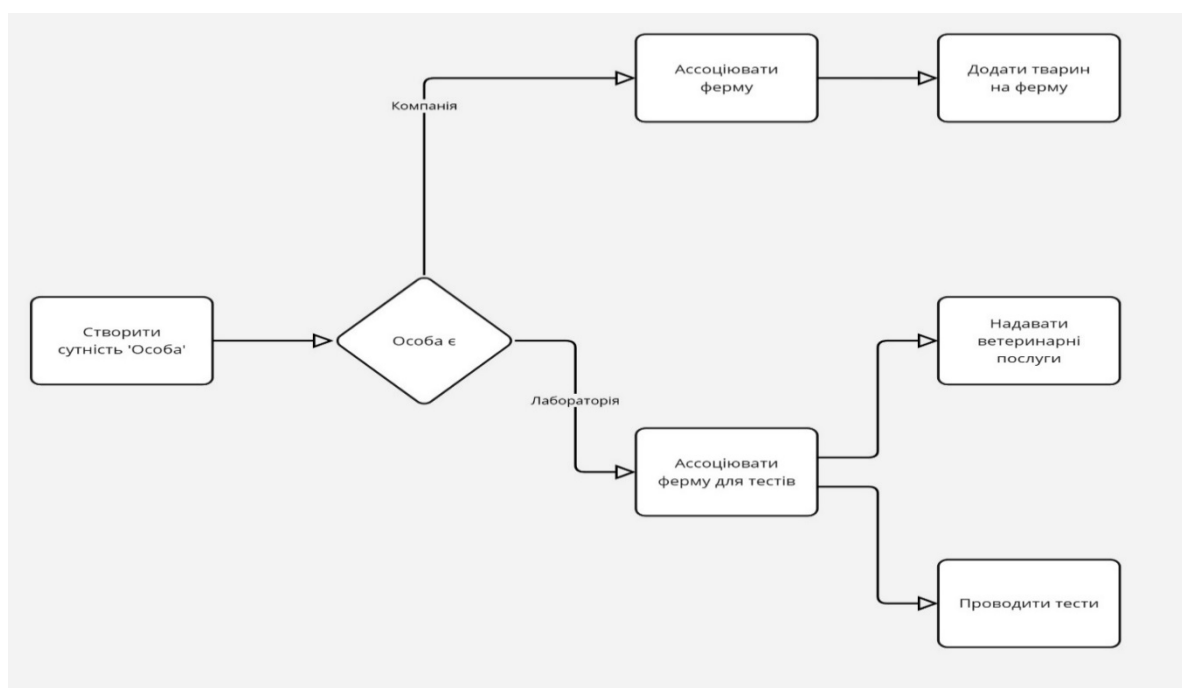


Рисунок 5.6 – Процес створення сутності Особа та присвоєння до неї ферм

Якщо користувач створив сутність з типом лабораторії, у нього також є можливість додавати тести хвороб, їх методи та протоколи тестування до себе, щоб використовувати це в тестуваннях тварин (рис. 5.7). Ці можливості дозволять іншим користувачам обирати цю лабораторію, як необхідну для тестування.

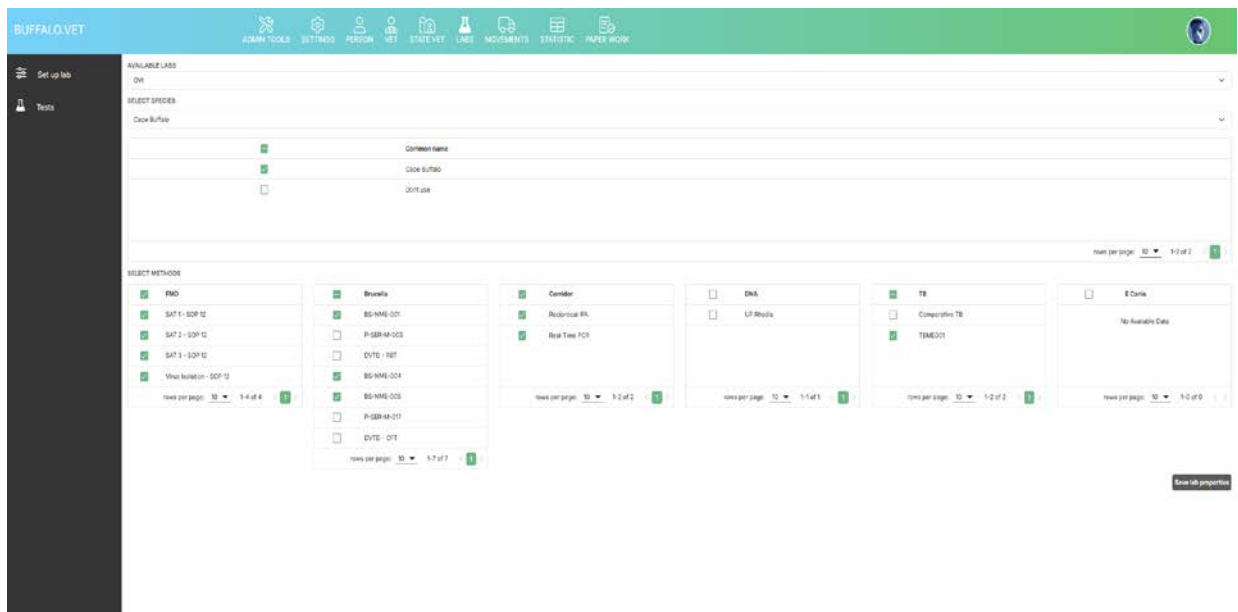


Рисунок 5.7 – Сторінка для назначення тестів та методів лабораторії

Наступні дії, які може зробити користувач – це додати собі ферму та створити тварин, щоб зберегти їх на своїй фермі (рис. 5.8).

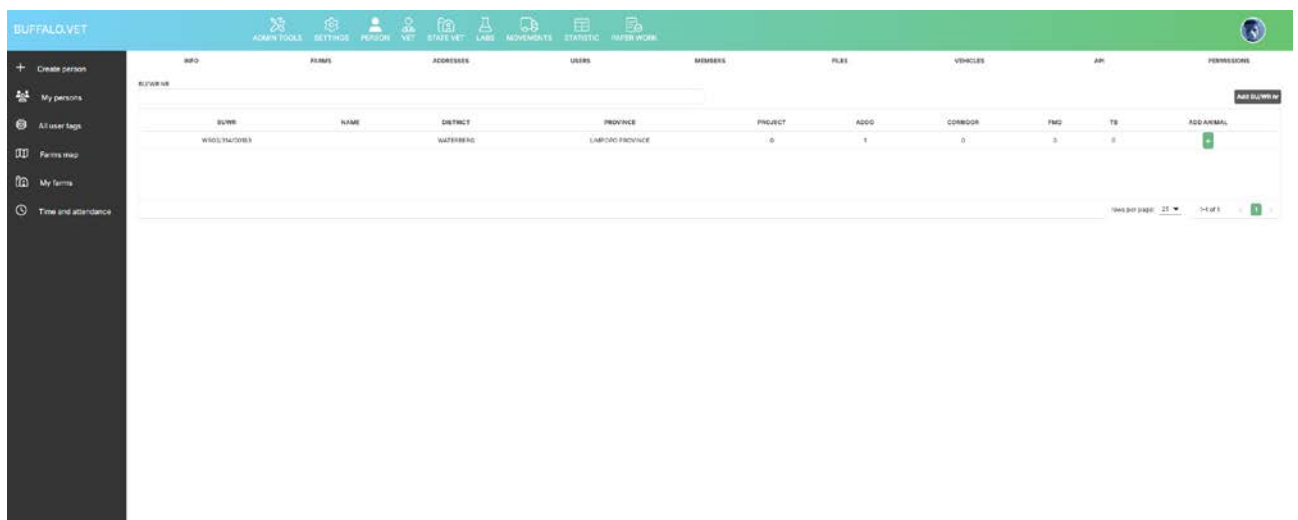


Рисунок 5.8 – Сторінка зі списком ферм, належачих до сутності

Наступним кроком іде додавання тварин до ферми (рис. 5.9). Після чого, їм створюють унікальні теги та додають у ізоляцію.

SPECIES
 Select species ▼

☐ Male ☐ Female ☐ Unknown

MICROCHIP ID

EXTERNAL TAG

BODY PLACEMENT
 ▼

DATE OF BIRTH
 dd.mm.pppp 📅

DATE OF DEATH
 dd.mm.pppp 📅

Create animal

Рисунок 5.9 – Форма створення тварини та збереження її на фермі

Якщо у БД немає потрібної зареєстрованої ферми, то у користувача є можливість зареєструвати її самому. Для цього є сторінка (рисунок 5.10), яка дозволяє це зробити за допомогою інтерфейса мапи.

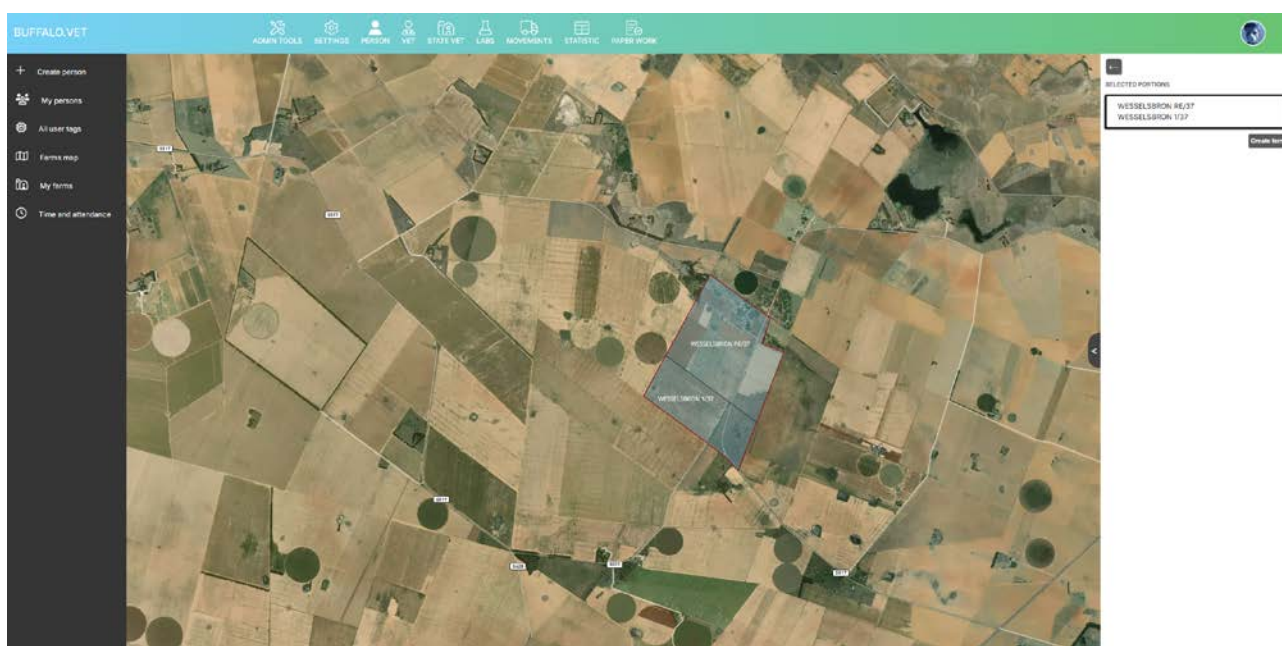


Рисунок 5.10 – Сторінка реєстрації нової ферми

Сторінка ізоляції тварин зображена на рисунку 5.11. А покроковий алгоритм створення ізоляції для тестування тварин наведено на рисунку 5.12.

BUFFALO.VET

ADMIN TOOLS SETTINGS PERSON VET STATE VET LABS MOVEMENTS STATISTIC PAPER WORK

TBC00017 [Unlink isolation](#)

OFFICE EMAIL: david.motsum@gmail.com

OWNER EMAIL: richard.york@buffalo.vet

ACCOUNTS EMAIL: richard.york@buffalo.vet

SENDER EMAIL: david.motsum@gmail.com

BURN NUMBER: WPG2314100783

ISOLATION START PERIOD: 2024-08-05

ISOLATION END PERIOD:

OWNER NAME: Richard York

ACCOUNT NAME: Richard York

SENDER NAME: David Hermanus Schalk Pretorius

OWNER REFERENCE:

Cape Buffalo

References:

INSTITUTE REF: 2024-0-TM82

LABORATORY REF: TAD 24/TB80

Extra Dates:

SAMPLING DATE: 2024-08-05

REPORT DATE: 2024-08-23

Purpose for Testing:

NUMBER FOR TESTING: Routine

Specimens:

SPECIMEN TYPES:

EDTA: ☐

Serum: ☐

Heparin: ☐

NUMBER OF SPECIMENS: 1

MICROCHIP ID: 992001000084370

SAT 1: negative

SAT 2: negative

SAT 3: negative

SAMPLE REF:

5.11 – Сторінка ізоляції тварин

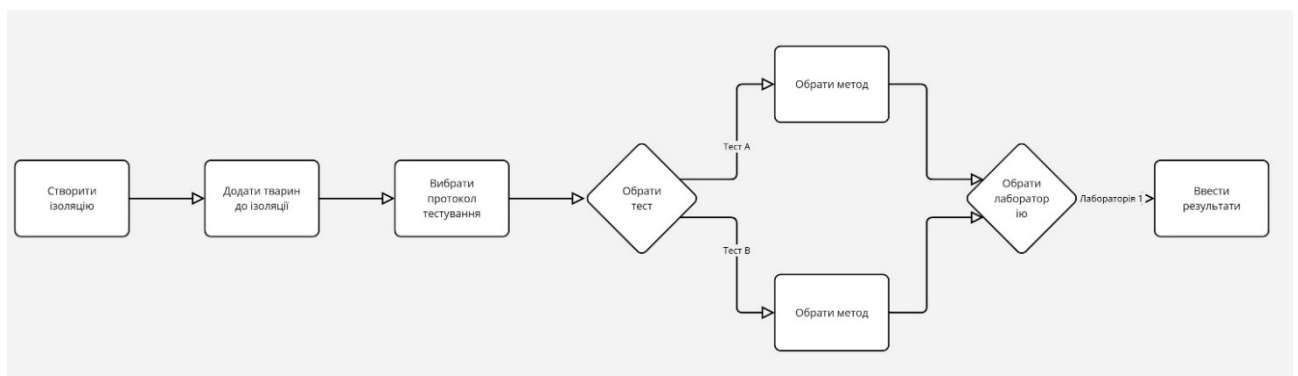


Рисунок 5.12 – Послідовність створення ізоляції для тестування тварин

5.5 Висновки до п'ятого розділу

Розроблений вебзастосунок BUFFALO.VET є сучасним рішенням для автоматизації процесів у ветеринарній галузі та на тваринницьких фермах. Його

функціонал охоплює широкий спектр завдань: від реєстрації користувачів і управління профілями до створення та ведення обліку тварин, ферм, лабораторій і процесів, пов'язаних із перевезенням та тестуванням.

Програмний продукт відповідає всім сучасним вимогам до апаратного та програмного забезпечення, що забезпечує його стабільну роботу на різних платформах та пристроях. Він інтегрується з популярними сервісами, такими як Google та платіжні системи, підтримує геолокацію, цифровий підпис і багаторівневу систему доступу, що робить його інструментом для ефективного управління бізнес-процесами.

У ході тестування вебзастосунок показав високу стабільність і відповідність заявленому функціоналу. Результати тестів підтвердили коректність роботи основних модулів у різних веббраузерах та на різних пристроях. Інтуїтивний інтерфейс, функціональні можливості та забезпечення прозорості даних роблять BUFFALO.VET зручним і корисним інструментом для користувачів.

