НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

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

"____" _____ 2025 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему

Програмне забезпечення автоматизованого робочого місця для управління складом станції технічного обслуговування

Спеціальність 121 – «Інженерія програмного забезпечення»

Гарант освітньої програми
К.т.н., доцент _____ Вайганг Г.О.

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент _____ Вайганг Г.О.
науковий ступінь та вчене звання) (підпис) (ПІБ)

Виконав _____ Біба Д.С.
(підпис) (ПІБ студента)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

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

"____" _____ 2025 р.

ЗАВДАННЯ
на виконання бакалаврської кваліфікаційної роботи студенту

Бібі Дмитру Сергійовичу

Спеціальність 121 «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи: Програмне забезпечення автоматизованого робочого місця для управління складом станції технічного обслуговування

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

Термін подання завершеної роботи на кафедру 2025.05.26
(рік, місяць, число)

Вихідні дані до роботи: Опис програмного забезпечення, основні параметри управління складом, дані щодо запасів та руху запчастин, нормативно-правові акти

Перелік питань що розглядаються:
1. Системний аналіз предметної області інформаційної системи
2. Проєктування інформаційного та програмного забезпечення
3. Розробка інформаційного та програмного забезпечення
4. Рекомендації щодо впровадження та експлуатації системи

Дата видачі завдання «_____»_____2025 р.
Керівник бакалаврської кваліфікаційної роботи

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

Завдання прийняв до виконання _____ Біба Д.С.
(підпис) (ПІБ студента)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

.NET – безкоштовна та з відкритим вихідним кодом платформа для розробки програмного забезпечення

ACID – це набір властивостей, що гарантують надійну роботу транзакцій бази даних

C# – об’єктно-орієнтована мова програмування створена Microsoft

DirectX – це набір компонентів у Windows, які дають змогу програмному забезпеченню працювати безпосередньо з відео- та аудіопристроями

HDD – Hard Disk Drive

MVVM – Model-View-ViewModel, один з MV патернів (архітектурний шаблон у WPF)

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

PDF – Portable Document Format (переносний формат документів)

SKU – Stock Keeping Unit (складський артикул)

SQL – це декларативна мова запитів для реляційних СУБД

SSD – Solid State Drive

UI – User Interface (інтерфейс користувача)

UML – уніфікована мова моделювання

WMS – система управління складом

WPF – Windows Presentation Foundation (технологія побудови UI в середовищі .NET)

XAML – декларативна мова розмітки

АРМ – автоматизоване робоче місце

БД – база даних

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

СТО – станція технічного обслуговування

СУБД – система управління базами даних

ВСТУП

У сучасних умовах розвитку автомобільної галузі та постійного зростання кількості транспортних засобів на дорогах України. Особливої актуальності набуває питання їхнього технічного обслуговування та ремонту. Станції технічного обслуговування (СТО) відіграють ключову роль у забезпеченні надійності та безпеки експлуатації автомобілів та пропонують широкий спектр послуг від діагностики до капітального ремонту різноманітних систем транспортних засобів.

Одним із найважливіших складових ефективної роботи СТО є належна організація складського господарства. Складське господарство забезпечує безперебійну роботу всіх підрозділів СТО шляхом своєчасного постачання необхідних запасних частин, матеріалів та інструментів. Саме від ефективності управління складськими процесами залежить швидкість виконання ремонту, якість обслуговування клієнтів та, як наслідок, рентабельність підприємства.

Актуальність теми зумовлена тим, що в умовах сучасного ринку автосервісних послуг критичного значення набувають швидкість та якість обслуговування клієнтів, які безпосередньо залежать від наявності необхідних запасних частин, витратних матеріалів та інструментів на складі. Ефективне управління складом дозволяє скоротити час очікування клієнтів, уникнути надлишкових запасів і, одночасно, не допустити дефіциту необхідних комплектуючих, що суттєво впливає на репутацію СТО та лояльність клієнтів.

Традиційні методи ведення складського обліку на підприємствах автосервісу із використанням паперової документації або базових електронних таблиць уже не відповідають вимогам сучасного бізнесу. Вони характеризуються низькою оперативністю та схильністю до помилок. У цьому контексті розробка та впровадження спеціалізованого програмного забезпечення для автоматизації робочого місця завідувача складу СТО є надзвичайно актуальним завданням.

Мета розробки – створення програмного забезпечення автоматизованого робочого місця (АРМ) для управління складом станції технічного

обслуговування, яке забезпечить підвищення ефективності обліку та контролю складських операцій, оптимізацію запасів та покращення якості обслуговування клієнтів СТО.

Створюване програмне забезпечення має вирішувати комплекс завдань, пов'язаних з автоматизацією основних складських процесів, включаючи облік надходження та видачі товарно-матеріальних цінностей, контроль залишків, та формування звітності. Особлива увага приділяється розробці зручного інтерфейсу користувача та забезпеченню надійності зберігання даних, що є критично важливим для безперебійної роботи СТО.

Структура роботи. Структурні елементи роботи розміщені в такій послідовності.

У першому розділі розглядається предметна область, проведено огляд існуючих рішень та сформульована постановка задачі дослідження.

Другий розділ присвячено моделюванню предметної області. Побудовані діаграми прецедентів використання, послідовності, активності, класів.

Третій розділ присвячено проектуванню програмної системи. Були розроблені логічна модель даних, фізична модель даних, архітектура системи, вибір та обґрунтування інструментарію.

В четвертому розділі розглянуто впровадження та експлуатація системи, описані вимоги та проведено перевірку якості програмного продукту.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ІНФОРМАЦІЙНОЇ СИСТЕМИ

1.1 Опис предметної області

Станція технічного обслуговування автомобілів (СТО) представляє собою підприємство, що надає комплекс послуг з технічного обслуговування та ремонту транспортних засобів. Важливою складовою частиною будь-якої СТО є її складське господарство, яке відіграє ключову роль у забезпеченні безперебійної роботи всіх підрозділів станції.

Складське господарство СТО забезпечує зберігання, облік та своєчасне постачання запасних частин, матеріалів, інструментів та обладнання, необхідних для виконання ремонтних робіт. Ефективна організація складу дозволяє мінімізувати час простою автомобілів на обслуговуванні та підвищити якість послуг, що надаються клієнтам.

Типовий склад СТО має зонування з урахуванням особливостей зберігання різних груп товарів. Так, окремо розміщуються зони прийому товарів, зберігання запасних частин, витратних матеріалів, шин та дисків, інструментів та обладнання, а також зона видачі товарів. Таке зонування дозволяє оптимізувати логістичні процеси та забезпечити належні умови зберігання для кожної категорії товарів.

На складі СТО зазвичай зберігаються запасні частини для різних систем автомобіля: деталі двигуна, трансмісії, підвіски тощо. Окрему групу складають витратні матеріали, такі як мастильні матеріали, фільтри та автохімія. Також на складі можуть зберігатися автомобільні аксесуари, шини, диски, а також інструменти та обладнання, необхідні для проведення ремонтних робіт.

Управління складом СТО включає кілька взаємопов'язаних процесів. Закупівля та прийом товарів починається з аналізу потреб у запасних частинах та матеріалах, вибору постачальників, оформлення замовлень, прийому товарів

та перевірки їх якості. Процес зберігання передбачає правильне розміщення товарів з урахуванням умов зберігання та забезпечення їх схоронності.

Облік товарів є одним з найважливіших процесів, який включає ведення документації, проведення інвентаризацій, контроль залишків та формування звітності. Завершальним етапом є видача товарів, яка передбачає обробку заявок від майстрів СТО, комплектацію замовлень та оформлення відповідної документації.

Сучасні склади СТО активно використовують спеціалізовані інформаційні системи, які дозволяють автоматизувати основні процеси управління запасами. Такі системи забезпечують точний облік товарів, оптимізують процеси закупівлі та розміщення товарів, контролюють залишки та формують аналітичну звітність. Інтеграція складської системи з загальною системою обліку СТО дозволяє підвищити ефективність роботи підприємства в цілому.

1.2 Аналіз вимог до програмної системи

Функціональні вимоги до програмного забезпечення автоматизованого робочого місця для управління складом станції технічного обслуговування охоплюють низку ключових аспектів, які забезпечують ефективне функціонування системи. Перш за все, система повинна забезпечити точне ведення каталогу запчастин, щоб користувачі могли легко здійснювати пошук та навігацію. Крім того, система повинна здійснювати облік поточних залишків на складі та надходження нових партій для ефективного управління запасами.

Додатково, функціональні вимоги включають можливість прийому електронних запитів від автомеханіків та реєстрації видачі запчастин. Це сприяє покращенню координації роботи автосервісу та забезпечує прозорість процесів. Крім того, система повинна забезпечити розмежування прав доступу для різних категорій користувачів, що підвищує безпеку даних.

З погляду компонентів системи, кожен з них має відповідати функціональним вимогам. Наприклад, компонент ведення каталогу повинен забезпечувати зручний пошук та навігацію по запчастинах, а компонент обліку

надходжень - здійснювати реєстрацію нових партій. Кожен компонент має бути розроблений таким чином, щоб відповідати вимогам функціональності та забезпечувати високу ефективність та надійність системи в цілому.

В табл. 1.1 наведено специфікацію функціональних вимог до системи складу СТО.

Таблиця 1.1

Специфікація функціональних вимог

№	Вимоги	Призначення та використання
1	Ведення каталогу запчастин	Система повинна забезпечувати створення, збереження та навігацію по каталогу запчастин із можливістю пошуку за назвою, артикулом, категорією та виробником.
2	Відображення поточних залишків на складі	Система повинна надавати актуальну інформацію про кількість запчастин у наявності, оновлюючи дані в режимі реального часу для оперативного реагування на запити.
3	Облік надходження партій запчастин	Система повинна дозволяти реєструвати надходження нових партій запчастин із можливістю додавання через інтерфейс.
4	Прийом електронних запитів від автомеханіків	Система повинна приймати заявки від автомеханіків через інтерфейс, автоматизувати перевірку наявності та генерувати повідомлення про статус виконання запиту.
5	Реєстрація видачі запчастин співробітнику	Система повинна фіксувати факт передачі запчастини: кому, коли і для яких робіт видано, формувати видаткові накладні та вести історію видач.
6	Розмежування прав доступу	Система повинна підтримувати ролі користувачів (оператор, менеджер, адміністратор).
7	Редагування та доповнення каталогу запчастин	Система повинна дозволяти уповноваженим користувачам додавати нові позиції, редагувати характеристики існуючих запчастин та підтримувати історію змін.
8	Облік переміщення та використання запчастин	Система повинна автоматично фіксувати переміщення запчастин між зонами зберігання, а також їхнє використання, із вказанням дати, часу та відповідальних осіб.

Нефункціональні вимоги до системи обліку запчастин на автосервісі (табл. 1.2) визначають параметри та властивості системи, які не прямо пов'язані з її функціональністю, але визначають її якість, надійність, безпеку та інші аспекти, що впливають на задоволення потреб користувачів.

Таблиця 1.2

Специфікація нефункціональних вимог

№	Категорія вимог	Вимоги	Призначення та використання
1	Безпека	Автентифікація	Система повинна підтримувати автентифікацію для різних користувачів.
2		Шифрування паролів	Система повинна шифрувати паролі користувачів перед збереженням у базі даних, використовуючи сучасні алгоритми.
3	Інтерфейс	Простий інтерфейс	Інтерфейс повинен бути простим та зрозумілим для користувачів, що дозволяє швидко виконувати необхідні дії без складних налаштувань.
4	Сумісність	Операційні системи	Програмне забезпечення повинно бути сумісне з операційними системами Windows 10 і вище.
5	Продуктивність	Оптимізація запитів до БД	Запити до бази даних повинні бути оптимізовані для забезпечення швидкої обробки даних, без затримок при великих обсягах інформації.
6		Робота з великими обсягами даних	Система повинна ефективно працювати з великими обсягами даних, забезпечуючи швидке завантаження, пошук та обробку даних без значних затримок.

Однією з ключових нефункціональних вимог є безпека. Система повинна гарантувати конфіденційність, цілісність та доступність даних про запчастини та операції з ними, а також забезпечити захист від несанкціонованого доступу до цієї інформації. Безпека також включає надійну автентифікацію користувачів та шифрування чутливих даних.

Ще однією важливою нефункціональною вимогою є простота інтерфейсу. Система повинна мати зрозумілий та інтуїтивний інтерфейс, який дозволяє користувачам швидко виконувати необхідні дії без складних налаштувань чи додаткового навчання.

Також важливою є продуктивність системи. Вона повинна забезпечувати швидку реакцію на запити користувачів та оптимальний час відповіді на них. Це включає як оптимізацію запитів до бази даних, так і ефективну роботу з великими обсягами даних без значних затримок.

Крім того, система повинна бути сумісною з операційними системами Windows 10 і вище, що забезпечує її широке впровадження та використання на існуючому обладнанні автосервісу.

Ці функціональні та нефункціональні вимоги визначають основні можливості та характеристики системи обліку запчастин на автосервісі. Отже, системні вимоги є важливими для забезпечення стабільної та ефективної роботи системи та впевненості в її відповідності стандартам безпеки, доступності та ефективності.

1.3 Моделювання предметної області

Моделювання предметної області для розробки програмного модуля забезпечення є ключовим етапом у процесі розробки програмного забезпечення. Цей етап включає ідентифікацію основних сутностей та встановлення взаємозв'язків між ними, що служить базою для подальшої розробки системи. Ми розглянемо ключові сутності та їх взаємодії у нашій предметній області, що дозволить краще зрозуміти основні об'єкти та процеси, які відбуваються в системі.

Мова UML є універсальною мовою візуального моделювання, призначеною для специфікації, візуалізації, проектування та документування компонентів програмного забезпечення, бізнес-процесів та інших систем. Вона є простим і потужним інструментом моделювання, який ефективно використовується для створення концептуальних, логічних та графічних

моделей складних систем різного призначення. Використання UML дозволяє розробникам чітко представляти структуру і поведінку системи, що полегшує розуміння і комунікацію між членами команди, а також забезпечує єдину основу для розробки і підтримки програмного забезпечення.

Для уточнення структури та логіки роботи складського підрозділу СТО була побудована діаграма прецедентів (рис.1.1), яка відображає основних учасників процесу та їхню взаємодію з системою. У центрі цієї взаємодії перебуває інформаційна система управління складом, яка підтримує кілька ключових сценаріїв для різних категорій користувачів: оператора складу, менеджера складу, автомеханіка та постачальника запчастин.

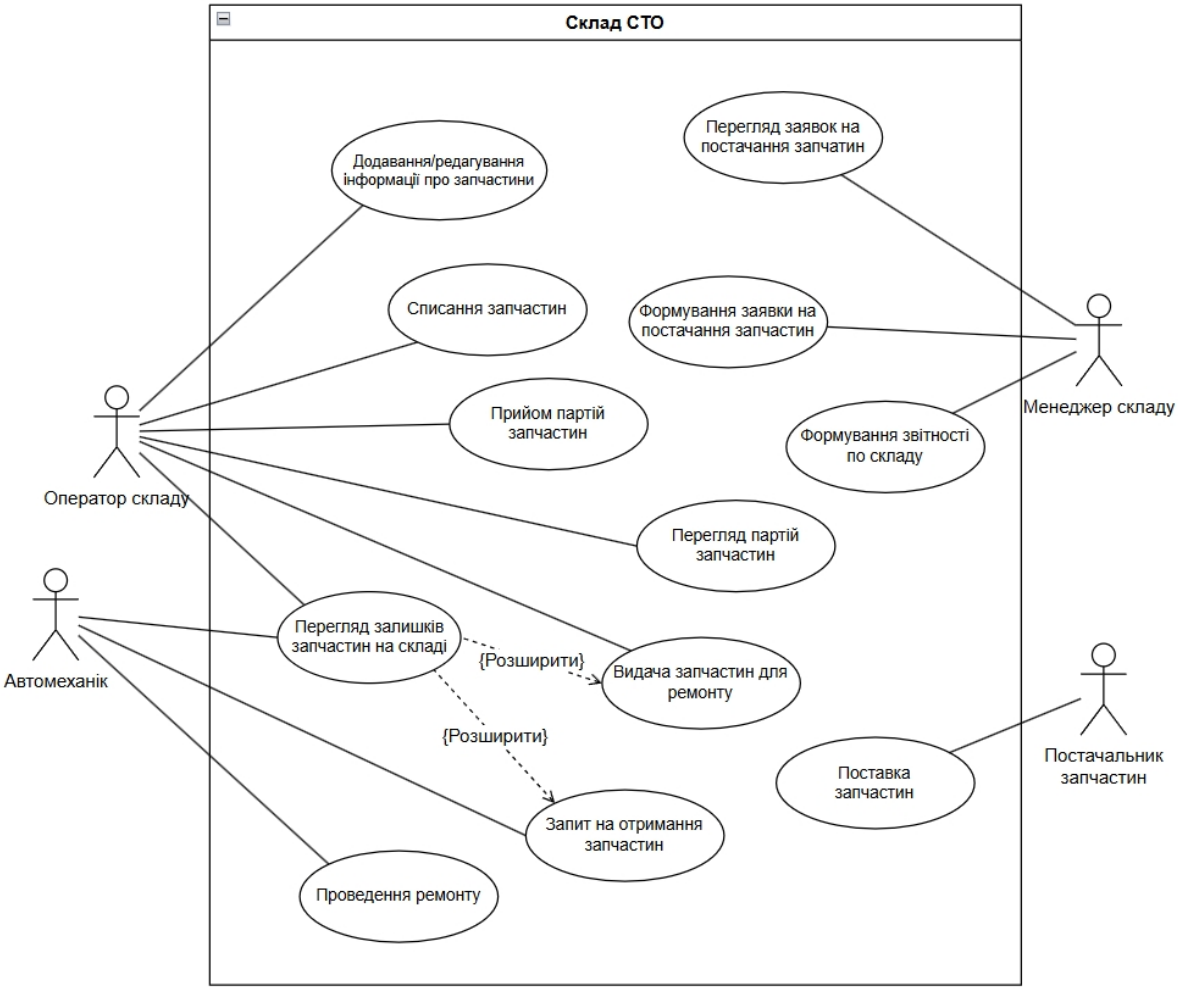

Рис.1.1 Діаграма прецедентів складу СТО

Оператор складу виконує основний обсяг щоденних операцій. До його функціоналу належать:

- додавання та редагування інформації про запчастини, що дозволяє оновлювати довідник номенклатури відповідно до надходжень чи змін асортименту;
- списання запчастин, що списуються з обліку у разі пошкодження, закінчення терміну придатності або внутрішнього використання;
- прийом партій запчастин, що надходять від постачальників - із фіксацією кількості, якості, дати надходження та супровідних документів;
- перегляд партій запчастин, які вже знаходяться на складі, з можливістю аналізу їхнього походження та залишкового ресурсу;
- видача запчастин для ремонту, яка здійснюється на підставі запиту від автомеханіка та реєструється для забезпечення обліку.

Автомеханік, як безпосередній користувач матеріальних ресурсів, може:

- переглядати залишки запчастин на складі для перевірки наявності необхідних позицій перед виконанням ремонту;
- формувати запит на отримання запчастин, який передається оператору складу;
- проводити ремонт на основі виданих запчастин – цей прецедент є кінцевою метою всієї складської логістики.

Менеджер складу виконує функції, пов’язані з вищим рівнем управління:

- здійснює перегляд заявок на постачання запчастин, що формуються операторами складу або автоматично системою;
- формує нові заявки на постачання у разі виявлення нестачі ресурсів;
- складає звітність по складу.

Постачальник запчастин представлений у вигляді зовнішнього учасника, який взаємодіє з системою опосередковано. Його роль полягає в поставці запчастин за узгодженими заявками, після чого товари проходять приймання на складі.

Таким чином, діаграма прецедентів демонструє взаємозв’язки між підрозділами СТО та зовнішніми партнерами, а також відображає розподіл

відповідальності між працівниками. Така структуризація процесів дозволяє сформувати вимоги до інформаційної системи, яка буде автоматизувати управління запасами, облік операцій та взаємодію персоналу в рамках єдиного середовища.

Для уточнення структури та логіки роботи складу СТО в часі була побудована діаграма послідовності (рис.1.2), яка візуалізує взаємодію між основними учасниками процесу видачі запчастин на СТО.

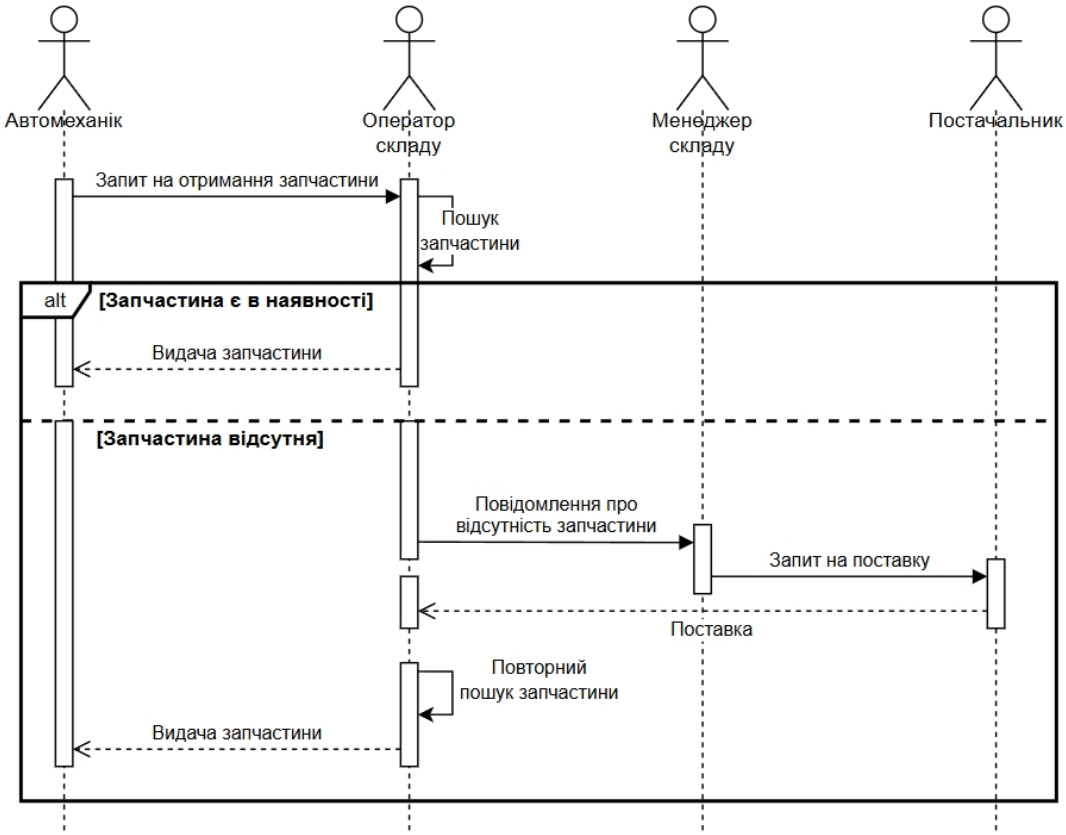

Рис.1.2 Діаграма послідовності складу СТО

Діаграма демонструє два можливі сценарії розвитку подій. У випадку, коли запчастина є в наявності, оператор складу видає запчастину автомеханіку, і процес завершується успішно. Якщо ж запчастина відсутня, вмикається альтернативний сценарій, при якому оператор складу повідомляє менеджера складу про відсутність необхідної запчастини. Менеджер складу формує запит на поставку до постачальника, який виконує поставку запчастини. Після

надходження запчастини оператор повторно здійснює пошук, і запчастина видається автомеханіку.

Така діаграма послідовності наочно демонструє логіку роботи складу СТО, послідовність операцій та взаємодію між різними учасниками процесу. Вона дозволяє зрозуміти не лише штатний порядок роботи, але й альтернативні сценарії, що виникають при відсутності необхідних запчастин. Це допомагає оптимізувати логістичні процеси, скоротити час очікування клієнтів та підвищити ефективність роботи СТО в цілому.

Для візуалізації бізнес-процесу видачі запчастин на СТО була побудована діаграма активності (рис.1.3), яка відображає послідовність дій та взаємодію між основними учасниками процесу – автомеханіком, оператором складу, менеджером складу та постачальником.

