ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення підтримки управління складом автозапчастин»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Денис КРИВОНОСЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-43

Денис КРИВОНОСЕНКО

Керівник: Владислав ЯСКЕВИЧ
к.т.н., доцент

Рецензент: _____

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кривоносенку Денису Миколайовичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення підтримки управління складом автозапчастин»
керівник кваліфікаційної роботи к.т.н., доцент Владислав ЯСКЕВИЧ,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «24» лютого 2025 р. № 56.

2. Строк подання кваліфікаційної роботи «02» червня 2025 р.

3. Вихідні дані до кваліфікаційної роботи: Теоретичні відомості про методи розробки програмного забезпечення для обліку товарів на складі, опис роботи складських програм, документація для програмних фреймворків, таких як C#, .NET, інструменти для роботи з базами даних.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд програмного забезпечення для управління складом автозапчастин та аналіз існуючих аналогів, визначення вимог до програмного забезпечення.
2. Огляд та аналіз засобів реалізації програмного забезпечення для управління складом автозапчастин.

3. Проєктування архітектури програмного забезпечення для підтримки управління складом автозапчастин.
4. Програмна реалізація та тестування програмного забезпечення.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Вимоги до програмного забезпечення.
   3. Програмні засоби реалізації.
   4. Діграма варіантів використання.
   5. Діаграма структури баз даних.
   6. Діаграма послідовності “Додавання нової автозапчастини” .
   7. Екранні форми.
   8. Демонстрація роботи застосунку.
   9. Апробація результатів дослідження.

6. Дата видачі завдання «25» лютого 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02-06.03.2025	
2	Аналіз та дослідження існуючих аналогів програмного забезпечення	07.03-13.03.2025	
3	Огляд та аналіз засобів реалізації програмного забезпечення для управління складом	14.03-18.03.2025	
4	Огляд та аналіз засобів реалізації інтерфейсу користувача (UI)	19.03-24.03.2025	
5	Проєктування архітектури програмного забезпечення для управління складом автозапчастин	25.03-27.03.2025	
6	Реалізація та тестування програмного забезпечення	28.03-28.04.2025	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2025	
8	Розробка демонстраційних матеріалів	06.05-12.05.2025	
9	Попередній захист роботи	13.05-31.05.2025	

Здобувач вищої освіти _____ Денис КРИВОНОСЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Владислав ЯСКЕВИЧ
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 57 стор., 2 табл., 25 рис., 12 джерел.

Мета роботи – полегшити процес управління складом автозапчастин.

Об’єкт дослідження – процес управління складськими запасами автозапчастин у малому та середньому бізнесі.

Предмет дослідження – програмні засоби та технології створення прикладного програмного забезпечення для локального збереження та обробки складських даних.

Короткий зміст роботи – У роботі проаналізовано предметну галузь та методи реалізації програмного забезпечення для обліку автозапчастин на складі. Проведено огляд існуючих програмних рішень, таких як BAS, Dilovod, Torgsoft, Remonline, для обліку товарів та управління запасами на складі. Розроблено програму для ручного обліку автозапчастин, яка включає функціональні можливості, такі як: авторизація користувачів, ведення каталогу автозапчастин, внесення та оновлення даних про запчастини, зокрема їх кількість, типи та моделі. Реалізовано систему пошуку та фільтрації товарів для швидкого доступу до інформації про запчастини. Для розробки використано мову програмування C# та фреймворк .NET для створення бекенду, зокрема Entity Framework для роботи з базою даних. Програма підтримує зручний інтерфейс для введення даних про запчастини вручну та дозволяє генерувати звіти щодо залишків товарів. Проведено тестування для забезпечення коректної роботи всіх функцій системи.

КЛЮЧОВІ СЛОВА: ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, АВТОЗАПЧАСТИНИ, УПРАВЛІННЯ СКЛАДОМ, ОБЛІК ТОВАРІВ, SQLITE, C#, .NET, ENTITY FRAMEWORK, WINDOWS FORMS, ІНТЕРФЕЙС КОРИСТУВАЧА (UI), ІНВЕНТАРИЗАЦІЯ, ЗВІТИ, ПОШУК ТОВАРІВ, ФІЛЬТРАЦІЯ, АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, КАТЕГОРИЗАЦІЯ ТОВАРІВ, МОВИ ПРОГРАМУВАННЯ, УПРАВЛІННЯ ЗАПАСАМИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Db - клас для взаємодії з базою даних.