Таким чином, розроблений продукт готовий до експлуатації, зберігаючи потенціал для подальшого вдосконалення і розширення функціоналу відповідно до потреб користувачів.

ВИСНОВКИ

У процесі виконання роботи було розглянуто, розроблено та проаналізовано функціонал CRM-системи, орієнтованої на потреби тваринницьких ферм. Система дозволяє автоматизувати ключові бізнес-процеси, зокрема управління тваринами, облік даних про їх здоров'я, організацію перевезень та підтримку взаємодії з ветеринарними службами. Було створено структуровану архітектуру, що охоплює всі аспекти роботи ферми, включаючи реєстрацію користувачів, розподіл ролей, управління доступами та інтеграцію з іншими сервісами. Одним із ключових результатів є побудова інтерфейсної частини та бази даних, які забезпечують зручність використання та гнучкість у розширенні системи. На основі принципів архітектурного патерну MVC було реалізовано чітке розділення бізнес-логіки, інтерфейсу та обробки даних. Також було впроваджено сервіси та репозиторії для забезпечення масштабованості й підтримуваності коду.

Розроблена CRM-система враховує специфіку тваринництва, пропонуючи функціонал, що дозволяє вести облік тварин із можливістю зберігання історичних даних, подавати заявки на перевезення та отримання спеціальних номерів, працювати з ізоляціями, тестами та протоколами здоров'я, організовувати роботу ветеринарів та зони їх відповідальності, а також ефективно управляти ролями та доступами для користувачів. Система забезпечує високу безпеку даних завдяки аутентифікації за допомогою Google OAuth 2.0 та можливості гнучкого налаштування прав доступу. Впровадження API дозволяє інтегрувати CRM-систему з іншими платформами для автоматизації більш складних процесів.

Результати роботи підтверджують, що розроблена CRM-система здатна значно оптимізувати бізнес-процеси тваринницьких ферм, підвищити ефективність їх управління та створити єдину інформаційну екосистему для фермерів, ветеринарів і супутніх служб. Це дозволяє зробити висновок, що система є перспективною для впровадження в практичну діяльність та подальшого вдосконалення.

Вперше запропоновано інтегрований метод оптимізації логістичних маршрутів, який базується на комбінованому використанні генетичних алгоритмів та моделей теорії ігор. Цей підхід дозволяє одночасно враховувати множинні обмеження, такі як час доставки, витрати на паливо та обсяги вантажу, що дозволяє зменшити загальні витрати на перевезення і знизити час розробки маршрутів. Крім того, метод забезпечує динамічну адаптацію до змін у транспортній мережі.

Удосконалено метод прогнозування динаміки попиту у логістичних системах шляхом інтеграції нейромереж із традиційними статистичними ARIMA-моделями. Це дозволяє моделювати нелінійні залежності та покращувати точність прогнозування, що зменшує ризики перевищення запасів або дефіциту товарів. Оновлений метод є адаптивним і дозволяє в режимі реального часу враховувати сезонність та інші зовнішні фактори.

Дістав подальшого розвитку метод кластеризації вантажних потоків для оптимізації роботи розподільчих центрів. Удосконалений алгоритм використовує методи глибокого навчання, що дозволяє не лише ідентифікувати вузькі місця у вантажних потоках, але й прогнозувати потребу у ресурсах на різних етапах логістичного процесу. Це забезпечує збільшення пропускної здатності розподільчих центрів та зменшення часу обробки замовлень.

Практична значимість роботи полягає у створенні методів та інструментів, які можуть бути впроваджені для підвищення ефективності логістичних систем і автоматизації бізнес-процесів. Розроблені підходи до оптимізації маршрутів, прогнозування попиту та управління запасами дозволяють значно скоротити витрати на перевезення, зменшити надлишкові запаси та прискорити виконання логістичних операцій.

CRM-система, розроблена для тваринницьких ферм, також має високу практичну значимість, оскільки сприяє автоматизації обліку тварин, управлінню ветеринарними послугами, плануванню ізоляцій і контролю перевезень. Її використання дозволяє покращити координацію між фермерськими господарствами та ветеринарними службами, зменшити кількість помилок у

роботі з даними та підвищити загальну продуктивність ферми. Завдяки впровадженню сучасних CRM-рішень фермери отримують можливість ефективно аналізувати інформацію та приймати обґрунтовані рішення.

Таким чином, розроблені методи та програмні рішення є універсальними інструментами, які сприяють зниженню витрат і підвищенню ефективності як у логістиці, так і в управлінні фермерськими господарствами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Top 10 Digital Transformation Technologies for Member-based Organizations [with definition and examples] [Electronic resource]. – Access mode: <https://www.glueup.com/blog/digital-transformation-technologies>.
2. Розробка CRM для агрокомплексу [Electronic resource]. – Access mode: <https://avada-media.ua/services/crm-dlya-agrokompleksa/>.
3. Як агрокомпанії обрати CRM-систему, та чи є українські альтернативи [Електрон. ресурс]. – Режим доступу: <https://agroportal.ua/blogs/yak-agrokompaniji-obrati-crm-sistemu-ta-chi-ye-ukrajinski-alternativi>.
4. Доступ до системи. Ролі [Електрон. ресурс]. – Режим доступу: <https://cleverbox-crm.com/dostup.html>.
5. USDA (U.S. Department of Agriculture) [Electronic resource]. – Access mode: <https://www.usda.gov/topics/farming>.
6. Use of CRM Strategy in Agriculture Industry [Electronic resource]. – Access mode: <https://enmain.com/customer-relationship-management/use-of-crm-strategy-in-agriculture-industry/>.
7. 5 Best Practices on How to use Veterinary Software & CRM for Vet Clinics [Electronic resource]. – Access mode: <https://www.nimble.com/blog/vet-clinics-crm/>.
8. Customer Relationship Management (CRM) for Veterinary Teams: Introducing Your Enhanced Campaigns and Dashboard [Electronic resource]. – Access mode: <https://petsapp.com/blog/customer-relationship-management-crm-for-veterinary-teams-introducing-your>.
9. Бізнес-логіка — Вікіпедія [Електрон. ресурс]. – Режим доступу: <https://uk.wikipedia.org/wiki/Бізнес-логіка>
10. Що таке "бізнес логіка"? І як почати її розуміти - FoxmindEd [Електрон. ресурс]. – Режим доступу: https://foxminded.ua/shcho_take_biznes_lohika_i_yak_pochaty/
11. Zoho CRM: коротко про головне [Електрон. ресурс]. – Режим

доступу: <https://alterego.digital/ua/blog/zoho-crm>.

12. Використання Zoho CRM для ефективного управління клієнтами та кадрами в Україні [Електрон. ресурс]. – Режим доступу: <https://shelfy.com.ua/newsroom/vykorystannya-zoho-crm-v-ukrayini/>.

13. Огляд Zoho CRM: Все, що потрібно знати [Електрон. ресурс]. – Режим доступу: <https://www.whitesales.ua/uk/blog/zoho-crm-review-everything-you-need-to-know/>.

14. AgriWebb: Комплексне управління тваринництвом [Електрон. ресурс]. – Режим доступу: <https://agtecher.com/uk/product-uk/agriwebb/>.

15. Getting Started with Livestock [Electronic resource]. – Access mode: <https://help.farmbrite.com/help/getting-started-with-livestock>.

16. Методологічний апарат логістики [Електрон. ресурс]. – Режим доступу: https://stud.com.ua/1703/logistika/metodologichniy_aparat_logistiki.

17. Метод ABC-аналізу [Електрон. ресурс]. – Режим доступу: https://stud.com.ua/14310/logistika/metod_analizu.

18. Логістичний аналіз і контролінг Методи аналізу та оптимізації в логістичних дослідженнях [Електрон. ресурс]. – Режим доступу: https://stud.com.ua/171064/logistika/logistichniy_analiz_kontrolling.

19. PHP [Electronic resource]. – Access mode: <https://en.wikipedia.org/wiki/PHP>.

20. JavaScript [Electronic resource]. – Access mode: <https://uk.wikipedia.org/wiki/JavaScript/>.

21. PHP Storm – Empowering PHP Developers [Electronic resource]. – Access mode: <https://www.jetbrains.com/phpstorm/>.

22. HTML [Electronic resource] / – Access mode: <https://influencemarketinghub.com/glossary/html/>.

23. CSS: Cascading Style Sheets [Electronic resource] / – Access mode: <https://medium.com/educipher/css-cascading-style-sheets-b9706e77224>.

24. Vue.js 3 – Introduction [Electronic resource] / - Access mode: <https://vuejs.org/guide/introduction.html>.

25. The Official Router for Vue.js [Electronic resource]. – Access mode: https://devdocs.io/vue_router~3-api-reference/
26. 8 основних переваг фреймворку Laravel для веб-розробки [Електрон. ресурс]. – Режим доступу: <https://wezom.com.ua/ua/blog/17-preimuschestv-ispolzovaniya-laravel-v-it-industrii>.
27. Medium - Advantages of Laravel [Electronic resource]. – Access mode: <https://medium.com/@techmngtr/top-10-advantages-of-laravel-framework-for-web-development-4c894aaad8f9>.
28. MariaDB [Electronic resource]. – Access mode: <https://uk.wikipedia.org/wiki/MariaDB>.
29. MariaDB Server: the innovate open source database [Electronic resource] / – Access mode: <https://mariadb.org/>.
30. Побудова динамічних вебзастосунків за допомогою MVC [Електрон. ресурс]. – Режим доступу: <https://blog.ithillel.ua/articles/building-dynamic-web-applications-using-mvc>.
31. Service Container [Electronic resource]. – Access mode: <https://laravel.com/docs/10.x/container#binding-classes-into-the-container>.
32. Впровадження патерну репозиторію для кращої організації коду [Електрон. ресурс]. – Режим доступу: <https://peerdh.com/uk/blogs/programming-insights/implementing-repository-pattern-for-better-code-organization>.

ДОДАТОК А
Технічне завдання

Вступ

Сфера тваринництва є важливим сегментом аграрної економіки, де ефективне управління фермерськими господарствами відіграє ключову роль у досягненні високої продуктивності. Розробка інструментів для автоматизації бізнес-процесів на тваринницьких фермах є перспективною ідеєю, яка дозволяє покращити облік, контроль та управління процесами, а також підвищити якість обслуговування ветеринарних послуг.

Програмні рішення для обліку бізнес-процесів у сфері тваринництва дозволяють зручно працювати з базою тварин, управляти ветеринарними оглядами, організовувати перевезення та контроль за станом ізоляцій. Такі рішення також сприяють оптимізації обліку кормів, медикаментів, витратних матеріалів та інших ресурсів. Крім того, програмні продукти забезпечують автоматизацію управління документацією, що економить час співробітників та покращує загальну продуктивність фермерського господарства.

A.1 Найменування програмного продукту

Назва вебзастосунку – BUFFALO.VET.

A.2 Підстави для розробки

Випускна кваліфікаційна робота виконується за темою «Дослідження та програмна реалізація CRM-системи для керування бізнес-процесами тваринницьких ферм» відповідно до наказу № 398 від 07 жовтя 2024 року, затвердженого Національним університетом «Запорізька політехніка».

A.3 Призначення та цілі реалізації програмного продукту

Призначенням розроблюваного вебзастосунку є автоматизація

управління бізнес-процесами тваринницьких ферм. Програмний продукт надає функціональні можливості залежно від ролі користувача, таких як керівник ферми, ветеринар або спеціаліст із перевезення тварин.

Вебзастосунок BUFFALO.VET використовується для додавання до бази тварин, ветеринарних оглядів, історичних даних, запитів на перевезення, тестувань на хвороби, створення ролей та зон відповідальності для ветеринарів, а також управління інвентарем. Основною ціллю реалізації вебзастосунку є оптимізація роботи фермерських господарств, підвищення ефективності ветеринарного контролю, спрощення перевезень тварин та підтримка ізоляцій.

A.4 Перелік вимог до програмного продукту

A.4.1 Функціональні вимоги

У вебзастосунку BUFFALO.VET повинні бути реалізовані наступні функції відповідно до ролей користувачів: керівник ферми, ветеринар, адміністратор та спеціаліст із перевезення тварин. Програмний продукт має забезпечувати наступні функціональні можливості: аутентифікація користувачів, додавання інформації про ферми та тварин, облік ветеринарних оглядів, подання заявок на перевезення тварин, створення та ведення зон відповідальності для ветеринарів, пошук та відображення тварин із повною історією їх здоров'я. Крім того, вебзастосунок повинен підтримувати роботу з ізоляціями, управління ресурсами, створення сафарі-тегів та подання звітів.

A.4.2 Вимоги до продуктивності

Вебзастосунок BUFFALO.VET має забезпечувати швидкість відповіді на запити користувачів не більше 3 секунд для ефективної взаємодії з системою.

A.4.3 Вимоги до користувацького інтерфейсу

При розробці інтерфейсу користувача необхідно забезпечити адаптивність вебдизайну до різних розмірів екранів, інтуїтивно зрозумілу навігацію та використання кольорової гами, яка відповідає тематиці аграрного сектору. Інтерфейс повинен бути зручним для використання всіма категоріями користувачів.

A.4.4 Експлуатаційні вимоги програмного продукту

Вебзастосунок не потребує специфічного програмного забезпечення або операційної системи, але повинен працювати на сучасних браузерях. Для забезпечення ефективної роботи необхідний стабільний доступ до мережі Інтернет та апаратного забезпечення із середніми технічними характеристиками.

A.4.5 Вимоги до апаратного забезпечення

Для коректного функціонування вебзастосунку BUFFALO.VET рекомендується використовувати апаратне забезпечення з такими характеристиками: процесор Intel Core i5 або новіший, оперативна пам'ять 8 Гб, накопичувач не менше 512 Гб, відеокарта середнього рівня продуктивності та стабільне мережеве підключення.

A.4.6 Вимоги до безпеки

Кожен користувач повинен мати власний обліковий запис із персональним логіном та паролем. Аутентифікація здійснюється через Google OAuth 2.0 або внутрішню систему авторизації. Адміністратор має можливість призначати права доступу відповідно до ролі користувача в системі.

A.5 Етапи виконання магістерської роботи

Виконання дипломної роботи включає етапи дослідження, проектування, реалізації та тестування вебзастосунку BUFFALO.VET. Остаточним етапом є підготовка пояснювальної записки та захист роботи перед комісією.

A.6 Послідовність контролю виконання та приймання роботи

Дипломна робота виконується згідно з затвердженим графіком. Після перевірки на відповідність вимогам та унікальність робота подається до екзаменаційної комісії для захисту.