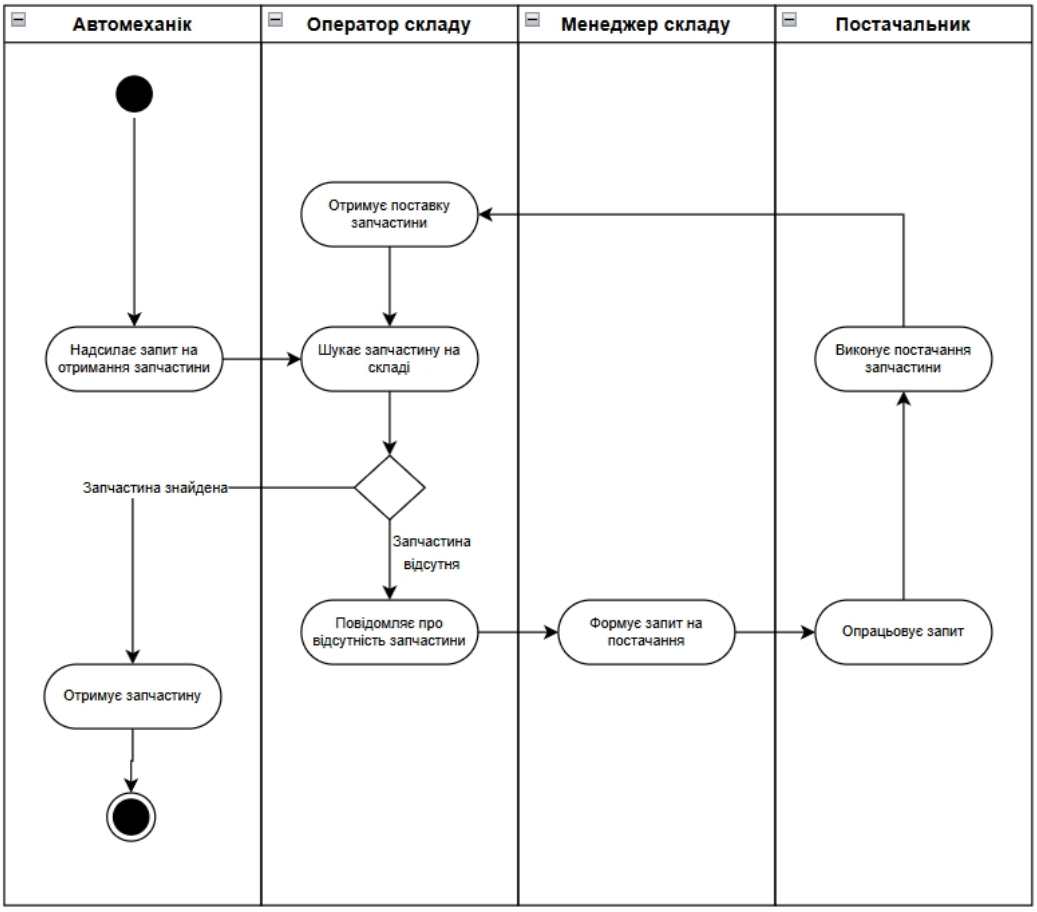

Рис.1.3 Діаграма активності складу СТО

Процес починається з того, що автомеханік надсилає запит на отримання запчастини. Оператор складу, отримавши запит, перевіряє наявність потрібної запчастини на складі. У випадку, якщо запчастина є в наявності, вона одразу видається автомеханіку, і процес завершується.

Якщо ж запчастина відсутня, оператор повідомляє про це менеджера складу. Менеджер, у свою чергу, формує запит на постачання запчастини та надсилає його постачальнику. Після обробки запиту постачальник виконує постачання запчастини на склад. Оператор складу приймає поставку, після чого здійснює повторну видачу запчастини автомеханіку.

Дана діаграма активності дозволяє побачити як основний сценарій обробки запиту, так і альтернативний, що передбачає додаткові дії у разі відсутності запчастин. Така модель є необхідною для аналізу та оптимізації внутрішніх логістичних процесів, зменшення часу простою та забезпечення безперебійної роботи СТО.

1.4 Огляд інформаційних джерел та існуючих рішень

Перед тим як розпочати розробку будь-якого програмного продукту, важливо здійснити аналіз існуючих джерел інформації та аналогічних рішень у відповідній галузі. Цей етап має вирішальне значення для забезпечення успішного розвитку проекту та досягнення його мети. Відомості, отримані під час огляду інформаційних джерел та аналізу існуючих рішень, можуть стати основою для розробки нового продукту або для вдосконалення існуючих систем. Проведемо аналіз інформаційних джерел та існуючих рішень у контексті управління складом.

Огляд цих джерел та рішень дозволяє отримати уявлення про наявні технології, методи та підходи до розробки програмного продукту для управління складом СТО.

Tocan WMS є повнофункціональною системою управління складом, розробленою для автоматизації всіх основних процесів складів різних типів і

розмірів. Система підтримує як хмарну (SaaS), так і стаціонарну версії, що дозволяє адаптуватися до потреб малого, середнього та великого бізнесу.

Tocan WMS забезпечує автоматизацію повного циклу складських операцій: приймання, розміщення, відвантаження. Гнучке налаштування топології складу із застосуванням 2D-3D конструктора дозволяє оптимізувати розміщення товарів. Система підтримує адресне зберігання (статичне та динамічне), управління товарами за термінами придатності, партіями та серійними номерами, а також інтеграцію з ERP-системами, такими як 1С, SAP, MS Dynamics. Інтерфейс системи адаптований для роботи на різних пристроях, включаючи термінали збору даних (ТЗД), і забезпечує інтуїтивно зрозумілий доступ до функціоналу для комірників та диспетчерів складу.

Крім основних функцій, Tocan WMS включає потужний модуль аналітики, який надає детальну статистику ефективності складських операцій, продуктивності персоналу та оборотності товарів. Система дозволяє автоматично розраховувати KPI для складського персоналу та оптимізувати маршрути переміщення товарів за допомогою алгоритмів машинного навчання. Важливою перевагою Tocan WMS є також інтегрована система управління якістю, яка дозволяє контролювати відповідність товарів стандартам на всіх етапах складського процесу, включаючи можливість організації карантинних зон для товарів, що потребують додаткової перевірки.

Незважаючи на широкий функціонал, Tocan WMS має певні недоліки. Система відрізняється високою вартістю впровадження та обслуговування, що може бути проблематичним для малих підприємств. Крім того, складність налаштування 3D-конструктора вимагає спеціальних навичок та тривалого навчання персоналу, а інтеграція з деякими ERP-системами, особливо нестандартними або застарілими, часто викликає технічні складнощі та потребує додаткових доопрацювань. Користувачі також відзначають обмежену гнучкість у налаштуванні звітів та недостатню оптимізацію системи для роботи на мобільних пристроях з низькою продуктивністю, що може створювати затримки при високому навантаженні на склад.

LSC WMS є системою управління складом, призначеною для комплексної автоматизації складської логістики в різних галузях, включаючи виробництво, дистрибуцію, роздрібну торгівлю та e-commerce.

Система підтримує різні технології відбору, управління ресурсами, інвентаризацію, аналітику та гнучку систему звітності. Інтерфейс LSC WMS включає монітори для управління складом, відвантаженнями та документами, що дозволяє в режимі реального часу контролювати всі процеси на складі. Система також підтримує роботу з терміналами збору даних, що забезпечує оперативність та точність виконання складських операцій.

LSC WMS вирізняється розширеними можливостями для автоматизації складських комплексів з високим рівнем механізації. Система інтегрується з конвеєрними лініями, автоматизованими складськими системами (AS/RS), сортувальними установками та роботизованими комплексами, забезпечуючи високу ефективність обробки великих обсягів товарів. Особливістю LSC WMS є також інноваційна технологія голосового управління (voice-picking), яка дозволяє співробітникам складу отримувати голосові інструкції та підтверджувати виконання операцій без використання ручних пристроїв, що значно підвищує швидкість роботи та зменшує кількість помилок.

Вбудована система прогнозування та планування ресурсів LSC WMS автоматично розраховує оптимальну кількість персоналу та техніки для виконання поточних завдань, базуючись на аналізі історичних даних та прогнозі навантаження. Це дозволяє ефективно розподіляти робоче навантаження та уникати пікових перевантажень складської інфраструктури, що особливо важливо для підприємств з сезонними коливаннями попиту та компаній електронної комерції.

До недоліків LSC WMS можна віднести надмірну складність системи для невеликих складів, що призводить до використання лише частини функціоналу при високій вартості ліцензії. Користувачі відзначають значні витрати на початкове налаштування та інтеграцію з існуючими системами підприємства, а також тривалий період адаптації персоналу до роботи з голосовим управлінням.

Система також вимагає потужної серверної інфраструктури для стабільної роботи, особливо при використанні всіх аналітичних модулів, що додатково збільшує сукупну вартість володіння. Технічна підтримка LSC WMS часто критикується за повільне реагування на запити та недостатню гнучкість у вирішенні специфічних проблем клієнтів.

1.5 Постановка завдання

Програмне забезпечення автоматизованого робочого місця (АРМ) призначене для забезпечення ефективного управління складським господарством станції технічного обслуговування (СТО). СТО є підприємством, яке виконує технічне обслуговування та ремонт транспортних засобів, і його ефективна діяльність значною мірою залежить від чіткої організації процесів постачання та обліку запасних частин, витратних матеріалів, інструментів та обладнання. Основне призначення складського господарства СТО полягає у забезпеченні безперебійного забезпечення ремонтних зон усіма необхідними матеріалами, що зменшує час простою автомобілів і підвищує якість наданих послуг.

Розроблюване програмне забезпечення є десктопним застосунком, який функціонує на операційній системі Windows. Воно має підтримувати повний цикл складських операцій, включаючи приймання товарів від постачальників, зберігання із дотриманням зонування, облік, інвентаризацію, формування звітів, а також видачу запчастин автомеханікам на підставі їхніх запитів.

Для забезпечення збереження даних і захисту від втрати інформації реалізовано механізм автоматичного резервного копіювання бази даних. Копії створюються регулярно за визначеним розкладом і зберігаються в безпечному каталозі на локальному або зовнішньому носії. У разі збою або пошкодження основної бази даних адміністратор може швидко відновити її з останньої резервної копії, мінімізуючи ризики простою системи.

Інтерфейс програми має бути розроблений з урахуванням принципів простоти та зручності користування. Основні функції згруповані за вкладками

відповідно до ролей користувачів, що дозволяє швидко орієнтуватися в системі навіть новим працівникам. Візуальне оформлення мінімалістичне, із чітко структурованими формами, таблицями та кнопками для виконання типових дій. Усі поля супроводжуються підказками, що знижує ймовірність помилок і сприяє швидкому навчанню персоналу.

У системі передбачено три основні ролі користувачів: оператор, менеджер і автомеханік. Кожна роль має свої повноваження та функціональну зону відповідальності.

Оператор виконує всі основні операції зі складським обліком. Він відповідає за введення нових партій запчастин, облік надходжень від постачальників, оформлення видачі запчастин на основі запитів, а також контроль залишків у режимі реального часу. Оператор має право редагувати довідник номенклатури, включаючи зміну назви, артикулу, категорії, виробника тощо, і веде облік переміщення товарів по зонах складу.

Менеджер має доступ до аналітичної та звітної інформації. Він може переглядати поточні залишки запчастин, аналізувати історію операцій, формувати звіти за визначеними параметрами, а також оцінювати ефективність складських процесів. Менеджер не редагує дані, але використовує систему для прийняття рішень на основі актуальної інформації зі складу.

Автомеханік є користувачем, який створює запити на видачу запчастин для виконання ремонтних робіт. Він має можливість переглядати наявні запчастини в системі – зокрема, їх назви, характеристики, доступну кількість – і на основі цього обирати необхідні деталі. Доступ автомеханіка обмежено лише переглядом та формуванням запитів без можливості змінювати будь-які дані.

2 ПРОЕКТУВАННЯ ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Логічна модель даних у вигляді ER-діаграми

Модель «сутність-зв'язок» (ER-модель) – модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. Існує ряд моделей для представлення знань, але одним з найзручніших інструментів уніфікованого представлення даних, незалежного від програмного забезпечення що його реалізує, є модель «сутність-зв'язок». Важливим є той факт, що з моделі «сутність-зв'язок» можуть бути породжені всі існуючі моделі даних (ієрархічна, мережева, реляційна, об'єктна), тому вона є найзагальнішою.

Модель сутність-зв'язок є результатом систематичного процесу, який описує та визначає деяку предметну область. Вона не визначає сам процес, а лише візуалізує його. Дані представлені у вигляді компонентів (сутностей), які пов'язані між собою певними зв'язками, які виражають залежності і вимоги між ними, такі як: одна будівля може бути розділена на нуль або більше квартир, але одна квартира може бути розташована лише в одній будівлі. Сутності можуть мати різні властивості (атрибути), які характеризують їх. Діаграми, створені для представлення цих сутностей, атрибутів і зв'язків графічно, називають сутність-зв'язок діаграмами.

ER-модель зазвичай реалізується в вигляді баз даних. У разі реляційної бази даних, в якій зберігаються дані в таблицях, кожен рядок кожної таблиці являє собою один екземпляр сутності. Деякі поля даних в цих таблицях вказують на індекси в інших таблицях. Такі поля є покажчиками фізичної реалізації зв'язків між сутностями.

Коли ми говоримо про сутність, ми зазвичай говоримо про деякий аспект реального світу, який можна виділити поміж інших аспектів. Сутність – це збірне поняття, деяка абстракція реально існуючого об'єкта, процесу, явища чи деякого уявлення про об'єкт. Хоча термін сутність найбільш вживаний, потрібно

розрізняти поняття типу сутності та екземпляру сутності. Поняття тип сутності відноситься до набору однорідних особистостей, предметів, подій або ідей, виступаючих як ціле. Екземпляр сутності відноситься до конкретної речі в наборі. Наприклад, типом сутності може бути МІСТО, а екземпляром – Київ, Львів і т. д.

Сутність – реальний або уявний об’єкт, що має істотне значення для аналізованої предметної області, інформація про який підлягає збереженню. Кожна сутність має унікальний ідентифікатор. Кожний екземпляр сутності однозначно ідентифікується і відрізняється від усіх інших екземплярів даного типу сутності.

Розглянемо ER-модель для системи управління складом СТО на рис. 2.1.

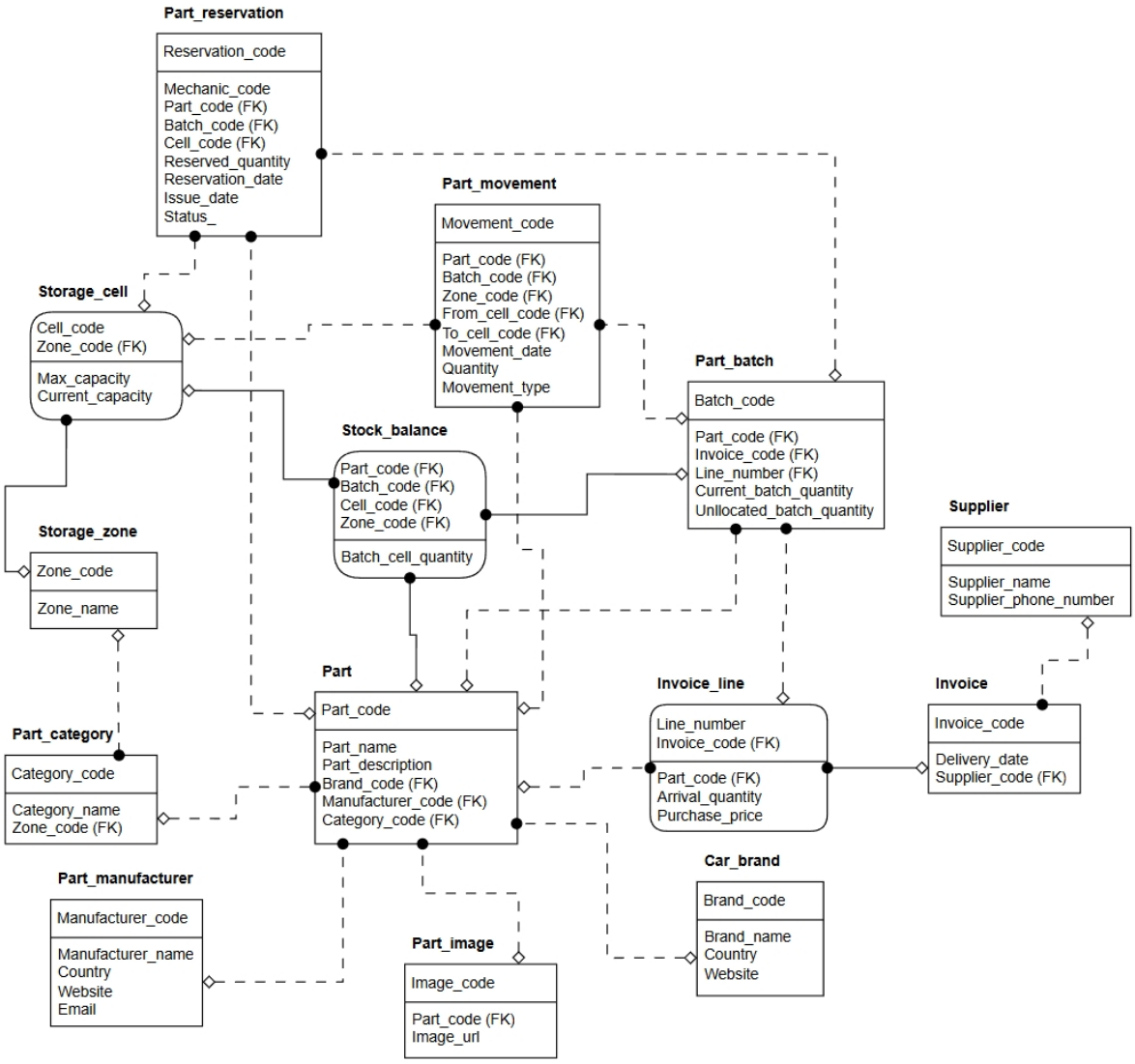

Рис.2.1 ER-діаграма додатку

В табл. 2.1 описано кожну сутніть ER-діаграми додатку.

Таблиця 2.1

Опис сутностей ER-діаграми

№	Сутність	Опис
1	Part	Основна сутність – деталь. Містить назву, опис, бренд, категорію та виробника
2	Part_batch	Партія деталей. Зв’язана з поставкою. Містить кількість деталей у партії і кількість ще не розподілених деталей
3	Part_reservation	Резервація деталі для механіка. Містить кількість, дату бронювання та статус
4	Part_movement	Переміщення деталей між комірками. Вказує звідки/куди, дату, тип руху та кількість
5	Stock_balance	Залишок деталей у комірці для певної партії
6	Storage_cell	Комірка зберігання з максимальною і поточною місткістю
7	Storage_zone	Зона складу, до якої належать комірки
8	Invoice	Документ поставки. Зв’язаний з постачальником та містить дату
9	Invoice_line	Рядок поставки: яка деталь, у якій кількості та за якою ціною була доставлена
10	Supplier	Постачальник деталей
11	Part_category	Категорія деталі (наприклад, двигуни, фільтри), може бути прив’язана до зони
12	Part_manufacturer	Виробник деталі. Має назву, країну, сайт, email
13	Car_brand	Автомобільний бренд, з яким сумісна деталь
14	Part_image	Зображення деталі, зберігається URL. Кожне зображення прив’язане до конкретної деталі

Таким чином, побудована ER-діаграма охоплює всі ключові сутності та взаємозв’язки, що формують основу інформаційної структури системи. Такий підхід дозволяє забезпечити цілісність і узгодженість даних у базі, а також створює надійну базу для розробки механізмів обліку, резервування та переміщення запчастин у межах складу СТО. Визначені сутності охоплюють як логістику зберігання, так і взаємодію з постачальниками, користувачами та супровідною документацією, що є критично важливим для ефективного функціонування всієї системи.

2.2 Діаграма класів та кооперації

Діаграма класів – це статична структурна діаграма, що демонструє властивості системи, класи, операції та зв’язки між об’єктами для опису структури системи.

Діаграма класів служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. На цій діаграмі показують класи, інтерфейси, об'єкти й кооперації, а також їхні відносини (зв'язки).

На діаграмі класи представлені у вигляді прямокутників, які містять три відділення:

- верхня частина містить назву класу. Вона надрукована напівжирним шрифтом і вирівняна по центру, починається з великої літери;
- середнє відділення містить атрибути класу. Вони вирівняні по лівому краю і перша літера мала;
- нижній відсік містить операції, які може виконувати клас. Вони також вирівняні по лівому краю і перша літера мала.

Не менш важливими є зв’язки між класами. Всього існує п’ять типів зв’язків: асоціація, узагальнення, залежність, агрегація і композиція.

Асоціація показує, що об'єкти однієї сутності (класу) пов'язані з об'єктами іншої сутності. Графічно асоціація зображується у вигляді лінії, що з'єднує клас сам з собою або з іншими класами.

Узагальнення (наслідування) вказує на ієрархічну структуру класів, де один клас (дочірній) успадковує атрибути та методи іншого класу (батьківський). Узагальнення означає, що об'єкти класу-нащадка можуть використовуватися скрізь, де зустрічаються об'єкти класу-батька, але не навпаки. Іншими словами, нащадок може бути підставлений замість батька. При цьому він успадковує властивості батьків, зокрема його атрибути і операції. Часто, хоча і не завжди, у нащадків є і свої власні атрибути і операції, крім тих, що існують у батька.

Залежність показує, як один клас використовує інший. Зміна в описі одного класу може вплинути на інший клас. Наприклад, клас "Замовлення" може залежати від класу "Клієнт", якщо він має посилання на клієнта, який оформив замовлення.

Агрегація є вкладенням одного класу в інший, але клас обгортки не контролює термін служби вкладеного об'єкта (який представлений посиланням на об'єкт, що використовується).

Композиція подібний до агрегації, але має більшу залежність. Об'єкти класу, що використовується в композиції, створюються разом з об'єктом іншого класу і повністю контролюються ним. Наприклад, клас "Автомобіль" може містити клас "Двигун", при цьому двигун існує тільки як частина автомобіля і не може існувати окремо.

Діаграма класів ідеально підходить для відображення майбутньої структури класів у розроблюваному програмному забезпеченні. На рисунку 2.2 наведена діаграма класів для декстопного додатку управління складом СТО.

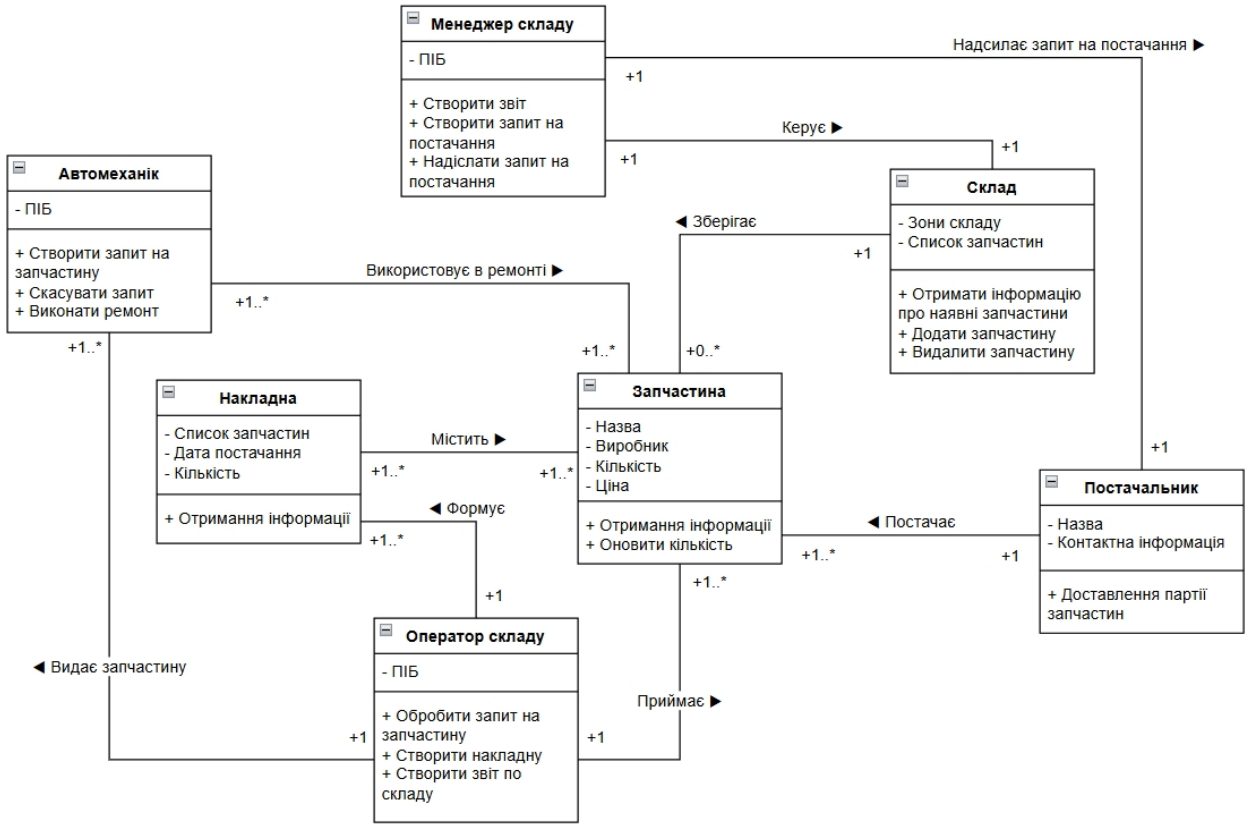

Рис.2.2 Діаграма класів додатку

Діаграма включає класи, які представляють ключові елементи системи, такі як: запчастина, накладна, менеджер складу, склад, автомеханік, оператор складу та постачальник. Автомеханік використовує запчастини у процесі ремонту, створюючи запит на їх видачу. Оператор складу обробляє ці запити та видає запчастини автомеханику. Окрім того, оператор формує накладну на отримані партії. Накладна містить перелік запчастин, дату та кількість постачання. Постачальник постачає запчастини на склад відповідно до запиту, який формує менеджер складу. Менеджер керує складом, а також створює звіти і запити на постачання. Склад зберігає запчастини та дозволяє додавати або вилучати їх. Запчастини асоційовані як з накладними, так і з постачальниками, складами й автомеханіками.

