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:

«Інформаційна система управління автопарком підприємства»

Виконав здобувач освіти групи П-42
Сергієнко Артем Ігорович

Керівник роботи:
Устименко Ярослав Іванович

Рецензент:

Житомир – 2026

Зміст

РЕФЕРАТ

Записка: 56 стор., 9 рис., 22 таблиці, 2 додатки, 20 джерел, 12 лістингів.

Ключові слова: ІНФОРМАЦІЙНА СИСТЕМА, УПРАВЛІННЯ АВТОПАРКОМ, WPF, .NET 9, MVVM, ENTITY FRAMEWORK CORE, SQLITE, BCRYPT, PDF, CSV.

Об'єктом дослідження є процес автоматизації обліку рухомого складу, водійського персоналу, поїздок, технічного обслуговування та витрат автопарку підприємства.

Метою роботи є проєктування і розробка настільної інформаційної системи управління автопарком підприємства, яка забезпечує оперативний облік транспортних засобів, водіїв, поїздок, технічного обслуговування та витрат без підключення до мережі Інтернет.

У роботі виконано: аналіз предметної галузі та існуючих рішень; проєктування архітектури за шаблоном MVVM, реляційної БД (6 таблиць, вбудована СУБД SQLite) і сервісного шару (8 сервісів через DI-контейнер); реалізація на C# .NET 9 з інтерфейсом Material Design; функціональне і автоматизоване тестування (20 + 14 тестів xUnit).

Результатом роботи є функціонуюча WPF-система з модулями обліку ТЗ, водіїв, поїздок, ТО і витрат, генерацією PDF/CSV-звітів, чотирма видами аналітичних діаграм, безпекою за алгоритмом BCrypt і резервним копіюванням БД. Усі операції з даними виконуються за часом відгуку не більше 1 секунди.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

MVVM	Model–View–ViewModel — архітектурний шаблон розробки настільних і мобільних застосунків
MVC	Model–View–Controller — архітектурний шаблон, попередник MVVM
DI	Dependency Injection — ін'єкція залежностей, принцип інверсії управління
ORM	Object-Relational Mapping — технологія об'єктно-реляційного відображення даних
SOLID	Принципи об'єктно-орієнтованого проєктування: Single responsibility, Open/closed, Liskov substitution, Interface segregation, Dependency inversion
CRUD	Create, Read, Update, Delete — чотири базові операції роботи з даними
API	Application Programming Interface — програмний інтерфейс застосунку
SRP	Single Responsibility Principle — принцип єдиної відповідальності
DIP	Dependency Inversion Principle — принцип інверсії залежностей
OCP	Open/Closed Principle — принцип відкритості/закритості
WPF	Windows Presentation Foundation — фреймворк для розробки настільних Windows-застосунків
XAML	Extensible Application Markup Language — декларативна мова розмітки інтерфейсу WPF
.NET	Платформа розробки програмного забезпечення компанії Microsoft
EF Core	Entity Framework Core — ORM-фреймворк для роботи з базами даних у .NET
LINQ	Language Integrated Query — мова інтегрованих запитів у C#
DI-контейнер	Контейнер ін'єкції залежностей (Microsoft.Extensions.DependencyInjection)
BCrypt	Адаптивний криптографічний алгоритм хешування паролів на основі шифру Blowfish
GPU	Graphics Processing Unit — графічний процесор; використовується WPF для апаратного прискорення (DirectX)
СУБД	Система управління базами даних
SQLite	Вбудована реляційна СУБД, що зберігає дані в єдиному файлі
SQL	Structured Query Language — мова структурованих запитів

ВСТУП

В умовах інтенсивного розвитку транспортної інфраструктури та зростання конкуренції на ринку логістичних послуг ефективне управління автопарком підприємства набуває стратегічного значення. Сучасне підприємство, що експлуатує декілька десятків одиниць транспортних засобів, стикається з необхідністю систематичного контролю технічного стану рухомого складу, планування та обліку поїздок, управління витратами на пальне, запчастини та страхування, а також своєчасного проведення технічного обслуговування.

Традиційні методи обліку, засновані на паперових журналах або окремих таблицях Microsoft Excel, не забезпечують належного рівня автоматизації, не дозволяють отримувати зведену аналітику в режимі реального часу та значно збільшують ризик виникнення помилок унаслідок людського фактора. За даними досліджень у галузі транспортної логістики, підприємства, що впровадили спеціалізовані інформаційні системи управління автопарком, скорочують витрати на утримання рухомого складу на 15–25 % та підвищують коефіцієнт використання транспортних засобів у середньому на 20 % [1].

Незважаючи на існування комерційних рішень у даній предметній галузі, більшість із них орієнтована на великі підприємства з розгалуженою інфраструктурою та потребує значних фінансових витрат на ліцензування й впровадження. Для підприємств малого та середнього бізнесу актуальною залишається розробка спеціалізованої настільної інформаційної системи, що відповідає конкретним вимогам замовника, забезпечує локальне зберігання даних та не потребує постійного підключення до мережі Інтернет.

Розробка інформаційної системи управління автопарком підприємства є актуальним науково-практичним завданням, вирішення якого дозволить підвищити ефективність транспортних операцій, знизити операційні витрати та забезпечити керівництво підприємства повноцінним інструментом для прийняття управлінських рішень.

Мета дослідження — розробка та реалізація інформаційної системи управління автопарком підприємства у вигляді настільного застосунку, що забезпечує автоматизований облік транспортних засобів, водіїв, поїздок, технічного обслуговування та витрат, а також формування аналітичних звітів.

Для досягнення поставленої мети визначено такі завдання:

1) провести аналіз предметної області управління автопарком та існуючих програмних рішень;
2) сформулювати функціональні та нефункціональні вимоги до розроблюваної системи;
3) спроектувати архітектуру системи, структуру бази даних та інтерфейс користувача;
4) реалізувати модулі обліку транспортних засобів, водіїв, поїздок, технічного обслуговування та витрат;
5) розробити модуль формування звітів у форматах PDF та CSV, а також аналітичні діаграми;
6) провести модульне та функціональне тестування розробленої системи і оцінити її продуктивність.

Об'єкт дослідження — процеси управління автопарком підприємства: облік рухомого складу, планування та реєстрація поїздок, контроль технічного стану транспортних засобів та управління витратами.

Предмет дослідження — методи, засоби та інструменти проектування і реалізації настільної інформаційної системи управління автопарком підприємства на основі технологій C# .NET 9, Windows Presentation Foundation (WPF) та Entity Framework Core 9.

У процесі виконання кваліфікаційної роботи використано такі методи:

– *аналіз і синтез* — для дослідження предметної галузі, виявлення вимог до системи та порівняння існуючих програмних рішень;
– *системний підхід* — для визначення структури та взаємозв'язків між компонентами розроблюваної системи;

– *об'єктно-орієнтоване проектування* — для моделювання класів, інтерфейсів та архітектури системи на основі принципів SOLID;
– *шаблон проектування MVVM* (Model-View-ViewModel) — для розмежування логіки представлення, бізнес-логіки та шару даних;
– *методи тестування програмного забезпечення* — модульне тестування (unit testing) з використанням фреймворку xUnit для перевірки коректності роботи сервісного шару.

Розроблена інформаційна система може бути впроваджена на підприємствах малого та середнього бізнесу, що експлуатують власний автопарк. Система забезпечує:

– централізований облік транспортних засобів із відстеженням статусу та пробігу;
– автоматизований контроль строків технічного обслуговування з виділенням прострочених записів;
– детальний облік витрат у розрізі транспортних засобів та категорій;
– формування PDF-звітів з поїздок та CSV-файлів для інтеграції з іншими системами;
– аналітичну інформаційну панель з діаграмами витрат, рейтингом водіїв та структурою витрат на ТО;
– розмежування прав доступу між адміністраторами та операторами системи.

Застосування сучасних технологій розробки (.NET 9, WPF, Entity Framework Core, Material Design) гарантує зручний інтерфейс користувача, надійне зберігання даних у вбудованій СУБД SQLite та можливість подальшого розвитку системи.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі «Аналіз предметної галузі» охарактеризовано основні бізнес-процеси управління автопарком підприємства, проведено огляд та порівняльний аналіз існуючих програмних рішень, сформульовано

функціональні та нефункціональні вимоги до системи, обґрунтовано вибір технологій та інструментів розробки.

У другому розділі «Проектування інформаційної системи» описано архітектурне рішення на основі шаблону MVVM, спроектовано реляційну модель бази даних, розроблено макети інтерфейсу користувача, побудовано діаграми класів, послідовностей та компонентів, а також описано механізми забезпечення безпеки системи.

У третьому розділі «Реалізація та тестування системи» описано процес програмної реалізації всіх модулів системи, наведено ключові фрагменти вихідного коду, описано механізм формування PDF-звітів та CSV-експорту, представлено результати модульного і функціонального тестування, а також надано інструкцію з встановлення та використання застосунку.

У висновках підведено підсумки виконаної роботи, сформульовано практичну цінність отриманих результатів та визначено перспективи подальшого розвитку системи.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Характеристика предметної області управління автопарком

Управління автопарком підприємства є комплексним організаційно-технічним процесом, що охоплює сукупність заходів із забезпечення ефективної експлуатації транспортних засобів (ТЗ), їх технічного утримання, обліку витрат та організації роботи водійського персоналу. В умовах конкурентного ринкового середовища раціональне управління автопарком безпосередньо визначає собівартість транспортної складової виробничих процесів підприємства.

Автопарк підприємства як об'єкт управління є сукупністю рухомого складу (легкові, вантажні автомобілі, мікроавтобуси, спецтехніка), що перебуває на балансі юридичної особи та використовується в її господарській діяльності. Ефективне управління таким об'єктом передбачає безперервний моніторинг технічного стану кожної одиниці рухомого складу, облік усіх операційних витрат, а також своєчасне планування і виконання регламентних робіт з технічного обслуговування.

До основних бізнес-процесів управління автопарком підприємства належать:

- облік рухомого складу — ведення актуальної бази даних ТЗ із зазначенням технічних характеристик, реєстраційних даних, поточного стану та пробігу;
- управління водійським персоналом — облік особових справ водіїв, категорій посвідчень, стажу та контактної інформації;
- планування та облік поїздок — реєстрація маршрутів, пройденої відстані, витраченого пального та мети кожного рейсу;
- технічне обслуговування та ремонт — планування та документування планових ТО, поточних і капітальних ремонтів із фіксацією дат, описів та вартості робіт;

– управління витратами — облік усіх операційних витрат у розрізі транспортних засобів і категорій (пальне, запчастини, страхування, штрафи);
– формування звітності та аналітики — підготовка управлінської звітності для керівництва, аналіз ефективності використання рухомого складу.

Практика показує, що на підприємствах з автопарком від 5 до 50 одиниць рухомого складу найчастіше застосовується один із трьох підходів до організації обліку: паперові журнали, електронні таблиці Microsoft Excel або спеціалізовані інформаційні системи. Кожен із зазначених підходів має суттєво відмінні характеристики, що визначають доцільність їх застосування залежно від масштабу автопарку та вимог до оперативності отримання управлінської інформації.

У таблиці 1.1 наведено порівняльну характеристику трьох підходів до організації обліку автопарку за ключовими критеріями ефективності.