Entity Framework - ORM для взаємодії з реляційними базами даних.

SQLite - реляційна база даних для зберігання даних.

UI - інтерфейс користувача.

DialogResult - результат закриття діалогового вікна в програмі.

SHA-256 - алгоритм хешування для безпеки паролів.

C# - мова програмування для розробки програмного забезпечення.

.NET - платформа для створення програм на C#.

Windows Forms - технологія для розробки графічних інтерфейсів на платформі .NET.

XLSX - формат Excel для експорту та імпорту даних.

UI/UX - проектування інтерфейсу та взаємодії користувача.

ВСТУП

Актуальність: Управління складом автозапчастин є ключовим елементом у діяльності підприємств, що займаються ремонтом автотранспортних засобів або продажем запчастин. В умовах швидко змінюваного ринку важливо забезпечити точний облік запчастин, контролювати їх рух і зберігати необхідні запаси для безперервної роботи підприємства. Часто підприємства мають справу з проблемами неточності в обліку запчастин через застосування застарілих методів, що ускладнюють оперативне оновлення даних і не забезпечують легкий доступ до потрібної інформації. [1] Ці проблеми можуть призводити до затримок у виконанні операцій, помилок при веденні документації, а також до некоректного відображення залишків запчастин. Розробка програмного забезпечення для управління складом автозапчастин здатна покращити точність обліку запчастин, скоротити час на виконання операцій та забезпечити зручний доступ до актуальної інформації. Це також дозволить оптимізувати процеси пошуку товарів та формування необхідних звітів.

Об’єкт дослідження: процес управління складськими запасами автозапчастин у малому та середньому бізнесі.

Предмет дослідження: програмні засоби та технології створення прикладного програмного забезпечення для локального збереження та обробки складських даних.

Мета роботи: полегшити процес управління складом автозапчастин.

Методи дослідження: У роботі було проаналізовано існуючі методи управління складом автозапчастин, що використовуються на ринку, а також програми, які підтримують облік запчастин на складах. Далі було проведено дослідження технологій для створення програмного забезпечення, що включає вибір фреймворку для розробки інтерфейсу та логіки додатку.

Для роботи з базою даних обрано технології SQLite та Entity Framework, які забезпечують надійне зберігання і швидкий доступ до даних. Вибір C# та .NET як

основних технологій для розробки був обумовлений їх універсальністю, ефективністю та широкими можливостями для створення програмного забезпечення під операційну систему Windows.

Наукова новизна: полягає у розробці програмного забезпечення для підтримки управління складом автозапчастин, яке дозволяє не тільки обліковувати запчастини, але й забезпечити ефективний пошук, фільтрацію та створення звітів. Всі дані про запчастини зберігаються у базі даних з можливістю їх редагування та створення звітів, що дозволяє отримувати точну інформацію про запаси та наявність товарів на складі.

Практична значущість результатів роботи полягає в розробці інструменту, який дозволяє підприємствам зменшити ймовірність помилок при веденні обліку запчастин, покращити організацію роботи складу, зберігати точні дані про наявність товарів і забезпечити зручний доступ до цієї інформації. Програмне забезпечення дозволяє значно зменшити час, необхідний для пошуку товарів на складі, а також підвищити ефективність роботи складу через створення автоматизованих звітів для подальшого аналізу і планування.

1 АНАЛІЗ ПРОГРАМИ ДЛЯ УПРАВЛІННЯ СКЛАДОМ АВТОЗАПЧАСТИ

1.1 Специфіка управління складом автозапчастин

Управління складом автозапчастин є важливою частиною діяльності підприємств, які займаються обслуговуванням або ремонтом автотранспортних засобів. Оскільки асортимент запчастин може бути дуже великим і різноманітним, управління такими запасами потребує ретельної організації та точного обліку. Всі операції на складі мають бути чітко регламентовані, щоб забезпечити точність даних і зручність використання програми.

