МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЖИТОМИРСЬКИЙ АГРОТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ

ВІДДІЛЕННЯ
«ІНЖЕНЕРНА ІНФРАСТРУКТУРА ТА КОМП’ЮТЕРНІ НАУКИ»

ЦИКЛОВА КОМІСІЯ СПЕЦІАЛЬНОСТІ
«КОМП'ЮТЕРНІ НАУКИ»

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня фаховий молодший бакалавр
за спеціальністю 122 Комп'ютерні науки
на тему:

«Система обліку використання палива та експлуатаційних витрат сільськогосподарської техніки»

Виконав здобувач освіти групи П-43стн
Степанчук Юрій Петрович

Керівник роботи:
Студзінський Володимир Вікторович

Рецензент:

Житомир – 2026

Зміст

					КРФМБ.122.043стн.021.2026.ПЗ		
Змн.	Арк.	№ докум.	Підпис	Дата			
Розробив		Степанчук Ю П			Система обліку використання палива та експлуатаційних витрат сільськогосподарської техніки	Літ.	Арк. 3 / Аркушів 52
Перевірив		Студзінський В В					
Рецензент		.				ЖАТФК	
Н. Контр.							
Затвердив.		Гнатюк О.Ф.					

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи освітнього ступеня фаховий молодший бакалавр за спеціальністю 122 Комп'ютерні науки на тему «Система обліку використання палива та експлуатаційних витрат сільськогосподарської техніки».

У роботі розроблено спеціалізовану настільну програмну систему для автоматизації обліку витрат палива та експлуатаційних витрат машинно-тракторного парку аграрного підприємства.

Мета — створення ефективного інструменту для реєстрації, зберігання та аналізу даних про споживання ПММ і технічне обслуговування техніки.

Для досягнення мети вирішено завдання: проаналізовано предметну галузь та існуючі рішення, визначено функціональні й нефункціональні вимоги, спроектовано архітектуру на основі патерну MVVM, розроблено реляційну базу даних, реалізовано графічний інтерфейс та модуль генерації PDF-звітів.

Система побудована на платформі .NET 9, мові C#, технології WPF з використанням Entity Framework Core 9, SQLite, бібліотек CommunityToolkit.Mvvm, MaterialDesignThemes та QuestPDF. Забезпечено автономну роботу без Інтернету, фільтрацію даних, автоматичний розрахунок питомих витрат палива та резервне копіювання бази.

Обсяг пояснювальної записки — 52 сторінки, 20 рисунків, 16 таблиць, 2 додатки, 21 джерело.

Ключові слова: облік палива, сільськогосподарська техніка, MVVM, WPF, Entity Framework Core, SQLite, .NET 9.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

EF Core – Entity Framework Core

ERP – Enterprise Resource Planning (система планування ресурсів підприємства)

FK – Foreign Key (зовнішній ключ)

MVVM – Model-View-ViewModel (модель–представлення–модель представлення)

ORM – Object-Relational Mapping (об’єктно-реляційне відображення)

PDF – Portable Document Format

PK – Primary Key (первинний ключ)

.NET – кросплатформна програмна платформа Microsoft .NET

C# – мова програмування C Sharp

SQLite – вбудована реляційна система управління базами даних SQLite

WPF – Windows Presentation Foundation

XAML – Extensible Application Markup Language (розширювана мова розмітки додатків)

МТП – машинно-тракторний парк

ПММ – паливно-мастильні матеріали

СУБД – система управління базами даних

ТО – технічне обслуговування

ФВ – функціональні вимоги

л/год – літрів на годину

га – гектар

ВСТУП

Сільськогосподарське виробництво є однією з ключових галузей національної економіки, у якій ефективність виробничих процесів безпосередньо залежить від рівня технічного забезпечення та раціонального використання матеріальних ресурсів. Сучасні аграрні підприємства функціонують в умовах значного навантаження на машинно-тракторний парк, де паливно-мастильні матеріали та витрати на технічне обслуговування формують суттєву частку собівартості сільськогосподарської продукції.

Відсутність систематизованого обліку витрат на експлуатацію техніки призводить до неконтрольованого зростання виробничих витрат, унеможливлює об'єктивний аналіз ефективності використання окремих одиниць парку та ускладнює прийняття управлінських рішень щодо планування технічного обслуговування й оновлення матеріально-технічної бази. Зазначені проблеми зумовлюють нагальну потребу у впровадженні спеціалізованих інформаційних систем для автоматизації обліку палива та експлуатаційних витрат.

Метою кваліфікаційної роботи є проектування та розробка спеціалізованої програмної системи для обліку використання палива та експлуатаційних витрат сільськогосподарської техніки, яка забезпечує автоматизацію процесів реєстрації даних, їх зберігання та формування аналітичної звітності.

Для досягнення поставленої мети в роботі вирішуються такі завдання:

1) аналіз предметної галузі та існуючих підходів до обліку витрат на експлуатацію сільськогосподарської техніки;
2) визначення функціональних і нефункціональних вимог до розроблюваної системи;
3) проектування архітектури програмного забезпечення та структури бази даних;
4) розробка та реалізація програмної системи засобами платформи .NET 9, мови програмування C# та технології Windows Presentation Foundation;

5) тестування розробленої системи та перевірка відповідності результатів встановленим вимогам.

Об'єктом дослідження є процеси обліку та контролю витрат при експлуатації сільськогосподарської техніки в умовах аграрного підприємства.

Предметом дослідження є методи автоматизації обліку витрати палива та експлуатаційних витрат, а також принципи побудови настільних програмних систем для аграрної галузі.

Методи дослідження. У процесі виконання кваліфікаційної роботи використано: методи системного аналізу — для дослідження предметної галузі та формулювання вимог до системи; об'єктно-орієнтований підхід — для проектування моделі даних і архітектури програмного забезпечення; шаблон проектування Model–View–ViewModel (MVVM) — для організації взаємодії між рівнями представлення, бізнес-логіки та даних; методи реляційного моделювання — для проектування схеми бази даних.

Практичне значення роботи полягає у розробці програмної системи, що дозволяє керівникам та спеціалістам аграрних підприємств вести деталізований облік витрат палива в розрізі кожної одиниці техніки та оператора, фіксувати всі види експлуатаційних витрат (технічне обслуговування, поточний і капітальний ремонти, придбання запасних частин), формувати зведені аналітичні звіти у форматі PDF, а також виконувати фільтрацію і пошук даних за заданими критеріями. Впровадження системи сприяє підвищенню прозорості господарської діяльності та дозволяє знизити нераціональні витрати ресурсів.

Програмну реалізацію виконано з використанням платформи .NET 9 та мови програмування C#. Графічний інтерфейс користувача розроблено на основі технології Windows Presentation Foundation (WPF) із застосуванням архітектурного шаблону MVVM та бібліотеки MaterialDesignThemes для забезпечення сучасного вигляду інтерфейсу. Для зберігання даних використано вбудовану реляційну СУБД SQLite у поєднанні з об'єктно-

реляційним відображувачем Entity Framework Core 9. Генерацію звітів у форматі PDF реалізовано із застосуванням бібліотеки QuestPDF.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі проведено аналіз предметної галузі та обґрунтовано вибір засобів розробки. Другий розділ присвячено проектуванню системи: визначенню вимог, побудові структурних діаграм та проектуванню схеми бази даних. У третьому розділі описано практичну реалізацію програмного забезпечення та наведено результати тестування розробленої системи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ВИБІР ЗАСОБІВ РОЗРОБКИ

1.1 Характеристика предметної галузі

Сільськогосподарська техніка є основним виробничим засобом аграрного підприємства. До її складу входять трактори, комбайни зернозбиральні та кормозбиральні, обприскувачі, сівалки, культиватори, плуги, вантажний автотранспорт та допоміжні машини. Кожна одиниця техніки виконує певний перелік технологічних операцій, споживає паливо та потребує регулярного технічного обслуговування.

Виробничий цикл аграрного підприємства охоплює весь календарний рік і передбачає сезонне навантаження на різні типи техніки. У весняний період активно задіяні трактори з ґрунтообробним обладнанням та сівалки; влітку — комбайни та обприскувачі; восени — ґрунтообробна техніка та вантажний транспорт для перевезення врожаю. Відповідно, витрати палива та потреба в технічному обслуговуванні суттєво варіюються залежно від сезону.

Облік витрат на утримання машинно-тракторного парку охоплює кілька категорій витрат. Основною статтею є витрати на паливно-мастильні матеріали, що включають дизельне паливо, бензин, мастильні матеріали та технічні рідини. Другою за значущістю є витрати на технічне обслуговування (ТО) та ремонт: планові ТО виконуються через визначений інтервал моточасів, а поточний і капітальний ремонти — за потребою. Окрему статтю складають витрати на придбання запасних частин, шин, страхування та реєстрацію транспортних засобів.

Ефективна система обліку цих витрат повинна забезпечувати фіксацію кожного факту заправлення техніки із зазначенням кількості та вартості палива, виду виконаних робіт, оператора та оброблюваної площі. Аналогічно мають реєструватися всі факти витрат на технічне обслуговування з документальним підтвердженням. Наявність такої системи дозволяє розраховувати питомі витрати палива на одиницю площі або продуктивності,

порівнювати ефективність різних одиниць техніки та планувати бюджет підприємства.

Таблиця 1.1 — Класифікація сільськогосподарської техніки за типами

Тип техніки	Призначення	Основний вид палива	Одиниця обліку роботи
Трактор	Буксирування агрегатів, ґрунтообробіток	Дизель	Моточаси / га
Комбайн зернозбиральний	Збирання зернових та олійних культур	Дизель	Моточаси / га
Комбайн кормозбиральний	Збирання та подрібнення кормових культур	Дизель	Моточаси / т
Обприскувач	Внесення засобів захисту рослин	Дизель	Га / зміна
Сівалка	Точний висів насіння	Дизель (від трактора)	Га
Вантажний автомобіль	Перевезення зерна, добрив	Дизель	Км / т

1.2 Аналіз існуючих програмних рішень

На ринку програмного забезпечення для аграрних підприємств існує кілька класів рішень для обліку техніки та витрат. Їх можна умовно поділити на три групи: великі комплексні ERP-системи, спеціалізовані агрономічні платформи та універсальні облікові програми.

До першої групи належать ERP-системи класу SAP Agriculture та Microsoft Dynamics 365 Business Central. Ці продукти охоплюють повний спектр управлінського обліку, однак характеризуються значною вартістю впровадження та обслуговування, складністю налаштування під специфіку українського аграрного законодавства та необхідністю постійного підключення до серверної інфраструктури. Їх застосування економічно доцільне лише для великих агрохолдингів.

До другої групи належать платформи на кшталт Cropio, OneSoil та Agri-ERP, які орієнтовані на точне землеробство та управління польовими операціями. Хоча ці системи мають модулі обліку техніки, їх основна функціональність спрямована на геолокаційне відстеження та агрономічне планування, а не на деталізований фінансовий облік витрат.