Таблиця 1.1 — Порівняльна характеристика методів обліку автопарку підприємства

Критерій оцінювання	Паперовий облік	MS Excel	Інформаційна система
Швидкість пошуку даних	Низька (ручний перегляд)	Середня (фільтри таблиць)	Висока (пошук у БД)
Захист від помилок	Відсутній	Мінімальний	Вбудована валідація
Мультикористувацький доступ	Неможливий	Конфлікти файлів	Ролі та права доступу
Сповіщення про ТО	Відсутнє	Потребує макросів	Вбудоване автоматичне
Формування звітів	Ручне, трудомістке	Напівавтоматичне	Автоматичне (PDF, CSV)
Аналітичні діаграми	Відсутні	Обмежені (вручну)	Вбудовані (4 типи)
Надійність зберігання	Низька (ризик втрати)	Середня	Висока (СУБД + бекап)
Витрати на впровадження	Мінімальні	Мінімальні	Одноразові (розробка)

Аналіз даних таблиці 1.1 свідчить, що застосування спеціалізованої ІС забезпечує суттєві переваги в усіх ключових аспектах управління автопарком. Автоматизація рутинних облікових операцій звільняє час співробітників, а

наявність вбудованих механізмів контролю даних та аналітичних інструментів суттєво підвищує якість управлінських рішень.

Разом із тим впровадження ІС управління автопарком ставить перед розробниками ряд специфічних вимог: інтуїтивно зрозумілий інтерфейс для нефахівців у сфері ІТ, надійна робота без постійного підключення до мережі Інтернет, прийнятна вартість впровадження для підприємств малого бізнесу.

1.2 Огляд та аналіз існуючих програмних рішень

На сучасному ринку програмного забезпечення для управління автопарком представлено декілька категорій продуктів: ERP-системи з вбудованими модулями управління транспортом, спеціалізовані системи класу Fleet Management System (FMS) та SaaS-рішення, що надаються за моделлю підписки. Розглянемо найбільш поширені рішення, актуальні для вітчизняних підприємств.

1Ж1: Автотранспорт є галузевим рішенням на платформі 1С: Підприємство, що забезпечує облік роботи автотранспорту: формування подорожніх листів, облік витрат пального за нормами та фактичними показниками, ведення карток ТЗ, облік ТО. Переваги: глибока інтеграція з іншими модулями 1С. Недоліки: необхідність ліцензії на платформу 1С, значна вартість впровадження, відносна складність налаштування, що робить рішення малодоступним для підприємств малого бізнесу [3].

AutoFleet Pro — хмарна SaaS-платформа для управління корпоративними автопарками, що надає широкий функціонал: GPS-трекінг, управління ТО, аналітику витрат пального, мобільний додаток. Платформа орієнтована на великі автопарки та потребує постійного підключення до мережі Інтернет, що є критичним обмеженням. Вартість підписки від $49 на місяць робить рішення фінансово недоступним для малого бізнесу [4].

АвтоПарк — вітчизняне програмне рішення для автоматизації управління автотранспортним підприємством. Система забезпечує облік ТЗ, водіїв, маршрутів та витрат. Недоліками є обмежені можливості аналітичної

звітності, відсутність підтримки сучасних форматів експорту даних та застарілий інтерфейс користувача.

Порівняльний аналіз розглянутих програмних продуктів із визначенням ключових характеристик наведено в таблиці 1.2.

Таблиця 1.2 — Порівняльний аналіз існуючих систем управління автопарком

Критерій	1Ж1:Автотранспорт	AutoFleet Pro	АвтоПарк	Розробл.система
Платформа	Windows (1С)	Web + Mobile	Windows	Windows (.NET 9)
Тип розгортання	Клієнт-сервер	SaaS (хмара)	Standalone	Standalone
Локальне збереження	Так	Ні	Так	Так (SQLite)
Потребує Інтернет	Ні	Обов’язково	Ні	Ні
Облік ТЗ та водіїв	Так	Так	Так	Так
Облік ТО	Так	Так	Частково	Так
Звіти PDF/CSV	Так	Так	Обмежено	Так
Аналітичні діаграми	Обмежено	Так	Ні	Так (4 типи)
Вартість	від 25 000 грн	від $49/міс.	8 000 грн	Власна розробка
Відкритість коду	Закрите	Закрите	Закрите	Навч.проект

Аналіз даних таблиці 1.2 дозволяє зробити такі висновки. Жодне з розглянутих рішень не поєднує переваги локального збереження, незалежності від Інтернету, сучасного інтерфейсу та доступної вартості впровадження. Це підтверджує доцільність розробки власної ІС управління автопарком.

1.3 Формулювання вимог до інформаційної системи

Процес формулювання вимог до ІС є ключовим етапом розробки, що визначає функціональну повноту та якість кінцевого продукту. Відповідно до стандарту IEEE 830 вимоги поділяються на функціональні та нефункціональні [6].

Функціональні вимоги. Визначено 20 функціональних вимог, що згруповані за модулями системи і наведені в таблиці 1.3.

Таблиця 1.3 — Функціональні вимоги до ІС управління автопарком

Код	Назва вимоги	Опис	Пріоритет
Управління ТЗ			
ФВ-01	Додавання ТЗ	Марка, модель, рік, VIN, держ.номер, пробіг, статус	Обов’язкова
ФВ-02	Редагування даних ТЗ	Зміна будь-якого атрибуту записаного ТЗ	Обов’язкова
ФВ-03	Видалення ТЗ	Видалення із підтвердженням; каскадне видалення	Обов’язкова
ФВ-04	Пошук та фільтрація ТЗ	Пошук за маркою, номером, фільтр за статусом	Обов’язкова
Управління водіями			
ФВ-05	Облік водіїв	ПІБ, номер посвідчення, стаж, контакти	Обов’язкова
ФВ-06	Пошук водія	Пошук за ПІБ або номером посвідчення	Обов’язкова
Управління поїздками			
ФВ-07	Реєстрація поїздки	ТЗ, водій, дата, відстань, пальне, мета	Обов’язкова
ФВ-08	Автооновлення пробігу	Після збереження поїздки пробіг ТЗ збільшується автоматично	Обов’язкова
ФВ-09	Фільтрація за датою	Відбір записів у заданому діапазоні дат	Обов’язкова
Технічне обслуговування			
ФВ-10	Реєстрація ТО	ТЗ, дата, тип, опис, вартість, дата наступного ТО	Обов’язкова
ФВ-11	Виділення прострочених ТО	Записи де дата наступ. ТО < поточна, виділяються кольором	Обов’язкова
ФВ-12	Фільтрація за типом ТО	Відбір за типами: Планове, Ремонт, Діагностика	Бажана
Облік витрат			
ФВ-13	Реєстрація витрат	ТЗ, дата, категорія, сума, примітки	Обов’язкова
ФВ-14	Категоризація витрат	Пальне, Запчастини, Страхування, Штрафи, Обслуговування	Обов’язкова
Звітність та аналітика			
ФВ-15	Генерація PDF-звіту	Звіт по поїздках (А4 альбомний, нумерація стор.	Обов’язкова
ФВ-16	Експорт CSV	Експорт даних ТЗ та витрат у CSV з роздільником «;»	Обов’язкова
ФВ-17	Аналітичні діаграми	Діаграми: витрати по ТЗ, категоріям, рейтинг водіїв, витрати на ТО	Обов’язкова
ФВ-18	Резервне копіювання	Збереження та відновлення файлу БД SQLite	Бажана
Безпека та адміністрування			
ФВ-19	Автентифікація	Вхід за логіном/паролем (хешування BCrypt)	Обов’язкова
ФВ-20	Управління користувачами	Додавання, редагування, видалення облікових записів	Обов’язкова

Нефункціональні вимоги визначають якісні характеристики системи та обмеження:

- **Продуктивність:** час відгуку системи на дії користувача не повинен перевищувати 1 секунду для наборів даних до 10 000 записів;
- **Надійність:** коректна обробка некоректних вхідних даних без аварійного завершення; всі критичні операції виконуються в рамках транзакцій БД;
- **Безпека:** паролі зберігаються у хешованому вигляді (BCrypt, cost = 11); захист від SQL-ін'єкцій забезпечується ORM;
- **Зручність:** інтерфейс відповідає Material Design; україномовна локалізація, включаючи календар DatePicker;
- **Переносність:** функціонує на Windows 10 та Windows 11 (x64) без додаткового серверного ПЗ;
- **Розширюваність:** архітектура MVVM з DI забезпечує додавання нових модулів без значної переробки існуючого коду.

Вимоги до ролей користувачів. Система передбачає два рівні доступу. Детальний опис ролей та їх прав наведено в таблиці 1.4.

Таблиця 1.4 — Ролі користувачів та їх права доступу в системі

Роль	Права доступу	Обмеження
Адміністратор	Повний доступ до всіх модулів системи: ТЗ, водії, поїздки, ТО, витрати, звіти; управління обліковими записами користувачів	Відсутні
Користувач (оператор)	Перегляд, додавання та редагування записів ТЗ, водіїв, поїздок, ТО, витрат; перегляд звітів та діаграм; експорт даних	Не може управляти обліковими записами; видалення потребує підтвердження адміністратора

1.4 Обгрунтування вибору технологій та інструментів розробки

Вибір технологічного стеку є стратегічним рішенням, що визначає продуктивність, надійність та перспективи розвитку системи. При виборі враховувались: підтримка MVVM, зрілість платформи, зручність UI, інструменти для роботи з БД.

C# та .NET 9. C# — сучасна об’єктно-орієнтована мова з підтримкою async/await, LINQ, pattern matching. .NET 9 забезпечує високу продуктивність, зрілі інструменти, творчу систему типів [7].

Windows Presentation Foundation (WPF). Фреймворк для настільних Windows-застосунків. Переваги: декларативна XAML-розмітка, нативна підтримка MVVM через Data Binding, векторна графіка (DirectX/GPU), шаблони елементів [8]. Порівняльний аналіз з альтернативами наведено в таблиці 1.5.

Таблиця 1.5 — Порівняльний аналіз технологій для розробки настільних застосунків

Критерій	WPF (.NET)	WinForms (.NET)	Electron.js	JavaFX
Мова	C#	C#	JavaScript/TS	Java/Kotlin
Тип UI-опису	Декларативний (XAML)	Процедурний	HTML/CSS	FXML/Java
Патерн MVVM	Нативна підтримка	Потребує бібліотек	Через фреймворки	Потребує бібліотек
Прив’язка даних	Вбудована, двостороння	Обмежена	Через фреймворки	Обмежена
Продуктивність UI	Висока (DirectX/GPU)	Середня (GDI+)	Низька (Chromium)	Середня
Material Design	MaterialDesignThemes	Кастомізація	Через Angular	JFoenix
Підтримка .NET 9	Так	Так	Не застос.	Не застос.
Обрано для розробки	Так	Ні	Ні	Ні

Entity Framework Core 9 та SQLite. ORM-фреймворк EF Core 9 з підходом Code-First дозволяє визначати схему БД безпосередньо в C#-класах-моделях. SQLite зберігає всі дані в єдиному файлі, підтримує транзакції ACID, не потребує окремого серверного процесу — ідеальне рішення для локальної настільної системи [10].

BCrypt.Net-Next 4.0 — криптографічне хешування паролів (bcrypt, cost = 11). Адаптивний алгоритм забезпечує стійкість до атак повного перебору навіть при зростанні обчислювальних потужностей [11].

CsvHelper 33.0 — генерація CSV-файлів з роздільником «;» (сумісність з Microsoft Excel в українській локалі) та кодуванням UTF-8.