Опис структури класів наведений в табл. 2.2.

Таблиця 2.2

Опис структури класів додатку

№	Назва класу	Атрибути	Методи
1	Автомеханік	ПІБ	Створити запит на запчастину, Скасувати запит, Виконати ремонт
2	Менеджер складу	ПІБ	Створити звіт, Створити запит на постачання, Надіслати запит на постачання
3	Склад	Зони складу, Список запчастин	Отримати інформацію про наявні запчастини, Додати запчастину, Видалити запчастину
4	Запчастина	Назва, Виробник, Кількість, Ціна	Отримання інформації, Оновити кількість
5	Постачальник	Назва, Контактна інформація	Доставлення партії запчастин
6	Оператор складу	ПІБ	Обробити запит на запчастину, Створити накладну, Створити звіт по складу
7	Накладна	Список запчастин, Дата постачання, Кількість	Отримання інформації

Прості кооперації – взаємодія між кількома об'єктами з метою виконання певної функції або досягнення спільної цілі. Кооперації зручно відображати як діаграми класів за допомогою UML. Проста кооперація являється частиною діаграми класів, яка уточнюється для відображення взаємодії між групою класів. Далі будуть наведені приклади простих кооперацій, побудованих на основі діаграми класів. (див. рис. 2.2)

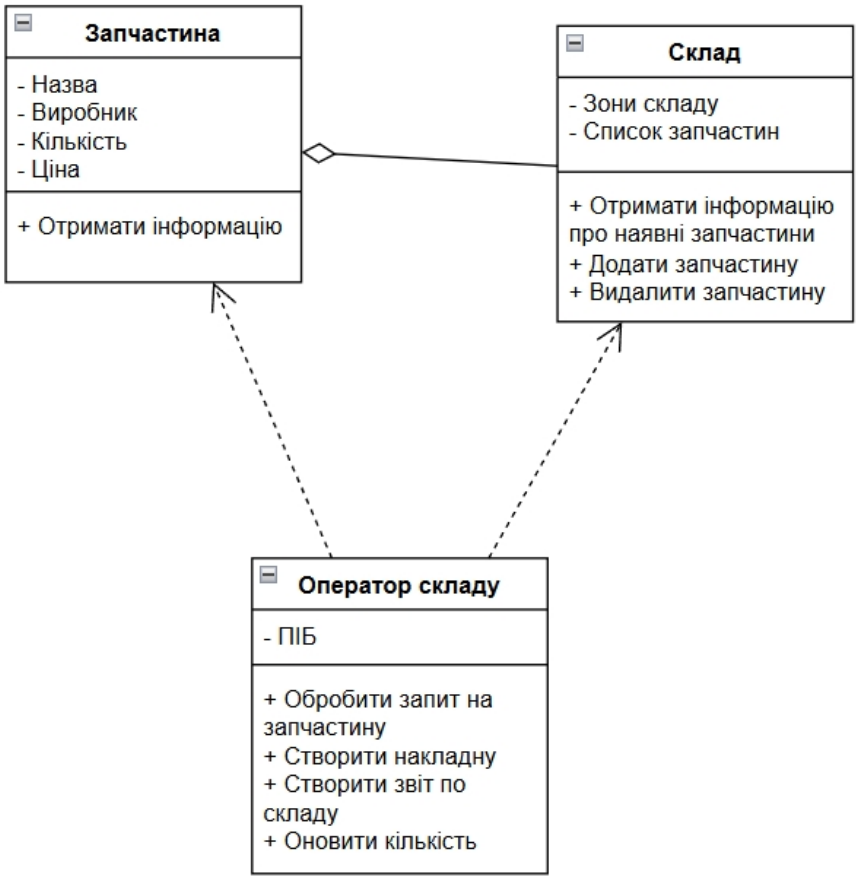

Рис.2.3 Управління запасами

В простій кооперації управління запасами (рис.2.3) ілюструється загальна логіка управління запасами. Основу системи становить склад, який містить запчастини – тобто всі запчастини пов’язані зі складом і саме через нього ведеться облік їхньої кількості, характеристик та інших даних.

Оператор складу виступає користувачем системи та взаємодіє зі складом. Він не працює з запчастинами напряму, а звертається до складу, щоб отримати інформацію про їхню наявність або вносити зміни. Наприклад, якщо потрібно

оновити кількість, сформувати звіт чи створити накладну – усе це відбувається через взаємодію з даними, що зберігаються на складі.

Таким чином, склад виконує роль централізованого сховища й посередника між інформацією про запчастини та оператором, а оператор – має доступ до цієї інформації й керує процесами на основі неї.

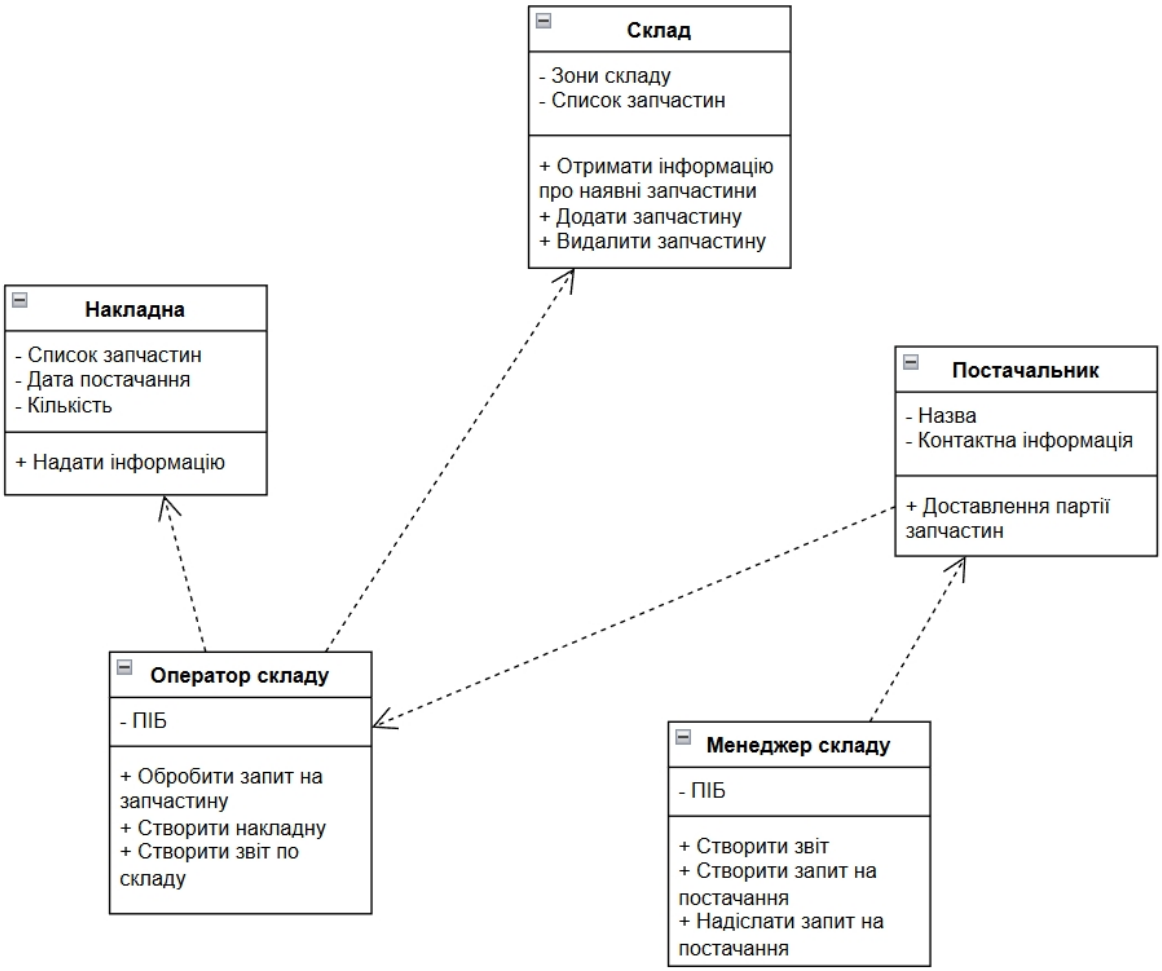

Рис.2.4 Поставка запчастин

В простій кооперації поставки запчастин (рис.2.4) показано, як організований процес постачання запчастин на склад. Менеджер складу ініціює постачання – він формує та надсилає запит до постачальника. Постачальник, отримавши запит, доставляє партію запчастин. Отримані запчастини обробляє оператор складу: він створює накладну, фіксує інформацію про доставку та додає запчастини на склад. Весь процес постачання закінчується на складі, який є місцем зберігання та обліку усіх запчастин.

Менеджер відповідає за ініціювання постачання, постачальник – за фізичну доставку, оператор складу – за оформлення та облік, а склад – за збереження запчастин.

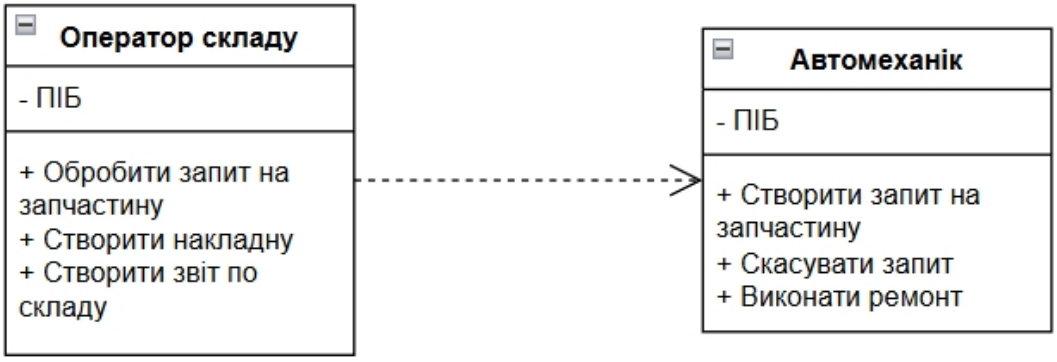

Рис.2.5 Видача запчастин

Проста кооперація видачі запчастин (рис.2.5) демонструє процес видачі запчастин зі складу. Автомеханік створює запит на отримання запчастин, які йому потрібні для виконання ремонту. Оператор складу отримує цей запит, обробляє його та формує накладну на видачу. Після цього запчастини списуються зі складу, а автомеханік може розпочати ремонт.

2.3 Діаграма пакетів

Діаграма пакетів – це різновид структурної діаграми UML, який дозволяє відображати архітектуру програмної системи на високому рівні абстракції. Основним елементом такої діаграми є пакет, що являє собою логічне об'єднання класів, підсистем або інших структурних одиниць. Такі діаграми використовується для візуалізації модульної організації системи, а також для показу залежностей між її елементами.

Діаграми пакетів корисні на етапі проектування програмного забезпечення, коли важливо зрозуміти загальну структуру системи, розділити її на підсистеми та встановити взаємодію між ними. Це допомагає створити гнучку архітектуру та спрощує супровід системи в майбутньому.

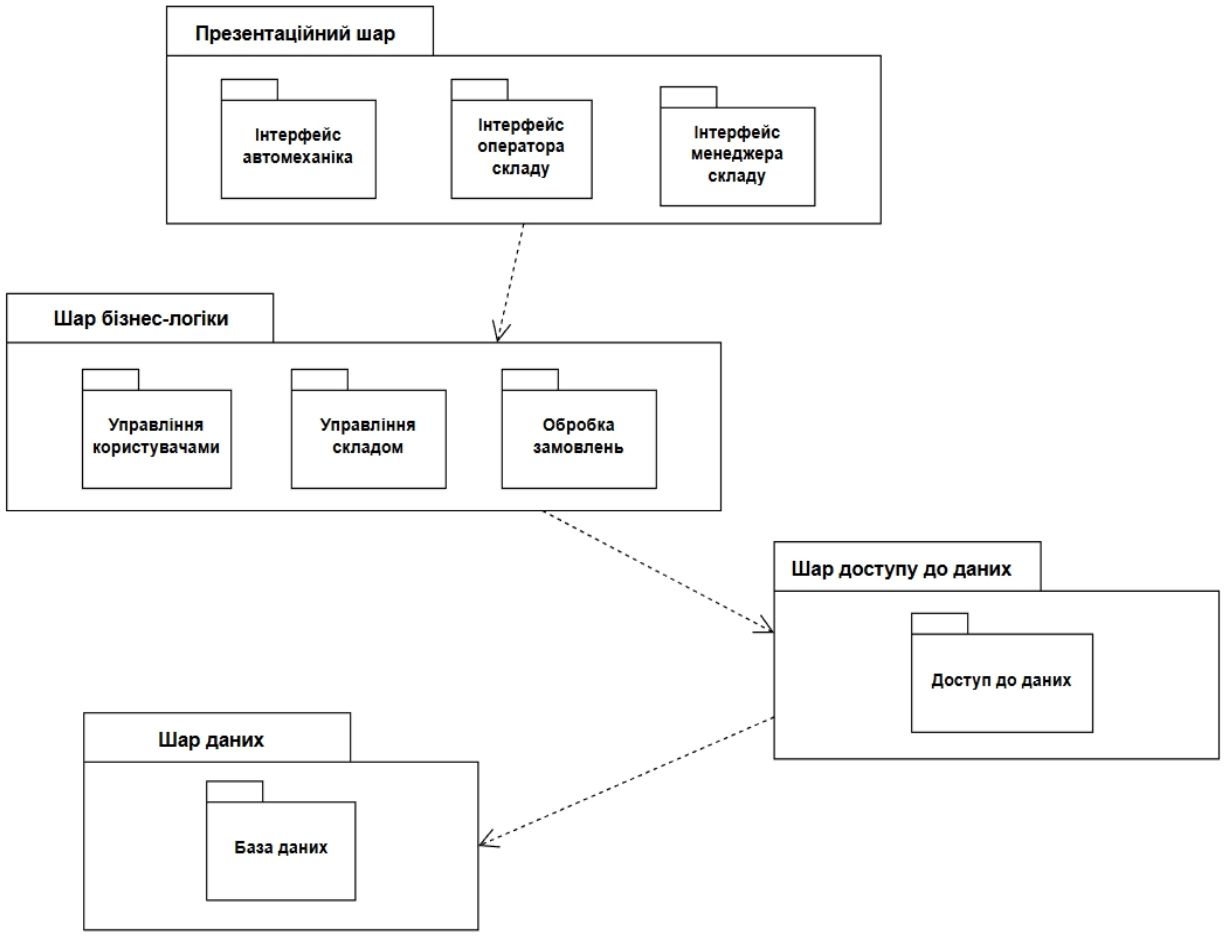

Рис.2.6 Діаграма пакетів додатку

Діаграма пакетів (рис.2.6) відображає архітектуру програмної системи, побудовану за принципами багаторівневої (шарової) моделі. На верхньому рівні знаходиться презентаційний шар, який має інтерфейси для різних користувачів: автомеханіка, оператора складу та менеджера. Цей шар взаємодіє з шаром бізнес-логіки, який включає модулі для управління користувачами, обробки замовлень та управління складом. Бізнес-логіка передає запити до шару доступу до даних, який є посередником між логікою програми та базою даних. Шар доступу до даних виконує запити до бази даних, яка розміщена в окремому шарі даних.

Багаторівнева архітектура чітко ділить систему на ізольовані компоненти. Кожен шар відповідає за свою функціональність та взаємодіє лише з сусідніми рівнями (шарами), забезпечуючи масштабованість системи.

2.3 Діаграма компонентів

Діаграма компонентів UML надає концептуальну картину взаємодії між різними системами. Можуть бути присутні як аспекти логічного, так і фізичного моделювання. Крім того, компоненти автономні. Це модульний системний елемент в UML, який можна замінити на альтернативні. Вони містять конструкції будь-якої складності та є самодостатніми. Лише через інтерфейси вкладені частини взаємодіють з іншими компонентами. Крім того, компоненти мають свої інтерфейси, але вони також можуть отримати доступ до операцій і служб інших компонентів за допомогою їхніх інтерфейсів. На діаграмі компонентів інтерфейси також показують зв’язки та залежності в архітектурі програмного забезпечення.

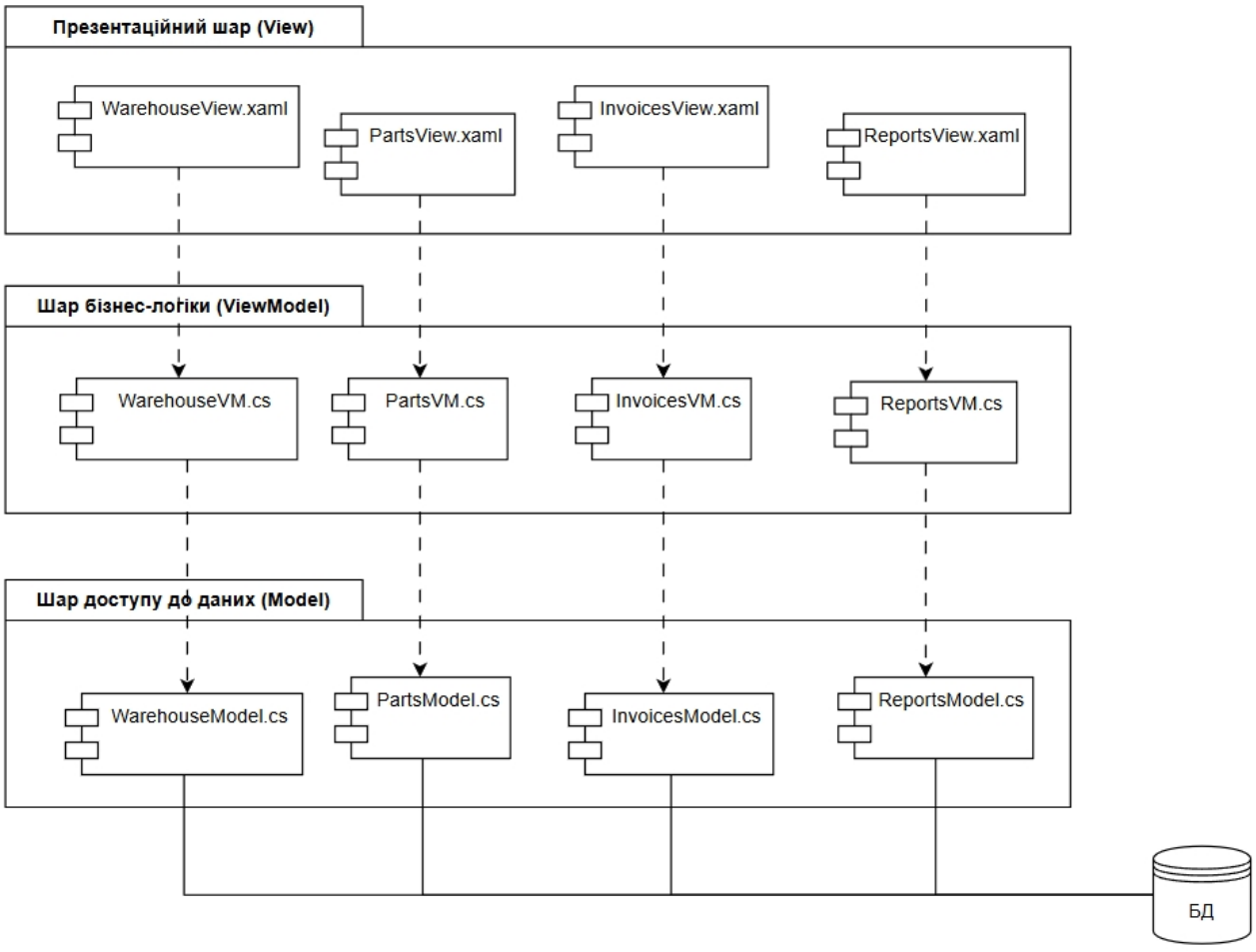

Рис.2.7 Діаграма компонентів додатку

Діаграма компонентів (рис.2.7) демонструє трирівневу архітектуру додатку, реалізовану за принципом MVVM (Model-View-ViewModel). Вона складається з трьох основних шарів: презентаційного (View), бізнес-логіки (ViewModel) та доступу до даних (Model). Кожен шар представлений окремими компонентами, які відповідають за роботу з відповідними сутностями системи: склад (Warehouse), запчастини (Parts), накладні (Invoices) та звіти (Reports). Компоненти верхнього рівня (XAML-представлення) взаємодіють із відповідними ViewModel, які взаємодіють із Model-класами для забезпечення доступу до бази даних. Дана діаграма ілюструє розділення відповідальностей і дотримання принципу слабкого зв’язку між шарами.

3 РОЗРОБКА ІНФОРМАЦІЙНОГО ТА ПРОГРАМНОГО ЗАБЗЕПЕЧЕННЯ

3.1 Система управління базою даних

Система управління базами даних (СУБД) – це спеціалізоване програмне забезпечення, яке забезпечує управління створенням, підтримкою та використанням баз даних. СУБД є посередником між прикладними програмами та даними для структурування збереження інформації та ефективного доступу до них. Основні функції СУБД включають:

- визначення даних через створення та модифікацію структури бази даних;
- маніпулювання даними шляхом додавання, видалення, оновлення та отримання даних;
- забезпечення цілісності даних через контроль за правильністю та несуперечливістю інформації;
- керування транзакціями згідно з принципами ACID;
- управління доступом для забезпечення безпеки та авторизації користувачів;
- оптимізацію продуктивності для ефективного виконання запитів.

СУБД можна класифікувати за різними ознаками. За моделлю даних вони поділяються на реляційні, об'єктно-орієнтовані, ієрархічні та NoSQL. За архітектурою СУБД бувають файл-серверними, клієнт-серверними, вбудованими та розподіленими.

У рамках розробки програмного забезпечення автоматизованого робочого місця для управління складом станції технічного обслуговування на базі технології Windows Presentation Foundation (WPF) було прийнято рішення використати реляційну СУБД Microsoft SQL Server. Цей вибір обумовлений специфікою предметної області та необхідністю забезпечити надійне зберігання

даних, швидкий доступ до інформації та зручну інтеграцію з обраною технологією розробки інтерфейсу користувача.

Вибір саме реляційної моделі бази даних для системи управління складом станції технічного обслуговування зумовлений кількома важливими факторами. По-перше, склад станції технічного обслуговування характеризується чіткою структурованістю даних: запчастини, товари, постачальники, замовлення, клієнти, історія операцій мають однозначно визначені атрибути та зв'язки між собою. По-друге, для подібних систем критично важливою є цілісність даних, яку реляційна модель забезпечує через систему первинних та зовнішніх ключів, унікальних обмежень та транзакцій. По-третє, для складського обліку необхідно забезпечити можливість виконання складних запитів для аналізу даних, формування звітів про залишки, оборотність товарів, що ефективно реалізується за допомогою мови SQL. Також система потребує надійного механізму паралельного доступу до даних, оскільки кілька співробітників можуть одночасно працювати з різними аспектами складського обліку.

Microsoft SQL Server надає оптимальне рішення для реалізації зберігання та обробки даних у системі завдяки тісній інтеграції з екосистемою Microsoft, що включає повну сумісність з .NET Framework, єдину екосистему розробки через Visual Studio та підтримку Entity Framework як ORM-рішення. SQL Server демонструє високу продуктивність та надійність у Windows-середовищі завдяки оптимізації операційної системи, ефективному кешуванню та потужним механізмам забезпечення цілісності даних.

Порівняно з іншими СУБД, SQL Server має суттєві переваги для інтеграції з WPF-додатками, включаючи нативну інтеграцію з .NET та спільний технологічний стек (.NET, XAML, C#), що сприяє ефективній розробці, тестуванню та подальшій підтримці системи.