Третю групу складають бухгалтерські системи — 1С:Підприємство з модулем управління автотранспортом, BAS ERP та аналогічні рішення. Вони забезпечують коректний бухгалтерський облік, але не адаптовані до специфіки сільськогосподарської техніки: не ведуть облік моточасів, не розраховують питомі витрати палива, не мають механізму прив'язки витрат до агрономічних операцій.

Таблиця 1.2 — Порівняльний аналіз існуючих програмних рішень

Критерій	**SAP Agriculture**	**Cropio**	**1С:Підприємство**	**Розроблювана система**
Облік палива за технікою	+	+	Частково	+
Облік моточасів	+	+	–	+
Облік ТО та ремонтів	+	–	+	+
Питомі витрати л/год	–	+	–	+
Генерація PDF-звітів	+	+	+	+
Робота без Інтернету	–	–	+	+
Простота впровадження	–	+	Середня	+
Вартість ліцензії	Висока	Середня	Середня	Відсутня

Проведений аналіз свідчить про відсутність на ринку простого, автономного і безкоштовного інструменту, який би поєднував детальний облік палива, облік експлуатаційних витрат та генерацію звітності в одному настільному застосунку без необхідності постійного інтернет-з'єднання. Саме цю нішу заповнює розроблювана система.

1.3 Обґрунтування вибору засобів розробки

Платформа .NET 9 та мова C#. .NET 9 є актуальною версією кросплатформної платформи від Microsoft, яка забезпечує продуктивність виконання, сучасні можливості мови C# 13 та розвинену екосистему бібліотек. Мова C# підтримує об'єктно-орієнтоване програмування, узагальнене програмування, асинхронні операції (async/await) та патерн-відповідність (pattern matching), що дозволяє писати чистий, лаконічний та підтримуваний код.

Windows Presentation Foundation (WPF). WPF є зрілою технологією побудови графічних інтерфейсів для Windows, яка використовує XAML для декларативного опису інтерфейсу та підтримує прив'язку даних (data binding) і стилізацію. Вибір WPF зумовлений тим, що цільовою операційною системою є Windows, а WPF забезпечує розвинену підтримку патерну MVVM та анімованих компонентів інтерфейсу.

SQLite та Entity Framework Core 9. SQLite є вбудованою реляційною СУБД, що зберігає базу даних у єдиному файлі без необхідності встановлення сервера. Це відповідає вимозі автономності системи та спрощує розгортання. Entity Framework Core (EF Core) — об'єктно-реляційний відображувач (ORM), що дозволяє працювати з базою даних через об'єкти C# без написання SQL-запитів вручну, забезпечуючи типову безпеку та зрозумілість коду.

Таблиця 1.3 — Порівняльний аналіз систем управління базами даних

Характеристика	SQLite	MySQL	PostgreSQL	SQL Server LocalDB
Серверний процес	Не потрібен	Потрібен	Потрібен	Потрібен
Файлове сховище	Один файл .db	Каталог з файлами	Каталог з файлами	Каталог MDF/LDF
Розмір бібліотеки	~500 КБ	~50 МБ	~30 МБ	~150 МБ
Підтримка EF Core	Повна	Повна	Повна	Повна
Ліцензія	Public Domain	GPL/Commercial	PostgreSQL (вільна)	Обмежена
Придатність для desktop	Відмінна	Задовільна	Задовільна	Задовільна

MaterialDesignThemes. Бібліотека MaterialDesignThemes реалізує специфікацію Material Design 3 від Google для WPF-застосунків. Вона надає готовий набір компонентів (кнопки, текстові поля, ComboBox, DataGrid, DatePicker, DialogHost) із сучасним виглядом та анімаціями, що суттєво скорочує час розробки інтерфейсу.

CommunityToolkit.Mvvm. Бібліотека CommunityToolkit.Mvvm забезпечує генерацію шаблонного коду для реалізації патерну MVVM через атрибути [ObservableProperty] та [RelayCommand]. Це дозволяє зосередитись на бізнес-логіці, уникнувши написання повторюваного коду для сповіщень про зміну властивостей.

QuestPDF. QuestPDF є сучасною бібліотекою для генерації PDF-документів у .NET із відкритим вихідним кодом та безкоштовною Community-ліцензією. Вона використовує fluent API для декларативного опису структури документа, підтримує таблиці, графіку, колонтитули та кириличні шрифти.

Висновки до розділу 1

У першому розділі проведено аналіз предметної галузі обліку витрат на сільськогосподарську техніку. Встановлено, що паливо та технічне обслуговування є суттєвими статтями витрат аграрного підприємства, облік яких потребує автоматизації. Аналіз існуючих рішень показав, що на ринку відсутній простий автономний інструмент, орієнтований на потреби середнього агровиробника без значних витрат на впровадження.

Обґрунтовано вибір технологічного стеку: платформи .NET 9, мови C#, технології WPF, СУБД SQLite з ORM Entity Framework Core 9, а також допоміжних бібліотек MaterialDesignThemes, CommunityToolkit.Mvvm та QuestPDF. Зазначений стек забезпечує автономність системи, сучасний інтерфейс, типову безпеку та можливість генерації звітів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ

2.1 Визначення вимог до системи

2.1.1 Функціональні вимоги

Функціональні вимоги визначають, яку поведінку повинна реалізовувати система. На основі аналізу предметної галузі та опитування потенційних користувачів сформовано перелік функціональних вимог, наведений у таблиці 2.1.

Таблиця 2.1 — Функціональні вимоги до системи

Код	Вимога	Пріоритет
ФВ-01	Ведення реєстру одиниць сільськогосподарської техніки з характеристиками (назва, тип, реєстраційний номер, виробник, модель, рік випуску, вартість)	Обов'язковий
ФВ-02	Реєстрація фактів використання палива: дата, техніка, оператор, вид палива, кількість літрів, ціна, відпрацьовані години, вид роботи, площа поля	Обов'язковий
ФВ-03	Облік експлуатаційних витрат: технічне обслуговування, поточний і капітальний ремонт, запасні частини, шини, страхування, реєстрація	Обов'язковий
ФВ-04	Ведення реєстру операторів (механізаторів та водіїв)	Обов'язковий
ФВ-05	Фільтрація записів за датою, технікою та типом витрат	Обов'язковий
ФВ-06	Пошук техніки за назвою, реєстраційним номером, виробником	Обов'язковий
ФВ-07	Формування зведеного звіту по техніці та помісячного звіту	Обов'язковий
ФВ-08	Експорт звітів у форматі PDF	Обов'язковий
ФВ-09	Автоматичне обчислення питомих витрат палива (л/год)	Бажаний
ФВ-10	Резервне копіювання та відновлення бази даних	Бажаний

2.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якісні характеристики системи та обмеження, у яких вона повинна функціонувати.

Таблиця 2.2 — Нефункціональні вимоги до системи

Категорія	Вимога
Продуктивність	Завантаження списку записів (до 10 000 рядків) — не більше 2 секунд; відгук на дії користувача — не більше 0,5 секунди
Надійність	Система не повинна втрачати дані при аварійному завершенні роботи; перед видаленням записів повинен відображатися запит підтвердження
Зручність	Інтерфейс повинен бути інтуїтивно зрозумілим для користувачів без технічної підготовки; усі написи — українською мовою
Портативність	Система повинна працювати на ОС Windows 10/11 без встановлення сервера СУБД; база даних зберігається у папці поруч з виконуваним файлом
Безпека	База даних зберігається локально, доступ до неї обмежений правами файлової системи Windows
Підтримуваність	Код системи має бути структурований відповідно до патерну MVVM для полегшення подальшого розвитку та тестування

2.2 Проектування архітектури системи

Архітектуру системи побудовано на основі патерну проектування Model–View–ViewModel (MVVM), що є стандартним підходом для WPF-застосунків. Патерн передбачає чіткий поділ відповідальностей між трьома рівнями: Model (модель даних), View (представлення) та ViewModel (модель представлення).

Model (Модель). Рівень моделі містить класи сутностей, що відображають таблиці бази даних: Machinery, Operator, FuelUsage та OperationalCost. Ці класи анотовані атрибутами Entity Framework Core для конфігурації схеми БД. Окрім класів сутностей, до рівня моделі належить контекст бази даних (AppDbContext), який налаштовує з'єднання та відносини між сутностями.

ViewModel (Модель представлення). Кожному розділу системи відповідає окремий ViewModel-клас: MachineryViewModel, FuelUsageViewModel, OperationalCostViewModel, OperatorsViewModel та

ReportsViewModel. Усі вони успадковують від базового класу BaseViewModel та реалізують логіку завантаження даних, фільтрації, валідації форм і збереження змін. Команди (RelayCommand) прив'язуються до кнопок у View, а властивості (ObservableProperty) — до полів введення та таблиць.

View (Представлення). Рівень представлення складається з XAML-файлів UserControl для кожного розділу та головного вікна MainWindow. Прив'язка даних здійснюється виключно через механізм Data Binding у XAML без коду логіки у code-behind. Перемикання між розділами реалізовано через DataTemplate у ресурсах головного вікна: ContentControl відображає відповідний UserControl залежно від типу поточного ViewModel.

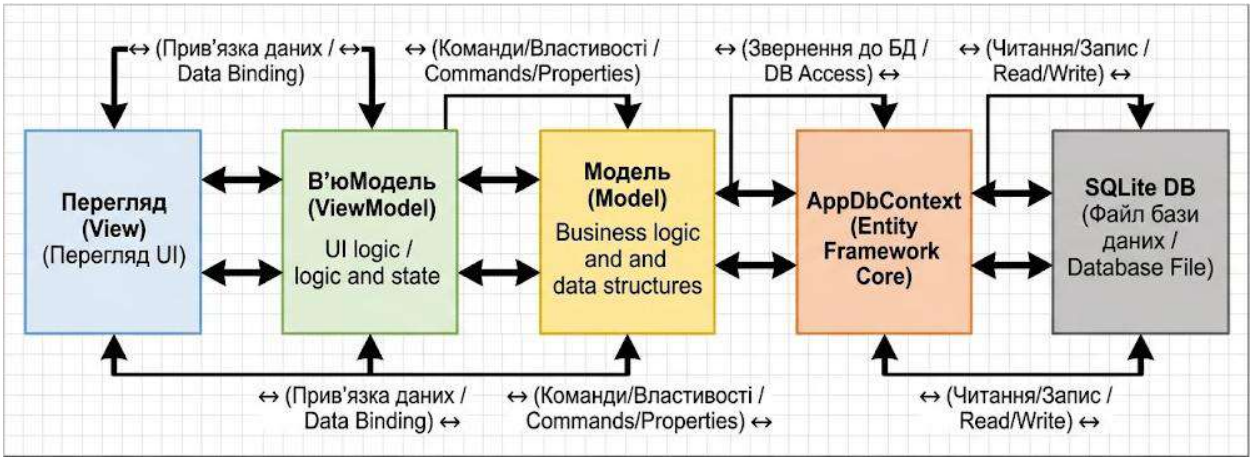


Рисунок 2.2.1. – Схема архітектури системи за патерном MVVM

Головна ViewModel — MainViewModel — зберігає посилання на всі дочірні ViewModel та керує навігацією між розділами. При переході на вкладку викликається метод LoadAsync() відповідного ViewModel, що оновлює дані з бази. Кожен ViewModel має окремий метод ApplyFilter(), який перезавантажує лише відфільтровані записи, не торкаючись довідникових колекцій (Machineries, Operators).