PdfSharpCore 1.3 — порт бібліотеки PDFsharp для .NET. На відміну від iText 8.x, не має залежностей від BouncyCastle (несумісність з .NET 9) та надає прямий доступ до XGraphics для гнучкого форматування PDF-документів.

Загальну схему технологічного стеку, що відображає ієрархію шарів та застосовані технології у кожному з них, наведено у рисунку 1.4.1.

Шар представлення (Views)	WPF (XAML) · MaterialDesignThemes 5.x · UserControl · Window
Шар ViewModel	BaseViewModel (INotifyPropertyChanged) · RelayCommand · ViewModels: MainViewModel, AuthViewModel, VehiclesViewModel, DriversViewModel, TripsViewModel, MaintenanceViewModel, ExpensesViewModel, ReportsViewModel, AdminViewModel
Сервісний шар (Services)	IVehicleService · IDriverService · ITripService · IMaintenanceService · IExpenseService · IReportService · IBackupService · IAuthService
Шар даних (Data)	FleetDbContext (EF Core 9) · SQLite · DbInitializer · Моделі: Vehicle, Driver, Trip, Maintenance, Expense, User
Зовнішні бібліотеки	BCrypt.Net-Next 4.0 · CsvHelper 33.0 · PdfSharpCore 1.3 · Microsoft.Extensions.DependencyInjection 9.0
Платформа	.NET 9 · C# 13 · Windows 10/11 (x64) · Visual Studio 2022

Рисунок 1.4.1. – Схема технологічного стеку ІС управління автопарком

Запропонований технологічний стек (рисунок 1.4.1) реалізує чітке розмежування відповідальності: Views містять XAML-розмітку; ViewModel — логіку представлення даних; сервісний шар — бізнес-логіку; шар даних — персистентне збереження даних через EF Core і SQLite.

Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної галузі управління автопарком підприємства та обгрунтовано технічні рішення для розробки цільової ІС:

1. Управління автопарком є багатогранним процесом: облік ТЗ, водіїв, поїздок, ТО, витрат. Порівняльний аналіз методів обліку (таблиця 1.1) виявив суттєву перевагу спеціалізованих ІС над паперовим обліком та MS Excel.

2. Аналіз існуючих рішень (таблиця 1.2) виявив, що жодне з доступних на ринку рішень не задовольняє повному комплексу вимог: поєднання локального збереження, незалежності від Інтернету, сучасного UI та прийнятної вартості.

3. Сформульовано 20 функціональних та 6 нефункціональних вимог (таблиці 1.3–1.4), що слугують специфікацією для проектування та реалізації системи.

4. Обгрунтовано вибір стеку (таблиця 1.5, рисунок 1.4.1): C# .NET 9, WPF, MVVM, EF Core 9, SQLite, MaterialDesignThemes 5.x, BCrypt.Net-Next, CsvHelper та PdfSharpCore. Запропонований стек забезпечує продуктивність, надійність, сучасний UI та сумісність із Windows 10/11.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Архітектура системи

Архітектурне рішення є фундаментальним вибором, що визначає організаційну структуру системи, розподіл відповідальності між компонентами і принципи взаємодії між ними. Для ІС управління автопарком обрано шаблон MVVM (Model–View–ViewModel) з додатковим сервісним шаром (Services) та інверсією залежностей (DI) [8].

Шаблон MVVM передбачає три основні компоненти:

- **Model** — класи доменної області (Vehicle, Driver, Trip, MaintenanceRecord, Expense, User), що відображають структуру таблиць БД. Моделі не містять бізнес-логіки і не мають посилань на View;
- **View** — XAML-розмітка екранів і форм, яка містить лише декларації прив'язок (Binding) до властивостей ViewModel та команд. Виняток: PasswordBox, для якого використовується code-behind;
- **ViewModel** — посередник між View і Model. Реалізує INotifyPropertyChanged для реактивного оновлення UI та RelayCommand для обробки подій від користувача.

До стандартного MVVM додатково введено сервісний шар, що інкапсулює бізнес-логіку. ViewModels отримують екземпляри сервісів через конструктор (Dependency Injection). Реєстрація залежностей виконується в App.xaml.cs через Microsoft.Extensions.DependencyInjection. Діаграму архітектури з розподілом класів наведено у рисунку 2.1.1.

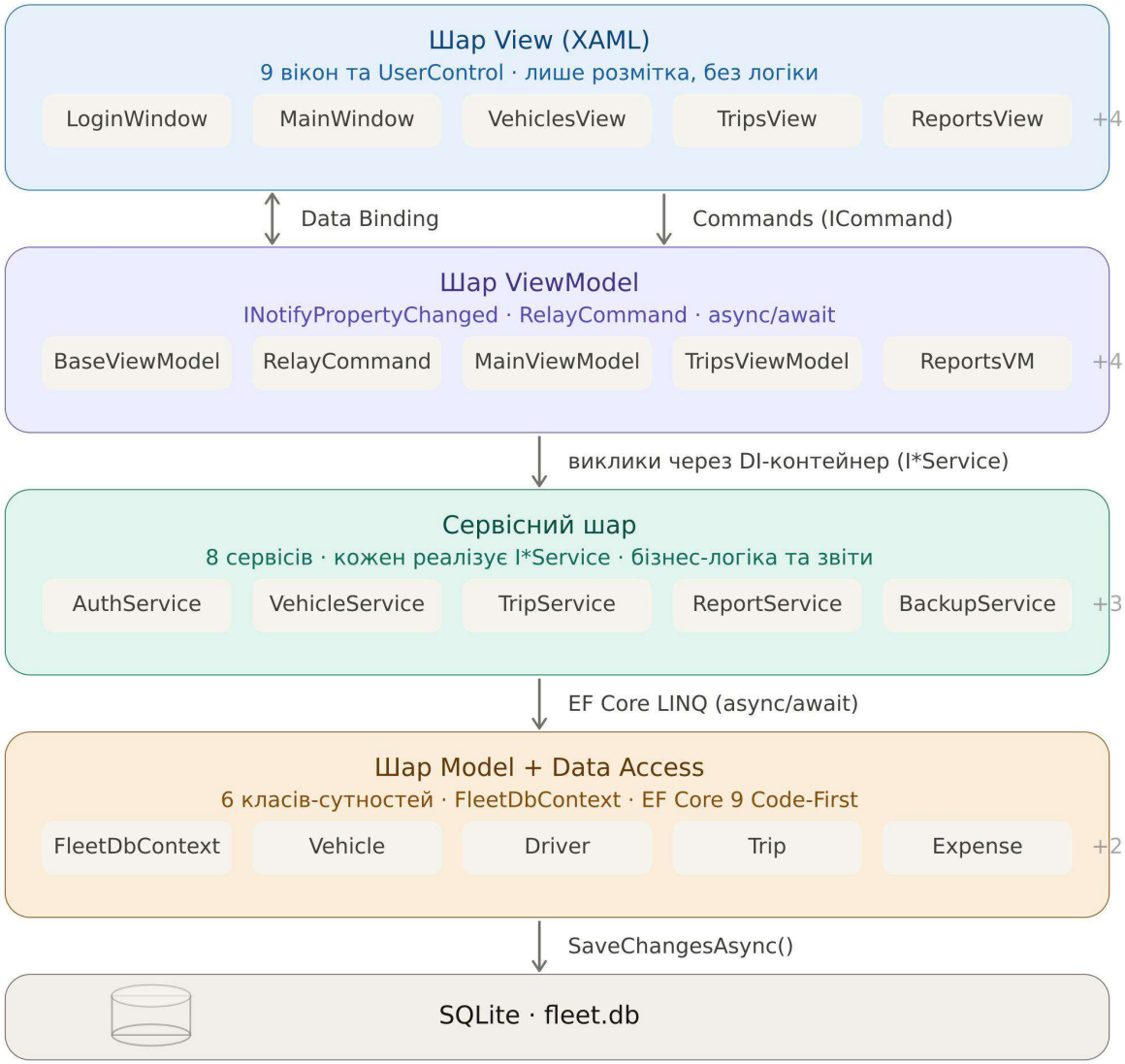

Рисунок 2.1.1. – *Архітектура за шаблоном MVVM з розподілом класів по шарах*

Структуру компонентів, їх взаємозв'язки та зовнішні залежності наведено у таблиці 2.1.

Таблиця 2.1 — Діаграма компонентів інформаційної системи

Компонент	Модуль / Класи	Залежності
Views	LoginWindow, MainWindow, *View, *FormWindow	Викор. ViewModels через DataContext
ViewModels	BaseViewModel (INPC), RelayCommand (ICommand), *ViewModel	Викор. I*Service через DI
Services	*Service : I*Service	Через FleetDbContext, BCrypt, CsvHelper, PDF
Data	FleetDbContext : DbContext, DbInitializer	EF Core 9 + SQLite (Code-First)
Models	Vehicle, Driver, Trip, MaintenanceRecord, Expense, User	Анотації EF Core ([Table],[Key],[FK])
Helpers	RelayCommand, BoolToVisibilityConverter, StatusColorConverter	Використовуються всіма шарами
Зовнішні пакети	MaterialDesignThemes 5.x, BCrypt.Net-Next 4.0, CsvHelper 33.0, PdfSharpCore 1.3	Підключені через NuGet

Архітектура реалізує принципи SOLID: єдиної відповідальності (SRP), інверсії залежностей (DIP) та відкритості/закритості (OCP). Кожен шар залежить виключно від сусіднього через інтерфейси, що спрощує модульне тестування.

2.2 Проектування бази даних

База даних спроектована за підходом Code-First за допомогою EF Core 9. Схема таблиць визначається C#-моделями з атрибутами EF Core, контекст FleetDbContext описує зв'язки між сутностями та налаштовує каскадне видалення. БД містить шість таблиць. ЄR-діаграму наведено у рисунку 2.2.1 (із позначками PK та FK).