3.2 Розробка інформаційної бази

При розробці інформаційної бази використовувалась СУБД Microsoft SQL Server. Процес створення розпочався з формування структури БД за допомогою

мови SQL. У процесі було сформовано 16 взаємопов'язаних таблиць, які охоплюють усі необхідні аспекти обліку та управління запчастинами на складі. Приклад коду для створення таблиці Part (запчастина) наведено на рис.3.1. Код інших таблиць БД наведено в ДОДАТКУ А.

```
-- Таблиця запчастин
CREATE TABLE Part
(
    Part_code CHAR(8) NOT NULL PRIMARY KEY,
    SKU_code VARCHAR(12) NOT NULL UNIQUE,
    Part_name VARCHAR(255) NOT NULL,
    Part_description VARCHAR(1000) NOT NULL,
    Manufacturer_code_FK CHAR(8) NOT NULL,
    Category_code_FK CHAR(8) NOT NULL,
    FOREIGN KEY (Manufacturer_code_FK) REFERENCES Part_manufacturer(Manufacturer_code),
    FOREIGN KEY (Category_code_FK) REFERENCES Part_category(Category_code)
);
```

Рис.3.1 Код для створення таблиці Part

Основу структури БД становлять такі ключові таблиці: Supplier (постачальники), Invoice (накладні), Storage_zone (зони складу), Storage_cell (комірки складу), Part_category (категорії запчастин), Part_manufacturer (виробники запчастин), Part (запчастини), Car_brand (бренди автомобілів), Part_image (зображення запчастин) та Part_batch (партії запчастин). Ці таблиці дозволяють структурувати всю базову інформацію про запчастини, їх характеристики та місцезнаходження.

Для підтримки функціональності системи були розроблені додаткові таблиці, такі як Stock_balance (залишки на складі), Part_movement (переміщення запчастин), Part_reservation (резервування запчастин) та Users (користувачі системи). Ці таблиці забезпечують облік кількості запчастин, відстеження їх переміщень між комірками складу та управління резервуванням запчастин для конкретних механіків.

Фізичну можель даних наведено на рис.3.2.

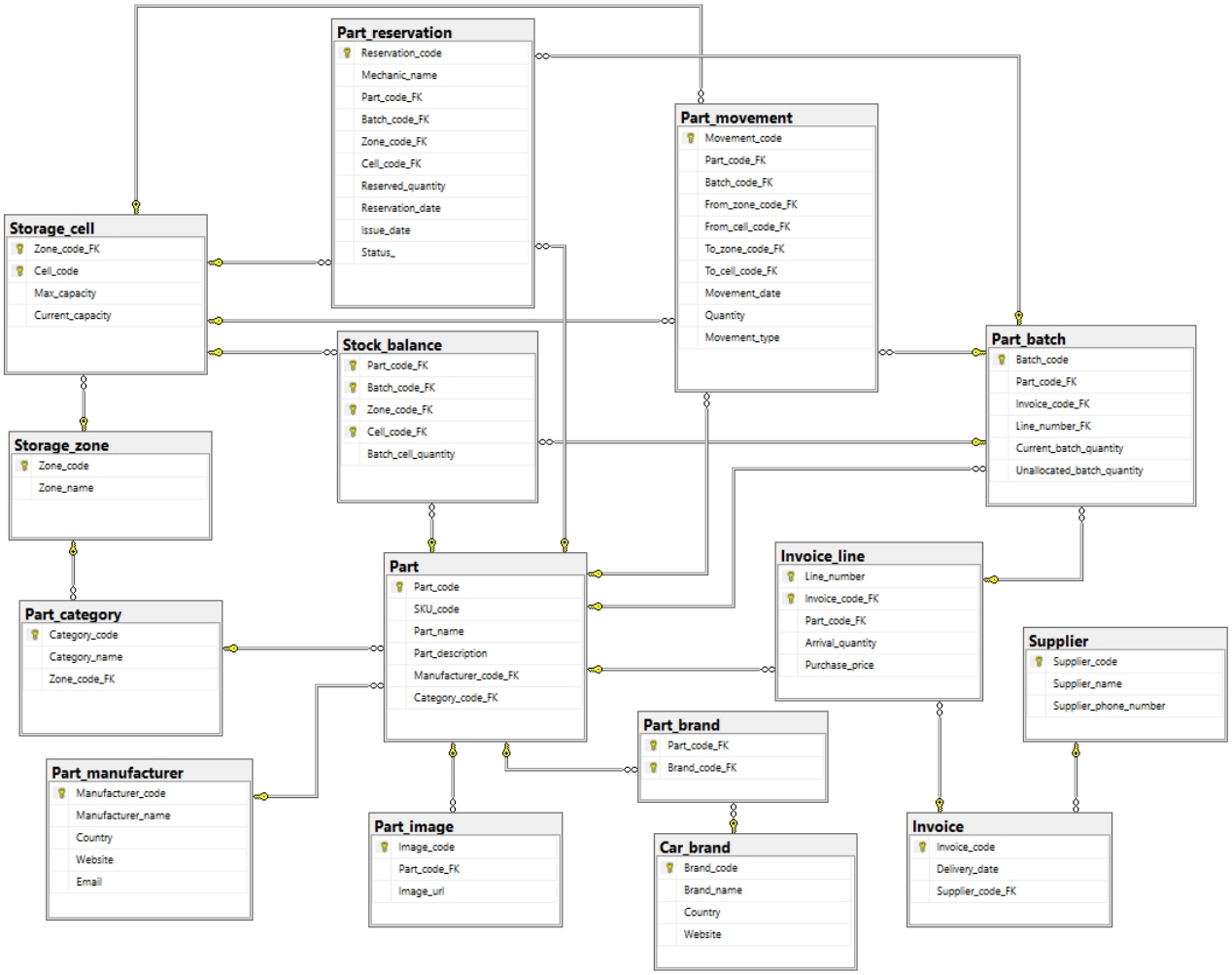

Рис.3.2 Фізична модель даних

При створенні таблиць було використано різні типи даних SQL Server, такі як CHAR та VARCHAR для текстових даних, INT для цілих чисел, DECIMAL для числових значень з фіксованою точністю, DATE та DATETIME для дат і часу, а також VARBINARY для зберігання бінарних даних, а саме хешів паролів.

Для забезпечення функціональності системи було розроблено сім збережених процедур, які потрібні для виконання основних операцій з управління запчастинами. Процедура DistributePartsToCell відповідає за розподіл запчастин з партії по коміркам складу. Процедури MovePartsBetweenCells, MovePartsBetweenCellsForReservation та MovePartsBetweenCellsForCancelReservation реалізують механізми переміщення запчастин між комірками для різних цілей. Процедури CreateReservation,

IssueReservedParts та CancelReservation забезпечують функціональність резервування, видачі та скасування резервацій запчастин для механіків. Код збережених процедур БД наведено в ДОДАТКУ Б.

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Розробка настільного додатку для операційної системи Windows з локальним розміщенням БД безпосередньо на складі станції технічного обслуговування. Основними завданнями системи є облік запчастин, управління товарними залишками, фіксація операцій з переміщення запчастин та формування звітності.

C# як мова програмування. C# є компільованою мовою програмування, що забезпечує вищу продуктивність порівняно з інтерпретованими мовами. Це важливо при обробці великих обсягів складських даних, коли швидкодія критично впливає на ефективність робочих процесів. Статична типізація C# знижує кількість помилок під час виконання програми, що є суттєвою перевагою для бізнес-критичних додатків обліку товарів.

Наявність Entity Framework суттєво спрощує роботу з даними та зменшує обсяг коду для типових операцій із базою даних, що прискорює розробку та підвищує її якість.

Важливою технічною перевагою C# є прямий доступ до Windows API, що в майбутньому забезпечить безпроблемну інтеграцію з периферійними пристроями: сканери штрих-кодів, принтери етикеток, термінали збору даних.

Автоматичне керування пам'яттю через механізм збирання сміття оптимізує використання системних ресурсів для стабільної роботи протягом тривалого часу без необхідності перезапуску програми під час робочої зміни.

Технологія Windows Presentation Foundation (WPF) для створення інтерфейсу користувача. WPF використовує апаратне прискорення DirectX для рендерингу інтерфейсу, що забезпечує високу швидкість відображення при роботі зі складними таблицями та великою кількістю елементів інтерфейсу. Це

особливо важливо при перегляді великих каталогів запчастин та історії складських операцій.

Архітектурний патерн MVVM, який є основою для WPF-додатків, забезпечує модульну структуру, що значно спрощує подальшу модифікацію та масштабування системи при змінах бізнес-процесів складу. Декларативне створення інтерфейсу через XAML прискорює розробку та забезпечує чітке розділення логіки і представлення, що підвищує підтримуваність коду.

Механізм прив'язки даних (data binding) суттєво зменшує кількість коду для синхронізації стану моделі та представлення, знижуючи ймовірність помилок при оновленні даних на екрані. Нативна підтримка Windows та можливість створення повноцінних встановлюваних додатків забезпечує стабільну продуктивність та доступ до системних ресурсів без обмежень, які характерні для веб-додатків.

MS SQL Server як СУБД забезпечує повну підтримку транзакційності за принципами ACID, що гарантує цілісність даних при одночасних операціях з боку кількох користувачів системи. Ця властивість критично важлива при обліку матеріальних цінностей, коли помилки в даних можуть призвести до фінансових втрат.

Безкоштовна версія SQL Server Express призначена саме для малого бізнесу та не потребує додаткових ліцензійних витрат, що робить рішення економічно доцільним для невеликих СТО. Потужна система індексації забезпечує швидкий пошук по великих каталогах запчастин та історії складських операцій навіть при значному накопиченні даних з часом.

Вбудовані механізми резервного копіювання мінімізують ризик втрати даних при апаратних збоях, що часто трапляються в умовах складських приміщень. Підтримка збережених процедур дозволяє реалізувати складну бізнес-логіку безпосередньо на рівні бази даних, зменшуючи мережевий трафік та підвищуючи загальну продуктивність системи.

Отже, обрана комбінація технологій оптимально відповідає технічним вимогам проєкту автоматизації робочого місця для управління складом СТО та

забезпечує найкраще співвідношення між продуктивністю, надійністю й зручністю подальшого супроводу системи.

3.4 Алгоритмізація та програмування програмних модулів

В ході розробки програмної системи було реалізовано багато програмних рішень для вирішення поставлених задач. Далі розглянемо код деяких з них. Код розглянутих програмних рішень надано в ДОДАТКУ В.

Для реалізації функціоналу додавання нової запчастини до бази даних було розроблено відповідний алгоритм, який охоплює усі необхідні етапи перевірки введених даних, формування об’єкта запчастини, обробки асоційованих сутностей та збереження у базу. Цей процес є одним із ключових у системі, адже забезпечує поповнення каталогу деталей з урахуванням їх категорій, виробників і сумісних брендів. На рисунку 3.3 наведено блок-схему, що ілюструє послідовність дій, які виконуються при додаванні нової запчастини.

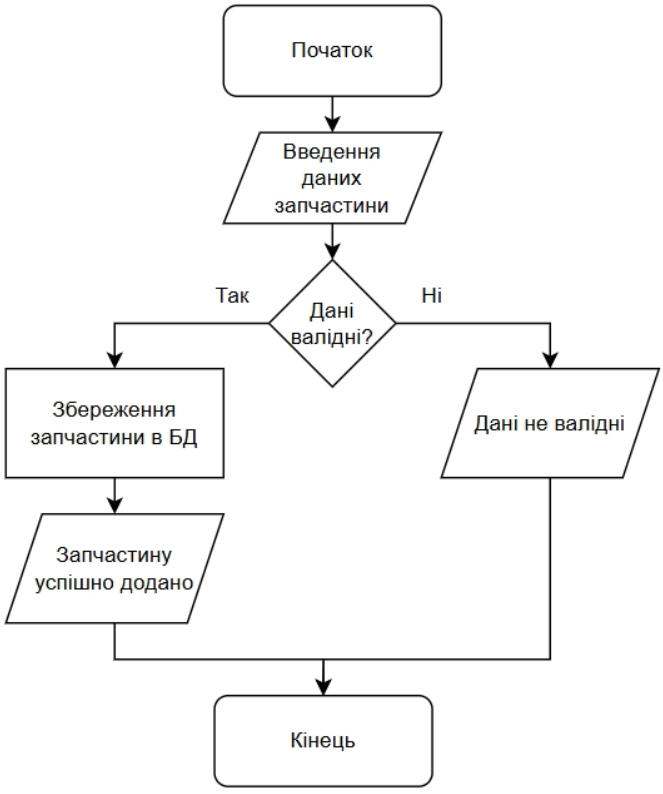

Рис.3.3 Блок-схема алгоритму додавання запчастини

У застосунку є спеціальна форма (рис.3.4) для додавання нової запчастини. Вона містить кілька полів для введення даних: артикул, назва, опис, категорія, марки авто та виробник. Частину з них користувач вводить вручну, а частину обирає з випадаючих списків.

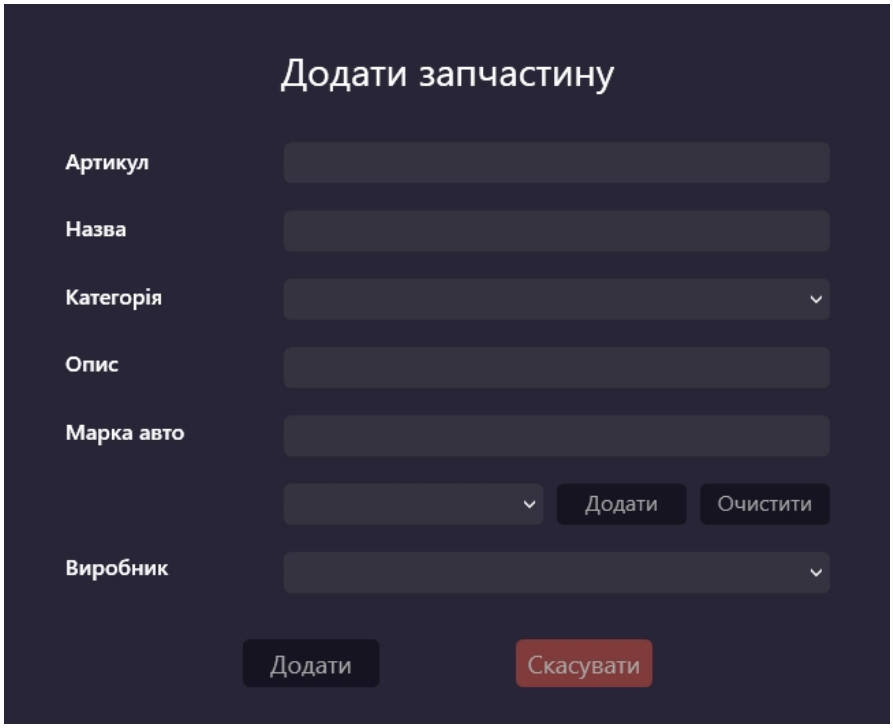

Рис.3.4 Форма додавання запчастини

Для марок авто є окрема логіка – користувач може обрати марку з випадаючого списку і додати її до запчастини за допомогою кнопки. Усі вибрані марки зберігаються у вигляді тексту, який пізніше обробляється перед збереженням у базу.

Після заповнення форми користувач натискає кнопку "Додати". У ViewModel викликається метод AddPartAsync(). Спочатку перевіряється, чи заповнені всі обов’язкові поля. Якщо якесь з них порожнє, з’являється повідомлення з підказкою, яке саме поле потрібно заповнити.

Якщо все гаразд, формується об’єкт Part, де:

1. генерується унікальний код запчастини;
2. категорія та виробник переводяться у відповідні коди через спеціальні сервіси;

3. усі введені значення зберігаються в модель.

Далі ця запчастина передається в сервіс збереження, який створює SQL-запити. Спочатку в таблицю Part додається основна інформація про запчастину. Потім для кожної вибраної марки авто виконується пошук її коду в таблиці Car_brand, після чого створюються записи у зв’язувальній таблиці Part_brand.

Усі ці операції відбуваються в рамках транзакції. Якщо якась із марок не знайдена в базі, то транзакція скасовується і користувач бачить повідомлення про помилку. Після успішного збереження форма очищується і користувачу показується повідомлення про успішне додавання запчастини.

Після додавання запчастин до системи важливо забезпечити їх правильне розміщення у комірках складу. Для цього реалізовано механізм ручного розподілу, який дозволяє оператору самостійно вказати кількість запчастин і обрати відповідну комірку. Процес супроводжується перевірками на коректність введених даних та обсяг доступного місця у комірці. На рисунку 3.5 представлено блок-схему алгоритму, що описує логіку ручного розподілення запчастин по складських комірках.

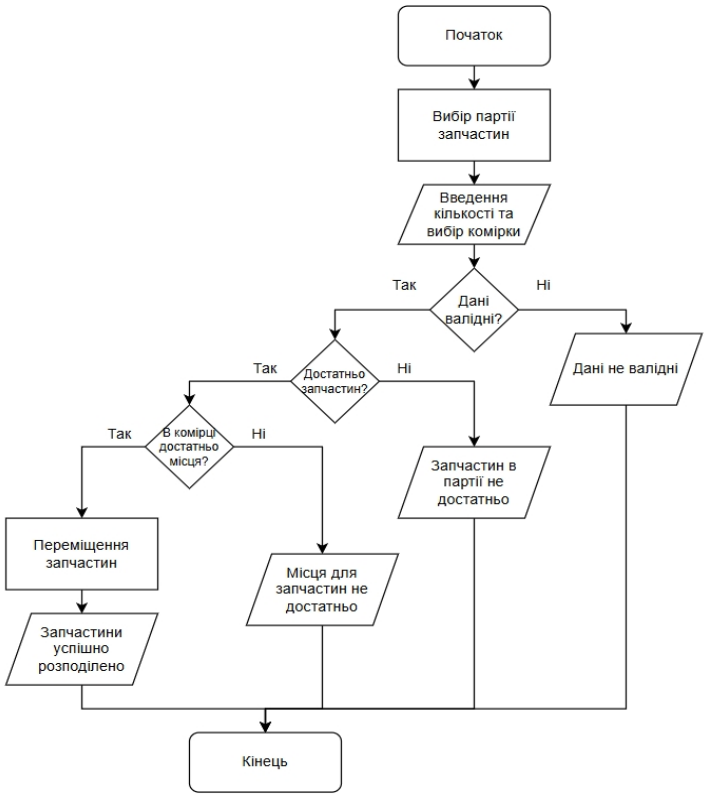

Рис.3.5 Блок-схема алгоритму розподілення запчастин в комірку

Після додавання партії запчастин у систему, її потрібно розподілити по комірках складу. Для цього користувач відкриває спеціальну форму (рис.3.6), де вводить кількість запчастин, яку потрібно перемістити, та обирає комірку зі списку доступних. У формі два поля: "Кількість" і "Комірка", які прив’язані до властивостей BatchQuantity та SelectedCell відповідно. Після введення даних користувач натискає кнопку "Перемістити", яка активує команду MoveBatchCommand.

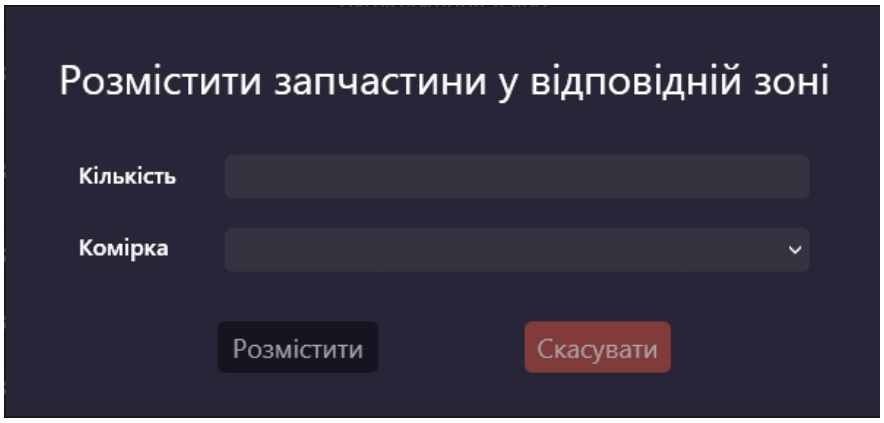

Рис.3.6 Форма розміщення запчастин

Ця команда викликає метод MoveBatchAsync, який відповідає за перевірку введених даних. Якщо хоча б одне поле порожнє, або кількість вказана некоректно, користувачу відображається повідомлення про помилку. У випадку, якщо все заповнено правильно, метод викликає сервіс _warehouseStockMovementService.DistributePartsToCellAsync, куди передає код партії, код зони, код обраної комірки та кількість.

Метод DistributePartsToCellAsync реалізує виклик збереженої процедури DistributePartsToCell у базі даних. Якщо ця процедура завершується успішно, на екран виводиться повідомлення про успішне переміщення і форма закривається. У разі помилки користувач бачить відповідне повідомлення.

Після надходження нових партій запчастин, користувач може скористатися автоматичним розподілом цих партій по комірках складу. У вікні (рис.3.8), яке відкривається, відображається коротке інформаційне повідомлення (Message), а також дві кнопки – "Розподілити" та "Скасувати". Кнопка

"Розподілити" активує команду AutoMoveCommand, яка викликає метод AutoMoveBatch.

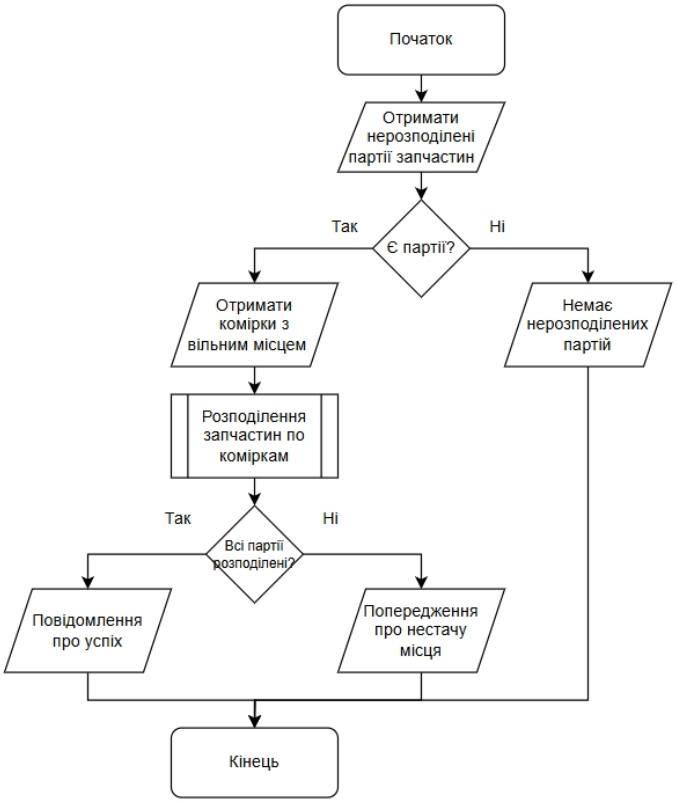

Рис.3.7 Блок-схема алгоритму автоматичного розподілення запчастин по коміркам

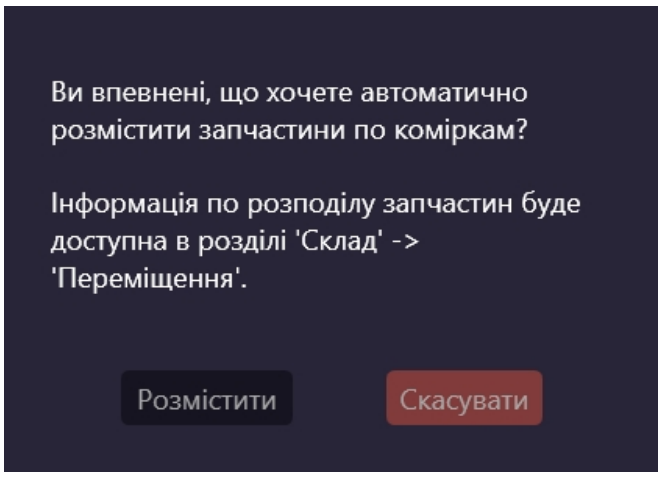

Рис.3.8 Вікно підтвердження автоматичного розподілення запчастин

Метод AutoMoveBatch починається з перевірки чи є партії з нерозподіленими залишками (Unallocated_batch_quantity). Якщо таких немає, процес завершується. Якщо є, то метод завантажує всі доступні комірки з бази даних через _loadDataService.GetStorageCellsAsync(), фільтрує ті, в яких ще є вільне місце і сортує їх за поточною заповненістю щоб використовувати найменш заповнені першими.

Для кожної партії виконується спроба розподілити її в комірки, які розташовані в тій зоні складу, що дозволена для даної запчастини (визначається методом _loadDataService.GetStorageZoneCodeByPartAsync). Партія поступово розподіляється в декілька комірок, у кожну з яких переноситься максимально можлива кількість запчастин, не перевищуючи наявного місця.

Для кожного переміщення викликається метод _warehouseStockMovementService.DistributePartsToCellAsync, який звертається до збереженої процедури DistributePartsToCell у базі даних, передаючи код партії, код зони, код комірки та кількість запчастин.