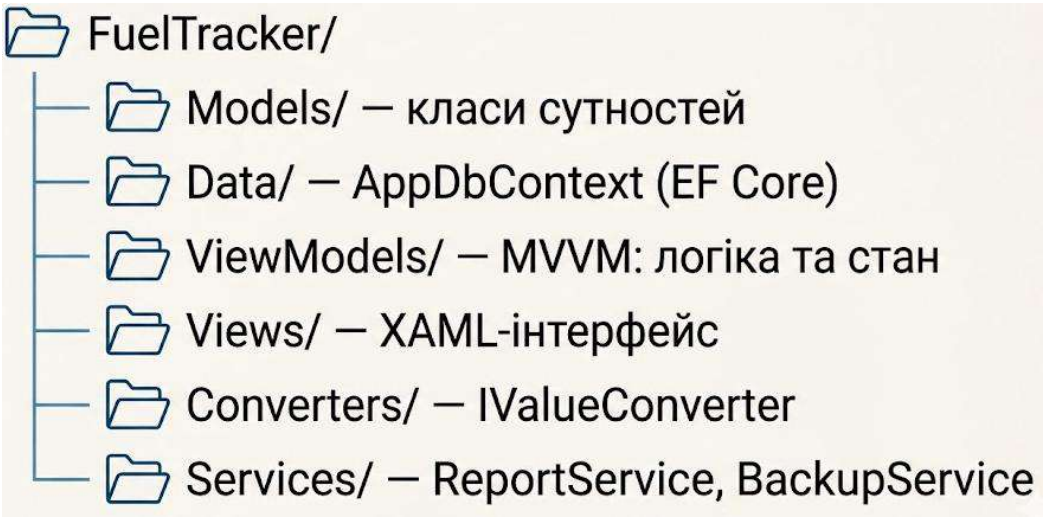


Рисунок 2.2.2. – Структура проєкту

2.3 Проектування бази даних

База даних містить чотири основні таблиці, пов'язані відносинами. Зберігання здійснюється у файлі SQLite, розміщеному в підпапці base поруч із виконуваним файлом застосунку. Схему бази даних наведено на рисунку 2.3.

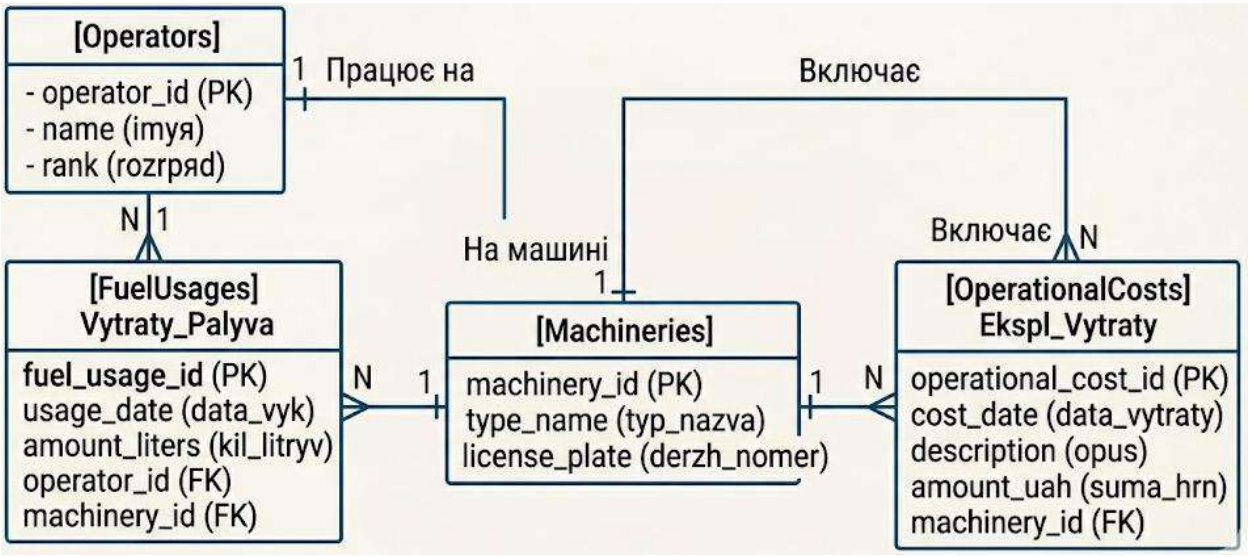


Рисунок 2.3.1. – Концептуальна ER-діаграма бази даних

2.3.1 Таблиця Machineries (Техніка)

Таблиця 2.3 — Структура таблиці Machineries

Поле	Тип	Обмеження	Опис
Id	INTEGER	PK, AUTO	Унікальний ідентифікатор
Name	TEXT	NOT NULL	Назва техніки
Type	TEXT	NOT NULL	Тип (Трактор, Комбайн тощо)
RegistrationNumber	TEXT	—	Реєстраційний номер
Manufacturer	TEXT	—	Виробник
Model	TEXT	—	Модель
YearOfManufacture	INTEGER	—	Рік випуску
PurchaseDate	TEXT	—	Дата придбання
PurchaseCost	REAL	—	Вартість придбання, грн
CurrentEngineHours	REAL	—	Поточний моточас, год
Status	TEXT	NOT NULL	Статус (Активна / На ремонті / ...)
Notes	TEXT	—	Примітки
CreatedAt	TEXT	NOT NULL	Дата створення запису
UpdatedAt	TEXT	NOT NULL	Дата останнього оновлення

2.3.2 Таблиця Operators (Оператори)

Таблиця 2.4 — Структура таблиці Operators

Поле	Тип	Обмеження	Опис
Id	INTEGER	PK, AUTO	Унікальний ідентифікатор
FullName	TEXT	NOT NULL	Прізвище, ім'я, по батькові
Position	TEXT	—	Посада (тракторист, комбайнер тощо)
Phone	TEXT	—	Контактний телефон
LicenseNumber	TEXT	—	Номер посвідчення
IsActive	INTEGER	DEFAULT 1	Ознака активності (1 — активний)
CreatedAt	TEXT	NOT NULL	Дата створення запису

2.3.3 Таблиця FuelUsages (Використання палива)

Таблиця 2.5 — Структура таблиці FuelUsages

Поле	Тип	Обмеження	Опис
Id	INTEGER	PK, AUTO	Унікальний ідентифікатор
MachineryId	INTEGER	FK → Machineries	Посилання на техніку
OperatorId	INTEGER	FK → Operators	Посилання на оператора
Date	TEXT	NOT NULL	Дата заправлення
FuelType	TEXT	NOT NULL	Тип палива (Дизель, Бензин тощо)
AmountLiters	REAL	NOT NULL	Кількість, л
PricePerLiter	REAL	NOT NULL	Ціна за літр, грн
WorkHours	REAL	—	Відпрацьовано годин
WorkType	TEXT	—	Вид роботи (Оранка, Сівба тощо)
FieldAreaHectares	REAL	—	Оброблена площа, га
Notes	TEXT	—	Примітки
CreatedAt	TEXT	NOT NULL	Дата створення запису

2.3.4 Таблиця OperationalCosts (Витрати)

Таблиця 2.6 — Структура таблиці OperationalCosts

Поле	Тип	Обмеження	Опис
Id	INTEGER	PK, AUTO	Унікальний ідентифікатор
MachineryId	INTEGER	FK → Machineries	Посилання на техніку
Date	TEXT	NOT NULL	Дата витрат
CostType	TEXT	NOT NULL	Тип витрат (ТО, Ремонт, Запчастини тощо)
Amount	REAL	NOT NULL	Сума витрат, грн
Description	TEXT	—	Опис виконаних робіт
PerformedBy	TEXT	—	Виконавець (сервіс, майстерня)
DocumentNumber	TEXT	—	Номер підтверджуючого документа
CreatedAt	TEXT	NOT NULL	Дата створення запису

Між таблицями встановлено такі зовнішні ключі: FuelUsages.MachineryId → Machineries.Id (каскадне видалення) та FuelUsages.OperatorId → Operators.Id (заборона видалення оператора при наявності записів). Аналогічно OperationalCosts.MachineryId → Machineries.Id (каскадне видалення).

2.4 Проектування інтерфейсу користувача

Головне вікно застосунку має компонування в дві колонки: ліворуч розміщено вертикальну панель навігації шириною 220 пікселів, праворуч — основна область контенту. Навігаційна панель містить кнопки переходу між п'ятьма розділами: «Техніка», «Використання палива», «Витрати», «Звіти» та «Оператори». Компонування відповідає концепції Material Design.

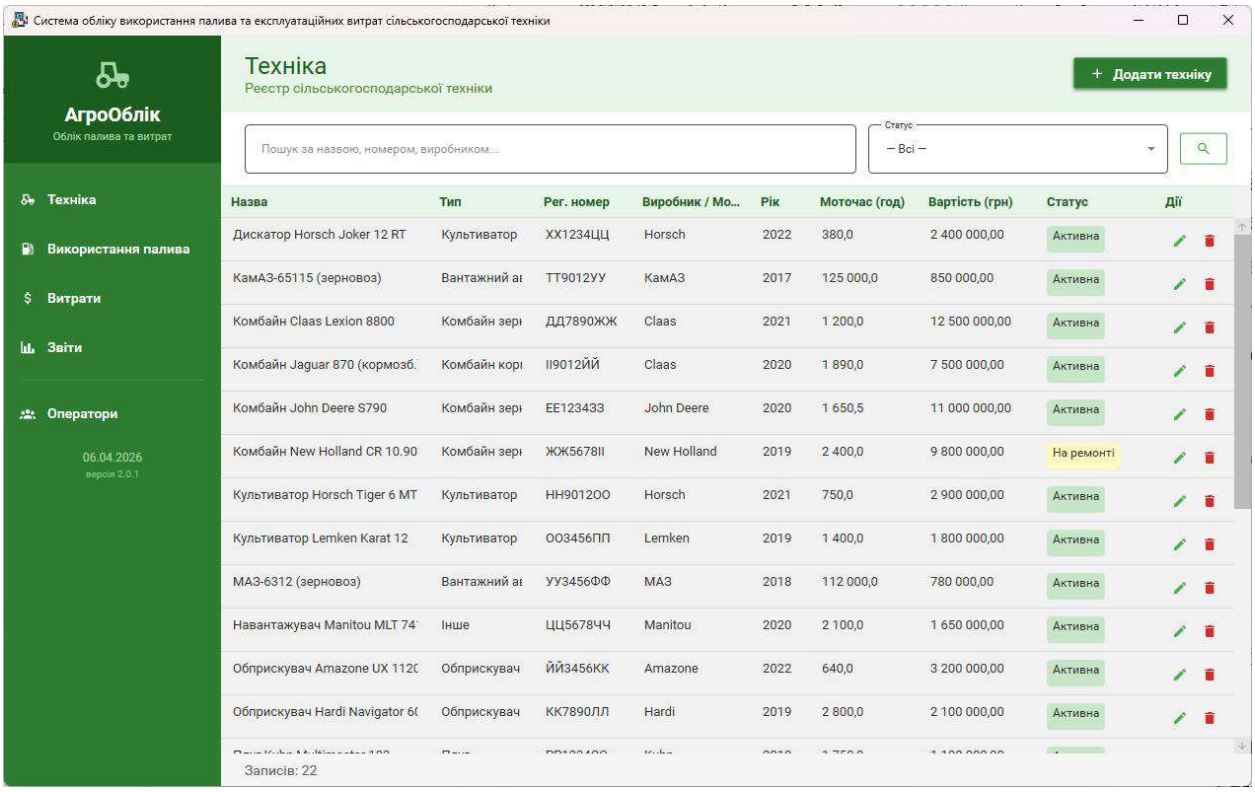


Рисунок 2.4.1. – Макет головного вікна застосунку

Кожен розділ побудований за однотипним шаблоном: верхня смуга із заголовком розділу та кнопкою додавання запису, рядок фільтрів, (де доречно) блок підсумкових карток, та таблиця даних. Форми додавання та редагування реалізовані як модальні діалоги DialogHost поверх основного контенту.