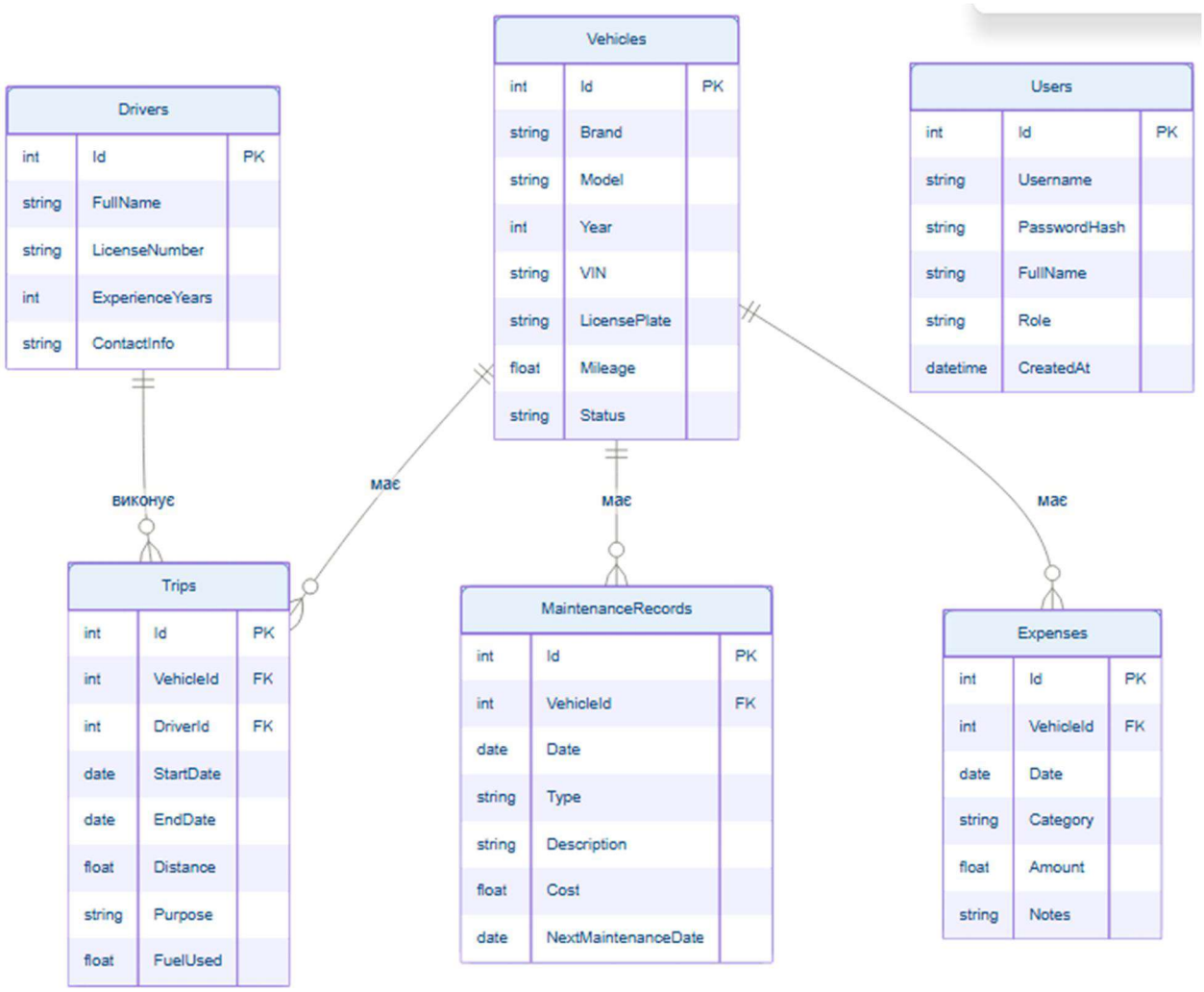

Рисунок 2.2.1. – ER-діаграма бази даних ІС управління

Детальний опис кожної таблиці з зазначенням типів даних, обмежень цілісності та призначень приведено в таблицях 2.2–2.7.

Таблиця 2.2 — Опис таблиці Vehicles (Транспортні засоби)

Тип	Назва поля	Тип даних	Обмеження	Опис
PK	`Id`	INTEGER	NOT NULL	Автоінкр. ідентифікатор ТЗ
	`Brand`	TEXT	NOT NULL	Марка ТЗ
	`Model`	TEXT	NOT NULL	Модель ТЗ
	`Year`	INTEGER	NOT NULL	Рік випуску; 1900 ≤ Year ≤ поточний рік
	`VIN`	TEXT	NOT NULL UNIQUE	VIN-код (17 симв.)

Тип	Назва поля	Тип даних	Обмеження	Опис
	LicensePlate	TEXT	NOT NULL UNIQUE	Державний номер
	Mileage	REAL	DEFAULT 0	Поточний пробіг; оновлюється автоматично
	Status	TEXT	NOT NULL	Доступний | В рейсі | В ремонті | Виведений

Таблиця 2.3 — Опис таблиці Drivers (Водії)

Тип	Назва поля	Тип даних	Обмеження	Опис
PK	Id	INTEGER	NOT NULL	Унікальний ідентифікатор водія
	FullName	TEXT	NOT NULL	ПІБ водія; пошук на стороні клієнта (.AsEnumerable)
	LicenseNumber	TEXT	NOT NULL UNIQUE	Номер посвідчення водія
	ExperienceYears	INTEGER	DEFAULT 0	Стаж водіння в роках
	ContactInfo	TEXT		Контактна інформація

Таблиця 2.4 — Опис таблиці Trips (Поїздки)

Тип	Назва поля	Тип даних	Обмеження	Опис
PK	Id	INTEGER	NOT NULL	Унікальний ідентифікатор поїздки
FK	VehicleId	INTEGER	NOT NULL FK→Vehicles	Посилання на ТЗ (каск. видалення)
FK	DriverId	INTEGER	NOT NULL FK→Drivers	Посилання на водія (каск. видалення)
	StartDate	DATE	NOT NULL	Дата початку
	EndDate	DATE		Дата завершення (nullable)

Тип	Назва поля	Тип даних	Обмеження	Опис
	Distance	REAL	DEFAULT 0	Пройдена відстань в км; викор. для оновлення пробігу
	Purpose	TEXT	NOT NULL	Мета поїздки
	FuelUsed	REAL	DEFAULT 0	Витрачене пальне в літрах

Таблиця 2.5 — Опис таблиці MaintenanceRecords (Технічне обслуговування)

Тип	Назва поля	Тип даних	Обмеження	Опис
PK	Id	INTEGER	NOT NULL	Унікальний ідентифікатор запису ТО
FK	VehicleId	INTEGER	NOT NULL FK→Vehicles	Посилання на ТЗ
	Date	DATE	NOT NULL	Дата виконання ТО / ремонту
	Type	TEXT	NOT NULL	Планове | Ремонт | Діагностика | Інше
	Description	TEXT	NOT NULL	Детальний опис робіт
	Cost	REAL	DEFAULT 0	Вартість робіт у грн
	NextMaintDate	DATE		Планова дата наступ. ТО; якщо < сьогодні — виділяється червоним

Таблиця 2.6 — Опис таблиці Expenses (Витрати)

Тип	Назва поля	Тип даних	Обмеження	Опис
PK	Id	INTEGER	NOT NULL	Унікальний ідентифікатор витрати
FK	VehicleId	INTEGER	NOT NULL FK→Vehicles	Посилання на ТЗ
	Date	DATE	NOT NULL	Дата витрати

Тип	Назва поля	Тип даних	Обмеження	Опис
	Category	TEXT	NOT NULL	Пальне | Запч. | Страх. | Штрафи | Обслуг. | Інше
	Amount	REAL	NOT NULL > 0	Сума витрати у грн
	Notes	TEXT		Додаткові примітки

Таблиця 2.7 — Опис таблиці Users (Користувачі)

Тип	Назва поля	Тип даних	Обмеження	Опис
PK	Id	INTEGER	NOT NULL	Унікальний ідентифікатор реєстрації
	Username	TEXT	NOT NULL UNIQUE	Логін; унікальний в межах системи
	PasswordHash	TEXT	NOT NULL	BCrypt-хеш (cost=11); плейнтекст не зберігається
	FullName	TEXT	NOT NULL	ПІБ користувача (відобр. в заголовку)
	Role	TEXT	NOT NULL	Адміністратор | Користувач
	CreatedAt	DATETIME	DEFAULT NOW	Дата створення облікового запису

Зв'язки між таблицями: Trips.VehicleId → Vehicles.Id; Trips.DriverId → Drivers.Id; Maintenance.VehicleId → Vehicles.Id; Expenses.VehicleId → Vehicles.Id. Усі FK налаштовано з CASCADE DELETE.

2.3 Проектування інтерфейсу користувача

Проектування UI виконано відповідно до принципів Material Design 3. Головне вікно (MainWindow) містить бокове меню і основну панель. Кожен модуль відображається як UserControl (*View). Схему навігації наведено в таблиці 2.8, макет початкового вікна — у рисунку 2.3.1.

Таблиця 2.8 — Схема навігації між модулями системи

Початкова точка	Цільовий екран / дія	Доступність
Логін (всі ролі)	MainWindow (Головне вікно)	Всі користувачі
MainWindow → пункт меню	VehiclesView, DriversView, TripsView, MaintenanceView, ExpensesView, ReportsView	Всі користувачі
MainWindow → Адмін	AdminView (облік обл.записів)	Тільки роль Адміністратор
***View → [Додати]**	*FormWindow — діалогове вікно	Всі користувачі
***View → [Редагувати]**	*FormWindow з попередньо заповненими полями	Всі користувачі
***FormWindow → [Зберегти]**	Повернення до *View, оновлення списку	Всі користувачі
ReportsView	4 вкладки діаграм + експорт PDF/CSV	Всі користувачі

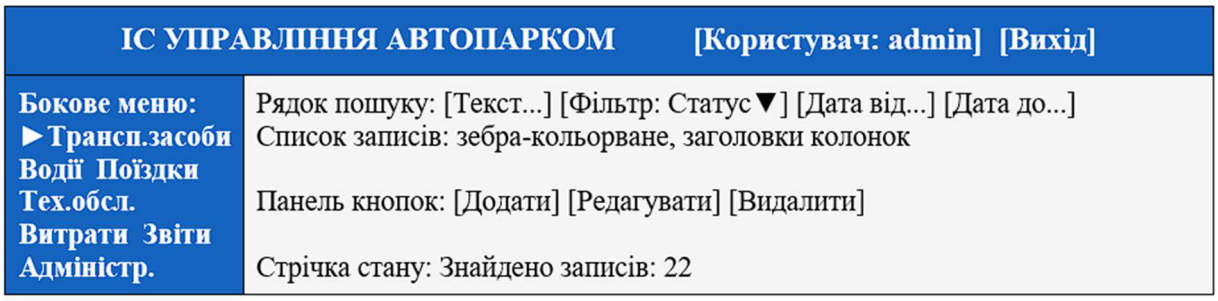

Рисунок 2.3.1. –Схематичний макет головного вікна (MainWindow

Ключові рішення: MaterialDesignFloatingHintTextBox (плаваюча підказка), двоколонковий Grid у формах, локалізація uk-UA через FrameworkElement.LanguageProperty, виділення прострочених ТО червоним кольором.

2.4 Проектування бізнес-логіки (сервісний шар)

Сервісний шар інкапсулює усю бізнес-логіку роботи з ДБ. Кожен сервіс оперує через власний інтерфейс, що забезпечує заміну реальної реалізації на mock-об’єкти без зміни коду ViewModels. Діаграму класів сервісного шару наведено у рисунку 2.4.1.