ДОДАТОК Б
Текст програми

Б.1 Маршрутизація сторінок вебзастосунку

<?php

```

use App\Http\Controllers\AgroEcologicalZonesController;
use App\Http\Controllers\AnimalParentGroupsController;
use App\Http\Controllers\Api\ApiUserRoleController;
use App\Http\Controllers\ApplicationsController;
use App\Http\Controllers\Auth\AnimalsController;
use App\Http\Controllers\Auth\CommunicationController;
use App\Http\Controllers\Auth\DiseasesController;
use App\Http\Controllers\Auth\FarmController;
use App\Http\Controllers\Auth\FilesController;
use App\Http\Controllers\Auth\FirebaseOAuth;
use App\Http\Controllers\Auth\IsolationsController;
use App\Http\Controllers\Auth\LabsController;
use App\Http\Controllers\Auth\LandingPageController;
use App\Http\Controllers\Auth\MovementsController;
use App\Http\Controllers\Auth\PermissionsController;
use App\Http\Controllers\Auth\PersonController;
use App\Http\Controllers\Auth\PersonUserRolesController;
use App\Http\Controllers\Auth\ShowAppController;
use App\Http\Controllers\Auth\SpeciesController;
use App\Http\Controllers\Auth\TestController;
use App\Http\Controllers\Auth\TestingProtocolController;
use App\Http\Controllers\Auth\TestMethodsController;
use App\Http\Controllers\Auth\TestResultController;
use App\Http\Controllers\Auth\TestsController;
use App\Http\Controllers\Auth\TextEntitiesController;
use App\Http\Controllers\Auth\TextEntityViewsController;
use App\Http\Controllers\Auth\UserController;
use App\Http\Controllers\Auth\UserRoleController;
use App\Http\Controllers\Auth\VerificationController;
use App\Http\Controllers\CountriesController;
use App\Http\Controllers\CsvFilesController;
use App\Http\Controllers\DavidController;
use App\Http\Controllers\DeviceController;
use App\Http\Controllers\Google2FAController;
use App\Http\Controllers\LabPdfFilesController;
use App\Http\Controllers\MeasurementController;
use App\Http\Controllers\MedicineBatchController;
use App\Http\Controllers\MedicineController;
use App\Http\Controllers\MovementAuthorizationController;
use App\Http\Controllers\MySQLCommandsController;
use App\Http\Controllers\PersonUserMembersController;
use App\Http\Controllers\RcpController;
use App\Http\Controllers\SafariTagsController;
use App\Http\Controllers\ShapesAbattoirsController;
use App\Http\Controllers\ShapesAdminController;
use App\Http\Controllers\ShapesFMDController;
use App\Http\Controllers\SignatureController;
use App\Http\Controllers\TrackingPixelController;
use App\Http\Controllers\YoutubeLinksController;
use App\Http\Middleware\AddRandomPadding;
use App\Http\Middleware\AuthCheck;
use App\Http\Middleware\MedicineAccess;
use App\Http\Middleware\ZonesMapAccess;
use Illuminate\Support\Facades\Route;

```

```

/*
|-----
|
| Web Routes
|-----
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "web" middleware group. Make something great!
|
*/

Route::fallback(function () {
    return 'Fallback route working!';
});

Route::get('/iclock/cdata', [DeviceController::class, 'handleCData']);
Route::post('/iclock/cdata', [DeviceController::class,
'handleCDDDataPost']);

Route::domain('warthog.'.env('APP_DOMAIN'))->group(function () {
    Route::get('/warthog/{permitUuid}', [PersonController::class,
'showWarthogPermitFile'])->name('generate-warthog-permit-file');
});

Route::domain('st.'.env('APP_DOMAIN'))->
    //middleware([AuthCheck::class])->
    group(function () {
        // Route::get('/{countryCode}/{safariTag}',
[SafariTagsController::class, 'getAnimalBySafariTag'])->name('safari-
tag-page');
        Log::info('ST Branch hit again 1');
    });

Route::get('/tracking-pixel', [TrackingPixelController::class,
'track']);

Route::post('/agree-contract', [UserController::class,
'agreeWithContract'])->name('agree-contract');
Route::prefix('telegram')->group(function () {
    Route::post('/webhook', [CommunicationController::class,
'telegramWebhook'])->name('telegram-webhook');
    Route::post('/send-message', [CommunicationController::class,
'sendTelegramMessage'])->name('send-message');
    Route::get('/get-telegram-qr', [CommunicationController::class,
'getTelegramQr'])->name('get-telegram-qr');
});

Route::domain(env('APP_DOMAIN'))->middleware([AuthCheck::class,
AddRandomPadding::class])->group(function () {
    //Route::prefix('st')->group(function() {
    //    Route::get('/{countryCode}/{safariTag}',
[SafariTagsController::class, 'getAnimalBySafariTag'])->name('safari-
tag-page');
    //});

    //Routes for permissions and roles
    Route::prefix('permissions')->group(function () {

```

```

        Route::get('/check-user-role-permission',
[UserRoleController::class, 'checkUserRolePermission'])->name('check-
user-role-permission');
        Route::delete('/delete-role', [UserRoleController::class,
'deleteRole'])->name('delete-role');
        Route::post('/edit-role', [UserRoleController::class,
'editRole'])->name('edit-role');
        Route::get('/get-roles', [UserRoleController::class,
'getRoles'])->name('get-roles');
        Route::get('/get-role-by-name', [UserRoleController::class,
'getRoleByName'])->name('get-role-by-name');
        //Route::get('/get-role-by-id', [UserRoleController::class,
'getRoleById'])->name('get-role-by-id');
        Route::get('/get-all-permissions',
[PermissionsController::class, 'getAllPermissions'])->name('get-all-
permissions');
        Route::get('/get-all-permissions-names',
[PermissionsController::class, 'getAllPermissionsNames'])->name('get-
all-permissions-names');
        Route::get('/get-all-permission-for-persons',
[PermissionsController::class, 'getAllPersonPermissions'])->name('get-
all-permission-for-persons');
        Route::post('/create-role-with-permissions',
[UserRoleController::class, 'createRole'])->name('create-role-with-
permissions');
        Route::post('/save-person-permissions',
[PersonController::class, 'savePersonPermissions'])->name('save-person-
permissions');
        Route::get('/get-person-permissions', [PersonController::class,
'getPersonPermissions'])->name('get-person-permissions');
    });

    Route::prefix('api-roles')->group(function () {
        Route::post('/create-role', [ApiUserRoleController::class,
'createRole'])->name('create-role');
        Route::post('/update-role', [ApiUserRoleController::class,
'updateRole'])->name('update-role');
        Route::post('/delete-api-role', [ApiUserRoleController::class,
'deleteRole'])->name('delete-api-role');
        Route::get('/get-api-roles', [ApiUserRoleController::class,
'getRoles'])->name('get-api-roles');
        Route::get('/get-permissions', [ApiUserRoleController::class,
'getPermissions'])->name('get-permissions');
        Route::post('/update-user', [ApiUserRoleController::class,
'updateUser'])->name('update-user');
    });

    // Routes for functions with user model
    Route::prefix('users')->group(function () {
        Route::get('/set-role', [UserController::class, 'setRole'])->
>name('set-role');
        Route::get('/get-user', [UserController::class, 'getUser'])->
>name('get-user');
        Route::get('/get-all-users', [UserController::class,
'getAllUsers'])->name('get-all-users');
        Route::post('/link-with-telegram', [UserController::class,
'linkWithTelegram'])->name('link-with-telegram');
        Route::patch('/edit-user', [UserController::class, 'editUser'])->
>name('edit-user');

```

```

        Route::patch('/ban-user', [UserController::class, 'banUser'])->name('ban-user');
        Route::delete('/delete-user', [UserController::class, 'deleteUser'])->name('delete-user');
        Route::get('/search-user', [UserController::class, 'searchUser'])->name('search-user');
        Route::post('/save-signature', [UserController::class, 'saveSignature'])->name('save-signature');
        Route::post('/upload-signature-image', [UserController::class, 'uploadSignatureImage'])->name('upload-signature-image');
        Route::get('/get-auth-user-model', [UserController::class, 'getAuthUser'])->name('get-auth-user-model');
    });

    //Routes for Chats
    Route::prefix('communications')->group(function () {
        Route::get('/get-telegram-users', [CommunicationController::class, 'getTelegramUsers'])->name('get-telegram-users');
        Route::get('/create-chat', [CommunicationController::class, 'createTelegramChat'])->name('create-chat');
        Route::get('/get-user-chats', [CommunicationController::class, 'getUserChats'])->name('get-user-chats');
        Route::get('/get-telegram-chat-messages', [CommunicationController::class, 'getTelegramChat'])->name('get-telegram-chat');
        Route::get('/update-chat-status', [CommunicationController::class, 'updateChatStatus'])->name('update-chat-status');
        // Route::get('/check-for-notification', [\App\Http\Controllers\Auth\CommunicationController::class, 'checkForNotifications'])->name('check-for-notifications');
    });

    //Routes for person
    Route::prefix('persons')->group(function () {
        Route::get('/sign-warthog-permit', [PersonController::class, 'signWarthogPermit'])->name('sign-warthog-permit');
        Route::get('/check-warthog-permit-signed', [PersonController::class, 'checkWarthogPermitSigned'])->name('check-warthog-permit-signed');
        Route::get('/create-person', [PersonController::class, 'createPerson'])->name('create-person');
        Route::get('/get-persons', [PersonController::class, 'getPersons'])->name('get-persons');
        Route::get('/get-user-persons', [PersonController::class, 'getUserPersons'])->name('get-user-persons');
        Route::get('/get-person', [PersonController::class, 'getPerson'])->name('get-person');
        Route::get('/search-person', [PersonController::class, 'searchPerson'])->name('search-person');
        Route::get('/get-person-by-person-email', [PersonController::class, 'getPersonByPersonEmail'])->name('get-person-by-person-email');
        Route::get('/get-person-by-uuid', [PersonController::class, 'getPersonByUUID'])->name('get-person-by-uuid');
        // Route::get('/get-user-person-by-slug', [\App\Http\Controllers\Auth\PersonController::class, 'getUserPersonBySlug'])->name('get-user-person-by-slug');
    });

```

```

        Route::get('/verif-email', [PersonController::class,
'verifEmail'])->name('verif-email');
        //      Route::get('/get-user-persons',
[App\Http\Controllers\Auth\PersonController::class,
'getPersonByUserId'])->name('get-user-persons');
        Route::post('/check-person', [PersonController::class,
'checkPerson'])->name('check-person');
        Route::post('/delete-person', [PersonController::class,
'deletePerson'])->name('delete-person');
        Route::post('/add-users-to-person', [PersonController::class,
'addUsersToPerson'])->name('add-users-to-person');
        Route::get('/get-person-files-by-bunr',
[PersonController::class, 'getPersonFilesByBunr'])->name('get-person-
files-by-bunr');
        Route::get('/get-person-entity-types', [PersonController::class,
'getPersonEntityTypeTypes'])->name('get-person-entity-types');
        Route::get('/get-user-labs', [PersonController::class,
'getUserLabs'])->name('get-user-labs');
        Route::post('/update-person-address', [PersonController::class,
'updatePersonAddress'])->name('update-person-address');
        Route::get('/get-all-labs', [PersonController::class,
'getAllLabs'])->name('get-all-labs');
        Route::post('/update-person-identifier',
[PersonController::class, 'updatePersonIdentifier'])->name('update-
person-identifier');
        Route::get('/get-person-identifier', [PersonController::class,
'getPersonIdentifier'])->name('get-person-identifier');
        Route::post('/save-person-file', [PersonController::class,
'savePersonFile'])->name('save-person-file');
        Route::get('/generate-person-file/{fileName}',
[PersonController::class, 'generatePersonFile'])->name('generate-person-
file');
        Route::get('/get-lab-methods', [PersonController::class,
'getLabMethods'])->name('get-lab-methods');
        Route::post('/update-person-info', [PersonController::class,
'updatePersonInfo'])->name('update-person-info');
        Route::post('/generate-user-agreement/{fileName}',
[PersonController::class, 'generateUserAgreement'])->name('generate-
user-agreement');
        Route::get('/get-person-agreement', [PersonController::class,
'getPersonAgreement'])->name('get-person-agreement');
        Route::get('/get-person-titles', [PersonController::class,
'getPersonTitles'])->name('get-person-titles');
        Route::get('/get-responsible-person', [PersonController::class,
'getResponsiblePerson'])->name('get-responsible-person');
        Route::post('/set-responsible-person', [PersonController::class,
'setResponsiblePerson'])->name('set-responsible-person');
        Route::post('/remove-responsible-person',
[PersonController::class, 'removeResponsiblePerson'])->name('remove-
responsible-person');
        Route::get('/get-natural-persons', [PersonController::class,
'getNaturalPersons'])->name('get-natural-persons');
        Route::get('/get-all-natural-persons', [PersonController::class,
'getAllNaturalPersons'])->name('get-all-natural-persons');
        Route::get('/get-natural-persons-list',
[PersonController::class, 'getNaturalPersonsList'])->name('get-natural-
persons-list');
        Route::get('/get-citt-tester-persons', [PersonController::class,
'getCittTesterPersons'])->name('get-citt-tester-persons');

```

```

        Route::get('/get-api-users', [PersonController::class,
'getApiUsers'])->name('get-api-users');
        Route::get('/api-users-list', [PersonController::class,
'getApiUsersList'])->name('api-users-list');
        Route::get('/set-driver', [PersonController::class,
'setDriver'])->name('set-driver');
        Route::delete('/delete-driver', [PersonController::class,
'deleteDriver'])->name('delete-driver');
        Route::get('/get-driver', [PersonController::class,
'getDriver'])->name('get-driver');
        Route::get('/get-persons-with-drivers',
[PersonController::class, 'getPersonsWithDrivers'])->name('get-persons-
with-drivers');
        Route::get('/create-vehicle', [PersonController::class,
'createVehicle'])->name('create-vehicle');
        Route::get('/update-vehicle', [PersonController::class,
'updateVehicle'])->name('update-vehicle');
        Route::get('/get-vehicles', [PersonController::class,
'getVehicles'])->name('get-vehicles');
        Route::delete('/delete-vehicle', [PersonController::class,
'deleteVehicle'])->name('delete-vehicle');
        Route::get('/companies-and-natural-persons',
[PersonController::class, 'getCompaniesAndNaturalPersons'])->
>name('companies-and-natural-persons');
        Route::get('/persons-movement-auth-levels',
[PersonController::class, 'getPersonsWithMovementAuthLevels'])->
>name('persons-movement-auth-levels');
        Route::post('/add-passport-info', [PersonController::class,
'addPassportInfo'])->name('add-passport-in
fo');
        Route::get('/countries', [PersonController::class,
'getCountries'])->name('countries');
        Route::get('/get-person-user-role-permissions',
[PersonController::class, 'getPersonUserRolePermissions'])->name('get-
person-user-role-permissions');
        Route::post('/update-person-logo', [PersonController::class,
'updatePersonLogo'])->name('update-person-logo');
        Route::get('/get-issuer-persons', [PersonController::class,
'getIssuerPersons'])->name('get-issuer-persons');
        Route::get('/get-warthog-permit-issuer-persons',
[PersonController::class, 'getWarthogPermitIssuerPersons'])->name('get-
warthog-permit-issuer-persons');
        Route::get('/submit-permit', [PersonController::class,
'submitPermit'])->name('submit-permit');
        Route::get('/get-person-permits', [PersonController::class,
'getPersonPermits'])->name('get-person-permits');
        Route::get('/get-person-warthog-permits',
[PersonController::class, 'getPersonWarthogPermits'])->name('get-person-
warthog-permits');
        Route::get('/get-applications-for-permits',
[PersonController::class, 'getApplicationsForPermits'])->name('get-
applications-for-permits');
        Route::get('/get-persons-by-farms', [PersonController::class,
'getPersonsByFarms'])->name('get-persons-by-farms');
        Route::get('/allow-app-for-permit', [PersonController::class,
'allowAppForPermit'])->name('allow-app-for-permit');
        Route::get('/deny-app-for-permit', [PersonController::class,
'denyAppForPermit'])->name('deny-app-for-permit');
        Route::post('/issue-warthog-permit', [PersonController::class,