Основні аспекти управління складом автозапчастин:

1. Облік товарів на складі

Опис: Облік автозапчастин на складі є основною операцією, що забезпечує точність даних про кількість запчастин на складі. Це включає введення даних про кожну частину в програму, де фіксуються такі характеристики, як тип, модель, кількість, ціна та інші важливі параметри.

Ключові аспекти:

Додавання запчастин: Запчастини додаються до обліку вручну через інтерфейс програми, де вказуються всі необхідні характеристики запчастини. Це дозволяє зберігати точні дані про всі одиниці запчастин.

Оновлення даних: Коли змінюється кількість запчастин, ці зміни повинні бути відображені в обліковій системі, що дозволяє підтримувати актуальну інформацію.

Категоризація товарів: Запчастини повинні бути правильно класифіковані за категоріями, щоб полегшити пошук і зберігання запчастин на складі.

2. Інвентаризація

Опис: Інвентаризація є процесом перевірки фактичних запасів на складі та порівняння їх з обліковими даними. Це дозволяє виявити розбіжності між фактичними і обліковими залишками.

Ключові аспекти:

Періодичні перевірки: Інвентаризація повинна проводитись регулярно, щоб забезпечити точність обліку товарів на складі.

Виявлення розбіжностей: Якщо дані в системі не збігаються з фактичними запасами, необхідно вжити заходів для коригування обліку.

3. Облік руху товарів

Опис: У програмі передбачено облік руху товарів на складі, який включає операції з отримання та повернення запчастин. Це дає можливість точно відслідковувати, коли запчастини забираються зі складу, наприклад, під час продажу або використання на СТО, а також коли вони повертаються назад на склад після обробки. Кожна операція фіксується в системі, з зазначенням кількості товару, дати та відповідальних осіб, що забезпечує точний контроль за залишками та рухом запчастин на складі.

Ключові аспекти:

Запис операцій: Кожна операція, яка впливає на кількість товару (наприклад, списання через дефекти або продаж), повинна бути задокументована в програмі, щоб оновлювати запаси на складі.

4. Формування звітів

Опис: Програма повинна дозволяти формувати звіти про стан складу, зокрема звіти про залишки автозапчастин, їхній рух (надходження, витрати) та інші важливі параметри.

Ключові аспекти:

Звіти по залишках: Генерація звітів, які дозволяють переглядати поточні залишки на складі, допомагає в плануванні закупок і продажів.

5. Різноманітність запчастин

Запчастини на складі можуть мати багато варіацій - від різних марок і моделей до різних типів і характеристик. Це ускладнює облік товарів і вимагає

чіткої класифікації кожної одиниці товару, щоб підтримувати точність і зручність у пошуку. У разі великої кількості товарів, кожен елемент потребує індивідуального підходу до введення та обліку в програмі, що дозволяє уникнути помилок і спростити подальше управління.

6. Категоризація

Для спрощення обліку та пошуку запчастини повинні бути правильно класифіковані за групами, наприклад, за категоріями: двигуни, підвіски, електроніка та інші. Це дозволяє швидко знаходити необхідні деталі в базі даних та оптимізує процеси обслуговування складу. Чітка категоризація також допомагає в аналізі попиту на різні групи товарів, що в свою чергу полегшує планування закупок та поповнення складу.

7. Контроль за залишками:

Точність обліку залишків на складі є критично важливою для ефективної роботи складу. Для цього необхідно регулярно оновлювати дані про кількість запчастин на складі після кожної. Без актуальних даних склад не може ефективно працювати, тому важливо здійснювати постійний моніторинг запчастин запасів для попередження дефіциту або надлишку запчастин.