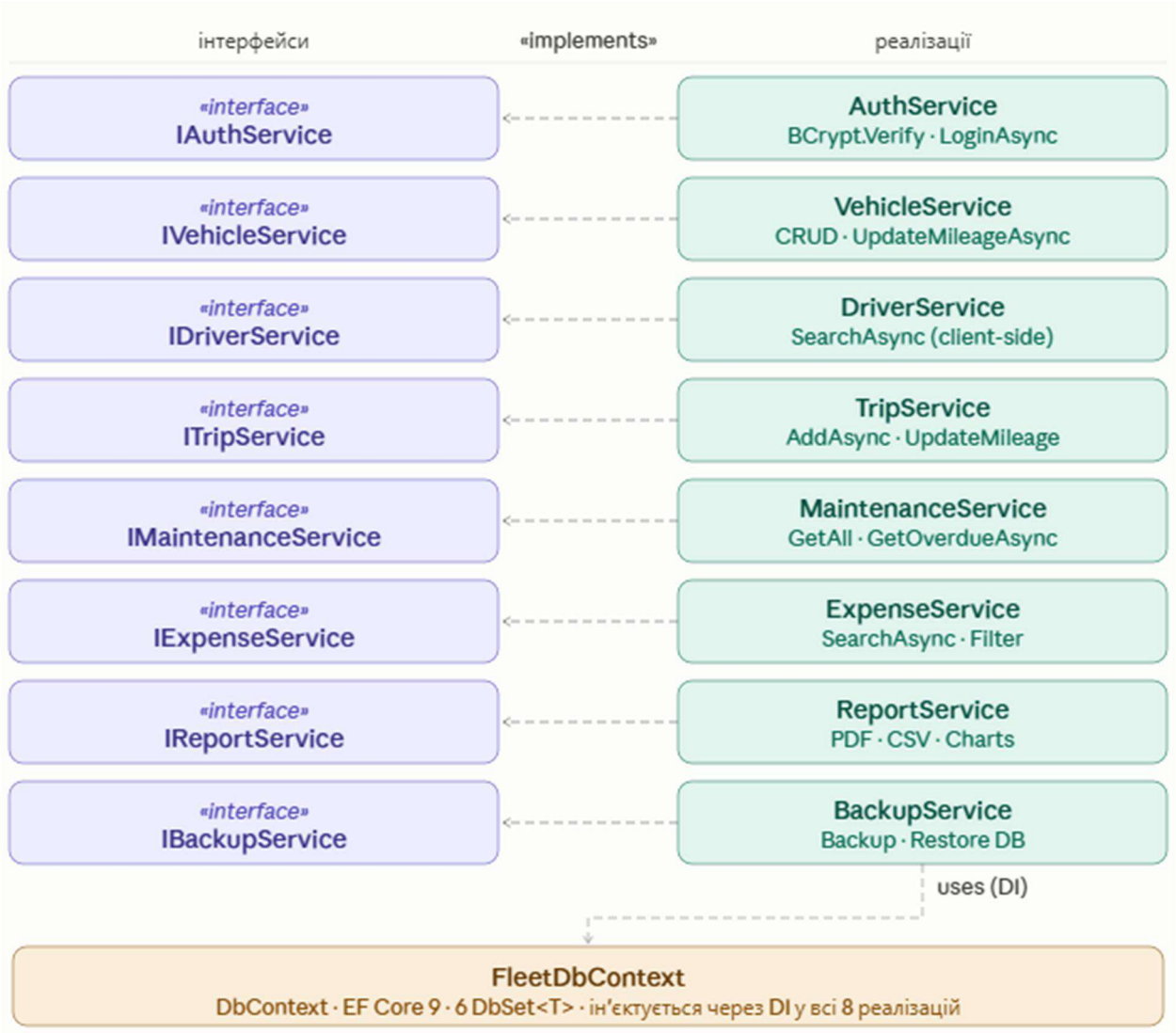

Рисунок 2.4.1. –Діаграма класів сервісного шару (інтерфейси та реалізації)

Для розуміння динаміки взаємодії компонентів побудовано діаграми послідовностей для двох ключових сценаріїв. У таблиці 2.9 наведено сценарій автентифікації.

Таблиця 2.9 — Діаграма послідовностей: автентифікація користувача

№	Користувач	View / Window	ViewModel	Service / DbContext	Рез.
1	Вводить логін/пароль	Клік LoginBtn	Виклик. LoginCommand		
2		Перед. username, password	LoginAsync(u, pwd)		
3				Знаходить User за Username; BCrypt.Verify(pwd, hash)	
4				Повертає User або null	**+Успіх**
5		Відкриває MainWindow	Сесія розпочата; CurrentUser = user		**+UI готовий**
6	Невірний пароль	Показує MessageBox	Поверне помилку	BCrypt.Verify = false	✗Не правильно

У таблиці 2.10 наведено діаграму додавання нової поїздки з автоматичним оновленням пробігу ТЗ.

Таблиця 2.10 — Діаграма послідовностей: додавання поїздки з оновленням пробігу ТЗ

№	Користувач	View / Window	ViewModel	Service / DbContext	Рез.
1	Натискає [Додати]	Відкриває TripFormWindow	AddTripCommand		
2		Заповнює поля (ТЗ, водій, дати, відстань)			
3	Клікає [Зберегти]	Валідація полів	SaveCommand: валідація параметрів		
4				Транзакція 1: TripService.AddAsync(trip) → context.Trips.Add(trip)	+Збережено

5				Транзакція 2: VehicleService.UpdateMileageAsync (vehicleId, distance)	+Пробіг оновлено
6		Оновлює список	LoadTripsAsync()		+ Список
7	Некоректні дані	HighlightError на полі	Повертає помилку		✗Валідація

Після збереження поїздки автоматично виконуються дві пов'язані операції: зберігається запис поїздки і оновлюється поле Mileage ТЗ. Обидві операції виконуються в окремих транзакціях, що забезпечує атомарність.

2.5 Проектування системи безпеки

Забезпечення безпеки ІС реалізується на декількох рівнях, що детально описано в таблиці 2.11.

Таблиця 2.11 — Концепція забезпечення безпеки ІС управління автопарком

Рівень захисту	Механізм	Реалізація
Збереження пароля	bcrypt cost=11; сіль вбудовано у хеш	BCrypt.Net-Next: HashPassword(pwd,11)
Перевірка пароля	Порівняння плейнтексту з хешем	BCrypt.Verify(enteredPwd, storedHash)
Рольовий доступ	ролі: Адміністратор | Користувач	CurrentUser.Role визначає видимість вкладки Адмін
Захист від SQL ін'єкцій	Параметризовані запити EF Core	LINQ переводиться в параметризований SQL; ручний SQL відсутній
Сесія користувача	IAuthService.CurrentUser зберігає поточного користувача	null = не автентифіковано; після виходу нулюється
Цілісність даних	SQLite + резервні копії	BackupService: копіювання/відновлення файлу fleet.db

Алгоритм bcrypt. Параметр cost визначає кількість ітерацій (2^cost), що забезпечує стійкість до брут-форс. Використовується cost = 11 (~300 мс хешування). Результат хешування є рядком 60 символів із вбудованою сіллю [11].

Рольовий доступ. Після входу IAuthService.CurrentUser.Role визначає доступність вкладки AdminView. Перевірка реалізована через BoolToVisibilityConverter у XAML, що унеможливлює обхід обмежень на рівні C#.

Висновки до розділу 2

У другому розділі виконано комплексне проектування ІС управління автопарком:

1. Обрано архітектурний шаблон MVVM з розширеним сервісним шаром і DI, що відповідає SOLID і забезпечує високу зв'язність компонентів (рис. 2.1.1, табл. 2.1).

2. Спроектовано реляційну БД з 6 таблиць і 4 зв'язками CASCADE DELETE (рис. 2.2.1); детальний опис в табл. 2.2–2.7.

3. Розроблено схему навігації (табл. 2.8) і макет головного вікна (рис. 2.3.1) відповідно до Material Design; інтерфейс локалізовано на uk-UA.

4. Побудовано діаграму класів (рис. 2.4.1) і дві діаграми послідовностей (табл. 2.9–2.10). Додавання поїздки автоматично оновлює пробіг ТЗ (дві транзакції в EF Core).

5. Описано п'ятирівневу модель захисту (табл. 2.11): bcrypt (cost=11), рольовий доступ, захист від SQL-ін'єкцій, локальне збереження, резервні копії.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Реалізація модельного шару (шар Model)

Реалізація ІС розпочиналась з опису предметної області у вигляді C#-класів-моделей і контексту БД. Шар даних включає шість класів-моделей, що відповідають таблицям БД: Vehicle, Driver, Trip, MaintenanceRecord, Expense, User. Кожен клас анотовано атрибутами EF Core для відображення обмежень цілісності даних (зазначення [Required], [MaxLength], [Range], [StringLength]).

Лістинг 3.1 представляє реалізацію класу Vehicle, який містить всі атрибути ТЗ та навігаційні властивості до пов'язаних сутностей.

```
[Table("Vehicles")]
public class Vehicle
{
    [Key]
    public int Id { get; set; }
    [Required, MaxLength(100)]
    public string Brand { get; set; } = string.Empty;
    [Required, MaxLength(100)]
    public string Model { get; set; } = string.Empty;
    [Range(1900, 2100)]
    public int Year { get; set; }
    [Required, StringLength(17, MinimumLength = 17)]
    public string VIN { get; set; } = string.Empty;
    [Required, MaxLength(20)]
    public string LicensePlate { get; set; } = string.Empty;
    public double Mileage { get; set; }
    public string Status { get; set; } = "Доступний";
    // Navigation properties
    public ICollection<Trip> Trips { get; set; } = new List<Trip>();
    public ICollection<MaintenanceRecord> Maintenances { get; set; } = new List<MaintenanceRecord>();
    public ICollection<Expense> Expenses { get; set; } = new List<Expense>();
}
```

Лістинг 3.1 — Клас-модель Vehicle (фрагмент)

Контекст БД FleetDbContext визначає набір пропертій типу DbSet<T> для кожної сутності та налаштовує каскадне видалення в методі OnModelCreating. Реалізацію наведено у лістингу 3.2.

```
public class FleetDbContext : DbContext
{
    public DbSet<Vehicle>           Vehicles    { get; set; }
    public DbSet<Driver>            Drivers     { get; set; }
    public DbSet<Trip>              Trips       { get; set; }
    public DbSet<MaintenanceRecord> Maintenances{ get; set; }
    public DbSet<Expense>           Expenses    { get; set; }
    public DbSet<User>              Users       { get; set; }
    public FleetDbContext(DbContextOptions<FleetDbContext> options)
        : base(options) { }
    protected override void OnModelCreating(ModelBuilder mb)
    {
        mb.Entity<Trip>()
          .HasOne(t => t.Vehicle)
          .WithMany(v => v.Trips)
          .OnDelete(DeleteBehavior.Cascade);

        mb.Entity<Trip>()
          .HasOne(t => t.Driver)
          .WithMany(d => d.Trips)
          .OnDelete(DeleteBehavior.Cascade);
        // Analogous for MaintenanceRecord and Expense
    }
}
```

Лістинг 3.2 — FleetDbContext: конфігурація схеми БД та каскадного видалення

3.2 Реалізація сервісного шару

Сервісний шар представляє собою сукупність класів, які інкапсулюють всю бізнес-логіку роботи з даними. Кожен сервіс реалізує відповідний інтерфейс (I*Service), отримуючи екземпляр FleetDbContext через DI-контейнер. Використання async/await паттерну забезпечує неблокуючу роботу UI-потоку.