```

```

'issueWarthogPermit']->name('issue-warthog-permit');
    Route::get('/search-persons-by-id', [PersonController::class,
'searchPersonById'])->name('search-persons-by-id');
    Route::get('/get-all-issued-permits-by-auth-code',
[PersonController::class, 'getAllIssuedPermitsByAuthCode'])->name('get-
all-issued-permits-by-auth-code');
    Route::delete('/delete-users-from-person',
[PersonController::class, 'deleteUsersFromPerson'])->name('delete-users-
from-person');
    Route::get('/get-person-users', [PersonController::class,
'getPersonUsers'])->name('get-person-users');

});

//Routes for farms
Route::prefix('farms')->group(function () {
    Route::get('/is-it-users-farm', [FarmController::class,
'isItUsersFarm'])->name('is-it-users-farm');
    Route::get('/get-bunr-pin', [FarmController::class,
'getBunrPin'])->name('get-bunr-pin');
    Route::get('/get-bunr-numbers', [FarmController::class,
'getBunrNumbers'])->name('get-all-bunr-numbers');
    Route::get('/get-bunr-by-uuid', [FarmController::class,
'getBunrByUuid'])->name('get-bunr-by-uuid');
    Route::get('/get-farm', [FarmController::class, 'getFarm'])->
>name('get-farm');
    Route::get('/get-farm-by-name', [FarmController::class,
'getFarmByName'])->name('get-farm-by-name');
    Route::get('/save-person-bunrs', [FarmController::class,
'savePersonBuNrs'])->name('save-person-bunrs');
    Route::get('/get-personBuNrs', [FarmController::class,
'getPersonBuNrs'])->name('get-get-personBuNrs');
    Route::get('/get-farms-by-uuid', [FarmController::class,
'getFarmsByUuid'])->name('get-farms-by-uuid');
    Route::get('/get-all-farms', [FarmController::class,
'getAllFarms'])->name('get-all-farms');
    Route::get('/search-farm-by-bunr', [FarmController::class,
'searchFarmByBuNr'])->name('search-farm-by-bunr');
    Route::get('/get-provinces-border', [FarmController::class,
'getProvincesBorders'])->name('get-provinces-border');
    Route::get('/get-provinces-info', [FarmController::class,
'getProvincesInfo'])->name('get-provinces-info');
    Route::get('/get-nearest-farms', [FarmController::class,
'getNearestFarm'])->name('get-nearest-farms');
    Route::post('/update-farm-borders', [FarmController::class,
'updateFarmBorders'])->name('update-farm-borders');
    Route::get('/get-person-farms', [FarmController::class,
'getPersonFarms'])->name('get-person-farms');
    Route::get('/get-farm-by-point', [FarmController::class,
'getFarmByPoint'])->name('get-farm-by-point');
    Route::get('/save-farm-features', [FarmController::class,
'saveFarmFeatures'])->name('save-farm-features');
    Route::get('/get-portions-by-region-and-tag',
[FarmController::class, 'getPortionsByRegionAndTag'])->name('get-
portions-by-region-and-tag');
    //    Route::post('/create-new-farm',
[FarmController::class, 'createNewFarm'])->name('create-new-farm');
    Route::get('/get-admin-shapes', [FarmController::class,
'getAdminShapes'])->name('get-admin-shapes');

```

```

        Route::get('/get-fmd-protection-zone', [FarmController::class,
'getFmdProtectionZone'])->name('get-fmd-protection-zone');
        Route::get('/get-user-farms', [FarmController::class,
'getUserFarms'])->name('get-user-farms');
        Route::get('/check-user-farm', [FarmController::class,
'checkUserFarm'])->name('check-user-farm');
        Route::post('/save-animal-csv-file', [FarmController::class,
'saveAnimalCsvFile'])->name('save-animal-csv-file');
        Route::get('/get-animal-csv-files', [FarmController::class,
'getAnimalCsvFiles'])->name('get-animal-csv-files');
        Route::get('/get-state-vet-area-by-bunr',
[FarmController::class, 'getStateVetAreaByBunr'])->name('get-sate-vet-
area-by-bunr');
        Route::post('/create-new-farm-application',
[FarmController::class, 'createNewFarmApplication'])->name('create-new-
farm-application');
        Route::post('/edit-bunr-data', [FarmController::class,
'editBunrData'])->name('edit-bunr-data');
        Route::post('/edit-farm-data', [FarmController::class,
'editFarmData'])->name('edit-farm-data');
        Route::post('/create-new-bunr', [FarmController::class,
'createNewBunr'])->name('create-new-bunr');
        Route::post('/add-file-to-bunr', [FarmController::class,
'addFileToBunr'])->name('add-file-to-bunr');
        Route::get('/generate-bunr-file/{fileName}',
[FarmController::class, 'generateBunrFile'])->name('generate-bunr-
file');
        Route::post('/delete-bunr-certificate', [FarmController::class,
'deleteBunrCertificate'])->name('delete-bunr-certificate');
        Route::get('/search-user-farms', [FarmController::class,
'searchUserFarms'])->name('search-user-farms');
    });

    Route::prefix('csv')->group(function () {
        Route::post('/parse-animals-csv-file',
[CsvFilesController::class, 'parseAnimalsCsvFile'])->name('parse-
animals-csv-file');
        Route::post('/update-animal-csv-file',
[CsvFilesController::class, 'updateAnimalCsvFile'])->name('update-
animals-csv-file');
        Route::post('/create-new-row-in-csv-file',
[CsvFilesController::class, 'createNewRowInCsvFile'])->name('create-new-
row-in-csv-file');
    });

    // Get CSV file parsers
    Route::get('/get-csv-parser-types', [FilesController::class,
'getCsvParserTypes'])->name('get-csv-parser-types');

    //Routes for species
    Route::prefix('species')->group(function () {
        Route::get('/get-species', [SpeciesController::class,
'getAllSpecies'])->name('get-all-species');
        Route::get('/get-species-by-farm', [SpeciesController::class,
'getSpeciesByFarmCountry'])->name('get-all-species-by-farm');
        Route::post('/create-species', [SpeciesController::class,
'createSpecies'])->name('create-species');
        Route::post('/delete-species', [SpeciesController::class,
'deleteSpecies'])->name('delete-species');
    });

```

```

        Route::get('/get-parent-species', [SpeciesController::class,
'getAllParentSpecies'])->name('get-parent-species');
        Route::post('/update-specie', [SpeciesController::class,
'updateSpecie'])->name('update-specie');
        Route::post('/assign-species-to-zones',
[SpeciesController::class, 'assignSpeciesToZones'])->name('assign-
species-to-zones');
        Route::get('/search-species', [SpeciesController::class,
'searchSpecies'])->name('search-species');
        Route::get('/country-species', [SpeciesController::class,
'getSpeciesByCountry'])->name('country-species');
    });

    //Routes for animals
    Route::prefix('animals')->group(function () {
        Route::post('/create-animal', [AnimalsController::class,
'createAnimal'])->name('create-animal');
        Route::post('/delete-animal', [AnimalsController::class,
'deleteAnimal'])->name('delete-animal');
        Route::post('/update-animal', [AnimalsController::class,
'updateAnimal'])->name('update-animal');
        Route::get('/get-all-animal-tags', [AnimalsController::class,
'getAllTags'])->name('get-all-animal-tags');
        Route::get('/get-user-animal-tags', [AnimalsController::class,
'getUserAnimalTags'])->name('get-user-animal-tags');
        Route::get('/create-animal-tag', [AnimalsController::class,
'createAnimalTag'])->name('create-animal-tag');
        Route::post('/delete-animal-tag', [AnimalsController::class,
'deleteAnimalTag'])->name('delete-animal-tag');
        Route::get('/update-animal-tags', [AnimalsController::class,
'updateAnimalTags'])->name('update-animal-tags');
        Route::get('/get-animals-by-isolation-id',
[AnimalsController::class, 'getAnimalsByIsolationId'])->name('get-
animals-by-farm-bunr');
        Route::get('/get-animals-by-isolation-id-and-bunr-uuid',
[AnimalsController::class, 'getAnimalsByIsolationIdAndBunrUuid'])->
name('get-animals-by-isolation-id-and-bunr-uuid');
        Route::get('/search-animal-tag', [AnimalsController::class,
'searchAnimalTags'])->name('search-animal-tags');
        Route::get('/search-animal-tag-with-gender',
[AnimalsController::class, 'searchAnimalTagsWithGender'])->name('search-
animal-tags-with-gender');
        Route::get('/search-user-animal-tag', [AnimalsController::class,
'searchUserAnimalTag'])->name('search-user-animal-tag');
        Route::post('/create-external-tags', [AnimalsController::class,
'createExternalTag'])->name('create-external-tags');
        Route::delete('/delete-external-tag', [AnimalsController::class,
'deleteExternalTag'])->name('delete-external-tags');
        Route::post('/edit-external-tag', [AnimalsController::class,
'editExternalTag'])->name('edit-external-tags');
        Route::get('/get-all-animals', [AnimalsController::class,
'getAllAnimals'])->name('get-all-animals');
        Route::get('/search-animals', [AnimalsController::class,
'searchAnimals'])->name('search-animals');
        Route::get('/get-animal-by-tag', [AnimalsController::class,
'getAnimalByTag'])->name('get-animal-by-tag');
        Route::get('/get-animal-by-uuid', [AnimalsController::class,
'getAnimalByUuid'])->name('get-animal-by-uuid');
        Route::get('/get-animal-family-tree-by-uuid',

```

```

[AnimalsController::class, 'getAnimalFamilyTreeByUuid']->name('get-
animal-family-tree-by-uuid');
    Route::get('/edit-animal-tag', [AnimalsController::class,
'editAnimalTag']->name('edit-animal-tag');
    Route::post('/save-animals-from-csv', [AnimalsController::class,
'saveAnimalsFromCsv']->name('save-animals-from-csv');
    Route::post('/create-measurement-for-animal',
[AnimalsController::class, 'createMeasurementForAnimal']->name('create-
measurement-for-animal');
    Route::get('/get-measurement-types',
[MeasurementController::class, 'getMeasurementTypes']->name('get-
measurement-types');
    Route::get('/get-measurement-by-type',
[MeasurementController::class, 'getMeasurementByType']->name('get-
measurement-by-type');
    Route::get('/get-all-measurements',
[MeasurementController::class, 'getAllMeasurements']->name('get-all-
measurements');
    Route::post('/create-measurement-type',
[MeasurementController::class, 'createMeasurementType']->name('create-
measurement-type');
    Route::post('/assign-measurement-to-animal',
[MeasurementController::class, 'assignMeasurementToAnimal']->
>name('assign-measurement-to-animal');
    Route::get('/get-animal-ownership-history',
[AnimalsController::class, 'getAnimalOwnershipHistory']->name('get-
animal-ownership-history');
    Route::get('/get-animal-history-by-history-id',
[AnimalsController::class, 'getAnimalHistoryByHistoryId']->name('get-
animal-history-by-history-id');
    Route::get('/get-animal-images', [AnimalsController::class,
'getAnimalImages']->name('get-animal-images');
    Route::post('/upload-images', [AnimalsController::class,
'uploadImages']->name('upload-animal-images');
    Route::post('/delete-animal-image', [AnimalsController::class,
'deleteAnimalImage']->name('delete-animal-image');
    });

    //Routes for animal parent groups
    Route::prefix('animal-parent-groups')->group(function () {
        Route::get('/create-parent-group',
[AnimalParentGroupsController::class, 'getAnimalParentGroups']->
>name('create-animal-parent-group');
        Route::get('/set-parents', [AnimalParentGroupsController::class,
'getAnimalParentGroups']->name('set-parents');
        Route::get('/add-animal-to-family-tree',
[AnimalParentGroupsController::class, 'addAnimalToFamilyTree']->
>name('add-animal-to-family-tree');
    });

    //Routes for diseases
    Route::prefix('diseases')->group(function () {
        Route::get('/get-all-diseases', [DiseasesController::class,
'getAllDiseases']->name('get-all-diseases');
        Route::get('/get-disease-by-name', [DiseasesController::class,
'getDiseaseByName']->name('get-disease-by-name');
        Route::get('/create-disease', [DiseasesController::class,
'createDisease']->name('create-disease');
        Route::get('/search-disease', [DiseasesController::class,

```

```

'searchDisease']->name('search-disease');
    Route::get('/get-parent-diseases', [DiseasesController::class,
'getParentDiseases']->name('get-parent-diseases'));
    Route::post('/delete-disease', [DiseasesController::class,
'deleteDisease']->name('delete-disease'));
    Route::post('/update-disease', [DiseasesController::class,
'updateDisease']->name('update-disease'));
});

//Routes for safari tags
Route::prefix('safari-tags')->group(function () {
    Route::get('/get-all-safari-tags', [SafariTagsController::class,
'getAllSafariTags']->name('get-all-safari-tags'));
    Route::get('/get-safari-tag-by-id',
[SafariTagsController::class, 'getSafariTagById']->name('get-safari-
tag-by-id'));
    Route::get('/get-safari-tag-by-tag',
[SafariTagsController::class, 'getSafariTagByTag']->name('get-safari-
tag-by-tag'));
    Route::post('/create-safari-tag', [SafariTagsController::class,
'createSafariTag']->name('create-safari-tag'));
    Route::post('/delete-safari-tag', [SafariTagsController::class,
'deleteSafariTag']->name('delete-safari-tag'));
    Route::get('/get-safari-tag-groups',
[SafariTagsController::class, 'getSafariTagGroups']->name('get-safari-
tag-groups'));
    Route::get('/search-safari-tag-group',
[SafariTagsController::class, 'searchSafariTagGroup']->name('search-
safari-tag-group'));
    Route::get('/get-bunr-safari-tags',
[SafariTagsController::class, 'getBunrSafariTags']->name('get-bunr-
safari-tags'));
    Route::get('/add-safari-tag-to-animal',
[SafariTagsController::class, 'addSafariTagToAnimal']->name('add-
safari-tag-to-animal'));
});

//Routes for isolations
Route::prefix('isolations')->group(function () {
    Route::get('/create-isolation', [IsolationsController::class,
'createIsolation']->name('create-isolation'));
    Route::get('/update-isolation', [IsolationsController::class,
'updateIsolation']->name('update-isolation'));
    Route::get('/get-all-isolations', [IsolationsController::class,
'getAllIsolations']->name('get-all-isolations'));
    Route::get('/get-isolations-for-specific-persons',
[IsolationsController::class, 'getIsolationsForSpecificPersons']->
>name('get-isolations-for-specific-persons');
    Route::post('/delete-isolation', [IsolationsController::class,
'deleteIsolation']->name('delete-isolation'));
    Route::get('/get-isolation-by-bunr',
[IsolationsController::class, 'getIsolationByBuNr']->name('get-
isolation-by-bunr'));
    Route::get('/get-isolation-by-id', [IsolationsController::class,
'getIsolationById']->name('get-isolation-by-id'));
    Route::get('/get-isolation-animals-by-bunr',
[IsolationsController::class, 'getIsolationAnimalsByIsolationId']->
>name('get-isolation-animals-by-bunr'));
    Route::post('/save-to-isolation', [IsolationsController::class,