Якщо для деякої партії не вистачає місця і частина залишилася нерозподіленою, користувачу виводиться попередження з вказаною кількістю залишку. Якщо ж усі партії розміщені успішно, на екран виводиться повідомлення про успішне завершення процесу.

Оскільки запчастини в системі можуть бути пов’язані з кількома автомобільними брендами, важливо реалізувати механізм додавання сумісних марок для кожної одиниці товару. Цей функціонал дозволяє забезпечити гнучкість у пошуку й фільтрації запчастин за брендами, що полегшує роботу персоналу та зменшує ризик помилок при видачі. На рисунку 3.9 зображено алгоритм, який демонструє послідовність дій при додаванні бренду до конкретної запчастини.

Процес генерації звітів у системі реалізовано через інтерактивну форму (рис.3.10), яка дозволяє користувачеві вказати період (дату початку та завершення), а також, за потреби, обрати додаткові параметри, як-от категорію. Введені дані обробляються після активації команди CreateReportCommand.

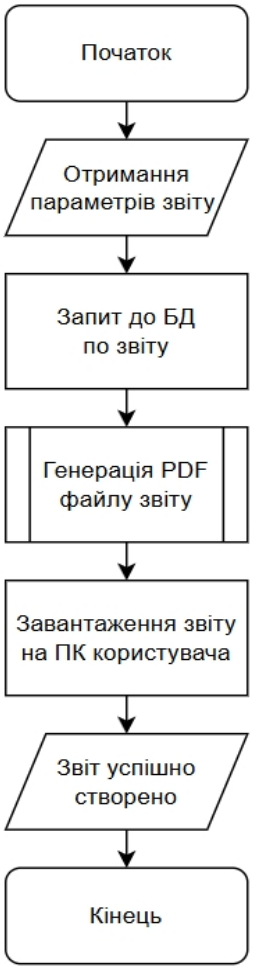

Рис.3.9 Блок-схема алгоритму генерації звітів

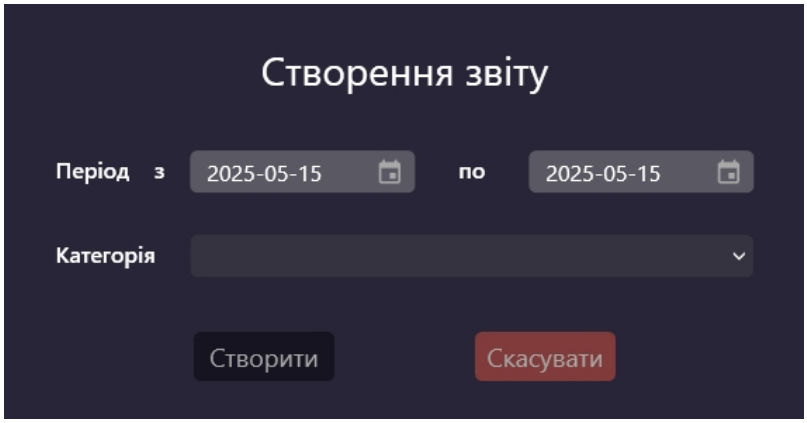

Рис.3.10 Форма генерації звіту

Після ініціювання створення звіту, ViewModel викликає асинхронний метод, який отримує потрібні записи з бази даних за допомогою _loadDataService. Ці записи фільтруються відповідно до вказаного діапазону дат і, за необхідності, категорії.

Далі система формує PDF-документ за допомогою бібліотеки iText7. Створюється заголовок із зазначенням обраного періоду, використовується шрифт Arial (звичайний та напівжирний). Основна частина звіту представлена у вигляді таблиці з фіксованими стовпцями, які заповнюються даними: код запчастини, назва, кількість, загальна сума тощо.

Після основного вмісту додається підсумок наприклад, обчислена загальна сума витрат, а також нижній колонтитул із поточною датою формування звіту та полем для підпису. На завершення викликається функція AddPageNumbers, яка додає номери сторінок на кожну сторінку PDF-документа.

Готовий звіт зберігається у вказане користувачем місце, після чого система покаже повідомлення про успішне завершення процесу або про помилку в разі її виникнення. Приклад звіту представлено в ДОДАТКУ Д.

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ ПРОГРАМНОЇ СИСТЕМИ

4.1 Тестування системи

Тестування програмного забезпечення – це процес, під час якого проводяться експерименти для виявлення помилок і дефектів у програмі. Воно дає змогу переконатися, що ПЗ працює коректно, відповідає вимогам і очікуванням користувачів, а також працює надійно і безпечно.

Тестування є критично важливим етапом життєвого циклу розробки програмного забезпечення, оскільки дозволяє ідентифікувати та усунути проблеми до того, як система потрапить до кінцевих користувачів.

В індустрії розробки ПЗ існує широкий спектр видів тестування, кожен з яких фокусується на певних аспектах системи.

Функціональне тестування перевіряє, чи відповідає програмне забезпечення заданим функціональним вимогам. Воно фокусується на тестуванні функцій, операцій і поведінки програми, а також включає перевірку вхідних даних, правильності обробки даних, роботи функцій і коректності вихідних результатів.

Нефункціональне тестування оцінює якісні атрибути програмного забезпечення, як-от продуктивність, надійність, безпека, зручність використання та сумісність. Приклади нефункціонального тестування включають навантажувальне тестування, регресійне тестування, тестування безпеки, тестування юзабіліті та інші.

Окрім цих основних категорій, у сучасній практиці застосовуються інтеграційне тестування, яке перевіряє взаємодію між різними модулями системи; системне тестування, що оцінює відповідність системи технічним вимогам; приймальне тестування, яке визначає готовність програмного продукту до передачі замовнику; регресійне тестування, що перевіряє, чи не порушили нові зміни існуючу функціональність; стрес-тестування, яке оцінює поведінку

системи при екстремальних навантаженнях; та юніт-тестування, що фокусується на перевірці окремих компонентів коду.

Для тестування розробленої системи було обрано функціональне тестування як найбільш релевантний підхід для вирішення поставлених задач. Цей вибір обумовлений кількома ключовими факторами. По-перше, розроблена система має чітко визначений набір функціональних вимог, тому функціональне тестування дозволяє безпосередньо перевірити відповідність реалізації цим вимогам. По-друге, функціональне тестування фокусується на перевірці що система виконує саме ті завдання, для яких вона була створена, без заглиблення в деталі реалізації.

Розглянемо приклад додавання нової накладної та розподілення поставлених запчастин по коміркам складу.

Після авторизації здійснюємо перехід на сторінку накладних (рис.4.1) та відкриваємо форму додавання накладної (рис.4.2).

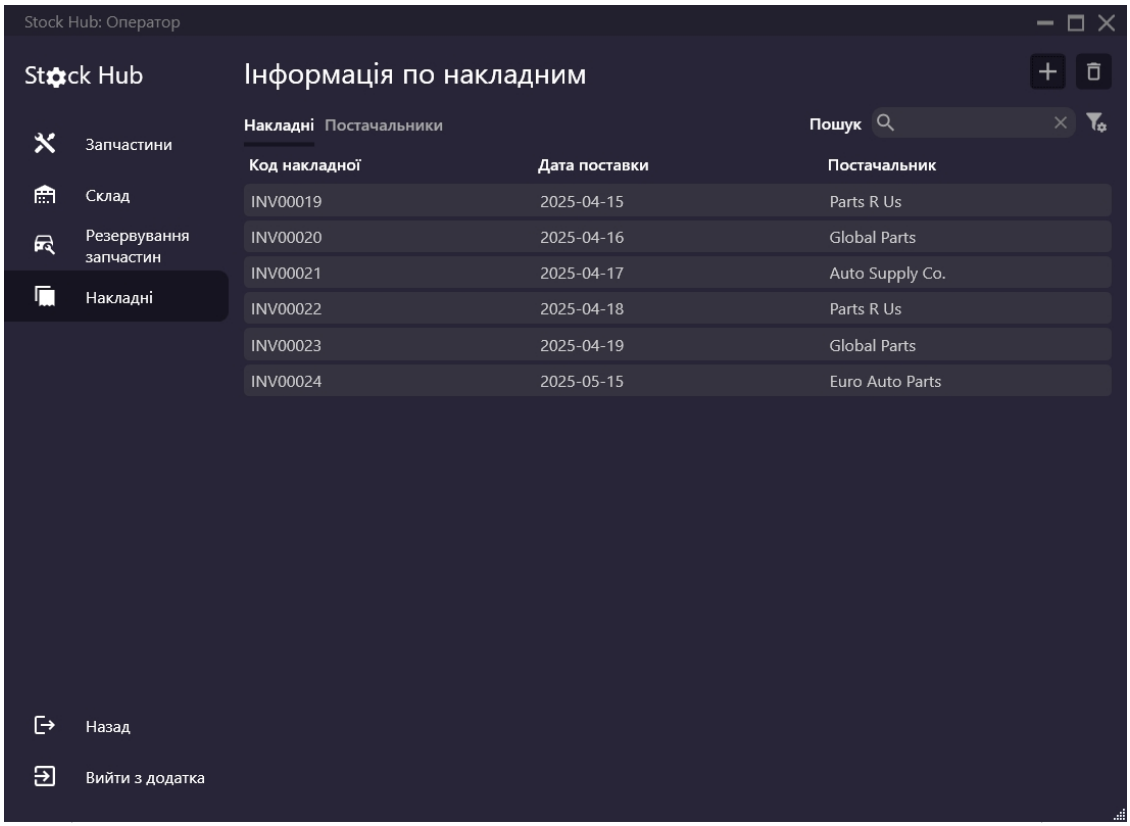

Рис.4.1 Сторінка накладних

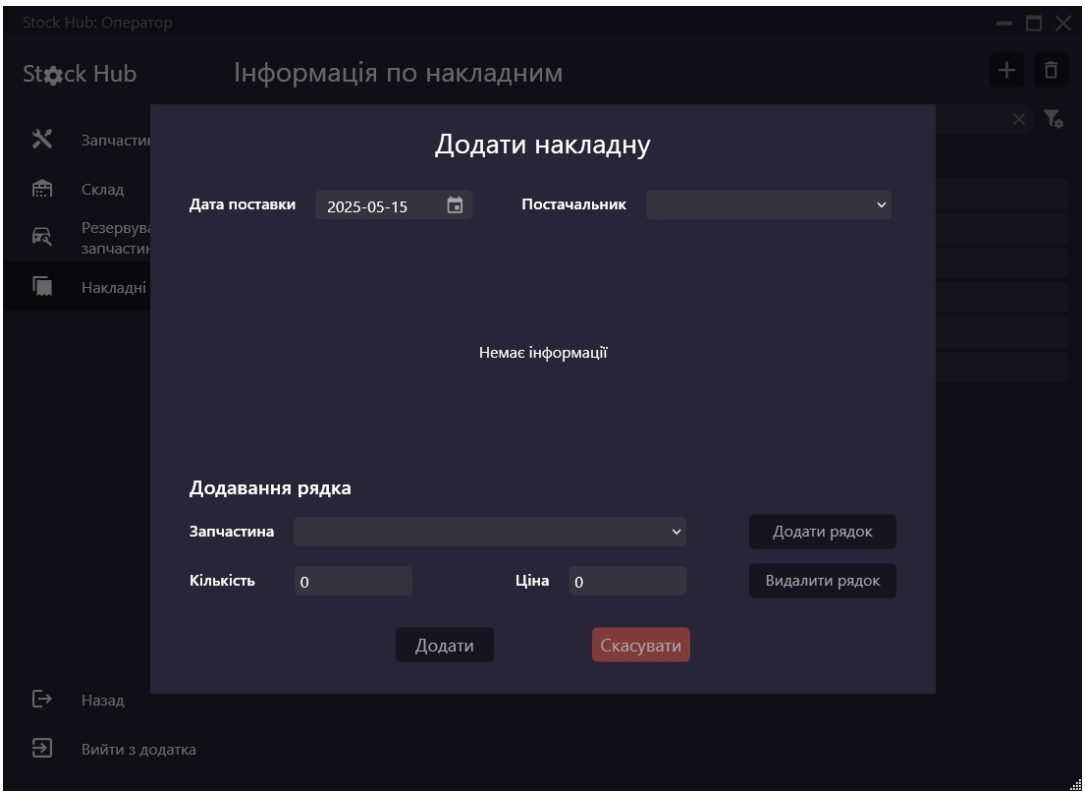

Рис.4.2 Форма додавання накладної

Для успішного додавання накладної необхідно виконати дві умови: обрати постачальника та додати хоча б один рядок накладної. При ігноруванні цих умов видає помилку (рис.4.3).

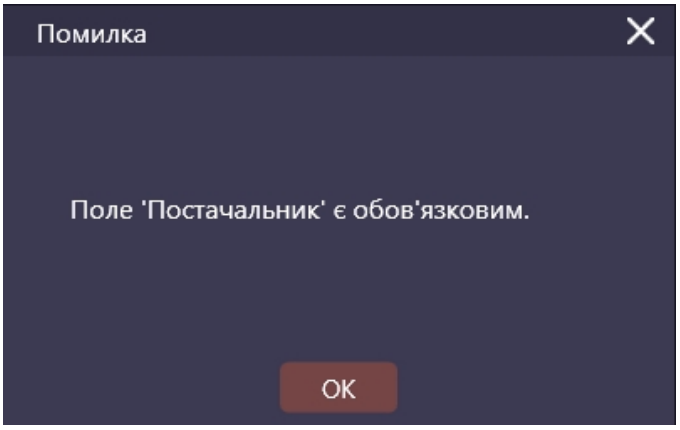

Рис.4.3 Помилка: поле “постачальник” порожнє

Для додавання накладної використовувалася перевірка чи поля заповнені коректно (рис.4.4).

```
var requiredFields = new (object FieldValue, string FieldName)[]
{
       (SelectedDate, "Дата"),
       (SelectedSupplier, "Постачальник"),
       (Lines, "Рядки")
};

foreach (var (value, name) in requiredFields)
{
       if (value == null || (value is string str && string.IsNullOrWhiteSpace(str))
|| (value is ObservableCollection<Invoice_line> coll && coll.Count == 0))
       {
              if (name == "Рядки")
              {
                     _modalNavigation.ShowMessage<ErrorMessageViewModel>(250, 400,
              "Накладна має мати хоча б 1 рядок!");
              }
       else
       {       _modalNavigation.ShowMessage<ErrorMessageViewModel>(250, 400, $"Поле
'{name}' є обов'язковим.");
       }
        return;
       }
}
```

Рис.4.4 Перевірка валідності полів накладної

Розглянемо випадок коли всі умови виконані, тоді накладна буде успішно додана в систему і з’явиться повідомлення про успіх операції (рис.4.5), крім цього, після закриття форми таблиця накладних оновилась (рис.4.6).

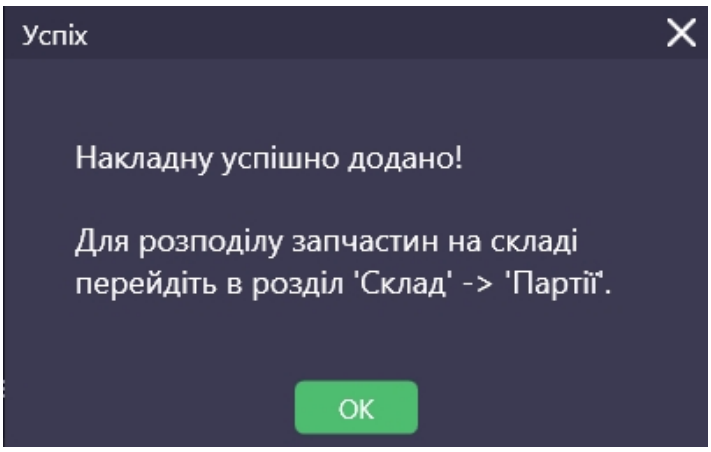

Рис.4.5 Успіх: накладна додана

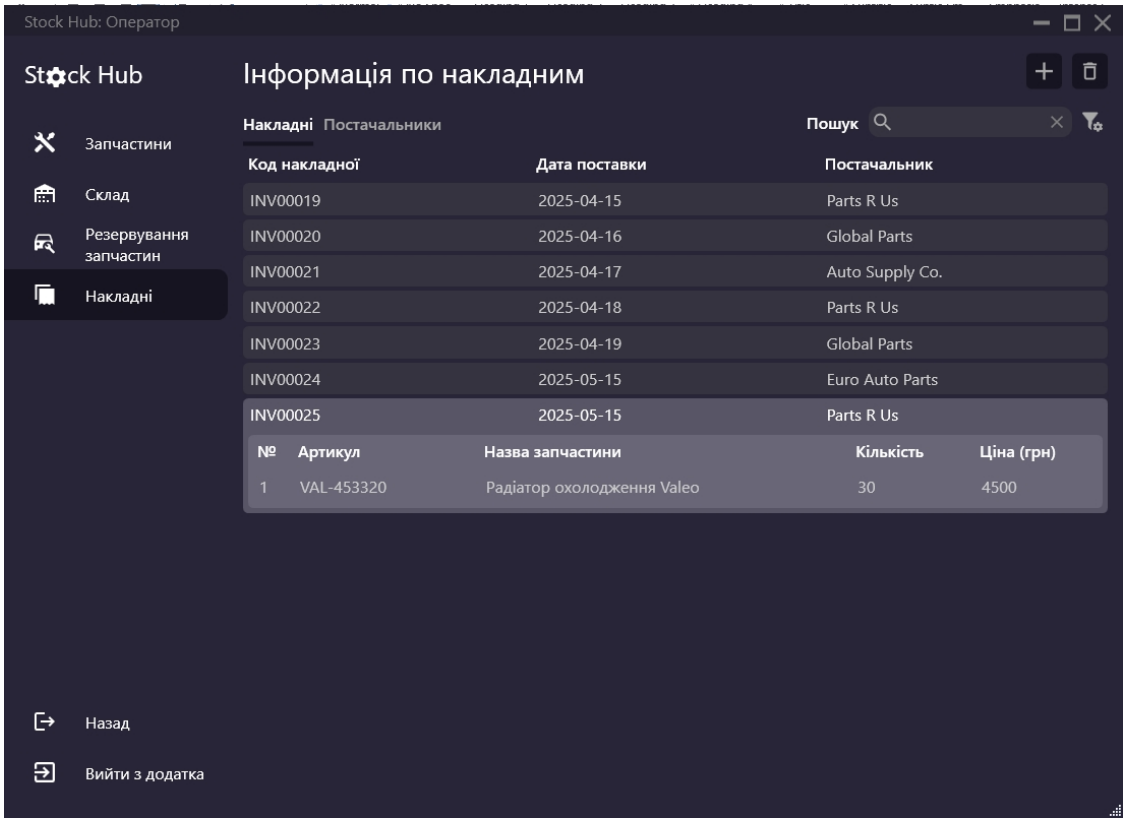

Рис.4.6 Таблиця накладних після оновлення

Оновлення таблиці відбувається після закриття модального вікна форми за допомогою оповіщення ViewModel сторінки з накладними, після сповіщення викликається функція OnModalClosed (рис.4.7) яка оновлює сторінку. Сповіщення було реалізовано за допомогою події, створеної в інтерфейсі сервісу для навігації між модальними вікнами (рис.4.8). Для коректної роботи сповіщення, ViewModel сторінки з накладними підписується на подію закриття модального вікна з сервісу навігації. Для підписання на подію необхідно в конструкторі ViewModel вказати функцію, яка виконується при оповіщенні.

```
private async void OnModalClosed(object sender, EventArgs e)
{
       await LoadInvoicesAsync();
       await FilterInvoicesAsync();
}
```

Рис.4.7 Функція OnModalClosed

```
    public interface IModalNavigationService
    {
        event EventHandler ModalClosed;
        event EventHandler ModalOpened;
        void ShowModal<TViewModel>(int height, int width) where TViewModel :
ViewModel;
        void ShowModal(int height, int width, ViewModel viewModel);
        void CloseModal();
        void ShowMessage<TViewModel>(int height, int width, string message) where
TViewModel : ViewModel;
        void ShowMessageForLogin<TViewModel>(int height, int width, string message)
where TViewModel : ViewModel;
        void ShowMessageForMainWindow<TViewModel>(int height, int width, string
message) where TViewModel : ViewModel;
        void CloseMessage();
    }
```

Рис.4.8 Інтерфейс сервісу навігації між модальними вікнами

Перейдемо до розподілення поставлених запчастин по коміркам складу. Для цього, по вказівці повідомлення про додавання накладної (див. рис.4.5), перейдено на сторінку складу у вкладку “Партії” (рис.4.9), де бачимо додану партію радіаторів охолодження. Далі спробуємо автоматично розподілити запчастини по коміркам (рис.4.10).

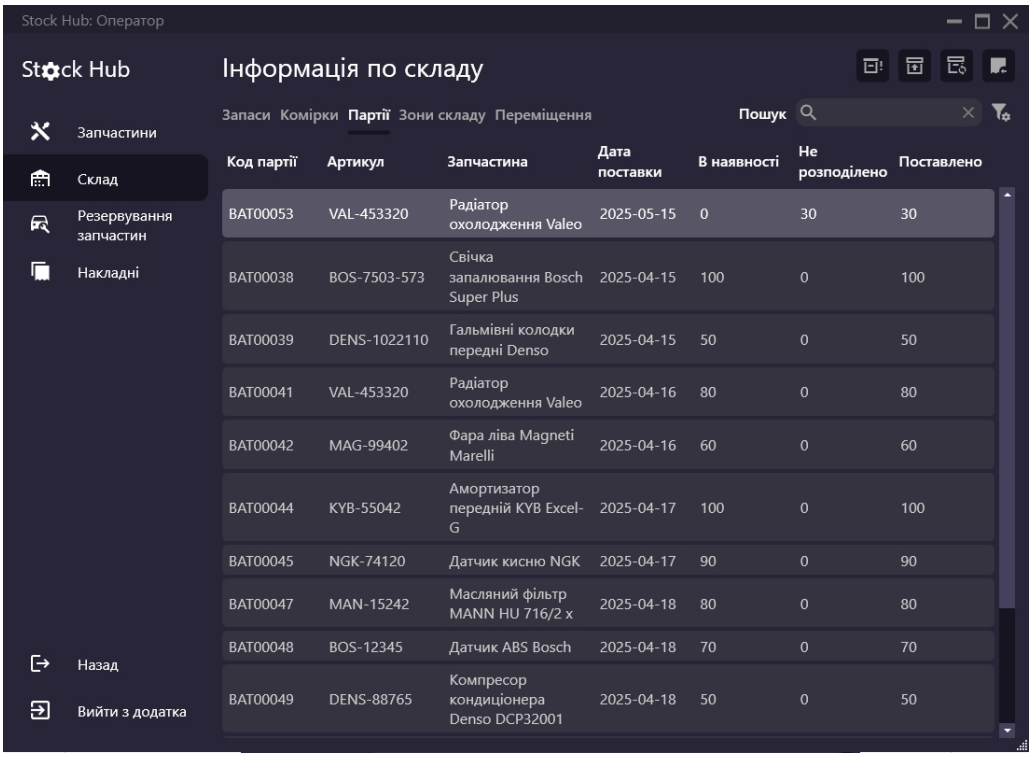

Рис.4.9 Таблиця поставлених партій

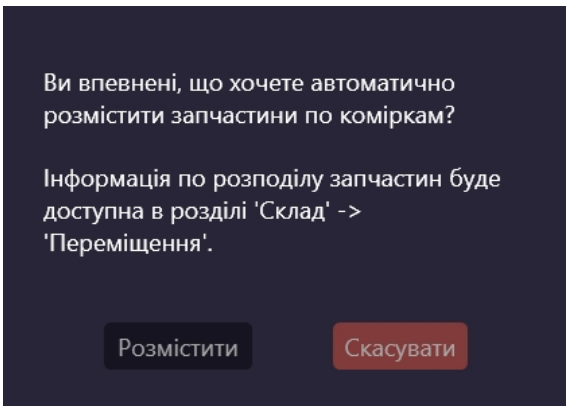

Рис.4.10 Підтвердження розподілення запчастин по коміркам

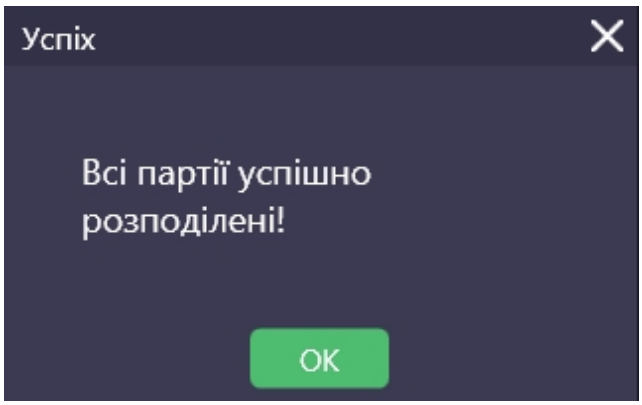

Рис.4.11 Успіх: партії розподілені по коміркам

Підтверджуємо розподілення. Успіх (рис.4.11). Бачимо, що радіатор змінив статус з “Не розподілено” на “В наявності” (рис.4.12).