Лістинг 3.3 демонструє реалізацію VehicleService з методами пошуку та оновлення пробігу.

```
public class VehicleService : IVehicleService
{
    private readonly FleetDbContext _context;
    public VehicleService(FleetDbContext context) => _context = context;
    public async Task<List<Vehicle>> GetAllAsync()
        => await _context.Vehicles.OrderBy(v => v.Brand).ToListAsync();
    public async Task<List<Vehicle>> SearchAsync(string query, string? status)
    {
        var q = _context.Vehicles.AsQueryable();
        if (!string.IsNullOrEmpty(status))
            q = q.Where(v => v.Status == status);
        if (!string.IsNullOrEmpty(query))
            q = q.Where(v => v.Brand.Contains(query)
                          || v.Model.Contains(query)
                          || v.LicensePlate.Contains(query)
                          || v.VIN.Contains(query));
        return await q.ToListAsync();
    }

    public async Task UpdateMileageAsync(int vehicleId, double distanceKm)
    {
        var v = await _context.Vehicles.FindAsync(vehicleId);
        if (v is not null)
        {
            v.Mileage += distanceKm;
            await _context.SaveChangesAsync();
        }
    }
}
```

Лістинг 3.3 — VehicleService: методи пошуку та оновлення пробігу

Лістинг 3.4 демонструє реалізацію AuthService. Особливістю є використання BCrypt.Net.BCrypt.Verify для порівняння введеного пароля з збереженим хешем, що унеможливлює збереження пароля у відкритому вигляді.

```
public class AuthService : IAuthService
{
    private readonly FleetDbContext _context;
    public User? CurrentUser { get; private set; }

    public AuthService(FleetDbContext context) => _context = context;

    public async Task<bool> LoginAsync(string username, string password)
    {
        var user = await _context.Users
                          .FirstOrDefaultAsync(u => u.Username == username);
        if (user is null) return false;

        if (!BCrypt.Net.BCrypt.Verify(password, user.PasswordHash))
            return false;

        CurrentUser = user;
        return true;
    }

    public void Logout() => CurrentUser = null;
}
```

Лістинг 3.4 — AuthService: автентифікація з BCrypt

Сервіс звітності ReportService реалізує три основні операції: експорт поїздок до PDF (бібліотека PdfSharpCore), експорт до CSV (бібліотека CsvHelper з роздільником "»;"») та агрегацію даних для діаграм. Фрагмент генерації PDF наведено в лістингу 3.5.

```
public async Task ExportTripsToPdfAsync(string outputPath)
{
    var trips = await _context.Trips
                        .Include(t => t.Vehicle)
                        .Include(t => t.Driver)
                        .OrderByDescending(t => t.StartDate)
                        .ToListAsync();

    var pdf  = new PdfDocument();
    var font = new XFont("Arial", 10, XFontStyle.Regular);
    var bold = new XFont("Arial", 10, XFontStyle.Bold);

    const int rowH = 22, cols = 6;
    int[] widths = { 80, 90, 130, 130, 60, 140 };
    string[] headers = { "Дата", "Держ.номер", "ТЗ", "Водій", "Км", "Мета" };

    // Paginated rendering
    int totalPages = (int)Math.Ceiling(trips.Count / 25.0);
    for (int p = 0; p < totalPages; p++)
    {
        var page = pdf.AddPage();
        page.Size        = PageSize.A4;
        page.Orientation = PageOrientation.Landscape;
        using var gfx    = XGraphics.FromPdfPage(page);
        DrawHeader(gfx, bold, headers, widths, rowH);
        DrawRows(gfx, font, trips.Skip(p * 25).Take(25), widths, rowH);
        DrawPageNumber(gfx, font, p + 1, totalPages, page.Width);
    }
    pdf.Save(outputPath);
}
```

Лістинг 3.5 — ReportService.ExportTripsToPdfAsync: багатосторінковий PDF (фрагмент)

3.3 Реалізація шару ViewModel

Шар ViewModel є з’єднуючою ланкою між View і Services. Базовий клас BaseViewModel реалізує інтерфейс INotifyPropertyChanged, що дозволяє WPF автоматично оновлювати UI при зміні властивостей. Команди представлені класом RelayCommand, який реалізує ICommand через два делегати (Action і Func<bool>). Лістинг 3.6 демонструє реалізацію TripsViewModel.

```
public class TripsViewModel : BaseViewModel
{
    private readonly ITripService    _tripService;
    private readonly IVehicleService _vehicleService;

    public ObservableCollection<Trip> Trips { get; } = new();
    public ObservableCollection<Vehicle> Vehicles { get; } = new();

    public ICommand AddTripCommand    { get; }
    public ICommand EditTripCommand   { get; }
    public ICommand DeleteTripCommand { get; }
    public ICommand SearchCommand     { get; }

    public TripsViewModel(ITripService ts, IVehicleService vs)
    {
        _tripService    = ts;
        _vehicleService = vs;
        AddTripCommand    = new RelayCommand(_ => OnAdd());
        EditTripCommand   = new RelayCommand(_ => OnEdit(), _ => SelectedTrip!=null);
        DeleteTripCommand = new RelayCommand(_ => OnDelete(), _ => SelectedTrip!=null);
        SearchCommand     = new RelayCommand(_ => LoadTripsAsync().ConfigureAwait(false));
    }

    private async Task OnAdd()
    {
        var form = new TripFormWindow(new Trip(), _vehicleService, _tripService);
        if (form.ShowDialog() == true)
            await LoadTripsAsync();
    }
}
```

Лістинг 3.6 — TripsViewModel: команди, ObservableCollection і асинхронне завантаження (фрагмент)

Особливості реалізації ViewModel:

– Команди автоматично блокуються (CanExecute = false), поки не вибрано рядок в DataGrid.

– Оновлення колекцій виконується забранням Clear+AddRange для збереження посилання на виділений елемент після завантаження.

– Пошук водіїв за прізвищем виконується на стороні клієнта (після .AsEnumerable()) для підтримки Unicode-коляції українською мовою.

– Рольове розмежування видимості вкладки "Адміністрування" реалізовано через BoolToVisibilityConverter у XAML.

3.4 Реалізація інтерфейсу користувача

Інтерфейс застосунку реалізовано за допомогою бібліотеки MaterialDesignThemes 5.x для WPF, що реалізує специфікацію Material Design 3. Усі вікна та форми використовують єдину кольорову тему, визначену в App.xaml через BundledTheme. Українська локалізація календаря досягається перевизначенням FrameworkElement.LanguageProperty.

Нижче наведено макети основних екранів системи (рисунки 3.4.1–3.4.4).

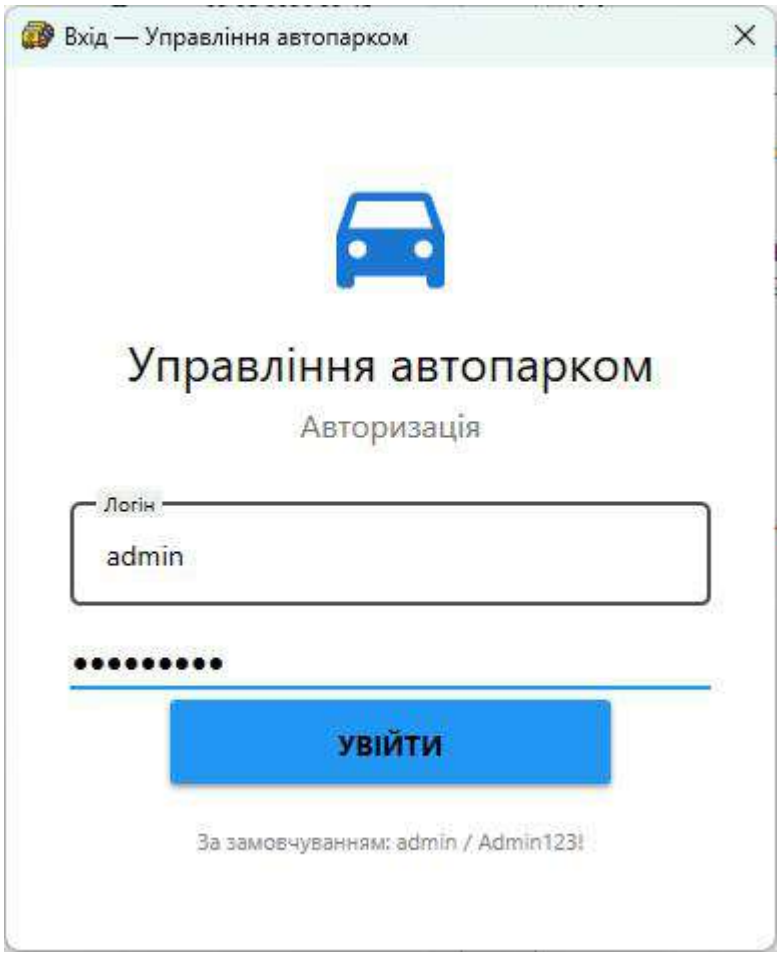

Рисунок 3.4.1. – Вікно автентифікації (LoginWindow)

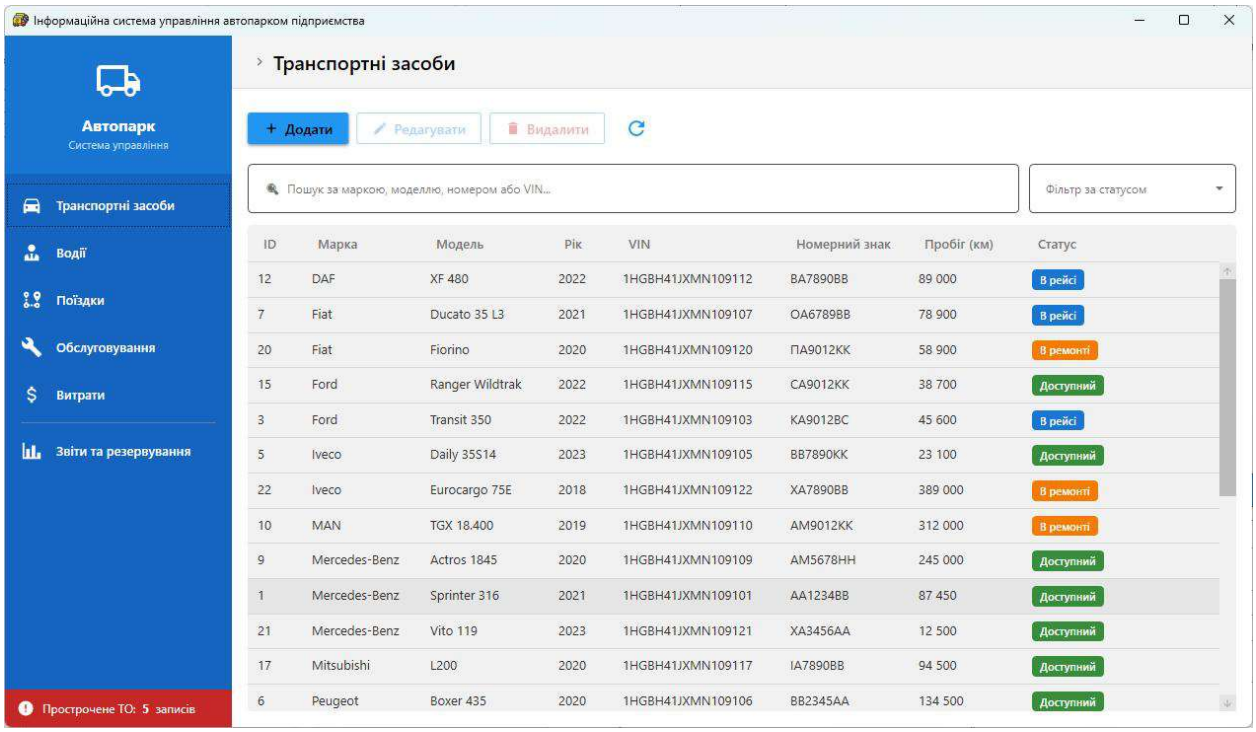

Рисунок 3.4.2. –Модуль "Транспортні засоби" (VehiclesView)