Висновки до розділу 2

У другому розділі сформульовано функціональні та нефункціональні вимоги до системи, визначено пріоритети реалізації. Спроектовано архітектуру системи на основі патерну MVVM із чітким розподілом рівнів Model, ViewModel та View. Розроблено схему бази даних, що включає чотири таблиці: Machineries, Operators, FuelUsages та OperationalCosts — з відносинами між ними. Визначено макет та логіку навігації графічного інтерфейсу.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Реалізація моделі даних та рівня доступу до бази даних

3.1.1 Класи сутностей

Рівень моделі даних системи реалізовано у вигляді чотирьох класів-сутностей, що безпосередньо відображаються на таблиці бази даних SQLite через механізм Object-Relational Mapping бібліотеки Entity Framework Core 9. Усі класи розташовані у просторі імен FuelTracker.Models та анотовані атрибутами з простору System.ComponentModel.DataAnnotations для декларативного визначення обмежень схеми бази даних. Між сутностями встановлено навігаційні властивості, що дозволяють Entity Framework Core автоматично виконувати JOIN-запити та підтримувати цілісність зв'язків.

Загальна схема взаємозв'язків між сутностями є такою: центральне місце займає сутність Machinery (Техніка), до якої прив'язані всі записи використання палива (FuelUsage) та всі записи експлуатаційних витрат (OperationalCost). Сутність Operator (Оператор) пов'язана з FuelUsage відношенням «один до багатьох», оскільки кожен запис заправлення реєструється на конкретного механізатора або водія.

Сутність Machinery (Техніка).

Клас Machinery є центральним елементом моделі та описує одиниці сільськогосподарської техніки підприємства. Кожен екземпляр класу відповідає одній фізичній одиниці парку — трактору, комбайну, обприскувачу, вантажному автомобілю тощо. Клас містить 14 збережуваних властивостей та дві навігаційні колекції.

Таблиця 3.1 — Властивості класу Machinery

Властивість	Тип C#	Атрибути / Обмеження	Опис
Id	int	[Key], AUTO INCREMENT	Первинний ключ
Name	string	[Required], MaxLength(200)	Назва техніки (обов'язкове поле)
Type	string	[Required], MaxLength(100)	Тип: Трактор, Комбайн, Сівалка тощо
RegistrationNumber	string	MaxLength(50)	Державний реєстраційний номер
Manufacturer	string	MaxLength(100)	Виробник (John Deere, Claas, МТЗ тощо)
Model	string	MaxLength(50)	Комерційна модель (8R 310, Axion 870 тощо)
YearOfManufacture	int	—	Рік виготовлення
PurchaseDate	DateOnly	—	Дата придбання техніки
PurchaseCost	decimal	decimal(18,2)	Вартість придбання, грн
CurrentEngineHours	decimal	decimal(18,2)	Поточний наробіток, моточасів
Status	string	За замовч.: "Активна"	Стан: Активна / На ремонті / На консервації / Списана
Notes	string	MaxLength(500)	Довільні примітки
CreatedAt	DateTime	За замовч.: UtcNow	Дата і час створення запису
UpdatedAt	DateTime	За замовч.: UtcNow	Дата і час останньої зміни
FuelUsages	ICollection<FuelUsage>	[NotMapped], навігаційна	Колекція записів витрати палива
OperationalCosts	ICollection< OperationalCost>	[NotMapped], навігаційна	Колекція записів витрат

Поле Status може набувати одного з чотирьох значень, визначених у допоміжному статичному класі MachineryStatuses: «Активна», «На ремонті», «На консервації», «Списана». Поле Type обмежено переліком значень класу

MachineryTypes (10 типів). Обидва поля зберігаються як рядки у базі даних без використання enum-типу задля спрощення відображення у ComboBox. Поле RegistrationNumber при збереженні автоматично перетворюється у верхній регістр.

Навігаційна властивість FuelUsages містить усі записи витрати палива для даної техніки. При видаленні запису Machinery EF Core автоматично видаляє всі пов'язані записи FuelUsage та OperationalCost завдяки налаштуванню OnDelete(DeleteBehavior.Cascade) у методі OnModelCreating контексту бази даних.

```
// ── Техніка (Agricultural Machinery) ──────────────────────────────────
public class Machinery
{
    [Key]
    public int Id { get; set; }

    [Required, MaxLength(200)]
    public string Name { get; set; } = string.Empty;          // Назва

    [Required, MaxLength(100)]
    public string Type { get; set; } = string.Empty;          // Тип (трактор, комбайн…)

    [MaxLength(50)]
    public string RegistrationNumber { get; set; } = string.Empty; // Реєстраційний номер

    [MaxLength(100)]
    public string Manufacturer { get; set; } = string.Empty;  // Виробник

    [MaxLength(50)]
    public string Model { get; set; } = string.Empty;         // Модель

    public int YearOfManufacture { get; set; }                // Рік виготовлення

    public DateOnly PurchaseDate { get; set; }                // Дата придбання

    [Column(TypeName = "decimal(18,2)")]
    public decimal PurchaseCost { get; set; }                 // Вартість придбання (грн)

    [Column(TypeName = "decimal(18,2)")]
    public decimal CurrentEngineHours { get; set; }           // Поточний моточас

    public string Status { get; set; } = "Активна";           // Статус: Активна / На ремонті / Списана

    [MaxLength(500)]
    public string Notes { get; set; } = string.Empty;         // Примітки

    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
    public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;

    // Navigation
    public ICollection<FuelUsage> FuelUsages { get; set; } = [];
    public ICollection<OperationalCost> OperationalCosts { get; set; } = [];
}
```

Рисунок 3.1.1. – Клас Machinery

Сутність Operator (Оператор).

Клас Operator зберігає відомості про механізаторів, комбайнерів і водіїв аграрного підприємства, які є відповідальними за використання техніки. Кожен оператор може бути прив'язаний до необмеженої кількості записів FuelUsage, утворюючи відношення «один до багатьох».

Таблиця 3.2 — Властивості класу Operator

Властивість	Тип C#	Атрибути / Обмеження	Опис
Id	int	[Key], AUTO INCREMENT	Первинний ключ
FullName	string	[Required], MaxLength(100)	Прізвище, ім'я та по батькові
Position	string	MaxLength(50)	Посада (тракторист, комбайнер, водій)
Phone	string	MaxLength(20)	Контактний номер телефону
LicenseNumber	string	MaxLength(100)	Номер посвідчення водія або тракториста
IsActive	bool	За замовч.: true	Ознака активності: true — працює, false — звільнений
CreatedAt	DateTime	За замовч.: UtcNow	Дата і час створення запису
FuelUsages	ICollection<FuelUsage>	Навігаційна	Колекція записів витрати палива оператора

Поле IsActive дозволяє «м'яко» деактивувати оператора, якого більше немає в штаті, без видалення його записів з журналу. Активні оператори (IsActive = true) відображаються у ComboBox при додаванні нових записів заправлення. При спробі видалити оператора, до якого прив'язані записи FuelUsage, EF Core відхиляє операцію завдяки налаштуванню OnDelete(DeleteBehavior.Restrict), що гарантує цілісність журналу.

```
// ── Оператори (Operators / Drivers) ──────────────────────────────────────
public class Operator
{
    [Key]
    public int Id { get; set; }

    [Required, MaxLength(100)]
    public string FullName { get; set; } = string.Empty;       // ПІБ

    [MaxLength(50)]
    public string Position { get; set; } = string.Empty;       // Посада

    [MaxLength(20)]
    public string Phone { get; set; } = string.Empty;          // Телефон

    [MaxLength(100)]
    public string LicenseNumber { get; set; } = string.Empty; // Номер посвідчення

    public bool IsActive { get; set; } = true;                 // Активний

    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    public ICollection<FuelUsage> FuelUsages { get; set; } = [];
}
```

Рисунок 3.1.2. – Клас Operator

Сутність FuelUsage (Використання палива).

Клас FuelUsage є основним журнальним класом системи та фіксує кожен окремий факт витрати палива. Один запис відповідає одному заправленню: з'єднує конкретну одиницю техніки та конкретного оператора з кількісними показниками витрати. Клас містить два зовнішніх ключі та дві обчислювані властивості, що не зберігаються в базі даних.

Таблиця 3.3 — Властивості класу FuelUsage

Властивість	Тип C#	Атрибути / Обмеження	Опис
Id	int	[Key], AUTO INCREMENT	Первинний ключ
MachineryId	int	[Required], FK → Machineries	Зовнішній ключ до техніки
Machinery	Machinery	Навігаційна	Пов'язана одиниця техніки
OperatorId	int	[Required], FK → Operators	Зовнішній ключ до оператора
Operator	Operator	Навігаційна	Пов'язаний оператор
Date	DateOnly	—	Дата заправлення / виконання роботи

Властивість	Тип C#	Атрибути / Обмеження	Опис
FuelType	string	[Required], MaxLength(50)	Тип палива (Дизель, Бензин, Газ, AdBlue)
AmountLiters	decimal	decimal(10,2)	Заправлено паливом, літрів
PricePerLiter	decimal	decimal(10,2)	Ціна одного літра, грн
WorkHours	decimal	decimal(10,2)	Відпрацьовано моточасів за зміну
WorkType	string	MaxLength(200)	Вид виконаної роботи (оранка, сівба, збирання тощо)
FieldAreaHectares	decimal	decimal(10,2)	Оброблена площа поля, га
Notes	string	MaxLength(500)	Додаткові примітки
CreatedAt	DateTime	За замовч.: UtcNow	Дата і час створення запису
TotalCost	decimal	[NotMapped], обчислювана	Загальна вартість палива = AmountLiters × PricePerLiter
LitersPerHour	decimal	[NotMapped], обчислювана	Питома витрата = AmountLiters / WorkHours (0 якщо WorkHours = 0)

Особливу увагу заслуговують обчислювані властивості, анотовані атрибутом [NotMapped]. Властивість TotalCost обчислює загальну вартість витраченого палива як добуток кількості та ціни за літр, що дозволяє відображати попередній розрахунок безпосередньо у формі введення при зміні будь-якого з операндів. Властивість LitersPerHour розраховує питому витрату палива на одну годину роботи техніки — ключовий показник ефективності, що виводиться у зведеному звіті:

```
[NotMapped]
public decimal TotalCost =>
    AmountLiters * PricePerLiter;

[NotMapped]
public decimal LitersPerHour =>
    WorkHours > 0 ? AmountLiters / WorkHours : 0;
```

Рисунок 3.1.3. – Обчислювані властивості

Тип поля Date обрано як DateOnly (введений у .NET 6) замість DateTime, оскільки для запису заправлення достатньо дати без часової компоненти. Це

дозволяє коректно порівнювати дати при фільтрації без урахування часового поясу. EF Core 9 зберігає DateOnly у SQLite як текстовий рядок формату «рррр-мм-дд».