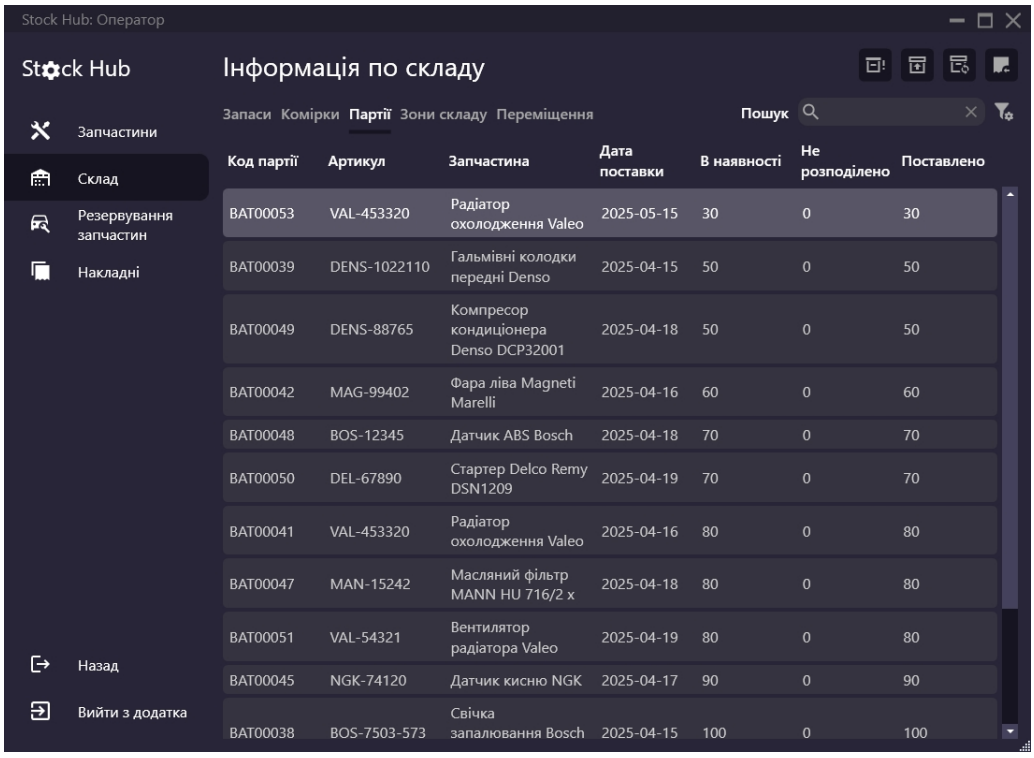

Рис.4.12 Зміна статусу партії

Далі перевіримо чи справді запчастини розподілились по коміркам. Для цього перейдемо у вкладку “Переміщення”, де бачимо лог переміщення запчастин (рис.4.13). Шукаємо “Радіатор охолодження Valeo”. Запчастини переміщені в комірку CELL-040 в зону охолодження та опалення.

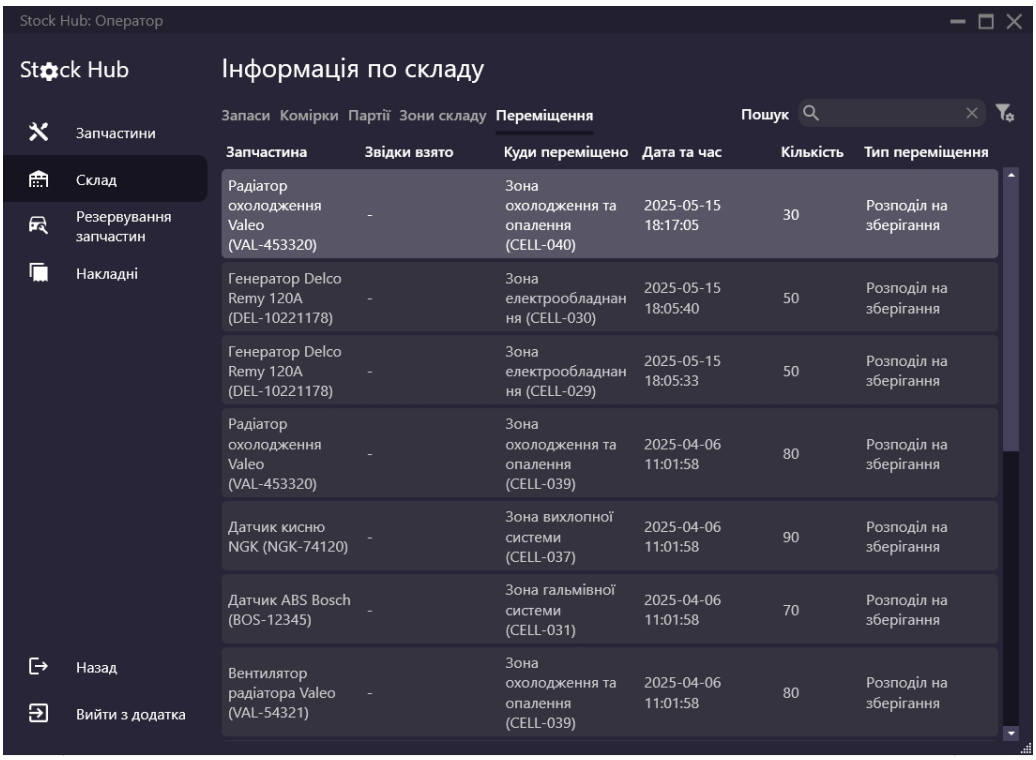

Рис.4.13 Логи переміщень запчастин

Для перевірки переміщення перейдемо у вкладку “Комірки” та знайдемо необхідну (рис.4.14). Переміщення пройшло успішно.

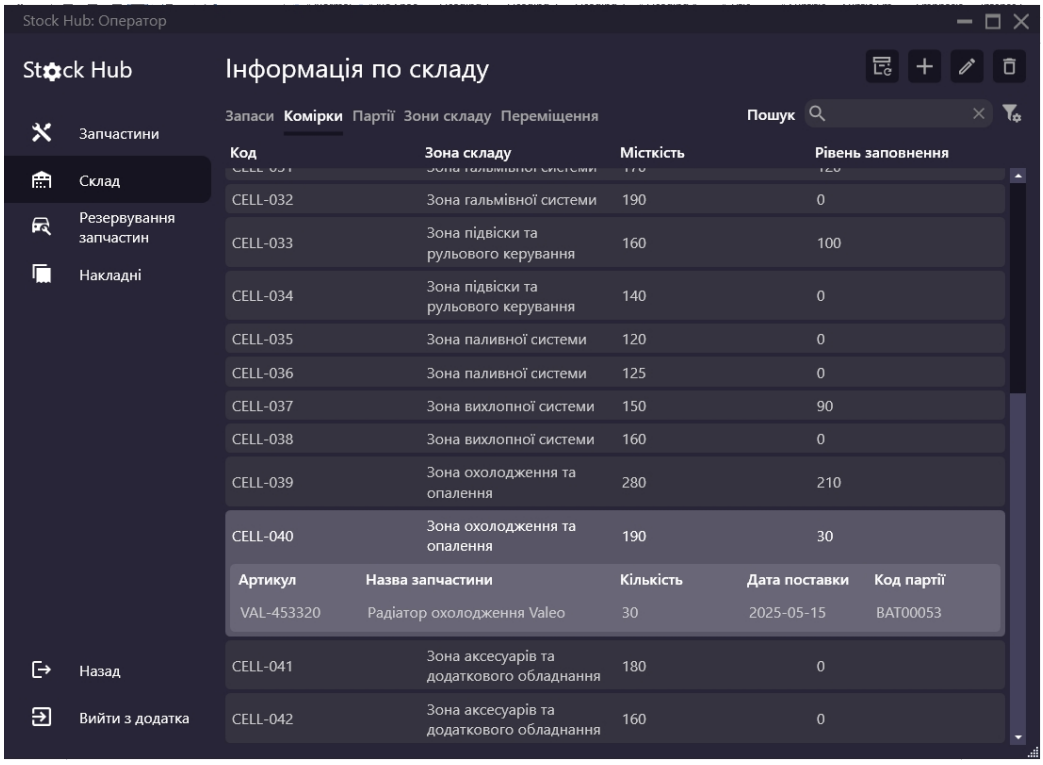

Рис.4.14 Таблиця комірок складу

Отже, функціональне тестування по створенню накладної та розміщення поставлених запчастин було проведено успішно, логіка працює добре та відображення UI частини здійснюється належним чином.

4.2 Вимоги до апаратного та програмного забезпечення

Подальший розгляд зосереджено на вимогах до системи з позиції власника, що дозволяє акцентувати увагу на очікуваннях щодо її функціональності, надійності та зручності використання.

Для ефективної роботи програмного забезпечення автоматизованого робочого місця для управління складом станції технічного обслуговування необхідно визначити його архітектуру та вимоги до апаратного і програмного забезпечення. Одним із важливих етапів є побудова діаграми розміщення (топології), яка відображає взаємодію між основними компонентами системи. На рис. 4.15 представлено архітектуру системи, що складається з двох основних вузлів: клієнтського пристрою (User PC) та сервера (Server), який містить вузол операційної системи (Windows 10/11).

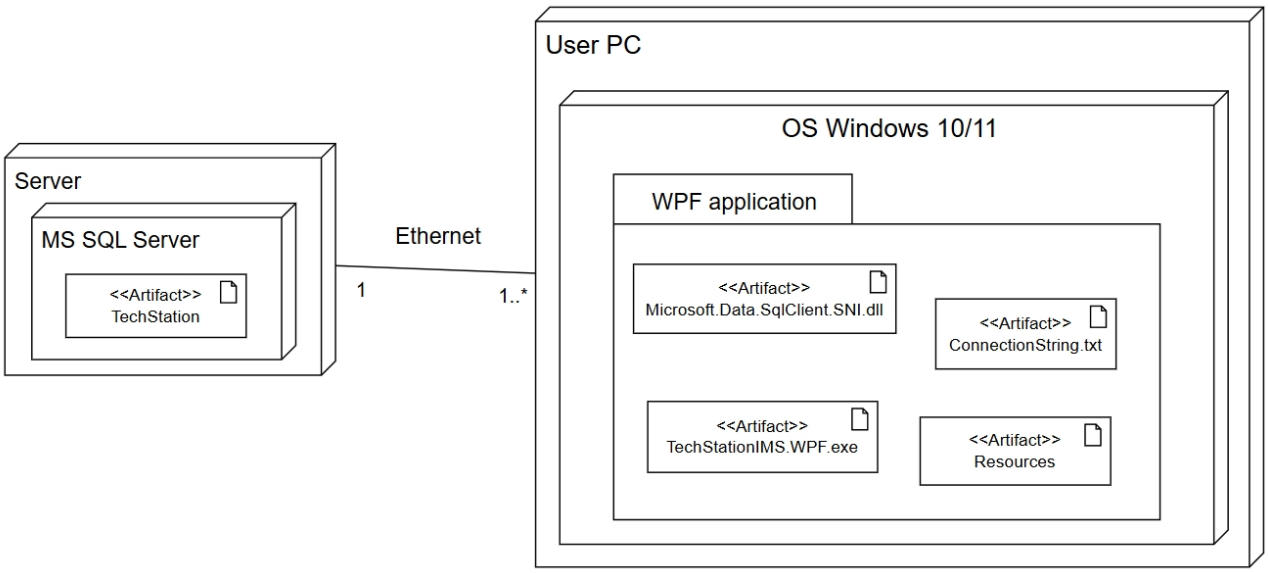

Рис.4.15 Діаграма розміщення

Система працює за класичною клієнт-серверною моделлю. Сервер включає MS SQL Server, який своєю чергою включає компонент «TechStation»,

який є базою даних. Сервер відповідає за зберігання даних, обробку запитів, керування бізнес-логікою та забезпечення цілісності даних.

Користувацький ПК працює під управлінням Windows 10/11 і містить WPF-застосунок, що забезпечує графічний інтерфейс для взаємодії з користувачем. До складу цього додатку входять бібліотека Microsoft Data SqlClient SNI.dll для мережевої взаємодії з SQL Server, файл ConnectionString.txt з параметрами підключення до сервера, виконуваний файл TechStationIMS.WPF.exe та папка Resources з ресурсами програми.

Коли користувач запускає додаток на своєму ПК, програма зчитує параметри підключення та встановлює з'єднання з сервером через Ethernet. Користувач взаємодіє з інтерфейсом програми, яка обмінюється даними з сервером. Сервер обробляє запити, виконує необхідні операції з базою даних і повертає результати клієнту, а користувацький додаток відображає отримані дані через інтерфейс.

Апаратні вимоги до впровадження програмної системи визначають мінімальну конфігурацію обладнання, необхідну для стабільної та ефективної роботи всіх компонентів застосунку, включаючи серверну частину та клієнтські пристрої. Деталізовану специфікацію апаратного забезпечення наведено в таблиці 4.1.

Таблиця 4.1

Апаратні вимоги

Компонент	Мінімальні	Рекомендовані
Процесор	4 ядра, 2.0 GHz (x64)	8+ ядер, 3.0+ GHz (x64)
Оперативна пам'ять	8 GB RAM	16+ GB RAM
Жорсткий диск	100 GB вільного місця (HDD 7200 RPM або кращий)	250+ GB вільного місця (SSD, бажано NVMe)
Мережа	Gigabit Ethernet	Gigabit Ethernet або швидша

Програмні вимоги охоплюють перелік необхідного системного та прикладного програмного забезпечення, яке забезпечує коректну роботу інформаційної системи, зокрема операційне середовище, середовище виконання, бібліотеки та інструменти розробки. Повний перелік рекомендованих програмних компонентів подано в таблиці 4.2.

Таблиця 4.2

Програмні вимоги

Компонент	Мінімальні	Рекомендовані
Операційна система	Windows Server 2019 Standard	Windows Server 2022 Datacenter
СУБД	Microsoft SQL Server 2022 Standard Edition	Microsoft SQL Server 2022 Enterprise Edition
Додаткове ПЗ	-	Система моніторингу серверів

Для функціонування додатка необхідні встановлений та налаштований MS SQL Server для зберігання даних, а також створена база даних з необхідними таблицями та зв'язками.

Наступним етапом є визначення вимог до системи з точки зору кінцевого користувача, що дозволяє врахувати зручність взаємодії, швидкість доступу до функцій, а також технічні умови для належного функціонування клієнтської частини застосунку.

Апаратні вимоги для персонального комп’ютера користувача включають мінімальну конфігурацію обладнання, необхідну для стабільної роботи інтерфейсу, обробки запитів та взаємодії з базою даних. Ці характеристики детально наведено в таблиці 4.3.

Таблиця 4.3

Апаратні вимоги

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Процесор	2 ядра, 1.8 GHz	4 ядра, 2.4+ GHz
Оперативна пам'ять	4 GB RAM	8+ GB RAM
Жорсткий диск	10 GB вільного місця (HDD)	20+ GB вільного місця (SSD)
Відеокарта	З підтримкою DirectX 11, 1 GB відеопам'яті	З підтримкою DirectX 12, 2+ GB відеопам'яті
Мережа	100 Mbps Ethernet	Gigabit Ethernet

Програмні вимоги для персонального комп’ютера користувача включають перелік необхідного програмного забезпечення, яке забезпечує запуск та повноцінне використання клієнтської частини системи, зокрема операційну систему, середовище виконання, підтримувані бібліотеки та допоміжні утиліти. Детальний перелік програмних компонентів і їх версій наведено в таблиці 4.4.

Таблиця 4.4

Програмні вимоги

Компонент	Мінімальні вимоги	Рекомендовані вимоги
Операційна система	Windows 10 версія 21H2 або новіша	Windows 11 22H2 або новіша
Системні компоненти	DirectX 11	DirectX 12

Для початку роботи користувачеві необхідно встановити WPF-додаток, який є основним інтерфейсом взаємодії з інформаційною системою. Встановлення передбачає копіювання виконуваних файлів на робочу станцію та, за потреби, попередню інсталяцію необхідних компонентів середовища виконання .NET, що забезпечує коректну роботу всіх елементів застосунку.

Таким чином, ретельне визначення апаратних і програмних вимог як на стороні сервера, так і для кінцевого користувача є необхідною умовою для впровадження ефективної та стабільної системи. Забезпечення відповідності цим вимогам дозволяє уникнути проблем сумісності, забезпечити швидкодію та гарантовану підтримку всіх функціональних можливостей програмного продукту в реальних умовах експлуатації.

4.3 Склад інсталяційного пакету

Для програмного забезпечення автоматизованого робочого місця для управління складом станції технічного обслуговування інсталяційний пакет склається лише з папки самого додатка. На рис. 4.16 показаний вміст папки додатка.

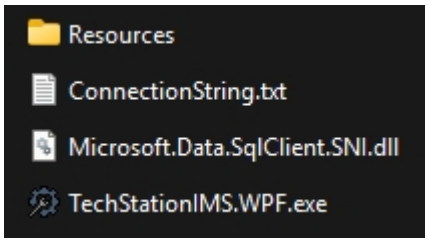

Рис.4.16 Вміст істаляційного пакету

Перед запуском програми необхідно вказати параметри підключення до бази даних у файлі ConnectingString.txt. У ньому слід записати рядок підключення з адресою сервера, назвою бази та обліковими даними. Програма зчитує ці дані під час запуску, тому їх коректність є критично важливою для стабільної роботи системи.

Таким чином, склад інсталяційного пакету є мінімальним, однак достатнім для автономної роботи застосунку на будь-якому клієнтському ПК, що відповідає зазначеним технічним вимогам. Завдяки простій процедурі розгортання система може бути оперативно встановлена та налаштована без додаткового програмного забезпечення або складних інсталяційних сценаріїв.

ВИСНОВКИ

У результаті виконання дипломного проєкту було проведено комплексне дослідження предметної області управління складом станції технічного обслуговування (СТО) та реалізовано основні етапи розробки прикладного програмного забезпечення для АРМ для управління складом станції технічного обслуговування. Основна увага приділялася моделюванню, проектуванню та реалізації інформаційної системи.

Було виконано системний аналіз діяльності складу СТО, в результаті чого визначено ключові бізнес-процеси, користувачі та їх взаємозв’язки. Для визначення вимог було побудовано діаграми прецедентів, активності, послідовності та діаграму класів для отримання цілісного уявлення про логіку функціонування майбутньої інформаційної системи.

Здійснено огляд систем-аналогів для управління складом (Tocan WMS, LSC WMS) з метою виявлення їхніх переваг та недоліків. На основі цього огляду сформульовано вимоги до створюваної системи та обґрунтовано доцільність її впровадження саме для невеликого СТО.

Було спроєктовано логічну та фізичну моделі бази даних. Для реалізації бази даних було обрано СУБД Microsoft SQL Server для забезпечення надійного збереження даних.

Обґрунтовано вибір архітектури програмного забезпечення. Застосовано трирівневу модель із розділенням на презентаційний шар (інтерфейс), шар бізнес-логіки та шар доступу до даних. Реалізація інтерфейсу здійснюється з використанням технології Windows Presentation Foundation (WPF) та архітектурного патерну MVVM для забезпечення модульності, гнучкості та зручність у подальшому супроводі системи.

У рамках розробки програмної частини було реалізовано окремі програмні модулі, зокрема: додавання нових запчастин до бази, розподіл запчастин по комірках, автоматичне переміщення деталей із партій у доступні зони складу,

реєстрація запитів від автомеханіків тощо. Особливу увагу приділено обробці помилок та перевірці коректності введених даних.

Отже, у ході реалізації дипломного проєкту було розроблено структуру та функціональні компоненти програмного забезпечення АРМ для управління складом станції технічного обслуговування. Розроблена система має потенціал для подальшого розвитку, масштабування та впровадження в умовах реального підприємства.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Марков О. Д. Організація автосервісу : навч. посіб. Львів: Оріяна-Нова, 2018. 332 с.
2. Канарчук В. Є., Лудченко О. А. Основи технічного обслуговування і ремонту автомобілів: підручник. Київ : Вища школа, 2019. 384 с.
3. Лудченко О. А. Технічне обслуговування і ремонт автомобілів: організація і управління : навч. посіб. Київ: Знання, 2017. 478 с.
4. Волгін В. В. Автосервіс. Створення і компоненти. Київ: Либідь, 2020. 408 с.
5. Фастовцев Г. Ф. Організація технічного обслуговування і ремонту автомобілів. Москва: Машинобудування, 2018. 256 с.
6. Управління автосервісом: навч. посіб. / за ред. Л. Б. Миротіна. Київ: Екзамен, 2021. 464 с.
7. Tocan WMS – система управління складом. URL: https://tocan.com.ua/uk/sistema-upravleniya-skladom-tocan-wms/ (дата звернення: 12.05.2025).
8. LSC WMS – система управління складською логістикою. URL: https://lscenter.com.ua/ua/logistics-optimization/lsc-wms (дата звернення: 12.05.2025).
9. Модель «сутність – зв'язок». Основні поняття моделі «сутність – зв'язок»: сутності, зв'язки, атрибути та їх класифікація. URL: посилання (дата звернення: 12.05.2025).
10. Діаграма класів. Вікіпедія – Вільна енциклопедія. URL: [Електронний ресурс] – Режим доступу: посилання (дата звернення: 12.05.2025).
11. Що таке діаграма класів UML і найкращий творець діаграм класів UML. URL: https://www.mindonmap.com/uk/blog/what-is-uml-class-diagram/ (дата звернення: 12.05.2025).
12. Повне розуміння діаграми компонентів UML за допомогою легкого методу. URL: https://www.mindonmap.com/uk/blog/uml-component-diagram/ (дата звернення: 14.05.2025).

13. XAML overview. URL: https://learn.microsoft.com/uk-ua/dotnet/desktop/wpf/xaml/ (дата звернення: 14.05.2025).
14. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань. Книга 2. Системи управління базами даних та знань. Львів : Магнолія-2006, 2018. 584 с.
15. Пасічник В. В., Резніченко В. А. Організація баз даних та знань. Київ : Видавнича група BHV, 2016. 384 с.
16. Що таке SQL і де його використовують: веб-сайт. URL: https://goit.global/ua/articles/shcho-take-sql-i-de-yoho-vykorystovuiut/ (дата звернення: 14.05.2025).
17. .NET. Вікіпедія – Вільна енциклопедія. URL: https://uk.wikipedia.org/wiki/.NET (дата звернення: 14.05.2025).
18. Завантаження пакета DirectX і його інсталяція: веб-сайт. URL: посилання (дата звернення: 14.05.2025).
19. SQL Server Database Engine. Microsoft SQL Server Documentation: веб-сайт. URL: https://learn.microsoft.com/uk-ua/sql/database-engine/sql-server-database-engine-overview (дата звернення: 14.05.2025).
20. Data Binding Overview. Microsoft Learn: веб-сайт. URL: https://learn.microsoft.com/uk-ua/dotnet/desktop/wpf/data/?view=netdesktop-8.0 (дата звернення: 14.05.2025).
21. ADO.NET Overview. Microsoft Learn: веб-сайт. URL: https://learn.microsoft.com/uk-ua/dotnet/framework/data/adonet/ado-net-overview (дата звернення: 14.05.2025).
22. Огляд архітектури клієнт-сервер. Microsoft Learn : веб-сайт. URL: https://learn.microsoft.com/uk-ua/dotnet/architecture/modern-web-apps-azure/common-client-side-web-technologies (дата звернення: 14.05.2025).
23. Smith J. Patterns: WPF Apps With the Model-View-ViewModel Design Pattern. MSDN Magazine. 2009. URL: https://docs.microsoft.com/uk-ua/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern (дата звернення: 14.05.2025).

24. SQL Server technical documentation / Microsoft Corporation. 2023. URL: https://docs.microsoft.com/uk-ua/sql/ (дата звернення: 14.05.2025).
25. Windows Presentation Foundation documentation / Microsoft Corporation. 2023. URL: https://docs.microsoft.com/uk-ua/dotnet/desktop/wpf/ (дата звернення: 14.05.2025).
26. C# documentation / Microsoft Corporation. 2023. URL: https://docs.microsoft.com/uk-ua/dotnet/csharp/ (дата звернення: 14.05.2025).
27. Тестування програмного забезпечення. Foxminded: веб-сайт. URL: https://foxminded.ua/testuvannia-prohramnoho-zabezpechennia/ (дата звернення: 15.05.2025).