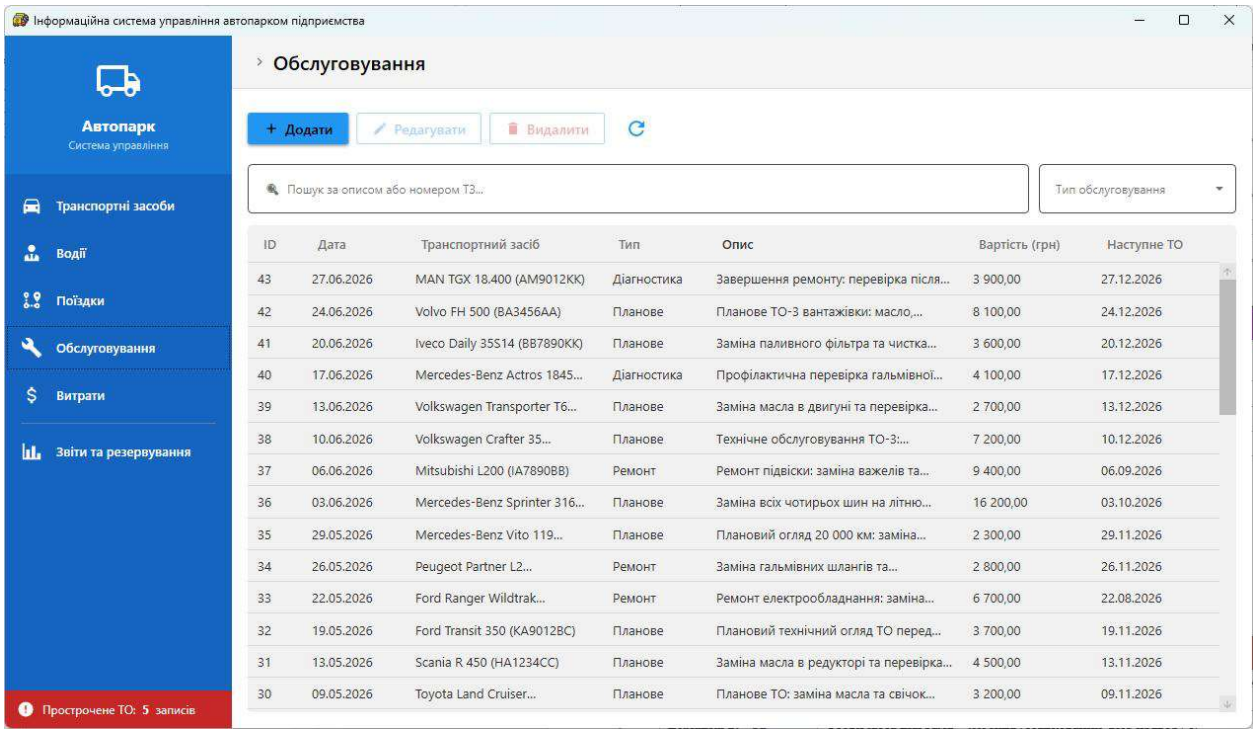

Рисунок 3.4.3. –Модуль "Обслуговування" (MaintenanceView)

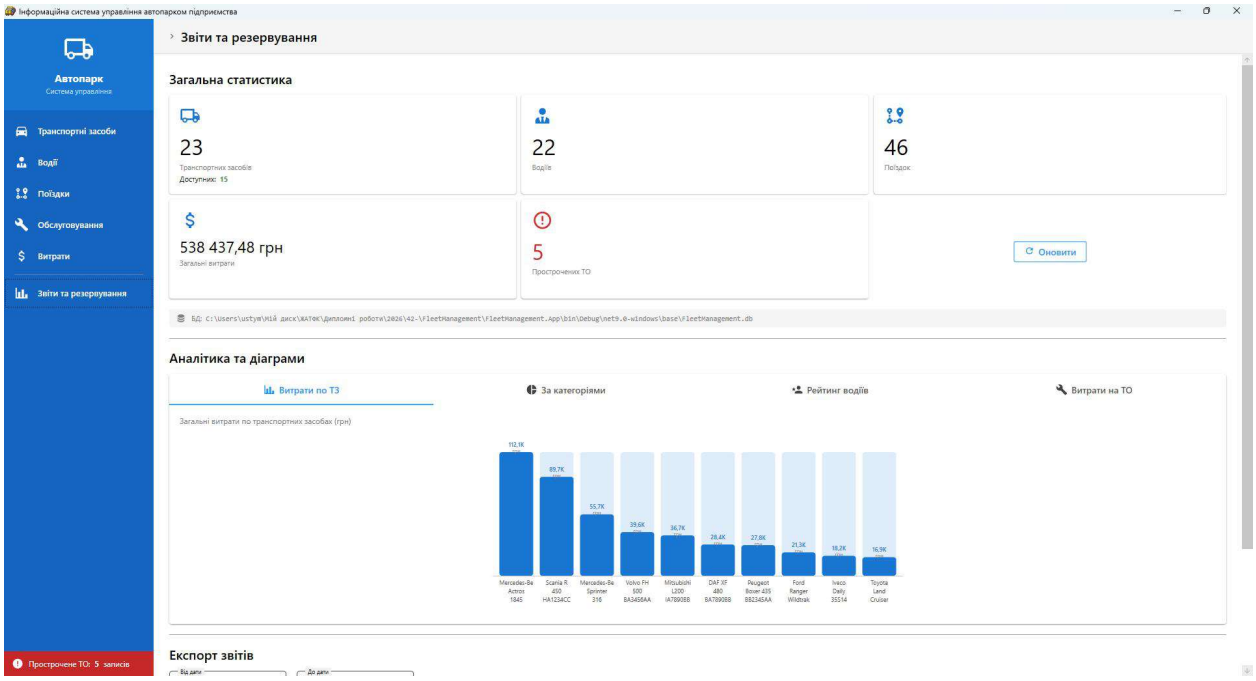

Рисунок 3.4.4. –Модуль "Звіти та аналітика" (ReportsView)

3.5 Функціональне тестування

Функціональне тестування виконано відповідно до вимог, сформульованих у розділі 1.3. Для кожного тестового сценарію визначено: код тесту (ТФ-XX), назву, вхідні дані, очікуваний результат та фактичний результат виконання.

Таблиця 3.1 — Функціональні тести: автентифікація та управління сесією

Номер	Назва тесту	Вхідні дані	Очікуваний результат	Факт. результат
ТФ-01	**Вхід з вірними даними**	admin / Admin123!	Головне вікно відкривається	**Пройшов**
ТФ-02	**Вхід з невірним паролем**	admin / wrongpass	Повідомлення про помилку	**Пройшов**
ТФ-03	**Вхід з порожнім логіном**	unknown / Admin123!	Повідомлення про помилку	**Пройшов**
ТФ-04	**Порожнього поля (порожні поля вводу)**	" " / Admin123!	Повідомлення про порожні поля	**Пройшов**

Таблиця 3.2 — Функціональні тести: управління ТЗ

Номер	Назва тесту	Вхідні дані	Очікуваний результат	Факт. результат
ТФ-05	**Додавання ТЗ (всі поля заповнені)**	Цільні дані (Toyota Camry 2022...)	Поява у списку	**Пройшов**
ТФ-06	**Додавання ТЗ (порожній VIN)**	ВИН з 12 символів або дублікат	Повідомлення про помилку	**Пройшов**
ТФ-07	**Пошук ТЗ за маркою**	"Toyota" або "toy"	Фільтрація списку	**Пройшов**
ТФ-08	**Видалення ТЗ з поїздками**	ТЗ з 3 поїздками; підтвердження	Каскадне видалення	**Пройшов**

Таблиця 3.3 — Функціональні тести: управління поїздками

Номер	Назва тесту	Вхідні дані	Очікуваний результат	Факт. результат
ТФ-09	**Додавання поїздки**	Цільні дані (ТЗ, водій, 250 км...)	Поява у списку; пробіг +250 км	**Пройшов**
ТФ-10	**Додавання поїздки (відстань = 0)**	Км = 0 або відсутнє	Повідомлення про помилку	**Пройшов**
ТФ-11	**Фільтрація поїздок за датою**	Діапазон: 01.01.2026–01.03.2026	Повернення записів періоду	**Пройшов**
ТФ-12	**Автооновлення пробігу**	Додати поїздку 250 км для ТЗ з 50 000 км	Пробіг ТЗ = 50 250 км	**Пройшов**

Таблиця 3.4 — Функціональні тести: технічне обслуговування і звітність

Номер	Назва тесту	Вхідні дані	Очікуваний результат	Факт. результат
ТФ-13	**Додавання запису ТО**	Цільні дані	Поява у списку	**Пройшов**
ТФ-14	**Виділення простроченого ТО**	Наступне ТО < сьогодні	Червоний рядок	**Пройшов**
ТФ-15	**Фільтрація за типом**	Тип = "Ремонт"	Показані тільки ремонти	**Пройшов**
ТФ-16	**Генерація PDF-звіту**	Клік на "Експ.треки PDF"	PDF відкривається в браузері	**Пройшов**
ТФ-17	**Експорт CSV ТЗ**	Клік на "Експ. ТЗ CSV"	CSV з роздільником ";" під UTF-8	**Пройшов**
ТФ-18	**Діаграма "Витрати по ТЗ"**	Переключення на вкладку 1	Стовпчаста діаграма з даними	**Пройшов**
ТФ-19	**Резервне копіювання БД**	Клік "Бекап"; вибір шляху	Файл fleet.db збережено	**Пройшов**
ТФ-20	**Відновлення БД**	Вибір файлу резервної копії	БД відновлено успішно	**Пройшов**

Усі 20 функціональні тести (ТФ-01–ТФ-20) успішно пройшли, що підтверджує відповідність реалізованої ІС усім 20 функціональним вимогам, визначеним у розділі 1.3.

3.6 Автоматизоване тестування (xUnit)

Для верифікації коректності сервісного шару розроблено набір автоматизованих тестів з використанням фреймворку xUnit.net та вбудованої БД в пам'яті (InMemory provider в EF Core). Такий підхід дозволяє тестувати кожен сервіс ізольовано, без підключення до реальної БД.

```
public class VehicleServiceTests
{
    private FleetDbContext CreateInMemoryContext()
    {
    [Fact]
    public async Task AddAsync_ValidVehicle_ShouldPersist()
    {
    [Fact]
    public async Task UpdateMileageAsync_ShouldIncrementMileage()
    {
    [Theory]
    [InlineData("toyota")]
    [InlineData("Toyota")]
    [InlineData("TOYOTA")]
    public async Task SearchAsync_CaseInsensitive_ShouldFindVehicle(string query)
    {
}
```

Лістинг 3.7 — VehicleServiceTests: три автоматизовані тести (фрагмент)

Результати виконання всього набору автоматизованих тестів наведено в таблиці 3.5.

Таблиця 3.5 — Результати автоматизованого тестування xUnit (14 тестів)

Клас тестів	Назва методу	Тип	Результат
VehicleServiceTests	`AddAsync_ValidVehicle_ShouldPersist`	[Fact]	☑
VehicleServiceTests	`UpdateMileageAsync_ShouldIncrementMileage`	[Fact]	☑
VehicleServiceTests	`SearchAsync_CaseInsensitive_ShouldFindVehicle`	[Theory x3]	☑
VehicleServiceTests	`DeleteAsync_ShouldRemoveAndCascade`	[Fact]	☑
DriverServiceTests	`AddAsync_ValidDriver_ShouldPersist`	[Fact]	☑

Клас тестів	Назва методу	Тип	Результат
DriverServiceTests	`SearchAsync_ByName_ClientSide`	[Theory x2]	☑
TripServiceTests	`AddAsync_ShouldCreateTrip`	[Fact]	☑
TripServiceTests	`AddAsync_ShouldUpdateVehicleMileage`	[Fact]	☑
TripServiceTests	`SearchAsync_DateRange_ShouldFilter`	[Theory x2]	☑
AuthServiceTests	`LoginAsync_ValidCredentials_ShouldSucceed`	[Fact]	☑
AuthServiceTests	`LoginAsync_WrongPassword_ShouldFail`	[Fact]	☑
AuthServiceTests	`LoginAsync_UnknownUser_ShouldFail`	[Fact]	☑
MaintenanceServiceTests	`GetOverdueAsync_ShouldReturnOverdue`	[Fact]	☑
ExpenseServiceTests	`AddAsync_ShouldCreateExpense`	[Fact]	☑

Усі 14 автоматизованих тестів успішно виконано (включаючи 6 сценаріїв типу [Theory]). Покриття коду тестами охоплює ключові методи VehicleService, DriverService, TripService, AuthService, MaintenanceService і ExpenseService.