Сутність OperationalCost (Експлуатаційні витрати).

Клас OperationalCost реєструє всі грошові витрати, пов'язані з утриманням техніки, що не є витратами на паливо: технічне обслуговування, різні види ремонту, придбання запасних частин, шин, страхування та державна реєстрація. Кожен запис прив'язаний до конкретної одиниці техніки. На відміну від FuelUsage, OperationalCost не потребує прив'язки до оператора, оскільки технічне обслуговування виконується сервісними службами.

Таблиця 3.4 — Властивості класу OperationalCost

Властивість	Тип C#	Атрибути / Обмеження	Опис
Id	int	[Key], AUTO INCREMENT	Первинний ключ
MachineryId	int	[Required], FK → Machineries	Зовнішній ключ до техніки
Machinery	Machinery	Навігаційна	Пов'язана одиниця техніки
Date	DateOnly	—	Дата здійснення витрат
CostType	string	[Required], MaxLength(100)	Тип витрат (з переліку CostTypes.All)
Amount	decimal	decimal(18,2)	Сума витрат, грн
Description	string	MaxLength(500)	Деталізований опис виконаних робіт або придбань
PerformedBy	string	MaxLength(200)	Виконавець: сервісний центр, майстерня тощо
DocumentNumber	string	MaxLength(100)	Номер акта, накладної або рахунку-фактури
CreatedAt	DateTime	За замовч.: UtcNow	Дата і час створення запису

Поле CostType набуває значень із статичного класу CostTypes.All, який містить дев'ять категорій: «Технічне обслуговування», «Поточний ремонт», «Капітальний ремонт», «Запасні частини», «Шини та гусениці», «Мастильні матеріали», «Страхування», «Реєстрація та техогляд», «Інше». Поле

DocumentNumber дозволяє зберігати посилання на первинний документ, що підтверджує факт витрат, — це підвищує юридичну достовірність обліку.

```
// ── Експлуатаційні витрати (Operational Costs) ──────────────────────
public class OperationalCost
{
    [Key]
    public int Id { get; set; }

    [Required]
    public int MachineryId { get; set; }
    public Machinery Machinery { get; set; } = null!;

    public DateOnly Date { get; set; }                          // Дата

    [Required, MaxLength(100)]
    public string CostType { get; set; } = string.Empty;        // Тип: ТО / Ремонт / Запчастини / Інше

    [Column(TypeName = "decimal(18,2)")]
    public decimal Amount { get; set; }                         // Сума (грн)

    [MaxLength(500)]
    public string Description { get; set; } = string.Empty;  // Опис

    [MaxLength(200)]
    public string PerformedBy { get; set; } = string.Empty;   // Виконавець

    [MaxLength(100)]
    public string DocumentNumber { get; set; } = string.Empty;// Номер документа

    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;
}
```

Рисунок 3.1.4. – Клас OperationalCost

Статичні класи-словники.

Окрім класів сутностей, файл Models.cs містить чотири статичні класи, що виконують роль словників допустимих значень для перелічуваних полів. Замість enum-типів обрано підхід із масивами рядків, що безпосередньо прив'язуються до ItemsSource компонентів ComboBox у XAML-інтерфейсі. Це дозволяє уникнути конвертерів між enum та рядком і спрощує локалізацію.

Таблиця 3.5 — Статичні класи-словники моделі

Клас	Поле-приймач	Кількість значень	Значення
MachineryTypes	Machinery.Type	10	Трактор, Комбайн зернозбиральний, Комбайн кормозбиральний, Сівалка, Обприскувач, Культиватор, Плуг, Причіп, Вантажний автомобіль, Інше

Клас	Поле-приймач	Кількість значень	Значення
MachineryStatuses	Machinery.Status	4	Активна, На ремонті, На консервації, Списана
FuelTypes	FuelUsage.FuelType	4	Дизель, Бензин, Газ (LPG), Адблю (AdBlue)
CostTypes	OperationalCost.CostType	9	Технічне обслуговування, Поточний ремонт, Капітальний ремонт, Запасні частини, Шини та гусениці, Мастильні матеріали, Страхування, Реєстрація та техогляд, Інше

Кожен статичний клас надає публічну константу All типу string[], що використовується у двох контекстах: у формах додавання/редагування — як повний перелік для ComboBox форми, та у рядках фільтрів — як основа для побудови масиву з доданим першим елементом «— Всі —» (наприклад, CostTypesFilter у OperationalCostViewModel). Така архітектура гарантує єдине джерело істини: зміна переліку значень в одному місці автоматично відображається у всіх частинах застосунку.

```
// ── Допоміжні enum-like статичні списки ──────────────────────────
public static class MachineryTypes
{
    public static readonly string[] All =
    [
        "Трактор", "Комбайн зернозбиральний", "Комбайн кормозбиральний",
        "Сівалка", "Обприскувач", "Культиватор", "Плуг",
        "Причіп", "Вантажний автомобіль", "Інше"
    ];
}

public static class FuelTypes
{
    public static readonly string[] All = ["Дизель", "Бензин", "Газ (LPG)", "Адблю (AdBlue)"];
}

public static class CostTypes
{
    public static readonly string[] All =
    [
        "Технічне обслуговування", "Поточний ремонт", "Капітальний ремонт",
        "Запасні частини", "Шини та гусениці", "Мастильні матеріали",
        "Страхування", "Реєстрація та техогляд", "Інше"
    ];
}

public static class MachineryStatuses
{
    public static readonly string[] All = ["Активна", "На ремонті", "На консервації", "Списана"];
}
```

Рисунок 3.1.5. – Статичні класи-словники

Зв'язки між сутностями та налаштування зовнішніх ключів.

Конфігурацію зовнішніх ключів і поведінки при каскадному видаленні виконано у методі OnModelCreating() класу AppDbContext. Визначено три відношення:

1) FuelUsage → Machinery: відношення «багато до одного». При видаленні техніки всі пов'язані записи журналу палива видаляються автоматично (DeleteBehavior.Cascade). Це забезпечує відсутність «осиротілих» записів у базі даних.
2) FuelUsage → Operator: відношення «багато до одного». При спробі видалити оператора, якого прив'язано до записів журналу, EF Core генерує виняток (DeleteBehavior.Restrict). Це захищає цілісність облікових даних.
3) OperationalCost → Machinery: відношення «багато до одного». Аналогічно до FuelUsage застосовується DeleteBehavior.Cascade: видалення техніки призводить до видалення всіх її записів витрат.

```
// FuelUsage → Machinery (каскадне видалення)
modelBuilder.Entity<FuelUsage>()
    .HasOne(f => f.Machinery)
    .WithMany(m => m.FuelUsages)
    .HasForeignKey(f => f.MachineryId)
    .OnDelete(DeleteBehavior.Cascade);

// FuelUsage → Operator (заборона видалення при наявності записів)
modelBuilder.Entity<FuelUsage>()
    .HasOne(f => f.Operator)
    .WithMany(o => o.FuelUsages)
    .HasForeignKey(f => f.OperatorId)
    .OnDelete(DeleteBehavior.Restrict);

// OperationalCost → Machinery (каскадне видалення)
modelBuilder.Entity<OperationalCost>()
    .HasOne(c => c.Machinery)
    .WithMany(m => m.OperationalCosts)
    .HasForeignKey(c => c.MachineryId)
    .OnDelete(DeleteBehavior.Cascade);
```

Рисунок 3.1.6. – Конфігурація у AppDbContext.OnModelCreating()

Таким чином, рівень моделі даних системи складається з чотирьох класів сутностей (Machinery, Operator, FuelUsage, OperationalCost), чотирьох статичних класів-словників та контексту бази даних AppDbContext.

Архітектурне рішення про використання DateOnly замість DateTime, атрибута [NotMapped] для обчислюваних властивостей та рядків замість enum прийнято з урахуванням специфіки ORM-взаємодії та вимог до зручності прив'язки даних у WPF-інтерфейсі.

3.1.2 Контекст бази даних

Клас AppDbContext успадковує від DbContext та конфігурує шлях до файлу бази даних, описує DbSet-властивості для кожної таблиці та налаштовує зовнішні ключі й поведінку при каскадному видаленні через OnModelCreating(). База даних зберігається у підпапці base поруч із виконуваним файлом застосунку:

```
private static string GetDefaultDatabasePath()
{
    // Отримуємо папку exe-файлу (або папку проєкту під час розробки)
    var exeFolder = Path.GetDirectoryName(
        Assembly.GetExecutingAssembly().Location) ?? AppDomain.CurrentDomain.BaseDirectory;

    var baseFolder = Path.Combine(exeFolder, "base");
    Directory.CreateDirectory(baseFolder);          // створити якщо не існує
    return Path.Combine(baseFolder, "fueltracker.db");
}
```

Рисунок 3.1.7. – Метод GetDefaultDatabasePath

Ініціалізація бази даних при першому запуску здійснюється методом EnsureCreated(), який автоматично створює всі таблиці відповідно до конфігурації OnModelCreating(). Seed-дані (початкові оператори та одиниця техніки для демонстрації) додаються через метод HasData().

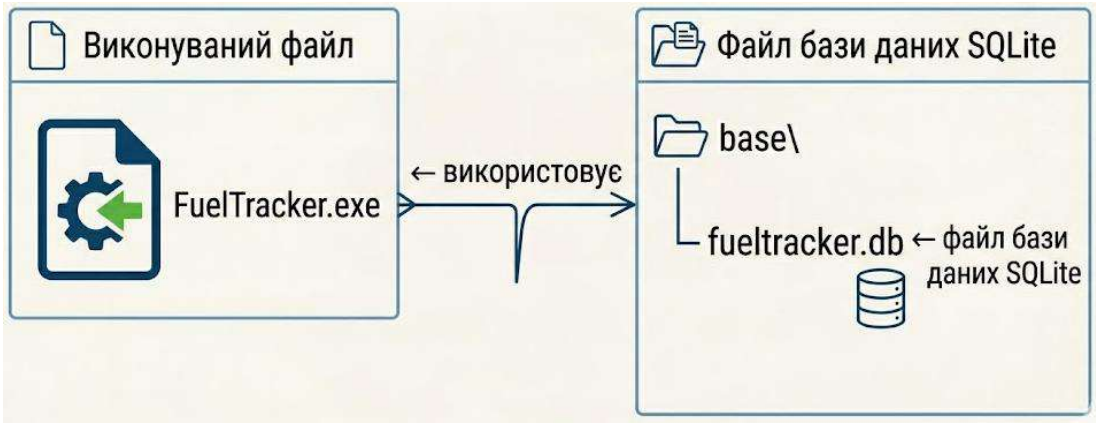


Рисунок 3.1.8. – Розташування файлу бази даних відносно виконуваного файлу

3.2 Реалізація рівня ViewModel

3.2.1 Базовий клас та головна ViewModel

Базовий клас BaseViewModel успадковує від ObservableObject бібліотеки CommunityToolkit.Mvvm та оголошує спільні властивості IsBusy і StatusMessage для відображення стану завантаження та повідомлень у рядку статусу. Утилітарні методи ShowError() та Confirm() централізують логіку відображення діалогів.