ДОДАТОК А

Сторінок – 3

Опис таблиць бази даних додатку

```
-- Таблиця брендів автомобілів
CREATE TABLE Car_brand
(
    Brand_code CHAR(8) NOT NULL PRIMARY KEY,
    Brand_name VARCHAR(255) NOT NULL,
    Country VARCHAR(100) NOT NULL,
    Website VARCHAR(255) NOT NULL
);

-- Таблиця постачальників
CREATE TABLE Supplier
(
    Supplier_code CHAR(8) NOT NULL PRIMARY KEY,
    Supplier_name VARCHAR(255) NOT NULL,
    Supplier_phone_number VARCHAR(13) NOT NULL UNIQUE
);

-- Таблиця накладних
CREATE TABLE Invoice
(
    Invoice_code CHAR(8) NOT NULL PRIMARY KEY,
    Delivery_date DATE NOT NULL,
    Supplier_code_FK CHAR(8) NOT NULL,
    FOREIGN KEY (Supplier_code_FK) REFERENCES Supplier(Supplier_code)
);

-- Таблиця зон складу
CREATE TABLE Storage_zone
(
    Zone_code CHAR(8) NOT NULL PRIMARY KEY,
    Zone_name VARCHAR(255) NOT NULL
);

-- Таблиця комірок у кожній зоні
CREATE TABLE Storage_cell
(
    Zone_code_FK CHAR(8) NOT NULL,
    Cell_code VARCHAR(8) NOT NULL,
    Max_capacity INT NOT NULL,  -- Максимальна місткість комірки
    Current_capacity INT NOT NULL DEFAULT 0, -- Поточний рівень заповнення
    PRIMARY KEY (Zone_code_FK, Cell_code),
    FOREIGN KEY (Zone_code_FK) REFERENCES Storage_zone(Zone_code)
);

-- Таблиця категорій запчастин
CREATE TABLE Part_category
(
    Category_code CHAR(8) NOT NULL PRIMARY KEY,
    Category_name VARCHAR(255) NOT NULL,
    Zone_code_FK CHAR(8) NOT NULL, -- Основна зона зберігання для категорії
    FOREIGN KEY (Zone_code_FK) REFERENCES Storage_zone(Zone_code)
);

-- Таблиця виробників запчастин
CREATE TABLE Part_manufacturer
(
    Manufacturer_code CHAR(8) NOT NULL PRIMARY KEY,
    Manufacturer_name VARCHAR(255) NOT NULL,
    Country VARCHAR(100) NOT NULL,
    Website VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL
);


-- Таблиця брендів для запчастини
```

```
CREATE TABLE Part_brand
(
    Part_code_FK CHAR(8) NOT NULL,
    Brand_code_FK CHAR(8) NOT NULL,
    PRIMARY KEY (Part_code_FK, Brand_code_FK),
    FOREIGN KEY (Part_code_FK) REFERENCES Part(Part_code),
    FOREIGN KEY (Brand_code_FK) REFERENCES Car_brand(Brand_code)
);

-- Таблиця зображень запчастин
CREATE TABLE Part_image
(
    Image_code CHAR(8) PRIMARY KEY,
    Part_code_FK CHAR(8) NOT NULL,
    Image_url VARCHAR(500) NULL,
    FOREIGN KEY (Part_code_FK) REFERENCES Part(Part_code) ON DELETE CASCADE
);

-- Таблиця рядків у накладній
CREATE TABLE Invoice_line
(
    Line_number SMALLINT NOT NULL,
    Invoice_code_FK CHAR(8) NOT NULL,
    Part_code_FK CHAR(8) NOT NULL,
    Arrival_quantity INT NOT NULL,
    Purchase_price DECIMAL(10, 2) NOT NULL,
    PRIMARY KEY (Invoice_code_FK, Line_number),
    FOREIGN KEY (Invoice_code_FK) REFERENCES Invoice(Invoice_code),
    FOREIGN KEY (Part_code_FK) REFERENCES Part(Part_code)
);

-- Таблиця партій запчастин
CREATE TABLE Part_batch
(
    Batch_code CHAR(8) NOT NULL PRIMARY KEY,
    Part_code_FK CHAR(8) NOT NULL,
    Invoice_code_FK CHAR(8) NOT NULL,
    Line_number_FK SMALLINT NOT NULL,
       Current_batch_quantity INT NOT NULL,
       Unallocated_batch_quantity INT NOT NULL,
    FOREIGN KEY (Part_code_FK) REFERENCES Part(Part_code),
    FOREIGN KEY (Invoice_code_FK, Line_number_FK) REFERENCES Invoice_line(Invoice_code_FK,
Line_number)
);

-- Таблиця кількості запчастин у комірках з конкретної партії
CREATE TABLE Stock_balance
(
    Part_code_FK CHAR(8) NOT NULL,
    Batch_code_FK CHAR(8) NOT NULL,
    Zone_code_FK CHAR(8) NOT NULL,
    Cell_code_FK VARCHAR(8) NOT NULL,
    Batch_cell_quantity INT NOT NULL,
    PRIMARY KEY (Part_code_FK, Batch_code_FK, Zone_code_FK, Cell_code_FK),
    FOREIGN KEY (Part_code_FK) REFERENCES Part(Part_code),
    FOREIGN KEY (Batch_code_FK) REFERENCES Part_batch(Batch_code),
    FOREIGN KEY (Zone_code_FK, Cell_code_FK) REFERENCES Storage_cell(Zone_code_FK,
Cell_code)
);

-- Таблиця логування переміщень запчастин
CREATE TABLE Part_movement
(
    Movement_code VARCHAR(36) PRIMARY KEY,
    Part_code_FK CHAR(8) NOT NULL,
```

```
    Batch_code_FK CHAR(8) NOT NULL,
    From_zone_code_FK CHAR(8) NULL,
    From_cell_code_FK VARCHAR(8) NULL,
    To_zone_code_FK CHAR(8) NULL,
    To_cell_code_FK VARCHAR(8) NULL,
    Movement_date DATETIME NOT NULL DEFAULT GETDATE(),
    Quantity INT NOT NULL,
    Movement_type VARCHAR(50) NOT NULL,
    FOREIGN KEY (Part_code_FK) REFERENCES Part(Part_code),
    FOREIGN KEY (Batch_code_FK) REFERENCES Part_batch(Batch_code),
    FOREIGN KEY (From_zone_code_FK, From_cell_code_FK) REFERENCES Storage_cell(Zone_code_FK,
Cell_code),
    FOREIGN KEY (To_zone_code_FK, To_cell_code_FK) REFERENCES Storage_cell(Zone_code_FK,
Cell_code)
);

-- Таблиця резервування запчастин
CREATE TABLE Part_reservation
(
    Reservation_code CHAR(12) PRIMARY KEY,
    Mechanic_name VARCHAR(40) NOT NULL,
    Part_code_FK CHAR(8) NOT NULL,
    Batch_code_FK CHAR(8) NOT NULL,
    Zone_code_FK CHAR(8) NOT NULL,
    Cell_code_FK VARCHAR(8) NOT NULL,
    Reserved_quantity INT NOT NULL,
    Reservation_date DATETIME NOT NULL DEFAULT GETDATE(),
    Issue_date DATETIME NULL,
    Status_ VARCHAR(20) NOT NULL,
    FOREIGN KEY (Zone_code_FK, Cell_code_FK) REFERENCES Storage_cell(Zone_code_FK,
Cell_code),
    FOREIGN KEY (Part_code_FK) REFERENCES Part(Part_code),
    FOREIGN KEY (Batch_code_FK) REFERENCES Part_batch(Batch_code)
);

-- Таблиця користувачів
CREATE TABLE Users
(
    User_code CHAR(8) NOT NULL PRIMARY KEY,
    Login_ VARCHAR(255) NOT NULL UNIQUE,
    Password_ VARBINARY(64) NOT NULL
);
```

ДОДАТОК Б

Сторінок – 8

Опис збережених процедур бази даних додатку

```
-- Процедура розподілу запчастин в комірку
CREATE PROCEDURE DistributePartsToCell
    @Batch_code CHAR(8),
    @Zone_code CHAR(8),
    @Cell_code VARCHAR(8),
    @Quantity INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Перевіряємо, чи достатньо не розподілених запчастин
    IF (SELECT Unallocated_batch_quantity FROM Part_batch WHERE Batch_code = @Batch_code) <
@Quantity
    BEGIN
        RAISERROR ('Недостатньо не розподілених запчастин у партії.', 16, 1);
        RETURN;
    END;

    -- Перевіряємо, чи є вільне місце в комірці
    IF (SELECT Max_capacity - Current_capacity FROM Storage_cell WHERE Zone_code_FK =
@Zone_code AND Cell_code = @Cell_code) < @Quantity
    BEGIN
        RAISERROR ('Недостатньо місця у комірці.', 16, 1);
        RETURN;
    END;

    BEGIN TRANSACTION;

    -- Додаємо запис у Stock_balance або оновлюємо існуючий
    MERGE INTO Stock_balance AS sb
    USING (SELECT @Batch_code AS Batch_code_FK, @Zone_code AS Zone_code_FK, @Cell_code AS
Cell_code_FK, @Quantity AS Batch_cell_quantity) AS src
    ON sb.Batch_code_FK = src.Batch_code_FK AND sb.Zone_code_FK = src.Zone_code_FK AND
sb.Cell_code_FK = src.Cell_code_FK
    WHEN MATCHED THEN
        UPDATE SET Batch_cell_quantity = sb.Batch_cell_quantity + src.Batch_cell_quantity
    WHEN NOT MATCHED THEN
        INSERT (Batch_code_FK, Part_code_FK, Zone_code_FK, Cell_code_FK,
Batch_cell_quantity)
        VALUES (@Batch_code, (SELECT Part_code_FK FROM Part_batch WHERE Batch_code =
@Batch_code), @Zone_code, @Cell_code, @Quantity);

    -- Оновлюємо кількість у Part_batch
    UPDATE Part_batch
    SET
        Current_batch_quantity = Current_batch_quantity + @Quantity,
        Unallocated_batch_quantity = Unallocated_batch_quantity - @Quantity
    WHERE Batch_code = @Batch_code;

    -- Оновлюємо поточну місткість комірки
    UPDATE Storage_cell
    SET Current_capacity = Current_capacity + @Quantity
    WHERE Zone_code_FK = @Zone_code AND Cell_code = @Cell_code;

    -- Лог переміщення запчастин
       INSERT INTO Part_movement (
              Movement_code, Part_code_FK, Batch_code_FK,
              From_zone_code_FK, From_cell_code_FK,
              To_zone_code_FK, To_cell_code_FK,
              Movement_date, Quantity, Movement_type)
       VALUES (
              NEWID(),
              (SELECT Part_code_FK FROM Part_batch WHERE Batch_code = @Batch_code),
              @Batch_code,
              NULL,
```

```
            NULL,
            @Zone_code,
            @Cell_code,
            GETDATE(),
            @Quantity,
            'Розподіл на зберігання'
       );

    COMMIT TRANSACTION;
       RETURN 1;
END;

GO
-- Процедура переміщення запчастиин з одної комірки в іншу
CREATE PROCEDURE MovePartsBetweenCells
    @Batch_code CHAR(8),
    @From_Zone CHAR(8),
       @From_Cell VARCHAR(8),
    @To_Zone CHAR(8),
    @To_Cell VARCHAR(8),
    @Quantity INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Перевірка, чи є достатня кількість запчастин у комірці
    IF NOT EXISTS (
        SELECT 1 FROM Stock_balance
        WHERE Batch_code_FK = @Batch_code
          AND Zone_code_FK = @From_Zone
          AND Cell_code_FK = @From_Cell
          AND Batch_cell_quantity >= @Quantity
    )
    BEGIN
        RAISERROR ('Недостатньо запчастин у вихідній комірці.', 16, 1);
        RETURN;
    END;

    BEGIN TRANSACTION;

    -- Віднімаємо кількість запчастин із поточної комірки
    UPDATE Stock_balance
    SET Batch_cell_quantity = Batch_cell_quantity - @Quantity
    WHERE Batch_code_FK = @Batch_code
      AND Zone_code_FK = @From_Zone
      AND Cell_code_FK = @From_Cell;

    -- Видаляємо запис, якщо кількість стала 0
    DELETE FROM Stock_balance
    WHERE Batch_code_FK = @Batch_code
      AND Zone_code_FK = @From_Zone
      AND Cell_code_FK = @From_Cell
      AND Batch_cell_quantity = 0;

       -- Оновлюємо поточну місткість вихідної комірки
    UPDATE Storage_cell
    SET Current_capacity = Current_capacity - @Quantity
    WHERE Zone_code_FK = @From_Zone AND Cell_code = @From_Cell;

    -- Додаємо кількість у нову комірку (оновлення або вставка)
    MERGE INTO Stock_balance AS sb
    USING (SELECT @Batch_code AS Batch_code_FK, @To_Zone AS Zone_code_FK, @To_Cell AS
Cell_code_FK, @Quantity AS Batch_cell_quantity) AS src
    ON sb.Batch_code_FK = src.Batch_code_FK AND sb.Zone_code_FK = src.Zone_code_FK AND
sb.Cell_code_FK = src.Cell_code_FK
```

```
    WHEN MATCHED THEN
        UPDATE SET Batch_cell_quantity = sb.Batch_cell_quantity + src.Batch_cell_quantity
    WHEN NOT MATCHED THEN
        INSERT (Batch_code_FK, Part_code_FK, Zone_code_FK, Cell_code_FK,
Batch_cell_quantity)
        VALUES (@Batch_code, (SELECT Part_code_FK FROM Part_batch WHERE Batch_code =
@Batch_code), @To_Zone, @To_Cell, @Quantity);

       -- Оновлюємо поточну місткість цільової комірки
    UPDATE Storage_cell
    SET Current_capacity = Current_capacity + @Quantity
    WHERE Zone_code_FK = @To_Zone AND Cell_code = @To_Cell;

    -- Лог переміщення запчастин
    INSERT INTO Part_movement (Movement_code, Part_code_FK, Batch_code_FK,
From_zone_code_FK, From_cell_code_FK, To_zone_code_FK, To_cell_code_FK, Movement_date,
Quantity, Movement_type)
    VALUES (
        NEWID(),
        (SELECT Part_code_FK FROM Part_batch WHERE Batch_code = @Batch_code),
        @Batch_code,
        @From_Zone,
        @From_Cell,
        @To_Zone,
        @To_Cell,
        GETDATE(),
        @Quantity,
        'Переміщення між комірками'
    );

    COMMIT TRANSACTION;
END;
       GO

-- Процедура переміщення запчастиин для резервування
CREATE PROCEDURE MovePartsBetweenCellsForReservation
    @Batch_code CHAR(8),
    @From_Zone CHAR(8),
       @From_Cell VARCHAR(8),
    @To_Zone CHAR(8),
    @To_Cell VARCHAR(8),
    @Quantity INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Перевірка, чи є достатня кількість запчастин у комірці
    IF NOT EXISTS (
        SELECT 1 FROM Stock_balance
        WHERE Batch_code_FK = @Batch_code
          AND Zone_code_FK = @From_Zone
          AND Cell_code_FK = @From_Cell
          AND Batch_cell_quantity >= @Quantity
    )
    BEGIN
        RAISERROR ('Недостатньо запчастин у вихідній комірці.', 16, 1);
        RETURN;
    END;

    BEGIN TRANSACTION;

    -- Віднімаємо кількість запчастин із поточної комірки
    UPDATE Stock_balance
    SET Batch_cell_quantity = Batch_cell_quantity - @Quantity
    WHERE Batch_code_FK = @Batch_code
      AND Zone_code_FK = @From_Zone
```

```
      AND Cell_code_FK = @From_Cell;

    -- Видаляємо запис, якщо кількість стала 0
    DELETE FROM Stock_balance
    WHERE Batch_code_FK = @Batch_code
      AND Zone_code_FK = @From_Zone
      AND Cell_code_FK = @From_Cell
      AND Batch_cell_quantity = 0;

       -- Оновлюємо поточну місткість вихідної комірки
    UPDATE Storage_cell
    SET Current_capacity = Current_capacity - @Quantity
    WHERE Zone_code_FK = @From_Zone AND Cell_code = @From_Cell;

    -- Додаємо кількість у нову комірку (оновлення або вставка)
    MERGE INTO Stock_balance AS sb
    USING (SELECT @Batch_code AS Batch_code_FK, @To_Zone AS Zone_code_FK, @To_Cell AS
Cell_code_FK, @Quantity AS Batch_cell_quantity) AS src
    ON sb.Batch_code_FK = src.Batch_code_FK AND sb.Zone_code_FK = src.Zone_code_FK AND
sb.Cell_code_FK = src.Cell_code_FK
    WHEN MATCHED THEN
        UPDATE SET Batch_cell_quantity = sb.Batch_cell_quantity + src.Batch_cell_quantity
    WHEN NOT MATCHED THEN
        INSERT (Batch_code_FK, Part_code_FK, Zone_code_FK, Cell_code_FK,
Batch_cell_quantity)
        VALUES (@Batch_code, (SELECT Part_code_FK FROM Part_batch WHERE Batch_code =
@Batch_code), @To_Zone, @To_Cell, @Quantity);

       -- Оновлюємо поточну місткість цільової комірки
    UPDATE Storage_cell
    SET Current_capacity = Current_capacity + @Quantity
    WHERE Zone_code_FK = @To_Zone AND Cell_code = @To_Cell;

    -- Лог переміщення запчастин
    INSERT INTO Part_movement (Movement_code, Part_code_FK, Batch_code_FK,
From_zone_code_FK, From_cell_code_FK, To_zone_code_FK, To_cell_code_FK, Movement_date,
Quantity, Movement_type)
    VALUES (
        NEWID(),
        (SELECT Part_code_FK FROM Part_batch WHERE Batch_code = @Batch_code),
        @Batch_code,
        @From_Zone,
        @From_Cell,
        @To_Zone,
        @To_Cell,
        GETDATE(),
        @Quantity,
        'Резервування'
    );

    COMMIT TRANSACTION;
END;
GO
-- Процедура переміщення запчастин для скасування резервування
CREATE PROCEDURE MovePartsBetweenCellsForCancelReservation
    @Batch_code CHAR(8),
    @From_Zone CHAR(8),
       @From_Cell VARCHAR(8),
    @To_Zone CHAR(8),
    @To_Cell VARCHAR(8),
    @Quantity INT
AS
BEGIN
    SET NOCOUNT ON;
```

```
    -- Перевірка, чи є достатня кількість запчастин у комірці
    IF NOT EXISTS (
        SELECT 1 FROM Stock_balance
        WHERE Batch_code_FK = @Batch_code
          AND Zone_code_FK = @From_Zone
          AND Cell_code_FK = @From_Cell
          AND Batch_cell_quantity >= @Quantity
    )
    BEGIN
        RAISERROR ('Недостатньо запчастин у вихідній комірці.', 16, 1);
        RETURN;
    END;

    BEGIN TRANSACTION;

    -- Віднімаємо кількість запчастин із поточної комірки
    UPDATE Stock_balance
    SET Batch_cell_quantity = Batch_cell_quantity - @Quantity
    WHERE Batch_code_FK = @Batch_code
      AND Zone_code_FK = @From_Zone
      AND Cell_code_FK = @From_Cell;

    -- Видаляємо запис, якщо кількість стала 0
    DELETE FROM Stock_balance
    WHERE Batch_code_FK = @Batch_code
      AND Zone_code_FK = @From_Zone
      AND Cell_code_FK = @From_Cell
      AND Batch_cell_quantity = 0;

       -- Оновлюємо поточну місткість вихідної комірки
    UPDATE Storage_cell
    SET Current_capacity = Current_capacity - @Quantity
    WHERE Zone_code_FK = @From_Zone AND Cell_code = @From_Cell;

    -- Додаємо кількість у нову комірку (оновлення або вставка)
    MERGE INTO Stock_balance AS sb
    USING (SELECT @Batch_code AS Batch_code_FK, @To_Zone AS Zone_code_FK, @To_Cell AS
Cell_code_FK, @Quantity AS Batch_cell_quantity) AS src
    ON sb.Batch_code_FK = src.Batch_code_FK AND sb.Zone_code_FK = src.Zone_code_FK AND
sb.Cell_code_FK = src.Cell_code_FK
    WHEN MATCHED THEN
        UPDATE SET Batch_cell_quantity = sb.Batch_cell_quantity + src.Batch_cell_quantity
    WHEN NOT MATCHED THEN
        INSERT (Batch_code_FK, Part_code_FK, Zone_code_FK, Cell_code_FK,
Batch_cell_quantity)
        VALUES (@Batch_code, (SELECT Part_code_FK FROM Part_batch WHERE Batch_code =
@Batch_code), @To_Zone, @To_Cell, @Quantity);

       -- Оновлюємо поточну місткість цільової комірки
    UPDATE Storage_cell
    SET Current_capacity = Current_capacity + @Quantity
    WHERE Zone_code_FK = @To_Zone AND Cell_code = @To_Cell;

    -- Лог переміщення запчастин
    INSERT INTO Part_movement (Movement_code, Part_code_FK, Batch_code_FK,
From_zone_code_FK, From_cell_code_FK, To_zone_code_FK, To_cell_code_FK, Movement_date,
Quantity, Movement_type)
    VALUES (
        NEWID(),
        (SELECT Part_code_FK FROM Part_batch WHERE Batch_code = @Batch_code),
        @Batch_code,
        @From_Zone,
        @From_Cell,
        @To_Zone,
        @To_Cell,
```

```
        GETDATE(),
        @Quantity,
        'Скасування резервування'
    );

    COMMIT TRANSACTION;
END;
      GO
-- "ПРОЦЕС ВИДАЧІ ЗАПЧАСТИН"
-- Процедура створення резервації
CREATE PROCEDURE CreateReservation
    @Reservation_code CHAR(12),
    @Mechanic_name VARCHAR(40),
    @Part_code CHAR(8),
    @Quantity INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Batch_code CHAR(8);
    DECLARE @Zone_code CHAR(8);
    DECLARE @Cell_code VARCHAR(8);
    DECLARE @AvailableQuantity INT;

    -- Вибір найбільш підходящої комірки (з найбільшим запасом, не у зоні видачі)
    SELECT TOP 1
        @Batch_code = Batch_code_FK,
        @Zone_code = Zone_code_FK,
        @Cell_code = Cell_code_FK,
        @AvailableQuantity = Batch_cell_quantity
    FROM Stock_balance
    WHERE Part_code_FK = @Part_code
      AND Zone_code_FK <> 'ZONE0004'
      AND Batch_cell_quantity >= @Quantity
    ORDER BY Batch_cell_quantity DESC; -- Обираємо комірку з найбільшим запасом

    -- Перевірка наявності достатньої кількості
    IF @Batch_code IS NULL
    BEGIN
        RAISERROR ('Недостатньо запчастин для резервування.', 16, 1);
        RETURN;
    END;

    BEGIN TRANSACTION;

    -- Додаємо запис у Part_reservation
    INSERT INTO Part_reservation (Reservation_code, Mechanic_name, Part_code_FK,
Batch_code_FK, Zone_code_FK, Cell_code_FK, Reserved_quantity, Reservation_date, Issue_date,
Status_)
    VALUES (@Reservation_code, @Mechanic_name, @Part_code, @Batch_code, @Zone_code,
@Cell_code, @Quantity, GETDATE(), NULL, 'Зарезервовано');

    -- Переміщення запчастин у зону видачі
    EXEC MovePartsBetweenCellsForReservation @Batch_code, @Zone_code, @Cell_code,
'ZONE0004', 'CELL-022', @Quantity;

    COMMIT TRANSACTION;
END;
GO
-- Далі оператор вручну змінює статус резерву на "Очікує видачі"

-- Процедура видачі зарезервованих запчастин
CREATE PROCEDURE IssueReservedParts
    @Reservation_code CHAR(12)
AS
```

```
BEGIN
    SET NOCOUNT ON;

    -- Перевіряємо, чи існує така резервація зі статусом "Зарезервовано"
    IF NOT EXISTS (
        SELECT 1 FROM Part_reservation WHERE Reservation_code = @Reservation_code AND
Status_ = 'Очікує видачі'
    )
    BEGIN
        RAISERROR ('Резервація не знайдена або вже видана.', 16, 1);
        RETURN;
    END;

    BEGIN TRANSACTION;

    -- Видаляємо запчастини з зони видачі
    DECLARE @Batch_code CHAR(8), @Part_code CHAR(8), @Quantity INT;

    SELECT @Batch_code = Batch_code_FK, @Part_code = Part_code_FK, @Quantity =
Reserved_quantity
    FROM Part_reservation WHERE Reservation_code = @Reservation_code;

    DELETE FROM Stock_balance
    WHERE Batch_code_FK = @Batch_code AND Zone_code_FK = 'ZONE0004' AND Cell_code_FK =
'CELL-022';

       -- Оновлюємо поточну місткість комірки
    UPDATE Storage_cell
    SET Current_capacity = Current_capacity - @Quantity
    WHERE Zone_code_FK = 'ZONE0004' AND Cell_code = 'CELL-022';

    -- Оновлюємо статус резервації
    UPDATE Part_reservation
    SET Status_ = 'Видано', Issue_date = GETDATE()
    WHERE Reservation_code = @Reservation_code;

    -- Лог видачі
    INSERT INTO Part_movement (Movement_code, Part_code_FK, Batch_code_FK,
From_zone_code_FK, From_cell_code_FK, To_zone_code_FK, To_cell_code_FK, Movement_date,
Quantity, Movement_type)
    VALUES (NEWID(), @Part_code, @Batch_code, 'ZONE0004', 'CELL-022', NULL, NULL, GETDATE(),
@Quantity, 'Видача механіку');

    COMMIT TRANSACTION;
END;
GO
-- Процедура скасування резервації
CREATE PROCEDURE CancelReservation
    @Reservation_code CHAR(12)
AS
BEGIN
    SET NOCOUNT ON;

    -- Перевіряємо, чи існує така резервація зі статусом "Зарезервовано"
    IF NOT EXISTS (
        SELECT 1 FROM Part_reservation WHERE Reservation_code = @Reservation_code AND
(Status_ = 'Зарезервовано' OR Status_ = 'Очікує видачі')
    )
    BEGIN
        RAISERROR ('Резервація не знайдена або вже оброблена.', 16, 1);
        RETURN;
    END;

    BEGIN TRANSACTION;
```

```
    -- Отримуємо інформацію про резервацію
    DECLARE @Batch_code CHAR(8), @Part_code CHAR(8), @Zone_code CHAR(8), @Cell_code
VARCHAR(8), @Quantity INT;

    SELECT @Batch_code = Batch_code_FK, @Part_code = Part_code_FK, @Zone_code =
Zone_code_FK, @Cell_code = Cell_code_FK, @Quantity = Reserved_quantity
    FROM Part_reservation WHERE Reservation_code = @Reservation_code;

    -- Повертаємо запчастини в початкову комірку
    EXEC MovePartsBetweenCellsForCancelReservation @Batch_code, 'ZONE0004', 'CELL-022',
@Zone_code, @Cell_code, @Quantity;

    -- Оновлюємо статус резервації
    UPDATE Part_reservation
    SET Status_ = 'Скасовано'
    WHERE Reservation_code = @Reservation_code;

    COMMIT TRANSACTION;
END;
```