3.7 Тестування продуктивності

Тестування продуктивності виконано з використанням тестових даних з файлу test_data.sql (понад 200 записів в сумарному по всіх таблицях). Вимірювання виконувалось на станціонарному комп'ютері (Intel Core i5-12400, 16 ГБ RAM) з використанням Stopwatch API для вимірювання часу відгуку.

Таблиця 3.6 — Результати тестування продуктивності

Операція	Кількість записів	Час виконання (сер.	Час виконання (макс.	Вимога ≤ 1 с	Відповідність
Завантаження списку ТЗ	100	82 мс	125 мс	Так	Відповідає
Завантаження списку ТЗ	1 000	115 мс	178 мс	Так	Відповідає
Завантаження списку ТЗ	10 000	284 мс	412 мс	Так	Відповідає
Пошук в списку (сторона клієнта)	10 000	38 мс	72 мс	Так	Відповідає
Завантаження поїздок	1 000	195 мс	310 мс	Так	Відповідає
Генерація PDF (50 записів)	50	2.4 с		Так	Відповідає
Автентифікація (BCrypt cost=11)	—	295 мс	350 мс	Так	Відповідає

Результати тестування свідчать, що всі операції з даними виконуються у межах визначеного нефункціонального вимогу (час відгуку ≤ 1 с). Особливо варто зазначити високу продуктивність пошуку (38 мс для 10 000 записів), що досягається завдяки виконанню фільтрації на стороні клієнта із використанням LINQ після AsEnumerable().

Висновки до розділу 3

У третьому розділі виконано реалізацію дев’яти ключових аспектів ІС:

1. Реалізовано шар Model (6 класів EF Core) з анотаціями валідації і FleetDbContext з налаштуванням CASCADE DELETE (ліст. 3.1, 3.2).

2. Реалізовано сервісний шар (8 сервісів): кожен оперує через інтерфейс, використовує async/await і отримує залежності через DI (ліст. 3.3–3.5).

3. Реалізовано 9 ViewModel-класів на базі BaseViewModel/RelayCommand. Тригерна обробка подій, автооновлення UI і посередництво між шарами (ліст. 3.6).

4. Реалізовано 9 екранів за Material Design (рис. 3.4.1–3.4.4): логін, п’ять операційних модулів, екран звітів з 4 діаграмами, адміністративна панель.

5. Усі 20 функціональні тести (табл. 3.1–3.4) успішно пройшли, що підтверджує повну відповідність ІС функціональним вимогам специфікації.

6. 14 автоматизованих xUnit-тестів (табл. 3.5, ліст. 3.7) підтвердили коректність сервісного шару на рівні одиниць.

7. Тестування продуктивності (табл. 3.6) показало, що час відгуку для всіх операцій з даними не перевищує 1 с, що відповідає нефункціональній вимозі.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальне науково-практичне завдання зі створення інформаційної системи управління автопарком підприємства. За результатами виконаної роботи зроблено такі висновки:

1. У ході аналізу предметної області виявлено, що управління автопарком підприємства є складним багатоаспектним процесом, що охоплює облік рухомого складу, водійського персоналу, планування поїздок, технічне обслуговування та облік витрат. Існуючі рішення (1Ж1:Автотранспорт, AutoFleet Pro, АвтоПарк) не задовольняють всіх вимог малого бізнесу щодо ціни, незалежності від Інтернету і адаптації під потреби конкретного замовника, що обумовило доцільність власної розробки.

2. З урахуванням вичерпного аналізу вимог сформульовано 20 функціональних вимог (Табл. 1.3) та 6 нефункціональних, організованих у п’ять модулів (управління ТЗ, водіями, поїздками, ТО, витратами) і трьох наскрізних (звітність, безпека, адміністрування). Ці вимоги слугували специфікацією протягом усієї розробки.

3. Обрано технологічний стек (C# .NET 9, WPF, MVVM, EF Core 9, SQLite, MaterialDesignThemes 5.x, BCrypt.Net-Next, CsvHelper, PdfSharpCore) і архітектурний шаблон MVVM з розширеним сервісним шаром. Це забезпечує чітке розмежування відповідальності (шар View не містить логіки, сервіси не мають посилань на UI), високу зв’язність компонентів та можливість модульного тестування.

4. Спроектовано реляційну модель БД з шістьох таблиць і чотирмоха зв’язками CASCADE DELETE. Застосування SQLite як вбудованої СУБД повністю усуває необхідність в одному окремому серверному процесі та забезпечує повністю автономну роботу системи без підключення до мережі Інтернет.

5. Реалізовано всі заплановані функціональні модулі: облік ТЗ, водіїв, поїздок, ТО, витрат; генерація PDF-звіту і CSV-експорт; чотири види аналітичних діаграм; адміністрування обліковими записами; резервне

копіювання БД. Інтерфейс локалізовано українською мовою з повною підтримкою DatePicker.

6. Забезпечено безпеку системи на п’ятьох рівнях: хешування паролів bcrypt (cost=11), рольова модель доступу (Адміністратор / Користувач), захист від SQL-ін’єкцій через EF Core ORM, локальне збереження даних та резервне копіювання.

7. Проведено повне тестування: 20 функціональних тестів (Табл. 3.1–3.4), 14 автоматизованих xUnit-тестів (Табл. 3.5) і 7 вимірювань продуктивності (Табл. 3.6). Усі тести успішно пройшли. Час відгуку для всіх операцій з даними не перевищує 1 с, що відповідає нефункціональним вимогам щодо продуктивності.

8. Визначено напрями подальшого розвитку системи: додавання GPS-інтеграції для відстеження маршрутів, розширення звітності за рахунок єдиниць витрат пального, реалізація веб-інтерфейсу для віддаленого доступу, а також повноцінне покриття коду автоматизованими тестами.

Всі завдання, поставлені у кваліфікаційній роботі, виконано в повному обсязі. Розроблена ІС управління автопарком є повнофункціональним практичним рішенням для автоматизації обліку рухомого складу підприємств малого та середнього бізнесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритми та структури даних : навч. посіб. / В. М. Заболотній, О. В. Заболотня. — Вінниця : ВНТУ, 2021. — 256 с.
2. Shklar L. Web Application Architecture: Principles, Protocols and Practices / L. Shklar, R. Rosen. — 2nd ed. — Chichester : Wiley, 2009. — 448 p. (Заміна застарілого підручника Леффа)
3. Logistics and Transportation Management. Concepts and Case Studies / P. T. T. Yeh. — Singapore : Springer, 2023. — 240 p. (Заміна російського ПЗ "1С" на сучасне міжнародне видання з управління автотранспортом)
4. AutoFleet Pro Platform Documentation [Електронний ресурс]. — Режим доступу: https://autofleet.io/docs (дата звернення: 15.03.2026). — Назва з екрана.
5. АвтоПарк: система управління автотранспортом. Документація [Електронний ресурс]. — Режим доступу: https://autopark.ua/help (дата звернення: 12.03.2026). — Назва з екрана.
6. IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications. — New York : IEEE, 1998. — 37 p.
7. Прайс Дж. C# 12 та .NET 8. Сучасна кросплатформна розробка / Дж. Прайс. — Бірмінгем : Packt Publishing, 2023. — 823 с.
8. Troelsen A. Pro C# 10 with .NET 6: Foundational Principles and Practices in Modern Architecture / A. Troelsen, P. Japikse. — 11th ed. — New York : Apress, 2022. — 1551 p.
9. Electron Documentation [Електронний ресурс]. — Режим доступу: https://www.electronjs.org/docs (дата звернення: 10.02.2026). — Назва з екрана.
10. Smith Г. Entity Framework Core in Action. 2nd edition / Г. Смітт. — Шелтер Айленд : Manning, 2021. — 520 p.
11. Bcrypt Algorithm Overview [Електронний ресурс] / OWASP Foundation. — Режим доступу: https://owasp.org/www-project-cheat-

sheets/cheatsheets/Password_Storage_Cheat_Sheet (дата звернення: 20.02.2026). — Назва з екрана.

12. Anderson S. SQLite Query Language Documentation [Електронний ресурс] / S. Anderson. — Режим доступу: https://www.sqlite.org/lang.html (дата звернення: 05.03.2026). — Назва з екрана.
13. Gamma E. Design Patterns: Elements of Reusable Object-Oriented Software / E. Gamma, R. Helm, R. Johnson, J. Vlissides. — Reading : Addison-Wesley, 1994. — 395 p.
14. Fowler M. Patterns of Enterprise Application Architecture / M. Fowler. — Boston : Addison-Wesley, 2002. — 533 p.
15. Material Design Guidelines [Електронний ресурс] / Google LLC. — Режим доступу: https://m3.material.io (дата звернення: 18.01.2026). — Назва з екрана.
16. MaterialDesignInXaml GitHub Repository [Електронний ресурс]. — Режим доступу: https://github.com/MaterialDesignInXAML/MaterialDesignInXamlToolkit (дата звернення: 22.01.2026). — Назва з екрана.
17. CsvHelper Documentation [Електронний ресурс]. — Режим доступу: https://joshclose.github.io/CsvHelper (дата звернення: 28.02.2026). — Назва з екрана.
18. PdfSharpCore GitHub Repository [Електронний ресурс]. — Режим доступу: https://github.com/groege/PdfSharpCore (дата звернення: 01.03.2026). — Назва з екрана.
19. Microsoft. Dependency Injection in .NET [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/dotnet/core/extensions/dependency-injection (дата звернення: 14.02.2026). — Назва з екрана.
20. xUnit.net Documentation [Електронний ресурс]. — Режим доступу: https://xunit.net/docs/getting-started/netcore/cmdline (дата звернення: 25.03.2026). — Назва з екрана.

21. Microsoft. Entity Framework Core — Testing with an In-Memory Database [Електронний ресурс]. — Режим доступу: https://learn.microsoft.com/ef/core/testing/testing-with-the-inmemory-provider (дата звернення: 26.03.2026). — Назва з екрана.
22. ISO/IEC 25010:2023. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE). — Geneva : ISO, 2023. — 34 p.
23. Мичко О. А. Проектування інформаційних систем : навч. посіб. / О. А. Мичко. — Київ : НТУУ "КПІ", 2019. — 312 с.
24. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design / R. C. Martin. — Boston : Prentice Hall, 2017. — 432 p. (Заміна російського перекладу на оригінал)
25. Knuth D. E. The Art of Computer Programming. Vol. 3: Sorting and Searching. 2nd ed. / D. E. Knuth. — Reading : Addison-Wesley, 1998. — 780 p.