```

```

'saveToIsolation']->name('save-to-isolation');
    Route::post('/remove-from-isolation',
[IsolationsController::class, 'removeFromIsolation']->name('remove-
from-isolation'));
    Route::post('/search-isolation', [IsolationsController::class,
'searchIsolation']->name('search-isolation');
    Route::post('/lock-isolation', [IsolationsController::class,
'lockIsolation']->name('lock-isolation');
    Route::post('/unlock-isolation', [IsolationsController::class,
'unlockIsolation']->name('unlock-isolation');
    Route::post('/create-isolation-round',
[IsolationsController::class, 'createIsolationRound']->name('create-
isolation-round');
    Route::post('/save-protocol-to-isolation-round',
[IsolationsController::class, 'saveProtocolToIsolationRound']->
>name('save-protocol-to-isolation-round');
    Route::get('/get-isolation-round-by-isolation-id',
[IsolationsController::class, 'getIsolationRoundByIsolationId']->
>name('get-isolation-round-by-isolation-id');
    Route::get('/get-lab-test-results-headers',
[IsolationsController::class, 'getTestLabResultsHeaders']->name('get-
lab-test-results-headers');
    Route::get('/get-citt-results', [IsolationsController::class,
'getCittResults']->name('get-citt-results');
    Route::get('/get-lab-test-results-items',
[IsolationsController::class, 'getTestLabResultsItems']->name('get-lab-
test-results-items');
    Route::get('/save-lab-to-isolation-round-test',
[IsolationsController::class, 'saveLabToRoundTest']->name('save-lab-to-
isolation-round-test');
    Route::get('/get-isolation-animals-by-bunr-and-species',
[IsolationsController::class, 'getIsolationAnimalsByBuNrAndSpecies']->
>name('get-isolation-animals-by-bunr-and-species');
    Route::post('/save-round-test-result',
[IsolationsController::class, 'saveRoundTestResult']->name('save-round-
test-result');
    Route::post('/add-new-round', [IsolationsController::class,
'addNewRound']->name('add-new-round');
    Route::get('/get-round-files', [IsolationsController::class,
'getRoundFiles']->name('get-round-files');
    Route::post('/save-parent-test-result-file',
[IsolationsController::class, 'saveParentTestResultFile']->name('save-
parent-test-result-file');
    Route::post('/set-negative-result',
[IsolationsController::class, 'setNegativeResult']->name('set-negative-
result');
    Route::post('/save-animal-test-results-file',
[IsolationsController::class, 'saveTestResultsFile']->name('save-
animal-test-results-file');
    Route::post('/save-animal-test-results-file-for-specific-
result', [IsolationsController::class,
'saveTestResultsFileForSpecificResult']->name('save-animal-test-
results-file-for-specific-result');
    Route::post('/save-tb-results', [IsolationsController::class,
'saveTBResults']->name('save-tb-results');
    Route::post('/save-test-results-file',
[IsolationsController::class, 'saveTestResultsFile']->name('save-test-
results-file');
    Route::get('/get-available-protocol-tests',

```

```

[IsolationsController::class, 'getAvailableProtocolTests']->name('get-
available-protocol-tests');
    Route::post('/update-animal-order',
[IsolationsController::class, 'updateAnimalOrder']->name('update-
animal-order'));
    Route::get('/get-available-isolations-for-user',
[IsolationsController::class, 'getAvailableIsolationsForUser']->
>name('get-available-isolations-for-user');
    Route::get('/get-lab-user-isolation',
[IsolationsController::class, 'getLabUserIsolation']->name('get-lab-
user-isolation'));
    Route::get('/get-user-specific-round',
[IsolationsController::class, 'getUserSpecificRound']->name('get-user-
specific-round');
    Route::get('/search-available-isolations-for-user',
[IsolationsController::class, 'searchAvailableIsolationsForUser']->
>name('search-available-isolations-for-user');
    Route::get('/get-isolation-statistics',
[IsolationsController::class, 'getIsolationStatistics']->name('get-
isolation-statistics');
    Route::post('/edit-isolation-test-result',
[IsolationsController::class, 'editIsolationTestResult']->name('edit-
isolation-test-result');
    Route::get('/create-institute-ref',
[IsolationsController::class, 'createInstituteRef']->name('create-
institute-ref');
    Route::get('/create-lab-ref', [IsolationsController::class,
'createLabRef']->name('create-lab-ref');
    Route::get('/get-round-refs', [IsolationsController::class,
'getRoundRefs']->name('get-round-refs');
    Route::get('/update-round-refs', [IsolationsController::class,
'updateRoundRefs']->name('update-round-refs');
    Route::post('/add-new-animals-in-round',
[IsolationsController::class, 'addNewAnimalsInRound']->name('add-new-
animals-in-round');
    Route::post('/update-owner-ref', [IsolationsController::class,
'updateOwnerReference']->name('update-owner-ref');
    Route::get('/get-round-animals', [IsolationsController::class,
'getRoundAnimals']->name('get-round-animals');
    Route::post('/delete-animals-from-round',
[IsolationsController::class, 'deleteAnimalsFromRound']->name('delete-
animals-from-round');
    Route::get('/update-citt-extra-data',
[IsolationsController::class, 'updateCittExtraData']->name('update-
citt-extra-data');
    Route::get('/get-citt-tester', [IsolationsController::class,
'getCittTester']->name('get-citt-tester');
    Route::post('/sign-citt-results', [IsolationsController::class,
'signCittResults']->name('sign-citt-results');
    Route::get('/set-extra-test-date', [IsolationsController::class,
'setExtraTestDate']->name('set-extra-test-date');
    Route::get('/get-extra-test-dates',
[IsolationsController::class, 'getExtraTestDates']->name('get-extra-
test-dates');
    Route::get('/update-extra-test-dates',
[IsolationsController::class, 'updateExtraTestDates']->name('update-
extra-test-dates');
    Route::get('/set-purpose-of-testing',
[IsolationsController::class, 'setPurposeOfTesting']->name('set-

```

```

purpose-of-testing');
    Route::get('/save-specimens', [IsolationsController::class,
'saveSpecimens'])->name('save-specimens');
    Route::get('/get-purpose-of-testing-and-specimens',
[IsolationsController::class, 'getPurposeOfTestingAndSpecimens'])->
name('get-purpose-of-testing-and-specimens');
    Route::get('/get-all-round-details',
[IsolationsController::class, 'getAllRoundDetails'])->name('get-all-
round-details');
    Route::post('/save-sample-ref', [IsolationsController::class,
'saveSampleRef'])->name('save-sample-ref');
    Route::get('/get-isolation-by-bunr-uuid',
[IsolationsController::class, 'getIsolationByBuNrUuid'])->name('get-
isolation-by-bunr-uuid');
    });

    //Routes for tests
    Route::prefix('tests')->group(function () {
        Route::get('/get-all-tests', [TestsController::class,
'getAllTest'])->name('get-all-tests');
        Route::get('/get-parent-tests', [TestsController::class,
'getParentTests'])->name('get-parent-tests');
        Route::get('/get-child-tests', [TestsController::class,
'getChildTests'])->name('get-child-tests');
        Route::get('/create-test', [TestsController::class,
'createTest'])->name('create-test');
        Route::get('/search-test', [TestsController::class,
'searchTest'])->name('search-test');
        Route::post('/delete-test', [TestsController::class,
'deleteTest'])->name('delete-test');
        Route::get('/edit-test', [TestsController::class, 'editTest'])->
name('edit-test');
    });

    //Routes for test methods
    Route::prefix('test-methods')->group(function () {
        Route::get('/get-all-test-methods',
[TestMethodsController::class, 'getAllTestMethod'])->name('get-all-test-
methods');
        Route::get('/create-test-method', [TestMethodsController::class,
'createTestMethod'])->name('create-test-method');
        Route::get('/search-test-method', [TestMethodsController::class,
'searchTestMethod'])->name('search-test-method');
        Route::post('/delete-test-method',
[TestMethodsController::class, 'deleteTestMethod'])->name('delete-test-
method');
        Route::get('/get-test-methods-by-test-id',
[TestMethodsController::class, 'getTestMethodsByTestId'])->name('get-
test-methods-by-test-id');
        Route::get('/get-all-test-methods-list',
[TestMethodsController::class, 'getAllTestMethodsList'])->name('get-all-
test-methods-list');
        Route::post('/update-test-method',
[TestMethodsController::class, 'updateTestMethod'])->name('update-test-
method');
    });

    //Routes for test results
    Route::prefix('test-results')->group(function () {

```

```

        Route::get('/get-all-test-result-values',
[TestResultController::class, 'getAllTestResultValues'])->name('get-all-
test-result-values');
        Route::post('/store-results', [TestResultController::class,
'storeTestResults'])->name('store-results');
        Route::get('/get-specific-test-results',
[TestResultController::class, 'getSpecificTestResults'])->name('get-
specific-test-results');
    });

    //Routes for rcp
    Route::prefix('rcp')->group(function () {
        Route::post('/create-rcp', [RcpController::class, 'createRcp'])->
name('create-rcp');
        Route::get('/get-rcp-by-uid', [RcpController::class,
'getRcpByUid'])->name('get-rcp-by-uid');
        Route::get('/get-all-rcps', [RcpController::class,
'getAllRcps'])->name('get-all-rcps');
        Route::get('/save-rcp-driver', [RcpController::class,
'saveRcpDriver'])->name('save-rcp-driver');
        Route::post('/save-rcp-departure', [RcpController::class,
'saveRcpDeparture'])->name('save-rcp-departure');
        Route::get('/save-rcp-arrival', [RcpController::class,
'saveRcpArrival'])->name('save-rcp-arrival');
        Route::get('/set-seal-status', [RcpController::class,
'setSealStatus'])->name('set-seal-status');
        Route::post('/upload-rcp-file', [RcpController::class,
'uploadRcpFile'])->name('upload-rcp-file');
        Route::post('/save-rcp-vehicles', [RcpController::class,
'saveVehicleForRcp'])->name('save-rcp-vehicles');
        Route::post('/edit-rcp-vehicles-list', [RcpController::class,
'editRcpVehiclesList'])->name('edit-rcp-vehicles-list');
        Route::post('/confirm-rcp', [RcpController::class,
'confirmRcp'])->name('confirm-rcp');
        Route::get('/remove-rcp-file', [RcpController::class,
'removeRcpFile'])->name('remove-rcp-file');
        Route::get('/edit-rcp-driver', [RcpController::class,
'editRcpDriver'])->name('edit-rcp-driver');
    });

    // Route for testing protocols
    Route::prefix('testing-protocols')->group(function () {
        Route::post('/create-protocol',
[TestingProtocolController::class, 'createProtocol'])->name('create-
protocol');
        Route::get('/get-all-protocols',
[TestingProtocolController::class, 'getAllProtocols'])->name('get-all-
protocols');
        Route::get('/get-protocol-by-name',
[TestingProtocolController::class, 'getProtocolByName'])->name('get-
protocol-by-name');
        Route::get('/get-protocol-test-methods',
[TestingProtocolController::class, 'getProtocolTestMethods'])->
name('get-protocol-test-methods');
        Route::get('/get-protocol-by-species-name',
[TestingProtocolController::class, 'getProtocolBySpeciesName'])->
name('get-protocol-by-species-name');
        Route::post('/update-protocol-name',
[TestingProtocolController::class, 'updateProtocolName'])->name('update-

```

```

protocol-name');
});

//Routes for text editor
Route::prefix('text-editor')->group(function () {
    Route::get('/get-entities', [TextEntitiesController::class,
'getTextEntities'])->name('get-entities');
    Route::post('/save-{entity}', [TextEntitiesController::class,
'saveAgreement'])->name('save-agreement');
    Route::get('/get-texts-by-entity-name',
[TextEntitiesController::class, 'getTextsByEntityName'])->name('get-
text-by-entity-name');
    Route::post('/edit-text', [TextEntitiesController::class,
'editText'])->name('edit-text');
    Route::get('/get-text-entity-files',
[TextEntitiesController::class, 'getTextEntityFiles'])->name('get-text-
entity-files');
    Route::post('/delete-files', [TextEntitiesController::class,
'deleteFilesFromEntity'])->name('delete-files');
});

//Routes for files
Route::prefix('files')->group(function () {
    Route::post('/store-files', [FilesController::class,
'storeFiles'])->name('store-files');
    Route::post('/delete-file', [FilesController::class,
'deleteFile'])->name('delete-file');
    Route::get('/save-file', [FilesController::class, 'saveFile'])->
name('save-file');
    Route::get('/get-all-files', [FilesController::class,
'getAllFiles'])->name('get-all-files');
    Route::get('/get-files-by-file-type', [FilesController::class,
'getFilesByFileType'])->name('get-files-by-file-type');
    Route::get('/get-file-types', [FilesController::class,
'getFileTypes'])->name('get-file-types');
    Route::get('/get-all-file-types', [FilesController::class,
'getAllFileTypes'])->name('get-all-file-types');
    Route::get('/get-file-by-name', [FilesController::class,
'getFileByName'])->name('get-file-by-name');
    Route::get('/download-file/{fileName}', [FilesController::class,
'downloadFile'])->name('download-file');
    Route::post('/store-file', [FilesController::class,
'storeFile'])->name('storeFile');
    Route::get('/search-file', [FilesController::class,
'searchFile'])->name('search-file');
    // Route::post('/save-person-file',
[FilesController::class, 'savePersonFile'])->name('save-person-file');
    Route::get('/get-person-files', [FilesController::class,
'getPersonFiles'])->name('get-person-files');
    Route::get('/search-person-file', [FilesController::class,
'searchPersonFile'])->name('search-person-file');
    Route::get('/get-file-path', [FilesController::class,
'getFilePath'])->name('get-file-path');
    Route::get('/get-lab-pdf-file-types', [FilesController::class,
'getLabPdfFileFileTypes'])->name('get-lab-pdf-file-types');
    Route::get('/get-lab-pdf-keys', [FilesController::class,
'getLabPdfKeys'])->name('get-lab-pdf-keys');
});

```

```

//Routes for labs
Route::prefix('labs')->group(function () {
    Route::post('/store-lab-properties', [LabsController::class,
'storeLabProperties'])->name('store-lab-properties');
    Route::get('/get-lab-properties', [LabsController::class,
'getLabProperties'])->name('get-lab-properties');
    Route::get('/get-available-lab-by-test-name',
[LabsController::class, 'getAvailableLabsByTestName'])->name('get-
available-lab-by-test-name');
    Route::get('/get-available-lab-methods-by-animal-tag',
[LabsController::class, 'getAvailableLabMethodsByAnimalTag'])->
>name('get-available-lab-methods-by-animal-tag');
});

//Routes for lab roles
// Route::prefix('lab-user-roles')->group(function() {
//     Route::get('/get-lab-user-role-permissions',
[PersonUserRolesController::class, 'getLabUserRolePermissions'])->
>name('get-lab-user-role-permissions');
//     Route::get('/get-lab-isolations',
[PersonUserRolesController::class, 'getIsolationsByLabId'])->name('get-
lab-isolations');
//     Route::post('/create-lab-user-role',
[PersonUserRolesController::class, 'createLabUserRole'])->name('create-
lab-user-role');
//     Route::get('/get-all-lab-user-roles',
[PersonUserRolesController::class, 'getAllLabUserRoles'])->name('get-
all-lab-user-roles');
//     Route::post('/edit-lab-user-role',
[PersonUserRolesController::class, 'editLabUserRole'])->name('edit-lab-
user-role');
//     Route::post('/delete-lab-user-role',
[PersonUserRolesController::class, 'deleteLabUserRole'])->name('delete-
lab-user-role');
//     Route::post('/set-lab-role-for-user',
[PersonUserRolesController::class, 'setLabRoleForUser'])->name('/set-
lab-role-for-user');
//     Route::post('/edit-lab-role-for-user',
[PersonUserRolesController::class, 'editLabRoleForUser'])->name('/edit-
lab-role-for-user');
// });

//Routes for person roles
Route::prefix('person-roles')->group(function () {
    Route::get('/get-person-user-roles',
[PersonUserRolesController::class, 'getPersonUserRoles'])->name('get-
person-user-roles');
    Route::get('/set-person-user-role-for-user',
[PersonUserRolesController::class, 'setPersonUserRoleForUser'])->
>name('set-person-user-role-for-user');
    Route::get('/edit-users-person-role',
[PersonUserRolesController::class, 'editUsersPersonRole'])->name('edit-
users-person-role');
    Route::post('/create-person-user-role',
[PersonUserRolesController::class, 'createPersonUserRole'])->
>name('create-person-user-role');
    Route::post('/update-person-user-role',
[PersonUserRolesController::class, 'updatePersonUserRole'])->
>name('update-person-user-role');
});