ДОДАТОК В

Сторінок – 10

Код програмних рішень додатку

Додавання запчастини

На фронт частині

```
 <!-- Text box form -->
<StackPanel Grid.Column="1"
            Orientation="Vertical">

    <Border Grid.Column="1"
        Style="{StaticResource BorderTextBoxFormStyle}">

        <TextBox Style="{StaticResource TextBoxFormStyle}"
                 Text="{Binding SKUCode}"
                 MaxLength="12"/>

    </Border>

    <Border Grid.Column="1"
        Style="{StaticResource BorderTextBoxFormStyle}">

        <TextBox Style="{StaticResource TextBoxFormStyle}"
             Text="{Binding PartName}"/>

    </Border>

    <Border Grid.Column="1" Style="{StaticResource BorderTextBoxFormStyle}">
        <ComboBox ItemsSource="{Binding PartCategories}"
                  SelectedValue="{Binding SelectedCategory}"
                  Text="{Binding SelectedCategory}">

            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Category_name}" />
                </DataTemplate>
            </ComboBox.ItemTemplate>
        </ComboBox>
    </Border>

    <Border Grid.Column="1"
        Style="{StaticResource BorderTextBoxFormStyle}">

        <TextBox Style="{StaticResource TextBoxFormStyle}"
             Text="{Binding PartDescription}"/>

    </Border>

    <Border Grid.Column="1"
        Style="{StaticResource BorderTextBoxFormStyle}">

        <TextBox Style="{StaticResource TextBoxFormStyle}"
                 Text="{Binding PartBrands}"
                 IsReadOnly="True"/>

    </Border>

    <StackPanel Orientation="Horizontal">
        <Border Grid.Column="1" Style="{StaticResource BorderTextBoxFormStyle}"
            Width="190">

            <ComboBox ItemsSource="{Binding CarBrands}"
                      SelectedValue="{Binding SelectedBrand}"
                      Text="{Binding SelectedBrand}">

                <ComboBox.ItemTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding Brand_name}" />
                    </DataTemplate>
                </ComboBox.ItemTemplate>
            </ComboBox>
        </Border>

        <Button Style="{StaticResource AddCarBrandToPartButtonStyle}"
                Command="{Binding AddBrandCommand}"
                ToolTip="Додати марку авто"/>

        <Button Style="{StaticResource DeleteCarBrandsFromPartButtonStyle}"
```

```
                                Command="{Binding ClearBrandsCommand}"
                                ToolTip="Очистити всі марки авто"/>

                        </StackPanel>

                        <Border Grid.Column="1" Style="{StaticResource BorderTextBoxFormStyle}">
                            <ComboBox ItemsSource="{Binding PartManufacturers}"
                                      SelectedValue="{Binding SelectedManufacturer}"
                                      Text="{Binding SelectedManufacturer}">

                                <ComboBox.ItemTemplate>
                                    <DataTemplate>
                                        <TextBlock Text="{Binding Manufacturer_name}" />
                                    </DataTemplate>
                                </ComboBox.ItemTemplate>
                            </ComboBox>
                        </Border>
                    </StackPanel>
                </Grid>

                <!-- Buttons -->
                <StackPanel Grid.Row="2"
                            Orientation="Horizontal"
                            HorizontalAlignment="Center"
                            VerticalAlignment="Center">

                    <Button Style="{StaticResource AddFormButtonStyle}"
                        Command="{Binding AddPartCommand}"/>

                    <Button Style="{StaticResource CancelFormButtonStyle}"
                        Command="{Binding CancelAddCommand}"/>

                </StackPanel>
```

На бек частині

```
private async Task AddPartAsync()
{
    try
    {
        IsOverlayVisible = true;

        var requiredFields = new (string FieldValue, string FieldName)[]
        {
            (SKUCode, "Артикул"),
            (PartName, "Назва"),
            (PartDescription, "Опис"),
            (SelectedCategory, "Категорія"),
            (PartBrands, "Марка авто"),
            (SelectedManufacturer, "Виробник")
        };

        foreach (var (value, name) in requiredFields)
        {
            if (string.IsNullOrWhiteSpace(value))
            {
                _modalNavigation.ShowMessage<ErrorMessageViewModel>(250, 400, $"Поле '{name}' є
обов'язковим.");
                return;
            }
        }

        var newPart = new Part(
            await _generateCodeService.GeneratePartCodeAsync(),
            SKUCode,
            PartName,
            await _codeByNameService.GetCategoryCodeByName(SelectedCategory),
            PartDescription,
            PartBrands,
            await _codeByNameService.GetManufacturerCodeByName(SelectedManufacturer)
        );

        await _addDataService.AddPartAsync(newPart);

        SKUCode = "";
        PartName = "";
        SelectedCategory = "";
        PartDescription = "";
```

```
        PartBrands = "";
        SelectedBrand = "";
        SelectedManufacturer = "";

        _modalNavigation.ShowMessage<SuccessMessageViewModel>(200, 270, "Запчастину успішно додано!");
    }
    catch (Exception ex)
    {
        _modalNavigation.ShowMessage<ErrorMessageViewModel>(250, 400, $"Не вдалося додати запчастину:
{ex.Message}");
    }
    finally
    {
        IsOverlayVisible = false;
    }
}
```

Запис даниз до БД

```
public async Task AddPartAsync(Part newPart)
{
    string insertPartQuery = @"
        INSERT INTO Part
        (Part_code, SKU_Code, Part_name, Part_description, Manufacturer_code_FK, Category_code_FK)
        VALUES
        (@PartCode, @SKUCode, @PartName, @PartDescription, @ManufacturerCodeFK, @CategoryCodeFK);";

    string getBrandCodeQuery = @"
        SELECT Brand_code
        FROM Car_brand
        WHERE Brand_name = @BrandName;";

    string insertPartBrandQuery = @"
        INSERT INTO Part_brand (Part_code_FK, Brand_code_FK)
        VALUES (@PartCodeFK, @BrandCodeFK);";

    using (var connection = new SqlConnection(DBConnection.GetConnectionString()))
    {
        await connection.OpenAsync();

        using (var transaction = connection.BeginTransaction())
        {
            try
            {
                using (SqlCommand insertPartCmd = new SqlCommand(insertPartQuery, connection,
transaction))
                {
                    insertPartCmd.Parameters.AddWithValue("@PartCode", newPart.Part_code);
                    insertPartCmd.Parameters.AddWithValue("@SKUCode", newPart.SKU_code);
                    insertPartCmd.Parameters.AddWithValue("@PartName", newPart.Part_name);
                    insertPartCmd.Parameters.AddWithValue("@PartDescription",
newPart.Part_description);
                    insertPartCmd.Parameters.AddWithValue("@ManufacturerCodeFK",
newPart.Manufacturer_name);
                    insertPartCmd.Parameters.AddWithValue("@CategoryCodeFK", newPart.Category_name);

                    await insertPartCmd.ExecuteNonQueryAsync();
                }

                var brandNames = newPart.Brand_name
                    .Split(',', StringSplitOptions.RemoveEmptyEntries)
                    .Select(b => b.Trim())
                    .Distinct(StringComparer.OrdinalIgnoreCase);

                foreach (var brandName in brandNames)
                {
                    string brandCode = null;

                    using (var getBrandCmd = new SqlCommand(getBrandCodeQuery, connection,
transaction))
                    {
                        getBrandCmd.Parameters.AddWithValue("@BrandName", brandName);
                        var result = await getBrandCmd.ExecuteScalarAsync();
                        brandCode = result?.ToString();
                    }

                    if (!string.IsNullOrEmpty(brandCode))
                    {
```

```
                        using (var insertBrandCmd = new SqlCommand(insertPartBrandQuery, connection,
transaction))
                        {
                            insertBrandCmd.Parameters.AddWithValue("@PartCodeFK", newPart.Part_code);
                            insertBrandCmd.Parameters.AddWithValue("@BrandCodeFK", brandCode);
                            await insertBrandCmd.ExecuteNonQueryAsync();
                        }
                    }
                    else
                    {
                        throw new Exception($"Бренд \"{brandName}\" не знайдено в таблиці
Car_brand.");
                    }
                }

                await transaction.CommitAsync();
            }
            catch (Exception)
            {
                await transaction.RollbackAsync();
                throw;
            }
        }

        connection.Close();
    }
}
```

Розподілення запчастин в комірку

На фронт частині

```
<!-- Input form -->
<Grid Grid.Row="1"
  HorizontalAlignment="Center"
  VerticalAlignment="Center">

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="100"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>

    <!-- Text form -->
    <StackPanel Grid.Column="0"
            Orientation="Vertical"
            HorizontalAlignment="Left">

        <TextBlock Style="{StaticResource TextFormStyle}"
               Text="Кількість"/>

        <TextBlock Style="{StaticResource TextFormStyle}"
               Text="Комірка"/>

    </StackPanel>

    <!-- Text box form -->
    <StackPanel Grid.Column="1"
            Orientation="Vertical">

        <Border Grid.Column="1"
        Style="{StaticResource BorderTextBoxFormStyle}">

            <TextBox Style="{StaticResource TextBoxFormStyle}"
                 Text="{Binding BatchQuantity}"/>

        </Border>

        <Border Grid.Column="1" Style="{StaticResource BorderTextBoxFormStyle}">
            <ComboBox ItemsSource="{Binding Cells}"
                  SelectedValue="{Binding SelectedCell}"
                  Text="{Binding SelectedCell}">

                <ComboBox.ItemTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding Cell_code}" />
                    </DataTemplate>
                </ComboBox.ItemTemplate>
```

```
            </ComboBox>
        </Border>
    </StackPanel>
</Grid>

<!-- Buttons -->
<StackPanel Grid.Row="2"
        Orientation="Horizontal"
        HorizontalAlignment="Center"
        VerticalAlignment="Center">

    <Button Style="{StaticResource MoveFormButtonStyle}"
            Command="{Binding MoveBatchCommand}"/>

    <Button Style="{StaticResource CancelFormButtonStyle}"
            Command="{Binding CancelMoveCommand}"/>

</StackPanel>
```

На бек частині

```
private async Task MoveBatchAsync()
{
    try
    {
        IsOverlayVisible = true;

        var requiredFields = new (string FieldValue, string FieldName)[]
        {
            (BatchQuantity, "Кількість"),
            (SelectedCell, "Комірка")
        };

        foreach (var (value, name) in requiredFields)
        {
            if (string.IsNullOrWhiteSpace(value))
            {
                _modalNavigation.ShowMessage<ErrorMessageViewModel>(250, 400, $"Поле '{name}' є
обов'язковим.");
                return;
            }
        }

        if (!int.TryParse(BatchQuantity, out int quantity) || quantity <= 0)
        {
            _modalNavigation.ShowMessage<ErrorMessageViewModel>(250, 400, "Введена некоректна
кількість.");
            return;
        }

        string cell = CleanSelectedCell(SelectedCell.ToString());
        bool success = await _warehouseStockMovementService.DistributePartsToCellAsync(BatchCode,
ZoneCode, cell, quantity);

        if (success)
        {
            _modalNavigation.ShowMessage<SuccessMessageViewModel>(200, 270, "Запчастини успішно
переміщені!");
            CloseModal();
        }
        else
        {
            _modalNavigation.ShowMessage<ErrorMessageViewModel>(250, 400, "Помилка переміщення
запчастин.");
        }
    }
    catch (Exception ex)
    {
        _modalNavigation.ShowMessage<ErrorMessageViewModel>(250, 400, "Помилка: " + ex.Message);
    }
    finally
    {
        IsOverlayVisible = false;
    }
}
```

Запис даних в БД

```
public async Task<bool> DistributePartsToCellAsync(string batchCode, string zoneCode, string cellCode,
int quantity)
{
    using (SqlConnection connection = new SqlConnection(DBConnection.GetConnectionString()))
    {
        await connection.OpenAsync();

        using (SqlCommand command = new SqlCommand("DistributePartsToCell", connection))
        {
            command.CommandType = CommandType.StoredProcedure;
            command.Parameters.AddWithValue("@Batch_code", batchCode);
            command.Parameters.AddWithValue("@Zone_code", zoneCode);
            command.Parameters.AddWithValue("@Cell_code", cellCode);
            command.Parameters.AddWithValue("@Quantity", quantity);

            try
            {
                await command.ExecuteNonQueryAsync();
                return true;
            }
            catch (SqlException ex)
            {
                Debug.WriteLine($"Помилка переміщення запчастин: {ex.Message}\n{ex.StackTrace}");
                return false;
            }
        }
    }
}
```

Автоматичне розподілення запчастин по коміркам

На фронт частині

```
<Grid>
                <Grid.RowDefinitions>
                    <RowDefinition Height="*"/>
                    <RowDefinition Height="Auto"/>
                </Grid.RowDefinitions>

                <!-- Text -->
                <TextBlock Style="{StaticResource DeleteTextStyle}"
                   Text="{Binding Message}"/>

                <!-- Buttons -->
                <StackPanel Grid.Row="2"
                    Orientation="Horizontal"
                    HorizontalAlignment="Center"
                    VerticalAlignment="Center">

                    <Button Style="{StaticResource MoveFormButtonStyle}"
                        Command="{Binding AutoMoveCommand}"
                        Margin="0 0 30 30"/>

                    <Button Style="{StaticResource CancelFormButtonStyle}"
                        Command="{Binding CancelMoveCommand}"
                        Margin="30 0 0 30"/>

                </StackPanel>
            </Grid>
```

На бек частині

```
private async Task AutoMoveBatch()
{
    try
    {
        IsOverlayVisible = true;

        var unallocatedBatches = BatchesStore.Where(b => b.Unallocated_batch_quantity > 0).ToList();
        if (!unallocatedBatches.Any()) return;

        var storageCells = (await _loadDataService.GetStorageCellsAsync())
            .Where(sc => sc.Current_capacity < sc.Max_capacity)
            .OrderBy(sc => sc.Current_capacity)
            .ToList();

        bool allBatchesDistributed = true;
```

```
        foreach (var batch in unallocatedBatches)
        {
            int remainingParts = batch.Unallocated_batch_quantity;

            var allowedZoneCode = await
_loadDataService.GetStorageZoneCodeByPartAsync(batch.Part_code);
            if (string.IsNullOrEmpty(allowedZoneCode)) continue;

            var allowedCells = storageCells
                .Where(sc => sc.Zone_code == allowedZoneCode)
                .ToList();

            foreach (var cell in allowedCells)
            {
                if (remainingParts <= 0) break;

                int availableSpace = cell.Max_capacity - cell.Current_capacity;
                if (availableSpace <= 0) continue;

                int partsToMove = Math.Min(remainingParts, availableSpace);

                bool success = await
_warehouseStockMovementService.DistributePartsToCellAsync(batch.Batch_code, cell.Zone_code,
cell.Cell_code, partsToMove);
                if (success)
                {
                    remainingParts -= partsToMove;
                }
            }

            if (remainingParts > 0)
            {
                _modalNavigation.ShowMessage<ErrorMessageViewModel>(300, 400, $"Попередження: не
вистачило місця для партії '{batch.Part_name}', " +
                                                                        $"залишилося
{remainingParts} нерозподілених запчастин.");
                allBatchesDistributed = false;
            }

            batch.Unallocated_batch_quantity = remainingParts;
        }

        if (allBatchesDistributed)
        {
            _modalNavigation.ShowMessage<SuccessMessageViewModel>(200, 320, "Всі партії успішно
розподілені!");
        }
    }
    catch (Exception ex)
    {
        _modalNavigation.ShowMessage<ErrorMessageViewModel>(250, 400, $"Не вдалося розподілити
запчастини: {ex.Message}");
    }
    finally
    {
        IsOverlayVisible = false;
        CloseModal();
    }
}
```

Генерація звіту

На фронт частині

```
<Grid>
    <!-- 3 Rows -->
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>

    <!-- Title -->
    <TextBlock Grid.Row="0"
               Style="{StaticResource TitleTextFormStyle}"
               Text="Створення звіту"/>
```

```
    <!-- Input form -->
    <Grid Grid.Row="1"
          HorizontalAlignment="Center"
          VerticalAlignment="Center">

        <Grid.RowDefinitions>
            <RowDefinition Height="*"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <StackPanel Grid.Row="0"
                    Orientation="Horizontal"
                    HorizontalAlignment="Left"
                    Margin="0 10 0 20">

            <TextBlock Style="{StaticResource TextFormStyle}"
                       Text="Період    з"
                       Margin="0 0 18 0"/>

            <DatePicker CalendarStyle="{DynamicResource DatePickerCalendarStyle}"
                        Style="{DynamicResource DatePickerStyle}"
                        SelectedDate="{Binding SelectedStartDate,
UpdateSourceTrigger=PropertyChanged}"
                        Margin="0 0 31 0"/>

            <TextBlock Style="{StaticResource TextFormStyle}"
                       Text="по"
                       Margin="0 0 31 0"/>

            <DatePicker CalendarStyle="{DynamicResource DatePickerCalendarStyle}"
                        Style="{DynamicResource DatePickerStyle}"
                        SelectedDate="{Binding SelectedEndDate, UpdateSourceTrigger=PropertyChanged}"
                        Margin="0"/>

        </StackPanel>

        <StackPanel Grid.Row="1"
                    Orientation="Horizontal"
                    Margin="0 0 0 0"
                    Visibility="{Binding IsByCategories, Converter={StaticResource
BoolToVisibilityConverter}}">

            <TextBlock Style="{StaticResource TextFormStyle}"
                       Text="Категорія"
                       Margin="0 0 25 0"/>

            <Border Grid.Column="1" Style="{StaticResource BorderTextBoxFormStyle}">
                <ComboBox ItemsSource="{Binding PartCategories}"
                          SelectedValue="{Binding SelectedCategory}"
                          Text="{Binding SelectedCategory}">

                    <ComboBox.ItemTemplate>
                        <DataTemplate>
                            <TextBlock Text="{Binding Category_name}" />
                        </DataTemplate>
                    </ComboBox.ItemTemplate>
                </ComboBox>
            </Border>
        </StackPanel>
    </Grid>

    <!-- Buttons -->
    <StackPanel Grid.Row="2"
                Orientation="Horizontal"
                HorizontalAlignment="Center"
                VerticalAlignment="Center">

        <Button Style="{StaticResource CreateFormButtonStyle}"
                Command="{Binding CreateReportCommand}"/>

        <Button Style="{StaticResource CancelFormButtonStyle}"
                Command="{Binding CancelCreateCommand}"/>

    </StackPanel>

</Grid>
```

На бек частині

```
public async Task CreatePurchaseExpensesByCategoriesReport(string filePath, DateTime startDate,
DateTime endDate, string categoryName)
{
    List<PurchaseExpensesReport> purchaseExpenses = await
_loadDataService.GetPurchaseExpensesByCategoriesAsync(startDate, endDate, categoryName);

    using (var writer = new PdfWriter(filePath))
    using (var pdf = new PdfDocument(writer))
    using (var document = new Document(pdf))
    {
        PdfFont fontRegular = PdfFontFactory.CreateFont("C:/Windows/Fonts/arial.ttf",
PdfEncodings.IDENTITY_H);
        PdfFont fontBold = PdfFontFactory.CreateFont("C:/Windows/Fonts/arialbd.ttf",
PdfEncodings.IDENTITY_H);

        string formattedStartDate = startDate.ToString("yyyy.MM.dd");
        string formattedEndDate = endDate.ToString("yyyy.MM.dd");
        document.Add(new Paragraph($"Витрати на закупівлі запчастин\nз {formattedStartDate} по
{formattedEndDate}\nкатегорії '{categoryName}'")
            .SetTextAlignment(TextAlignment.CENTER)
            .SetFont(fontBold)
            .SetFontSize(20));

        document.Add(new Paragraph().SetMarginTop(20));

        Table table = new Table(UnitValue.CreatePercentArray(4)).UseAllAvailableWidth();
        string[] headers = { "Артикул", "Запчастина", "Кількість", "Загальна ціна (грн)" };

        foreach (var header in headers)
        {
            table.AddHeaderCell(new Cell().Add(new Paragraph(header).SetFont(fontBold)));
        }

        foreach (var expense in purchaseExpenses)
        {
            table.AddCell(new Cell().Add(new Paragraph(expense.SKU).SetFont(fontRegular)));
            table.AddCell(new Cell().Add(new Paragraph(expense.PartName).SetFont(fontRegular)));
            table.AddCell(new Cell().Add(new
Paragraph(expense.TotalQuantity.ToString()).SetFont(fontRegular)));
            table.AddCell(new Cell().Add(new
Paragraph(expense.TotalCost.ToString("F2")).SetFont(fontRegular)));
        }

        // Calculate total expenses
        double totalExpenses = (double)purchaseExpenses.Sum(e => e.TotalCost);

        table.AddFooterCell(new Cell(1, 3).Add(new Paragraph("Загальні витрати:").SetFont(fontBold)));
        table.AddFooterCell(new Cell().Add(new
Paragraph(totalExpenses.ToString("F2")).SetFont(fontBold)));

        document.Add(table);

        DateTime dateTime = DateTime.Now;
        string formattedDate = dateTime.ToString("yyyy.MM.dd");

        Table footerTable = new Table(UnitValue.CreatePercentArray(new float[] { 1, 1 }))
            .UseAllAvailableWidth()
            .SetMarginTop(50);

        footerTable.AddCell(new Cell().Add(new Paragraph($"Дата:
{formattedDate}").SetFont(fontRegular))
            .SetBorder(Border.NO_BORDER)
            .SetTextAlignment(TextAlignment.LEFT));

        footerTable.AddCell(new Cell().Add(new Paragraph("Підпис _____").SetFont(fontRegular))
            .SetBorder(Border.NO_BORDER)
            .SetTextAlignment(TextAlignment.RIGHT));

        document.Add(footerTable);

        AddPageNumbers(pdf, fontRegular);
    }
}
```

ДОДАТОК Д

Звіт по поставкам запчастин

Поставки запчастин
з 2020.02.06 по 2025.05.09

Постачальник	Дата	Артикул	Запчастина	Кількість
Parts R Us	2025.04.15	BOS-7503-573	Свічка запалювання Bosch Super Plus	100
Parts R Us	2025.04.15	DENS-1022110	Гальмівні колодки передні Denso	50
Global Parts	2025.04.16	VAL-453320	Радіатор охолодження Valeo	80
Global Parts	2025.04.16	MAG-99402	Фара ліва Magneti Marelli	60
Auto Supply Co.	2025.04.17	KYB-55042	Амортизатор передній KYB Excel-G	100
Auto Supply Co.	2025.04.17	NGK-74120	Датчик кисню NGK	90
Parts R Us	2025.04.18	MAN-15242	Масляний фільтр MANN HU 716/2 x	80
Parts R Us	2025.04.18	BOS-12345	Датчик ABS Bosch	70
Parts R Us	2025.04.18	DENS-88765	Компресор кондиціонера Denso DCP32001	50
Global Parts	2025.04.19	DEL-67890	Стартер Delco Remy DSN1209	70
Global Parts	2025.04.19	VAL-54321	Вентилятор радіатора Valeo	80

Дата: 2025.04.06 Підпис _____