MainViewModel зберігає екземпляри всіх дочірніх ViewModel та керує навігацією через команду NavigateToCommand. При переході на вкладку викликається відповідний метод LoadAsync(). При першому запуску у конструкторі MainViewModel ініціюється завантаження даних першої вкладки (MachineryViewModel).

3.2.2 Патерн розділення довідників та фільтрів

Ключовою архітектурною проблемою при реалізації FuelUsageViewModel та OperationalCostViewModel було збереження вибраного значення в ComboBox «Техніка» при оновленні даних. При кожному виклику Load() колекція Machineries очищувалась і наповнювалась новими об'єктами, що змушувало WPF ComboBox автоматично скидати SelectedItem.

Вирішення досягнуто шляхом розділення методу завантаження на два: Load() — виконується один раз при першому відкритті вкладки та наповнює довідникові колекції (Machineries, Operators); ApplyFilter() — викликається при кожному застосуванні фільтра та оновлює лише колекцію записів Items, не чіпаючи довідників. Після збереження та видалення записів також викликається ApplyFilter(), а не Load().

```
[RelayCommand]
public async Task Load()
{
    IsBusy = true;
    try
    {
        using var ctx = CreateContext();

        // Завантажити довідники лише якщо вони ще пусті
        if (!Machineries.Any())
        {
            var mach = await ctx.Machineries
                .OrderBy(m => m.Name).ToListAsync();
            foreach (var m in mach) Machineries.Add(m);
        }

        await LoadItemsAsync(ctx);
    }
    catch (Exception ex) { ShowError($"Помилка: {ex.Message}"); }
    finally { IsBusy = false; }
}

// ── Кнопка Фільтр: лише перезавантажити записи, НЕ чіпати довідники ──────
[RelayCommand]
public async Task ApplyFilter()
{
    IsBusy = true;
    try
    {
        using var ctx = CreateContext();
        await LoadItemsAsync(ctx);
    }
    catch (Exception ex) { ShowError($"Помилка фільтрації: {ex.Message}"); }
    finally { IsBusy = false; }
}
```

Рисунок 3.2.1. – Метод Load для завантаження довідників

3.2.3 Реалізація пошуку з урахуванням кирилиці

SQLite не підтримує регістронезалежне порівняння (LIKE) для символів кирилиці через обмеження стандартної бібліотеки. Спроба використати .ToLower() у LINQ-запиті не дала результату, оскільки EF Core транслює це в SQL LOWER(), який не розпізнає великі літери кирилиці. Проблему вирішено завантаженням даних з фільтром лише за точними полями (статус) та подальшим пошуком на стороні C#:

```
// Порівняння без урахування регістру для будь-якої мови (включно з кирилицею)
private static bool Contains(string source, string search) =>
    source.Contains(search, StringComparison.CurrentCultureIgnoreCase);

// ── Пошук на стороні клієнта — правильно обробляє кирилицю ──────
IEnumerable<Machinery> filtered = all;
if (!string.IsNullOrWhiteSpace(SearchText))
{
    var q = SearchText.Trim();
    filtered = all.Where(m =>
        Contains(m.Name, q) ||
        Contains(m.RegistrationNumber, q) ||
        Contains(m.Manufacturer, q) ||
        Contains(m.Model, q) ||
        Contains(m.Type, q));
}

Items.Clear();
foreach (var item in filtered) Items.Add(item);
StatusMessage = $"Записів: {Items.Count}";
```

Рисунок 3.2.2. – Реалізація пошуку

3.3 Реалізація графічного інтерфейсу

3.3.1 Налаштування теми та культури

При запуску застосунку в методі App.OnStartup() встановлюється культура uk-UA для всіх потоків та перевизначається властивість LanguageProperty для всіх FrameworkElement. Це забезпечує коректну роботу компонента DatePicker — відображення назв місяців і днів тижня українською мовою та початок тижня з понеділка:

```
var uaCulture = new CultureInfo("uk-UA");
Thread.CurrentThread.CurrentCulture   = uaCulture;
Thread.CurrentThread.CurrentUICulture = uaCulture;
CultureInfo.DefaultThreadCurrentCulture   = uaCulture;
CultureInfo.DefaultThreadCurrentUICulture = uaCulture;
FrameworkElement.LanguageProperty.OverrideMetadata(
    typeof(FrameworkElement),
    new FrameworkPropertyMetadata(
        XmlLanguage.GetLanguage(uaCulture.IetfLanguageTag)));
```

Рисунок 3.3.1. – Налаштування культури

3.3.2 Навігація між розділами

Навігацію між розділами реалізовано через DataTemplate у ресурсах головного вікна. Кожному типу ViewModel відповідає шаблон, що вказує на

відповідний UserControl. ContentControl у головному вікні відображає поточний ViewModel, а WPF автоматично застосовує відповідний шаблон:

```
<Window.Resources>
  <DataTemplate DataType="{x:Type vm:MachineryViewModel}">
    <views:MachineryView/>
  </DataTemplate>
  <DataTemplate DataType="{x:Type vm:FuelUsageViewModel}">
    <views:FuelUsageView/>
  </DataTemplate>
  <!-- ... -->
</Window.Resources>
<ContentControl Content="{Binding CurrentView}"/>
```

Рисунок 3.3.2. – Навігація між розділами

3.3.3 Рядок фільтрів

Рядок фільтрів реалізовано на основі Grid із фіксованими колонками для кожного елемента управління. Від StackPanel з властивістю Spacing (яка недоступна у WPF) відмовлено на користь Grid, оскільки він забезпечує коректне вертикальне вирівнювання кнопки «Фільтр» відносно інших елементів. Кнопка має VerticalAlignment="Stretch" та заповнює всю висоту рядка.

ComboBox для вибору типу витрат та типу палива прив'язано до масивів із рядковим значенням «— Всі —» першим елементом. При застосуванні фільтра перевіряється умова FilterCostType != "— Всі —" перед додаванням умови WHERE до запиту. Фільтр техніки є ComboBox з прив'язкою SelectedItem до властивості FilterMachinery — об'єкта типу Machinery. Оскільки довідникова колекція Machineries не перезавантажується при фільтрації, вибраний об'єкт залишається стабільним.

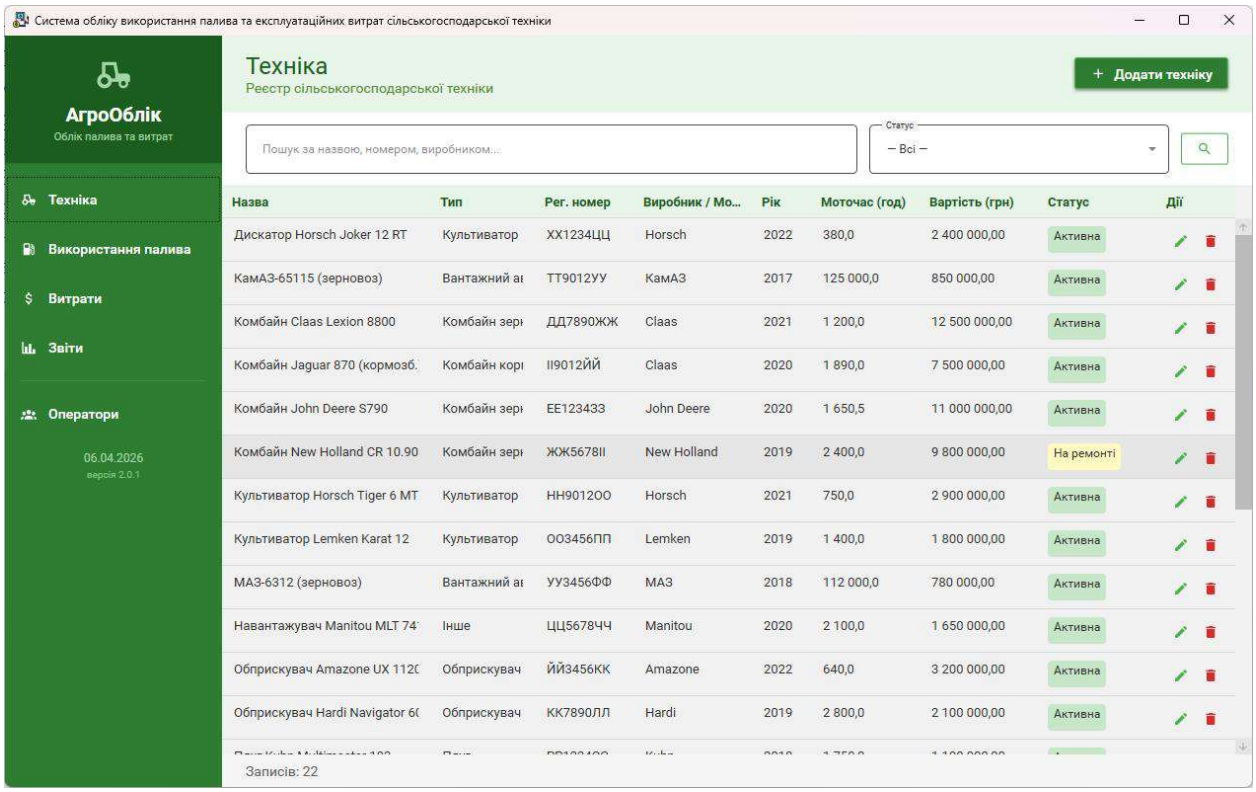


Рисунок 3.3.3. – Зовнішній вигляд вкладки «Техніка»