```

```

        Route::get('/edit-users-person-role',
[PersonUserRolesController::class, 'editUsersPersonRole'])->name('edit-
users-person-role');
        Route::get('/get-lab-user-permissions',
[PersonUserRolesController::class, 'getLabUserPermissions'])->
>name('/get-lab-user-permissions');
        Route::delete('/delete-person-user-role',
[PersonUserRolesController::class, 'deletePersonUserRole'])->
>name('/delete-person-user-role');
    });

    //Routes for movements
    Route::prefix('movements')->group(function () {
        Route::post('/create-movement', [MovementsController::class,
'createMovement'])->name('create-movement');
        Route::get('/get-all-movements', [MovementsController::class,
'getAllMovements'])->name('get-all-movements');
        Route::get('/get-movement-by-ref', [MovementsController::class,
'getMovementByRef'])->name('get-movement-by-ref');
        Route::post('/search-movement', [MovementsController::class,
'searchMovement'])->name('search-movement');
        Route::post('/search-my-movement', [MovementsController::class,
'searchMyMovement'])->name('search-my-movement');
        Route::post('/save-time-data', [MovementsController::class,
'setTimeData'])->name('save-time-data');
        Route::post('/save-redcross-permits-file',
[MovementsController::class, 'saveRedcrossPermitsFile'])->name('save-
redcross-permits-file');
        Route::post('/save-movement-gates', [MovementsController::class,
'saveMovementGates'])->name('save-movement-gates');
        Route::get('/get-animal-movements', [MovementsController::class,
'getAnimalMovements'])->name('get-animal-movements');
        Route::get('/get-movement-statuses',
[MovementsController::class, 'getMovementStatuses'])->name('get-
movement-statuses');
        Route::get('/decline-movement', [MovementsController::class,
'declineMovement'])->name('decline-movement');
        Route::get('/get-user-movements', [MovementsController::class,
'getUserMovements'])->name('get-user-movements');
        Route::post('/save-movement-auth-file',
[MovementsController::class, 'saveMovementAuthFile'])->name('save-
movement-auth-file');
        Route::get('/verify-movement-auth-file',
[MovementsController::class, 'verifyMovementAuthFile'])->name('verify-
movement-auth-file');
        Route::get('/approve-movement', [MovementsController::class,
'approveMovement'])->name('approve-movement');
    });

    Route::prefix('applications')->group(function () {
        Route::get('/get-all-applications',
[ApplicationsController::class, 'getAllApplications'])->name('get-all-
applications');
        Route::get('/get-all-application-headers',
[ApplicationsController::class, 'getAllApplicationHeaders'])->name('get-
all-application-headers');
        Route::get('/get-bunr-application-by-farm-name',
[ApplicationsController::class, 'getBunrApplicationByFarmName'])->
>name('get-bunr-application-by-farm-name');
    });

```

```

        Route::post('/create-bunr-application',
[ApplicationsController::class, 'createBunrApplication'])->name('create-
bunr-application');
        Route::get('/get-authorization-persons',
[ApplicationsController::class, 'getAuthorizationPersons'])->name('get-
authorization-persons');
    });

    //Route for lab pdf files
    Route::prefix('lab-pdf-files')->group(function () {
        Route::get('/generate-results-
file/{isolationId}/{roundNumber}/{labName}/{testName}',
[LabPdfFilesController::class, 'generateResultsFile'])->name('generate-
results-file');
        Route::get('/generate-movement-
file/{isolationRef}/{movementRef}', [LabPdfFilesController::class,
'generateMovementFile'])->name('generate-movement-file');
    });

    //Route for youtube links
    Route::prefix('youtube-links')->group(function () {
        Route::get('/create-youtube-link',
[YoutubeLinksController::class, 'addYoutubeLink'])->name('create-
youtube-links');
        Route::get('/get-all-youtube-links',
[YoutubeLinksController::class, 'getAllYoutubeLinks'])->name('get-all-
youtube-links');
        Route::get('/search-link', [YoutubeLinksController::class,
'searchLink'])->name('search-link');
        Route::get('/get-youtube-link', [YoutubeLinksController::class,
'getYoutubeLinkByPageLink'])->name('get-youtube-link');
        Route::get('/edit-link', [YoutubeLinksController::class,
'editLink'])->name('edit-link');
        Route::delete('/delete-link', [YoutubeLinksController::class,
'deleteLink'])->name('delete-link');
    });

    //Routes for mysql commands
    Route::prefix('mysql-commands')->group(function () {
        Route::get('/get-all-commands', [MySQLCommandsController::class,
'getAllCommands'])->name('get-all-commands');
        Route::get('/get-command-data', [MySQLCommandsController::class,
'getCommandData'])->name('get-command-data');
    });

    //Signature
    Route::get('/update-signature', [SignatureController::class,
'updateSignature'])->name('update-signature');

    //Google 2FA
    Route::post('/google2fa', [Google2FAController::class,
'getGoogle2FA'])->name('google2fa');
    //Route::post('/google2fa/verif-code', [Google2FAController::class,
'verifCode'])->name('verif-code');
    Route::get('/google2fa/recover-code', [Google2FAController::class,
'recoverGoogle2faCode'])->name('recover-code');

    //Routes for person members
    Route::prefix('person-members')->group(function () {

```

```

        Route::get('get-all-person-members',
[PersonUserMembersController::class, 'getAllMembers'])->name('get-
person-members');
        Route::post('save-person-member',
[PersonUserMembersController::class, 'saveMemberToPerson'])->name('save-
person-member');
        Route::delete('delete-member-from-person',
[PersonUserMembersController::class, 'deleteMemberFromPerson'])->
>name('delete-person-member');
    });

    //Routes for medicines
    Route::prefix('medicine')->group(function () {
        //      Route::group(["middleware"=>[MedicineAccess::class]],
function() {
            Route::get('/get-medicine-info-list',
[MedicineController::class, 'getMedicineInfoList'])->name('get-medicine-
info-list');
            Route::get('/search-medicine', [MedicineController::class,
'searchMedicine'])->name('search-medicine');
            Route::get('/get-medicine', [MedicineController::class,
'getMedicine'])->name('get-medicine');
            Route::post('/create-medicine', [MedicineController::class,
'createMedicine'])->name('create-medicine');
            Route::get('/get-manufacturers', [MedicineController::class,
'getManufacturers'])->name('get-manufacturers');
            Route::post('/delete-medicine', [MedicineController::class,
'deleteMedicine'])->name('delete-medicine');
            Route::post('/set-medicine-for-citt',
[MedicineController::class, 'setMedicineForCITT'])->name('set-medicine-
for-citt');
            Route::get('/get-medicine-for-citt', [MedicineController::class,
'getMedicineForCITT'])->name('get-medicine-for-citt');

            Route::get('/get-medicine-batch-info-list',
[MedicineBatchController::class, 'getMedicineBatchInfoList'])->
>name('get-medicine-batch-info-list');
            Route::get('/search-medicine-batch',
[MedicineBatchController::class, 'searchMedicineBatch'])->name('search-
medicine-batch');
            Route::get('/get-medicine-batch',
[MedicineBatchController::class, 'getMedicineBatch'])->name('get-
medicine-batch');
            Route::post('/create-medicine-batch',
[MedicineBatchController::class, 'createMedicineBatch'])->name('create-
medicine-batch');
            Route::get('/get-manufacturers',
[MedicineBatchController::class, 'getManufacturers'])->name('get-
manufacturers');
            Route::post('/delete-medicine-batch',
[MedicineBatchController::class, 'deleteMedicineBatch'])->name('delete-
medicine-batch');
            Route::get('/get-medicine-batch-for-citt',
[MedicineBatchController::class, 'getMedicineBatchForCITT'])->name('get-
medicine-batch-for-citt');
            Route::post('/set-medicine-batch-for-citt',
[MedicineBatchController::class, 'setMedicineBatchForCITT'])->name('set-
medicine-batch-for-citt');
            Route::get('/search-batch-by-type',

```

```

[MedicineBatchController::class, 'searchBatchByType']->name('search-
batch-by-type');

    Route::get('/get-medicine-batches',
[MedicineBatchController::class, 'getMedicineBatches']->name('get-
medicine-batches');
    Route::get('/get-medicines-list', [MedicineController::class,
'getMedicinesList']->name('get-medicines-list');
    Route::get('/get-medicine-types-list',
[MedicineController::class, 'getMedicineTypesList']->name('get-
medicine-types-list');
    Route::get('/get-medicine-type', [MedicineController::class,
'getMedicineType']->name('get-medicine-type');
    Route::post('/delete-medicine-type', [MedicineController::class,
'deleteMedicineType']->name('delete-medicine-type');
    Route::post('/create-medicine-type', [MedicineController::class,
'createMedicineType']->name('create-medicine-type');
    Route::get('/search-medicine-type', [MedicineController::class,
'searchMedicineType']->name('search-medicine-type');

    Route::post('/update-medicine', [MedicineController::class,
'updateMedicine']->name('update-medicine');
    Route::post('/update-medicine-batch',
[MedicineBatchController::class, 'updateMedicineBatch']->name('update-
medicine-batch');
    Route::post('/update-medicine-type', [MedicineController::class,
'updateMedicineType']->name('update-medicine-type');
    //    });
});

    Route::prefix('devices')->group(function () {
        Route::get('devices-list', [DeviceController::class, 'index']->
name('devices'));
        Route::get('devices/{device}/configure',
[DeviceController::class, 'configure']);
        Route::post('{device}/settings', [DeviceController::class,
'updateSettings']);
        Route::post('{device}/settings/create',
[DeviceController::class, 'createSettingForDevice']);
        Route::post('create', [DeviceController::class, 'create']);
        Route::post('{device}/update', [DeviceController::class,
'update']);
        Route::get('generate-getrequest-response',
[DeviceController::class, 'generateGetRequestResponse']);
        Route::get('get-device-pins', [DeviceController::class,
'getDevicePins']);
        Route::post('link-person', [DeviceController::class,
'linkPersonToDevice']);
        Route::get('/device-persons', [DeviceController::class,
'getDevicePersons']->name('device-persons');
        Route::get('/person-devices', [DeviceController::class,
'getPersonDevices']->name('person-devices');
    });

    Route::group(['middleware' => [ZonesMapAccess::class]], function ()
{
        Route::get('/agro-ecological-zones',
[AgroEcologicalZonesController::class, 'getAgroEcoZones']);
        Route::get('/shapes-abattoirs-zones',

```

```

[ShapesAbattoirsController::class, 'getAbattoirsZones']);
    Route::get('/shapes-fmd-zones', [ShapesFMDController::class,
'getFMDZones']);
    Route::get('/shapes-admins-zones',
[ShapesAdminController::class, 'getShapesAdminZones']);
    Route::post('/assign-zones', [PersonController::class,
'assignZones']);
    Route::get('/provinces', [FarmController::class,
'getProvinces']);
    Route::get('/movement-authorization-levels',
[MovementAuthorizationController::class,
'getMovementAuthorizationLevels']);
    Route::post('/assign-movement-authorization-levels',
[MovementAuthorizationController::class, 'assignLevelsAndZones']);
    Route::get('/user-movement-authorization-zones',
[MovementAuthorizationController::class,
'getUserMovementAuthorizationZones']);
    Route::get('/movement-authorization-zones',
[MovementAuthorizationController::class,
'getMovementAuthorizationZones']);
    Route::get('/get-movement-authorization-level',
[MovementAuthorizationController::class,
'getMovementAuthorizationLevel']);
    Route::get('/get-user-with-level',
[MovementAuthorizationController::class,
'getUserWithMovementAuthorizationLevel']);
    Route::post('/update-user-level',
[MovementAuthorizationController::class,
'updateUserMovementAuthorizationLevel']);
    Route::post('/create-movement-auth-level',
[MovementAuthorizationController::class,
'createMovementAuthorizationLevel']);
    Route::post('/delete-movement-auth-level',
[MovementAuthorizationController::class,
'deleteMovementAuthorizationLevel']);
    Route::get('/get-persons-zones',
[MovementAuthorizationController::class, 'getPersonsZones']);
    Route::get('/get-zones-for-person',
[MovementAuthorizationController::class, 'getZonesForPerson']);
    Route::get('/get-zones-types',
[MovementAuthorizationController::class, 'getZonesTypes']);
    Route::get('/get-persons-by-farms', [PersonController::class,
'getPersonsByFarms']);
    Route::post('/remove-person-zones', [PersonController::class,
'removePersonZones']);
    Route::post('/sign-movement-authorization',
[MovementsController::class, 'signMovementAuthorization']);
    Route::get('/get-movement-authorizations',
[MovementsController::class, 'getMovementAuthorizations']);
    Route::post('/update-persons-chain',
[MovementsController::class, 'updatePersonsChain']);
    Route::get('/get-state-vets', [MovementsController::class,
'getStateVets']);
});

Route::prefix('mails')->group(function () {
    Route::get('/mail-access', [TrackingPixelController::class,
'getMailAccessLogs']);
    Route::get('/search-mail-access',

```

```

[TrackingPixelController::class, 'searchMailAccessLogs']);
    Route::get('/email-access-log', [TrackingPixelController::class,
'getMailAccessLog']);
    });

    Route::get('/countries', [CountriesController::class,
'getAllCountries']);
    Route::get('/search-countries', [CountriesController::class,
'searchCountries']);

    Route::get('/test', [TestController::class, 'test']);
    Route::get('/test-david', [DavidController::class, 'test']);
    Route::get('/test-scanner', function () {
        return view('barcode-scanner');
    });

    // Route::post('/get-scanner-data',
[\App\Http\TelegramBot::class, 'getScannerData'])->name('get-scanner-
data');
});

//Routes for text entity views
Route::get('/get-user-agreement', [TextEntityViewsController::class,
'userAgreementView'])->name('get-user-agreement');
Route::get('/get-person-agreement-text',
[TextEntityViewsController::class, 'personAgreementView'])->name('get-
person-agreement-text');

//Routes for person email verification
Route::get('/verification-person/{email}/{token}',
[VerificationController::class, 'verifyEmail'])->name('verify-person-
email');

Route::get('/verification-user-email/{email}/{token}',
[UserController::class, 'verifyUserEmail'])->name('verification-user-
email');

//Routes for OAuth2.0
Route::prefix('auth')->group(function () {
    Route::get('/reg', [UserController::class, 'regUser'])->name('reg-
user');
    Route::get('/login', [UserController::class, 'loginUser'])->
>name('login-user');
    Route::get('/send-verification-email-mail', [UserController::class,
'sendVerificationMail'])->name('send-verification-email-mail');
    Route::get('/firebase-oauth', [FirebaseOAuth::class,
'FirebaseOAuth'])->middleware('firebase-oauth');
    Route::post('/sign-out', [FirebaseOAuth::class, 'signOut'])->
>name('sign-out');
    Route::get('/get-auth-user', [FirebaseOAuth::class, 'authUser'])->
>name('get-auth-user');
    Route::get('/get-auth-user-with-permissions', [FirebaseOAuth::class,
'authUserWithPermissions'])->name('get-auth-user-with-permissions');
});

//Route::get('/get-texts-by-entity-name',
[TextEntitiesController::class, 'getTextsByEntityName'])->name('get-
text-by-entity-name');

```

```
//Routes for permissions and roles
Route::prefix('permissions')->group(function () {
    Route::get('/get-role-by-id', [UserRoleController::class,
'getRoleId'])->name('get-role-by-id');
});

Route::get('/401', function () {
    return view('non-authorized', [
        'code' => 401,
        'message' => 'You are not authorized',
    ]);
})->name('401');

Route::get('/403', function () {
    return view('non-authorized', [
        'code' => 403,
        'message' => 'Access denied !',
    ]);
})->name('403');

//Route::get('/home',[LandingPageController::class,'showLandingPage']);
Route::get('/sign-in', [ShowAppController::class, 'showApp']);

Route::group([], function () {
    //    Route::get('/', [ShowAppController::class, 'showApp']);
    // If the page is not found (code 404), it automatically follows
    this route and return the main page
    Route::get('{any?}', [ShowAppController::class, 'showApp'])->
where('any', '.*');
});
```

Б.2 Код файлу UserService.php

```
<?php
```

```
namespace App\Services;

use App\Enums\FileTypeEnum;
use App\Enums\MailSubjectsEnum;
use App\Enums\TextEntitiesEnum;
use App\Http\SignatureActions;
use App\MailActions;
use App\Models\Signature;
use App\Models\TelegramUser;
use App\Models\TextEntity;
use App\Models\User;
use App\Models\UserRole;
use App\PdfActions;
use App\Repositories\FilesRepository;
use App\Repositories\TextEntitiesRepository;
use App\Repositories\UserRepository;
use App\TrackingEmailPixel;
use Barryvdh\DomPDF\Facade\Pdf;
use Carbon\Carbon;
```

```

use Dompdf\Exception;
use http\Exception\RuntimeException;
use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Crypt;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Mail;
use Illuminate\Support\Facades\Storage;
use Ramsey\Uuid\Uuid;

class UserService
{
    public function __construct(protected UserRepository
$userRepository, protected TextEntitiesRepository
$textEntitiesRepository, protected FilesRepository
$filesRepository){}

    /**
     * Create user (By Google)
     * @param array $user
     * @return void
     */
    public function createUser($user)
    {
        $guestRole = UserRole::where('role_name', 'guest')-
>get('id');

        $today = Carbon::today();

        $user = [
            'uid' => $user[0]->uid,
            'email' => $user[0]->email,
            'address' => null,
            'phone_number' => $user[0]->phoneNumber,
            'photo_url' => $user[0]->photoUrl,
            'ban_status' => 0,
            'role_id' => $guestRole[0]->id,
            'telegram_id' => null,
            'name' => $user[0]->displayName,
            'email_verified_at' => $today->toDateTimeString(),
        ];
        $this->userRepository->createUser($user);
    }

    public function loginUser($email, $password)
    {
        $user = $this->userRepository->getUserByEmail($email);

        if($user !== null && $user->email_verified_at === null){
            throw new \RuntimeException('Your email unverified
!');
        }

        $this->userRepository->loginUser($email, $password);
    }
}

```

```

    public function regUser($firstName, $surname, $email,
$password, $address, $number)
    {
        $userName = $firstName . ' ' . $surname;
        $hashedPassword = Hash::make($password);
        $guestRoleId = UserRole::where('role_name', 'guest')-
>first()->id;

        $user = [
            'email' => $email,
            'password' => $hashedPassword,
            'name' => $userName,
            'address' => $address,
            'phone_number' => $number,
            'uid' => Uuid::uuid1(),
            'role_id' => $guestRoleId ?? null
        ];

        return $this->userRepository->createUser($user);
    }

    public function linkWithTelegram($userId, $username)
    {
        if(TelegramUser::where('username', $username)->exists()){
            $telegramUserId = TelegramUser::where('username',
$username)->first()->id;
            if(User::where('telegram_id', $telegramUserId)-
>doesntExist()){
                $this->userRepository->editUser($userId,
['telegram_id' => $telegramUserId]);
            }else{
                throw new RuntimeException('This telegram is busy
!');
            }
        }else{
            throw new RuntimeException('Telegram profile doesnt
exists');
        }
    }

    public function googleUserExists($uid)
    {
        return $this->userRepository->googleUserExists($uid);
    }

    public function getUserById($userId)
    {
        return $this->userRepository->getUserById($userId);
    }

    /**
     * Get user by ID
     * @param integer $user_id

```

```

        * @return mixed
        */
    public function getUser($userId)
    {
        return $this->userRepository->getUserById($userId);
    }

    /**
     * Get all users from DB
     * @return
     * \Illuminate\Contracts\Pagination\LengthAwarePaginator
     */
    public function getAllUsers($pages)
    {
        return $this->userRepository->getAllUsers($pages);
    }

    /**
     * Update user by ID
     * @param integer $user_id
     * @param array $params
     * @return void
     */
    public function editUser($user_id, $params)
    {
        $this->userRepository->editUser($user_id, $params);
    }

    /**
     * Delete user by ID
     * @param integer $user_id
     * @return void
     */
    public function deleteUser($user_id)
    {
        $this->userRepository->deleteUser($user_id);
    }

    /**
     * Set role for user
     * @param $userId
     * @param $roleId
     * @return void
     */
    public function setRole($userId, $roleId)
    {
        $this->userRepository->setRole($userId, $roleId);
    }

    public function searchUser($search, $perPage)
    {
        return $this->userRepository->searchUser($search,

```

```

$perPage);
    }

    public function replaceSpecialsInAgreement($email, $time,
$text, $signature)
    {
        $patterns = [
            '###NAME###/' => $email,
            '###DATE###/' => $time,
        ];

        $newText = PdfActions::replaceSpecials($text, $patterns);

        return PdfActions::addImageToPdf($newText, $signature);
    }

    public function agreeWithContract($email, $signature)
    {
        $user = $this->userRepository->getUserByEmail($email);
        $agreement = $this->textEntitiesRepository-
>getTextsByEntityName('user_agreement');

        $carbon = new Carbon();

        $fileName = $email . '_' . time() . '.pdf';
        $filePath = '/public/'.

FileTypesEnum::tryFrom('user_agreement')->value . '/'. $fileName;

        try {
            $newText = $this->replaceSpecialsInAgreement($email,
$carbon->now(), $agreement[0]->text, $signature);
        } catch (\RuntimeException $exception){
            throw new $exception;
        }

        $this->userRepository->agreeWithContract($user->id,
$agreement[0]->entity_id);

        $pdf = PDF::loadHTML($newText);

        try {
            $this->filesRepository->storeFile($filePath, $pdf-
>output(), null);
        } catch (\RuntimeException $exception){
            throw new $exception;
        }

        try {
            $this->filesRepository-
>saveFile(FileTypesEnum::tryFrom('user_agreement')->value,
$fileName, $user->id);
        } catch (\RuntimeException $exception){

```

```

        throw new $exception;
    }

    $data = [
        'email' => $email,
        'title' => 'The Buffalo Company',
        'file_name' => $email . " user\'s agreement.pdf"
    ];

    try {
        PdfActions::sendPdfToEmail('user-agreement-pdf',
$filePath, $data);
    } catch (\RuntimeException $exception) {
        throw new $exception;
    }

    \Illuminate\Support\Facades\Auth::loginUsingId($user->id,
true);

    }

    public function saveSignature($signature)
    {
        $user = User::find(Auth::user()->id);
        $fileName = $user->email . '_signature_' . now() .
'.jpeg';
        $filePath = '/public/' .
FileTypesEnum::tryFrom('signatures')->value . '/' . $fileName;

        $commaPosition = strpos($signature, ',') + 1;
        $base64Str = substr($signature, $commaPosition);
        $imageData = base64_decode($base64Str);

        if(!Storage::exists($filePath)){
            $this->filesRepository->storeFile($filePath,
$imageData, null);
        }

        return Signature::create([
            'file_name' => $fileName,
            'user_id' => $user->id,
            'signature_img' => $signature,
        ]);
    }

    public function uploadSignatureImage($signature)
    {
        if($signature->getMimeType() != 'image/png' && $signature-
>getMimeType() != 'image/jpeg' && $signature->getMimeType() !=
'image/jpg'){
            throw new RuntimeException('Wrong signature file !');
        }
    }

```

```

        $mimeType = str_split($signature->getMimeType(), '6')[1];

        $user = User::find(Auth::user()->id);

        $fileName = $user->email . '_signature_' . now() . '.' .
$mimeType;
        $filePath = '/public/' .
FileTypesEnum::tryFrom('signatures')->value . '/';

        $fileRepository = new FilesRepository();

        $fileRepository->storeFile($filePath, $signature,
$fileName);

        $path = storage_path($filePath . $fileName);

        $type = pathinfo($path, PATHINFO_EXTENSION);
        $data = file_get_contents($signature);

        $base64 = 'data:image/' . $type . ';base64,' .
base64_encode($data);

        if(Signature::where('user_id', $user->id)->exists()){
            Signature::where('user_id', $user->id)->update([
                'file_name' => $fileName,
                'signature_img' => $base64,
            ]);
        }else{
            Signature::create([
                'file_name' => $fileName,
                'user_id' => $user->id,
                'signature_img' => $base64,
            ]);
        }
    }

    public function sendVerifyMail($email)
    {
        $text = TextEntity::where('entity_name',
TextEntitiesEnum::VERIFY_USER_EMAIL->value)->get();
        if($text->first()->text !== null){
            $hash = urlencode(Crypt::encryptString($email));

            $replacedText = preg_replace('/##EMAIL##/', $email,
$text->first()->text);
            $replacedText = preg_replace('/##VERIFLINK##/',
env("APP_URL") . 'verification-user-email/' . $email . '/' .
$hash, $replacedText);

            MailActions::sendMail($email, $replacedText, $text-
>first()->text_name, null, null, null);
        }
    }
}

```

```

public function verifyUserEmail($email, $token)
{
    $token = Crypt::decryptString($token);
    $currentTime = Carbon::now();
    if($email === $token){
        $user = User::where('email', $email)-
>where('email_verified_at', null)->first() ?? null;

        if($user !== null){
            $user->email_verified_at = $currentTime-
>toDateTimeString();
            $user->save();
        }
    }

    redirect()->to('/sign-in');
}

```

ДОДАТОК В
Слайди презентації

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

Дипломна кваліфікаційна робота магістра на тему:

«Дослідження методів логістики та розробка CRM-для тваринницьких ферм»

Виконав:
Степан ДОЦЕНКО,
ст. гр. КНТ-113м

Керівник:
Ольга ГЛАДКОВА,
к.т.н., доцент НУ «ЗП»

Рисунок В.1 – Слайд 1

Об'єкт дослідження – системи та методи логістики та бізнес-процеси діяльності тваринницьких ферм.

Предмет дослідження – CRM-система для керування бізнес-процесами та вирішення логістичних задач для ветеринарів та власників тваринницьких ферм.

Мета роботи – дослідження і розробка вебзастосунку, що надає функціональні можливості для керування бізнес-процесами ветеринарних лабораторій, клінік та власників тваринницьких ферм.

Задачі магістерського дослідження:

- дослідження програмних рішень, які надають аналогічні функції;
- розробка технічного завдання;
- вибір засобів для розробки CRM-системи для тваринницьких ферм;
- проектування структури навігації спт-системи для тваринницьких ферм;
- проектування архітектури спт-системи для тваринницьких ферм;
- конструювання CRM-системи для тваринницьких ферм.

Рисунок В.2 – Слайд 2

Порівняння досліджених CRM-систем для тваринницьких ферм

| Критерій порівняння | «Zoho CRM» | «AgriWebb» | «Farmbrite» |
|--|------------|------------|-------------|
| Облік тварин | + | + | + |
| Журнал ветеринарних оглядів та вакцинацій | + | + | + |
| Управління пасовищами та випасом | – | + | + |
| Картографування ферми | – | + | + |
| Контроль запасів кормів і медикаментів | + | + | + |
| Управління завданнями персоналу | + | + | + |
| Аналітичні звіти | + | + | + |
| Офлайн-доступ | – | + | + |
| Інтеграція з іншими сервісами | + | – | + |
| Управління фінансами та бюджетом | + | – | + |
| Управління добривами і покривними культурами | – | + | + |
| База постачальників ресурсів | + | – | – |
| Модуль для створення техкарт | – | + | + |
| Візуалізація та контроль виробничих процесів | + | + | + |
| Облік прибутків та витрат | + | + | + |
| Управління клієнтськими базами | + | – | + |
| Доступ до мобільного додатку | + | – | – |

Рисунок В.3 – Слайд 3

Порівняння мов програмування для розробки вебзастосунків

| Критерій порівняння | JS | C# | Python | PHP |
|--|----|----|--------|-----|
| Рішення різних завдань | ++ | ++ | ++ | ++ |
| Мультипарадигмальна мова | + | ++ | + | + |
| Інтерпретована мова | + | – | + | + |
| Динамічна типізація | + | – | + | + |
| Зручний синтаксис | + | + | +- | ++ |
| Реєстрозалежна мова | + | + | + | + |
| Високорівнева мова | + | + | + | + |
| Наявність менеджера пакетів | + | + | + | + |
| Прототип-орієнтований стиль | + | – | – | – |
| Велика кількість програмних каркасів для розробки вебзастосунків | ++ | + | ++ | ++ |
| Відносно менші витрати часу | + | +- | + | +- |
| Висока швидкість виконання | + | + | – | + |

Рисунок В.4 – Слайд 4

Порівняння текстових редакторів розробки

| Критерій порівняння | <u>Visual Studio Code</u> | PHP Storm |
|--|---------------------------|-----------|
| Кодування багатьма мовами | + | + |
| Перевірка правильності написання коду | + | + |
| Автоматична генерація каркаса <u>проєкту</u> або окремих фрагментів програмного коду | + | + |
| Вбудована підтримка <u>Git</u> | + | + |
| Вбудований <u>налагоджувач</u> для PHP | – | + |
| Вбудований інтерфейс бази даних | – | + |
| Встановлення розширень | – | + |
| Файловий менеджер | + | + |
| Діагностика помилок | + | + |
| Карта програмного коду | + | + |
| Розділення робочої області | + | + |
| Підтримка декількох курсорів | + | + |
| Вбудований інтерфейс командного рядка | + | + |
| Підтримка <u>Emmet</u> | + | + |
| Можливість роботи з базами даних | – | + |
| Високий рівень безпеки | – | + |
| Можливість роботи з декількома <u>проєктами</u> одночасно | + | + |
| Наявність командного рядка для швидкого виконання різних команд | + | + |
| Рефакторинг | + | + |
| Інтегрований термінал | + | + |
| Можливість тестування | + | + |

5

Рисунок В.5 – Слайд 5

Порівняння інструментів для розробки інтерфейсу користувача

| Критерій порівняння | <u>React</u> | <u>Vue</u> | <u>Angular</u> |
|--|--------------|------------|----------------|
| Мова програмування JS | + | + | – |
| Бібліотека | + | – | – |
| Віртуальна модель DOM | + | + | – |
| Двостороннє зв'язування даних | – | + | + |
| Швидке освоєння інструменту | + | ++ | – |
| Компонентний підхід розробки | + | + | + |
| Наявність вбудованого інструменту для маршрутизації <u>вебсторінок</u> | – | + | + |
| Наявність вбудованого інструменту для керування станами <u>вебзастосунку</u> | – | + | + |
| Розробка середніх <u>вебпроєктів</u> | + | +- | ++ |
| Наявність документації | + | + | + |
| Вбудовані стилі компонентів | + | – | – |
| Робота з REST API | + | + | + |
| Висока швидкість кодування | + | ++ | – |
| Розширена екосистема | + | + | + |