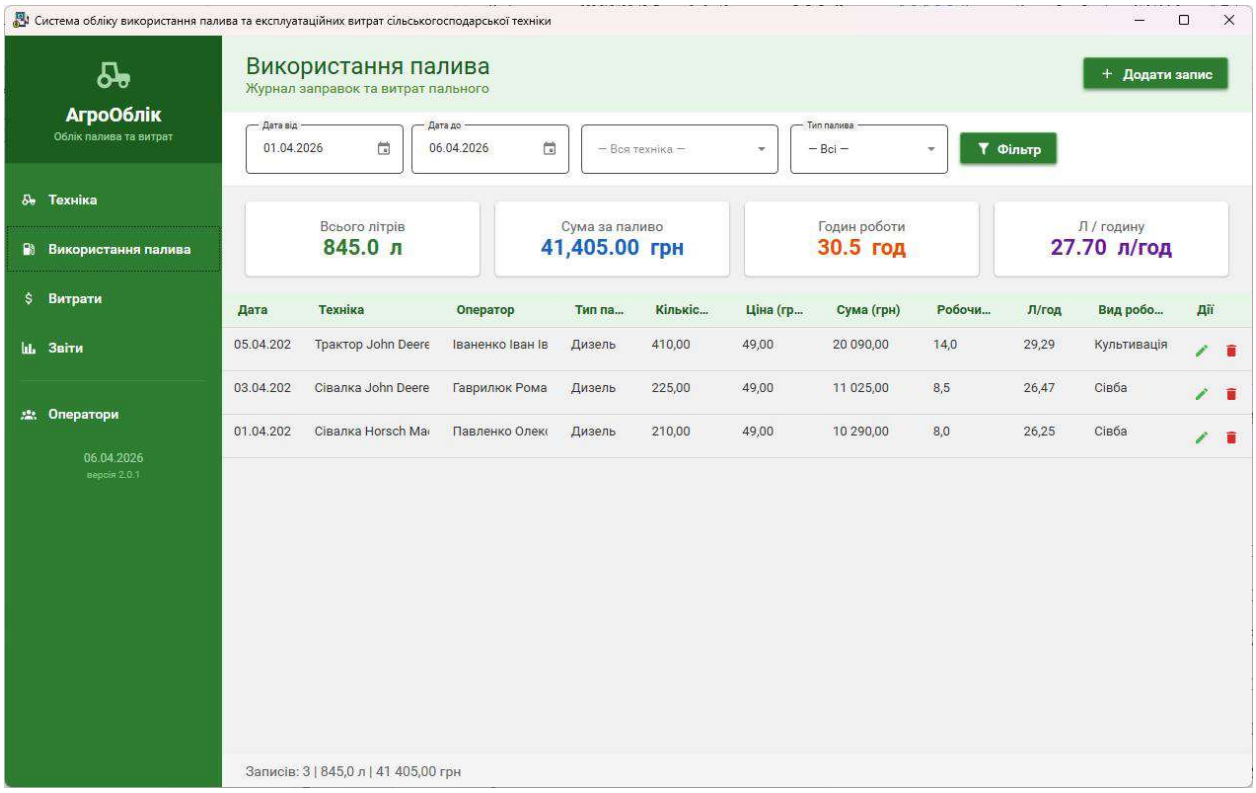


Рисунок 3.3.4. – Зовнішній вигляд вкладки «Використання палива»

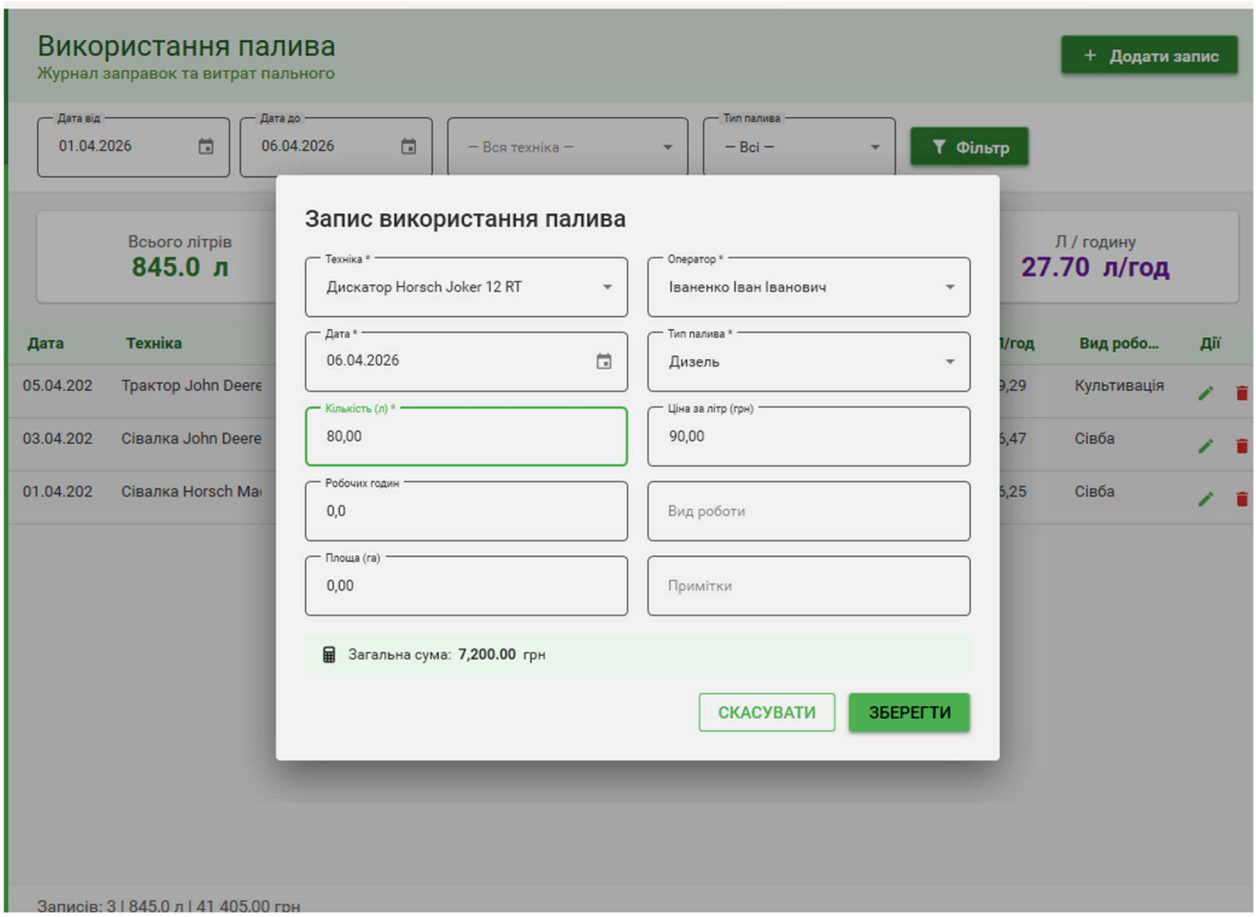
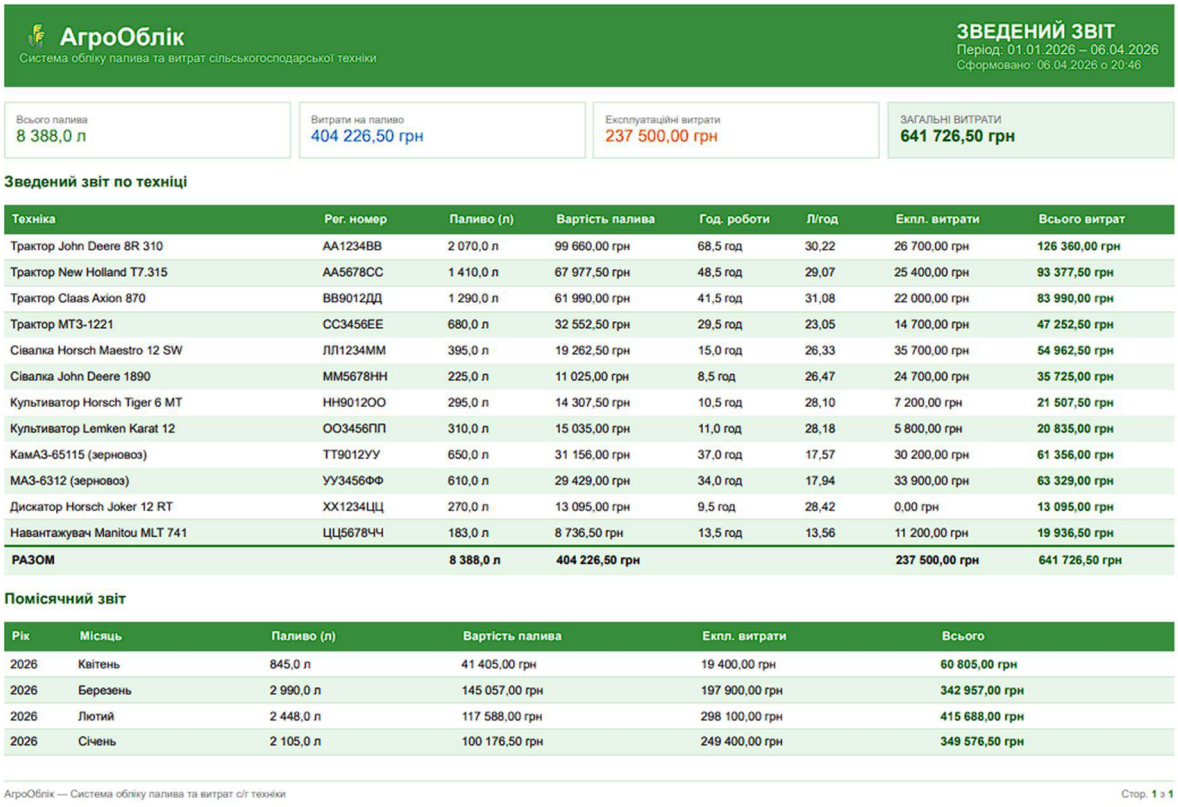


Рисунок 3.3.5. – Діалогова форма додавання запису про витрату палива

3.4 Реалізація генерації звітів

Генерацію PDF-звітів реалізовано у класі ReportService з використанням бібліотеки QuestPDF (Community License). Метод ExportSummaryReport() відкриває системний діалог збереження файлу, після чого будує документ та зберігає його на диск. При успішному збереженні користувачу пропонується відкрити файл у програмі перегляду PDF.

Документ має альбомну орієнтацію формату A4. Структура звіту включає: шапку з назвою системи та вказаним звітним періодом; блок підсумкових карток (загальний обсяг палива, вартість палива, експлуатаційні витрати, загальна сума); зведену таблицю по техніці; помісячну зведену таблицю; нижній колонтитул з номером сторінки.



АгроОблік
Система обліку палива та витрат сільськогосподарської техніки

ЗВЕДЕНИЙ ЗВІТ
Період: 01.01.2026 – 06.04.2026
Сформовано: 06.04.2026 о 20:46

Всього палива	Витрати на паливо	Експлуатаційні витрати	ЗАГАЛЬНІ ВИТРАТИ
8 388,0 л	404 226,50 грн	237 500,00 грн	641 726,50 грн

Зведений звіт по техніці

Техніка	Рег. номер	Паливо (л)	Вартість палива	Год. роботи	Л/год	Експл. витрати	Всього витрат
Трактор John Deere 8R 310	АА1234ВВ	2 070,0 л	99 660,00 грн	68,5 год	30,22	26 700,00 грн	126 360,00 грн
Трактор New Holland T7.315	АА5678СС	1 410,0 л	67 977,50 грн	48,5 год	29,07	25 400,00 грн	93 377,50 грн
Трактор Claas Axion 870	ВВ9012ДД	1 290,0 л	61 990,00 грн	41,5 год	31,08	22 000,00 грн	83 990,00 грн
Трактор МТЗ-1221	СС3456ЕЕ	680,0 л	32 552,50 грн	29,5 год	23,05	14 700,00 грн	47 252,50 грн
Сівалка Horsch Maestro 12 SW	ЛЛ1234ММ	395,0 л	19 262,50 грн	15,0 год	26,33	35 700,00 грн	54 962,50 грн
Сівалка John Deere 1890	ММ5678НН	225,0 л	11 025,00 грн	8,5 год	26,47	24 700,00 грн	35 725,00 грн
Культиватор Horsch Tiger 6 MT	НН9012ОО	295,0 л	14 307,50 грн	10,5 год	28,10	7 200,00 грн	21 507,50 грн
Культиватор Lemken Karat 12	ОО3456ПП	310,0 л	15 035,00 грн	11,0 год	28,18	5 800,00 грн	20 835,00 грн
КамАЗ-65115 (зерновоз)	ТТ9012УУ	650,0 л	31 156,00 грн	37,0 год	17,57	30 200,00 грн	61 356,00 грн
МАЗ-6312 (зерновоз)	УУ3456ФФ	610,0 л	29 429,00 грн	34,0 год	17,94	33 900,00 грн	63 329,00 грн
Дискатор Horsch Joker 12 RT	ХХ1234ЦЦ	270,0 л	13 095,00 грн	9,5 год	28,42	0,00 грн	13 095,00 грн
Навантажувач Manitou MLT 741	ЦЦ5678ЧЧ	183,0 л	8 736,50 грн	13,5 год	13,56	11 200,00 грн	19 936,50 грн
РАЗОМ		**8 388,0 л**	**404 226,50 грн**			**237 500,00 грн**	**641 726,50 грн**

Помісячний звіт

Рік	Місяць	Паливо (л)	Вартість палива	Експл. витрати	Всього
2026	Квітень	845,0 л	41 405,00 грн	19 400,00 грн	60 805,00 грн
2026	Березень	2 990,0 л	145 057,00 грн	197 900,00 грн	342 957,00 грн
2026	Лютий	2 448,0 л	117 588,00 грн	298 100,00 грн	415 688,00 грн
2026	Січень	2 105,0 л	100 176,50 грн	249 400,00 грн	349 576,50 грн

АгроОблік — Система обліку палива та витрат с/г техніки

Стор. 1 з 1

Рисунок 3.4.1. – Приклад згенерованого PDF-звіту

3.5 Резервне копіювання бази даних

Для захисту від втрати даних реалізовано функціонал резервного копіювання та відновлення бази даних. Клас DatabaseBackupService надає метод CreateBackup(), який відкриває діалог збереження файлу та виконує копіювання файлу fueltracker.db. Метод RestoreBackup() перед відновленням автоматично створює поточну резервну копію, після чого замінює файл бази даних обраним файлом.

3.6 Тестування системи

Тестування системи проводилось у два етапи: функціональне тестування вручну (manual testing) та перевірка граничних значень (boundary testing). Тестові дані включали 25 операторів, 22 одиниці техніки, 60 записів витрати палива та 55 записів експлуатаційних витрат за період з січня по червень 2026 року.

Таблиця 3.1 — Тест-кейси функціонального тестування

№	Тест-кейс	Очікуваний результат	Статус
1	Додавання нової одиниці техніки з усіма заповненими полями	Запис зберігається, відображається у таблиці	Пройдено
2	Спроба зберегти техніку без назви	Відображається повідомлення про помилку валідації	Пройдено
3	Пошук техніки за кириличним текстом з великої літери	Коректний пошук без урахування регістру	Пройдено
4	Фільтрація записів палива за вибраною технікою	Відображаються лише записи обраної техніки	Пройдено
5	Збереження запису палива і перевірка, що фільтр техніки не скинувся	Вибір у ComboBox зберігається	Пройдено
6	Видалення техніки з пов'язаними записами палива	Каскадне видалення всіх пов'язаних записів	Пройдено
7	Формування звіту за обраний період	Коректні підсумкові суми та помісячна розбивка	Пройдено
8	Експорт звіту у PDF	PDF-файл створено та відкрито	Пройдено
9	Резервне копіювання та відновлення БД	База відновлена з резервної копії	Пройдено
10	DatePicker відображає українські назви місяців	Коректна локалізація (uk-UA)	Пройдено

Таблиця 3.2 — Результати перевірки продуктивності

Операція	Кількість записів	Час виконання, мс	Відповідність вимозі
Завантаження списку техніки	22	< 100	Так (≤ 2000 мс)
Завантаження журналу палива без фільтра	60	< 200	Так
Застосування фільтра за технікою	60	< 150	Так
Формування зведеного звіту	60 + 55 записів	< 400	Так
Генерація PDF	—	< 1500	Так (≤ 2000 мс)

Результати тестування підтверджують, що всі функціональні вимоги виконано коректно, а продуктивність системи відповідає нефункціональним вимогам. Жодного критичного дефекту не виявлено.

Висновки до розділу 3

У третьому розділі описано практичну реалізацію програмної системи. Реалізовано рівень доступу до даних на основі EF Core 9 та SQLite з автоматичним створенням схеми бази даних. Реалізовано п'ять ViewModel-класів відповідно до патерну MVVM. Вирішено проблему збереження вибору техніки у ComboBox при оновленні даних шляхом розділення методів завантаження довідників та застосування фільтрів. Реалізовано регістронезалежний пошук з підтримкою кирилиці на стороні C#. Налаштовано культуру uk-UA для коректної локалізації DatePicker. Реалізовано генерацію PDF-звітів за допомогою QuestPDF та функціонал резервного копіювання бази даних. Проведено функціональне тестування, що підтвердило коректність реалізації всіх вимог.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи досягнуто поставлену мету: спроектовано та розроблено програмну систему обліку використання палива та експлуатаційних витрат сільськогосподарської техніки «АгроОблік».

У ході роботи отримано такі результати:

– проведено аналіз предметної галузі та досліджено процеси обліку витрат на сільськогосподарську техніку; встановлено, що існуючі рішення або надто складні й дорогі для середнього агровиробника, або не охоплюють специфіку сільськогосподарської техніки;
– обґрунтовано вибір технологічного стеку: платформа .NET 9, мова C#, технологія WPF, СУБД SQLite, ORM Entity Framework Core, бібліотеки MaterialDesignThemes, CommunityToolkit.Mvvm та QuestPDF;
– розроблено архітектуру системи на основі патерну MVVM із чітким поділом рівнів представлення, логіки та даних; спроектовано реляційну схему бази даних із чотирьох таблиць;
– реалізовано функціонал управління реєстром техніки (22 поля характеристик), журналу використання палива з автоматичним розрахунком питомих витрат (л/год), обліку всіх видів експлуатаційних витрат та реєстру операторів;
– реалізовано гнучку систему фільтрації за датою, технікою та типом витрат із збереженням вибраного значення ComboBox при оновленні даних; реалізовано регістронезалежний пошук з підтримкою кирилиці;
– реалізовано генерацію PDF-звітів (зведений по техніці та помісячний) з використанням бібліотеки QuestPDF; налаштовано повну локалізацію інтерфейсу та компонентів датчиків на українську мову;

– проведено функціональне тестування системи на тестовому наборі даних (понад 130 записів): всі тест-кейси пройдено успішно, продуктивність системи відповідає встановленим вимогам.

Практичне значення розробленої системи полягає у забезпеченні аграрних підприємств інструментом для об'єктивного обліку та аналізу витрат на машинно-тракторний парк, що сприяє скороченню нераціонального використання ресурсів та обґрунтованому плануванню технічного обслуговування. Система є автономною, не потребує серверної інфраструктури та може бути розгорнута на будь-якому комп'ютері під управлінням Windows 10/11.

Напрямами подальшого розвитку системи можуть бути: реалізація хмарної синхронізації бази даних для роботи в мережі підприємства; розробка модуля планування технічного обслуговування з нагадуваннями; додавання модуля імпорту даних з Excel; розробка мобільного застосунку для реєстрації заправлень безпосередньо

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Microsoft. .NET 9 Documentation. URL: https://learn.microsoft.com/en-us/dotnet/ (дата звернення: 06.04.2026).
2. Microsoft. Windows Presentation Foundation Overview. URL: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/ (дата звернення: 06.04.2026).
3. Microsoft. Entity Framework Core Documentation. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 06.04.2026).
4. SQLite Consortium. SQLite Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 06.04.2026).
5. CommunityToolkit. MVVM Toolkit Documentation. URL: https://learn.microsoft.com/en-us/dotnet/communitytoolkit/mvvm/ (дата звернення: 06.04.2026).
6. QuestPDF. QuestPDF Documentation. URL: https://www.questpdf.com/documentation.html (дата звернення: 06.04.2026).
7. MaterialDesignInXAML. Material Design In XAML Toolkit. URL: https://github.com/MaterialDesignInXAML/MaterialDesignInXamlToolkit (дата звернення: 06.04.2026).
8. Gossman J. Introduction to Model/View/ViewModel pattern for building WPF Apps. URL: https://blogs.msdn.microsoft.com (дата звернення: 06.04.2026).
9. Troelsen A., Japikse P. Pro C# 10 with .NET 6. Foundational Principles and Practices in Programming. Apress, 2022. 1481 p.
10. Price M. C# 11 and .NET 7 — Modern Cross-Platform Development Fundamentals. Packt Publishing, 2022. 818 p.
11. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994. 395 p.
12. Fowler M. Patterns of Enterprise Application Architecture. Addison-Wesley, 2002. 533 p.

13. Дудник О. В. Автоматизація обліку матеріально-технічних ресурсів у сільськогосподарських підприємствах. Вісник аграрної науки. 2023. № 4. С. 45–52.
14. Петренко В. С. Інформаційні системи в управлінні аграрним виробництвом : навч. посіб. К. : Аграрна освіта, 2021. 248 с.
15. ДСТУ 3008:2015. Звіти у сфері науки і техніки. Структура та правила оформлювання. К. : ДП «УкрНДНЦ», 2016. 26 с.
16. Richard E., Lerman J. Programming Entity Framework: DbContext. O'Reilly, 2012. 354 p.
17. Smith J. WPF Apps with the Model-View-ViewModel Design Pattern. MSDN Magazine. 2009. URL: https://learn.microsoft.com/ (дата звернення: 06.04.2026).
18. Smith J. P. Entity Framework Core in Action : Second Edition. Manning Publications, 2021. 752 p.
19. Albahari J. C# 12 in a Nutshell : The Definitive Reference. O'Reilly Media, 2023. 1056 p.
20. Погорєлова О. В. Оцінка забезпеченості сільського господарства України технікою. Інвестиції: практика та досвід. 2024. № 3. С. 71–76.
21. Черв'яков В. Аналіз ефективності застосування ERP-систем для обліку матеріальних ресурсів у сільськогосподарських підприємствах. Геральдика. 2025. URL: http://heraldes.khmnu.edu.ua/index.php/heraldes/article/download/2275/2324 (дата звернення: 06.04.2026).