6

Рисунок В.6 – Слайд 6

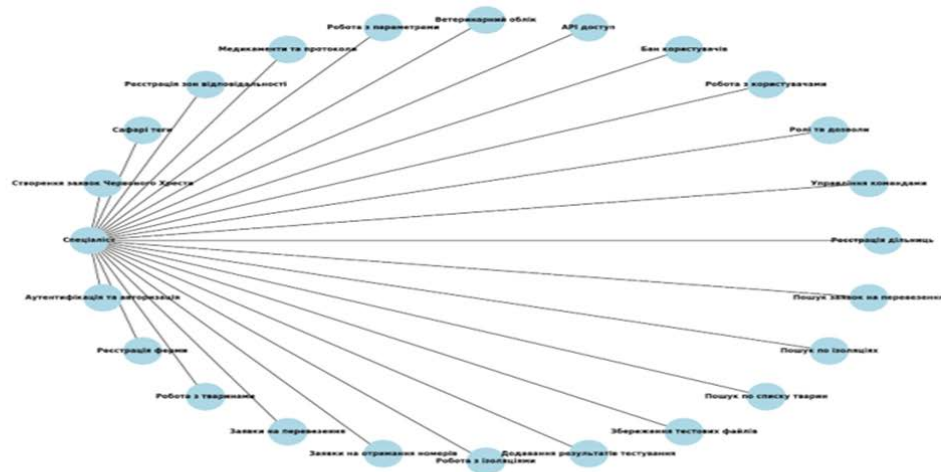
Порівняння СКБД

| Критерій порівняння | <u>MariaDB</u> | <u>PostgreSQL</u> | <u>MySQL</u> |
|---|--------------------|--------------------|--------------------|
| Сумісність з <u>MySQL</u> | ++ | – | ++ |
| Підтримка JSON | + | ++ | + |
| Реплікація | ++ | ++ | ++ |
| Масштабованість | ++ | ++ | + |
| Агрегація даних | ++ | ++ | + |
| Підтримка складних запитів (СТЕ) | + | ++ | – |
| Розробка високонавантажених систем | ++ | ++ | + |
| Швидкість виконання запитів | ++ | ++ | ++ |
| <u>Мультиплатформність</u> | + | + | + |
| Простота використання | ++ | + | ++ |
| Широкий вибір механізмів зберігання | ++ | + | + |
| Розширені можливості безпеки | + | ++ | + |
| Розширюваність через плагіни | + | ++ | + |
| Підтримка <u>геопросторових даних</u> (GIS) | ++ | ++ | + |
| Ліцензування | <u>Open Source</u> | <u>Open Source</u> | <u>Open Source</u> |
| Активність спільноти розробників | ++ | ++ | ++ |

7

Рисунок В.7 – Слайд 7

Схема взаємодії спеціаліста з вебзастосунком для тваринництва



8

Рисунок В.8 – Слайд 8

Розроблений алгоритм функціонування вебзастосунку

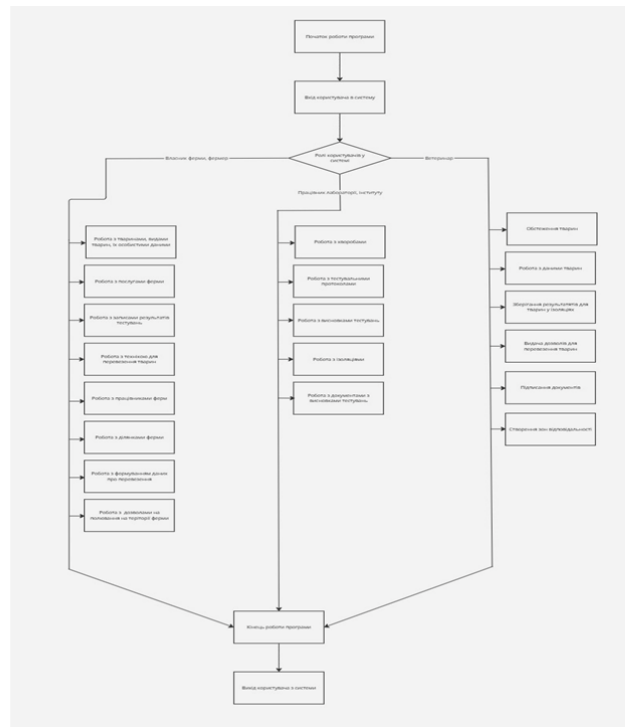


Рисунок В.11 – Слайд 11

Алгоритм виконання перевезення тварин з інтегрованим методом оптимізації логістичних маршрутів

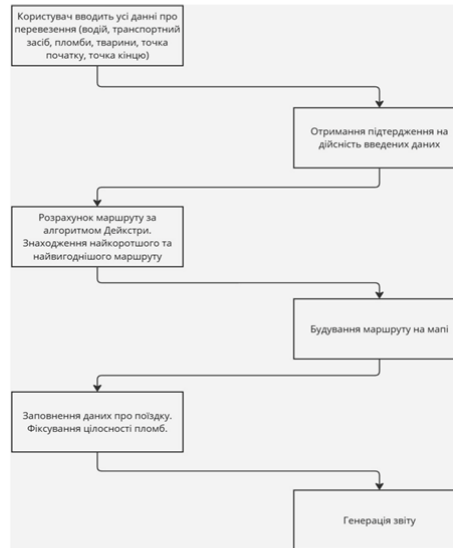


Рисунок В.12 – Слайд 12

Функції інтерфейсної та серверної частин вебзастосунку для керування бізнес-процесами у CRM-системі

| Назва функції | Опис |
|----------------------------|--|
| authUser() | Функція, яка авторизує користувача та зберігає його дані у сесії |
| agreeWithContract() | Функція, що зберігає файл та оновлює запис людини, якщо вона погодилася з користувацьким контрактом, що представлений на сайті |
| saveSignature() | Зберігає підпис користувача у БД |
| createRole() | Створює роль для користувача |
| editRole() | Змінює дозволу, або назву ролі |
| deleteRole() | Видаляє роль |
| banUser() | Банить користувача |
| editUser() | Змінює дані користувача |
| getUser() | Повертає дані про користувача |
| getAllUsers() | Повертає усіх користувачів |
| regUser() | Реєструє користувача |
| loginUser() | Логинить користувача у систему |
| linkWithTelegram() | Створює зв'язок з користувачем телеграму та користувачем сайту |
| createPerson() | Створює сутність компанія, лабораторія, або приватний ветеринар |
| getPerson() | Повертає сутність сутність компанія, лабораторія, або приватний ветеринар |
| deletePerson() | Видаляє сутність компанія, лабораторія, або приватний ветеринар |
| verifyPerson() | Верифікує електронну адресу сутності |
| addUserToPerson() | Додає до сутності користувачів (робітників) |
| createAnimalTag() | Створює мікрочип для тварини |
| createSafariTag() | Створює сафарі тегу |
| getFarm() | Повертає ферму |
| getFarmPin() | Повертає центр ферми за координатами |
| getFarmPortions() | Повертає окремі ділянки ферми |
| createFarmApplication() | Створює заявки на реєстрацію ферми |
| createWarthogApplication() | Створює заявки на дозвіл полювання на кабанів |
| createDisease() | Створює хворобу |
| createDiseaseProtocol() | Створює протокол хвороб для тестування |
| createTest() | Створює тест |
| createTestMethod() | Створює метод тестування |
| setTestToProtocol() | Додає тест до протоколу тестування |

| Назва функції | Опис |
|--------------------------|--|
| createIsolation() | Створює ізоляцію для тварин |
| setProtocolToIsolation() | Додає протокол до ізоляції |
| addAnimalToIsolation() | Додає тварину до ізоляції |
| saveAnimalResult() | Зберігає результат тестування тварини |
| saveCITTAnimalResult() | Зберігає результат тестування тварини на туберкульоз |
| createNewFarmPortion() | Створює нову ділянку для ферми |
| uploadIsolationFile() | Зберігає файл з результатами ізоляції |
| uploadAnimalImages() | Зберігає фотографії тварин |
| createMedicine() | Створює ліки та препарати |
| deleteMedicine() | Видаляє ліки |
| editMedicine() | Змінює назву, строк придатності, або виробника ліків |
| setMedicineToIsolation() | Зберігає ліки для ізоляції |

Рисунок В.13 – Слайд 13

Вебсторінка аутентифікації та реєстрації вебзастосунку «BUFFALO.VET»

Рисунок В.14 – Слайд 14

Вебсторінка налаштування ролі «Засновник»

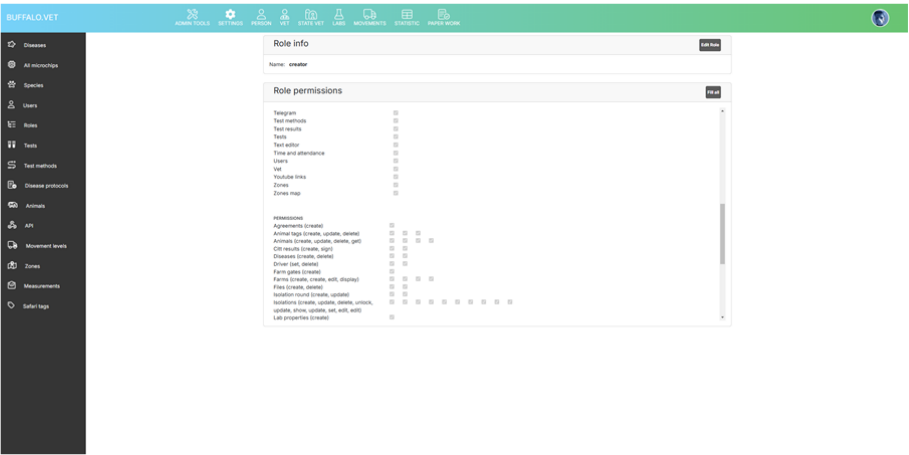


Рисунок В.15 – Слайд 15

Процес створення сутності Особа та присвоєння до неї ферм

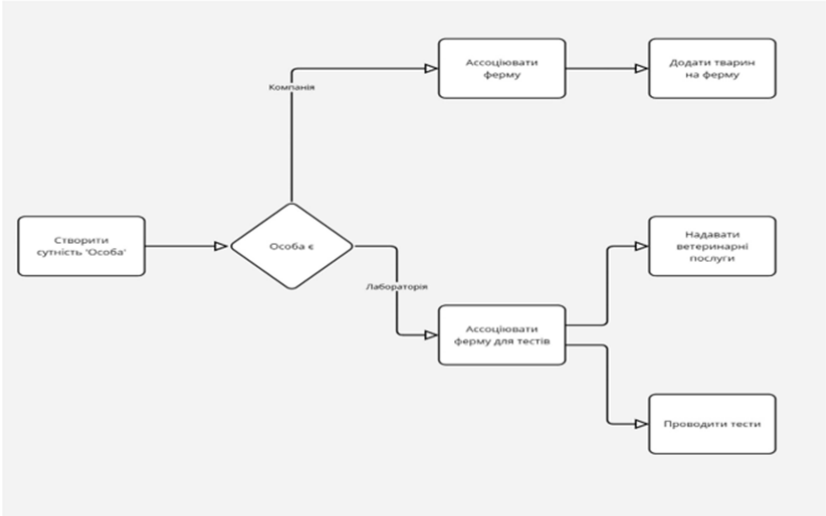
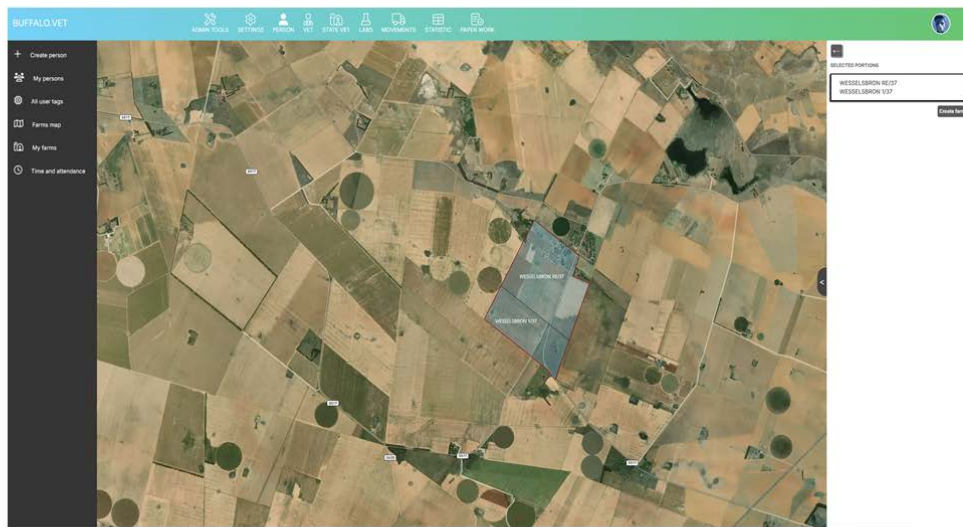


Рисунок В.16 – Слайд 16

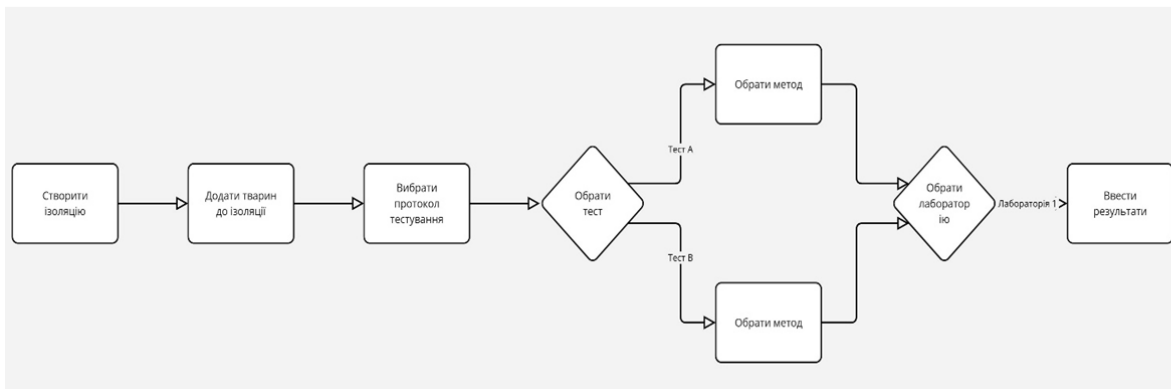
Сторінка реєстрації нової ферми



17

Рисунок В.17 – Слайд 17

Послідовність створення ізоляції для тестування тварин



18

Рисунок В.18 – Слайд 18

ВИСНОВКИ

У процесі виконання роботи було розглянуто, розроблено та проаналізовано функціонал CRM-системи, орієнтованої на потреби тваринницьких ферм. Також було створено структуровану архітектуру, що охоплює всі аспекти роботи ферми, включаючи реєстрацію користувачів, розподіл ролей, управління доступами та інтеграцію з іншими сервісами. На основі принципів архітектурного патерну MVC було реалізовано чітке розділення бізнес-логіки, інтерфейсу та обробки даних.

Розроблена CRM-система враховує специфіку тваринництва, пропонуючи функціонал, що дозволяє вести облік тварин із можливістю зберігання історичних даних, подавати заявки на перевезення та отримання спеціальних номерів, працювати з ізоляціями, тестами та протоколами здоров'я, організовувати роботу ветеринарів та зони їх відповідальності, а також ефективно управляти ролями та доступами для користувачів. Система забезпечує високу безпеку даних завдяки аутентифікації за допомогою Google OAuth 2.0 та можливості гнучкого налаштування прав доступу. Впровадження API дозволяє інтегрувати CRM-систему з іншими платформами для автоматизації більш складних процесів.

Запропоновано інтегрований метод оптимізації логістичних маршрутів, який базується на комбінованому використанні генетичних алгоритмів та моделей теорії ігор. Цей підхід дозволяє одночасно враховувати множинні обмеження, такі як час доставки, витрати на паливо та обсяги вантажу, що дозволяє зменшити загальні витрати на перевезення і знизити час розробки маршрутів. Крім того, метод забезпечує динамічну адаптацію до змін у транспортній мережі.

Практична значимість роботи полягає у створенні інструментів, які можуть бути впроваджені для підвищення ефективності логістичних систем і автоматизації бізнес-процесів. Розроблені підходи до оптимізації маршрутів, прогнозування попиту та управління запасами дозволяють значно скоротити витрати на перевезення, зменшити надлишкові запаси та прискорити виконання логістичних операцій. CRM-система, розроблена для тваринницьких ферм, також має високу практичну значимість, оскільки сприяє автоматизації обліку тварин, управлінню ветеринарними послугами, плануванню ізоляцій і контролю перевезень. Її використання дозволяє покращити координацію між фермерськими господарствами та ветеринарними службами, зменшити кількість помилок у роботі з даними та підвищити загальну продуктивність ферми. Завдяки впровадженню сучасних CRM-рішень фермери отримують можливість ефективно аналізувати інформацію та приймати обґрунтовані рішення.