1.2 Цільова аудиторія

Програмне забезпечення для обліку автозапчастин орієнтоване на різноманітні категорії користувачів, що працюють у сфері продажу та обслуговування автозапчастин. Кожна з цих груп має специфічні потреби в управлінні запасами, контролі за залишками та забезпеченні точності даних. Зокрема, основними групами користувачів є:

Малі та середні підприємства, що займаються продажем автозапчастин:

Ця категорія включає компанії, які здійснюють роздрібний та оптовий продаж автозапчастин. Для них важливо мати надійну систему для обліку запасів запчастин, що дозволяє зберігати точну інформацію про кількість запчастин, та види запчастин. Такі підприємства потребують інструментів для збереження

актуальних даних про наявність та рух запчастин на складі, що дозволяє зменшити ризик помилок, пов'язаних з відсутністю необхідних запчастин або їх надлишком на складі.

Станції технічного обслуговування (СТО):

Автосервіси, що виконують ремонт транспортних засобів, потребують системи для ефективного обліку використаних запчастин під час ремонтних робіт. Вони мають необхідність відслідковувати, які саме запчастини використовуються для кожного автомобіля, а також вести облік їх кількості, щоб забезпечити точність під час ремонту та уникнути помилок при поповненні запасів. Також важливо, щоб програма дозволяла вчасно оновлювати інформацію про залишки запчастин, що необхідно для планування закупок і поповнення складу.

Магазини автозапчастин:

Магазини, що спеціалізуються на продажу автозапчастин для різних марок автомобілів, мають постійну потребу в інструменті для контролю запасів запчастин. Вони повинні мати можливість отримувати актуальну інформацію щодо запасів запчастин на складі та можливість швидко реагувати на запити клієнтів. Програма дозволяє вести облік наявних запчастин, фіксувати операції з їх рухом на складі та формувати звіти для аналізу запасів, що забезпечує більш ефективне управління складом і бізнес-процесами. Магазини також використовують програму для управління запасами, щоб уникнути проблем з надлишком чи дефіцитом товарів.

Інші підприємства та організації, що мають потребу в управлінні запасами автозапчастин:

Це можуть бути різноманітні підприємства, які не є прямими продавцями або обслуговувачами автомобілів, але використовують автозапчастини в своїй діяльності. Наприклад, транспортні компанії, автопарки, корпоративні автосервіси або компанії, що займаються ремонтом техніки. Вони потребують простих і ефективних інструментів для обліку запчастин і забезпечення своєчасного поповнення запасів.

Усі ці категорії користувачів мають спільну мету - знизити операційні витрати, зменшити кількість помилок в обліку, підвищити ефективність управління запасами та забезпечити точність даних про запчастини. Програмне забезпечення, що надається для цих користувачів, дозволяє спростити багато процесів, що підвищує продуктивність і точність ведення бізнесу, спрощує інвентаризацію та допомагає оперативно реагувати на зміни в попиті та пропозиції.

1.3 Дослідження існуючих аналогів

Засоби для управління складом автозапчастин можуть бути представлені різними типами програмних рішень. Кожне програмне забезпечення має свої особливості, що дозволяють вибрати найбільш підходяще для конкретного підприємства або складських процесів. Усі вони спрямовані на оптимізацію обліку товарів, управління запасами, моніторинг руху товарів і навіть створення звітів для підприємства.[2]

У цьому розділі ми проаналізуємо кілька програмних рішень, їхні переваги та недоліки, а також надамо приклади можливих функцій, що можуть бути корисними для ефективного управління складом автозапчастин.

1.3.1 BAS Комплексне управління підприємством

Програма для автоматизації обліку та управління складом автозапчастин. Включає модулі для управління запасами, фінансами, закупівлею та продажами, що дозволяє створити комплексну систему для організації обліку на підприємстві.[3]

Функції програми:

Облік товарів та запасів: Можливість вести облік товарів з можливістю врахування місця зберігання, серійних номерів, кількості і вартості.

Переміщення товарів: Відстеження всіх переміщень товарів між складами і точками зберігання.

Формування звітів: Створення звітів про запаси, продажі, закупівлі, фінансові витрати.

Переваги:

Підтримка інтеграції з іншими модулями, що дозволяє автоматизувати більшість бізнес-процесів.

Можливість роботи з великими складами і великою кількістю товарів.

Недоліки:

Складність у налаштуванні для нових користувачів.

Висока вартість ліцензії для малого бізнесу.

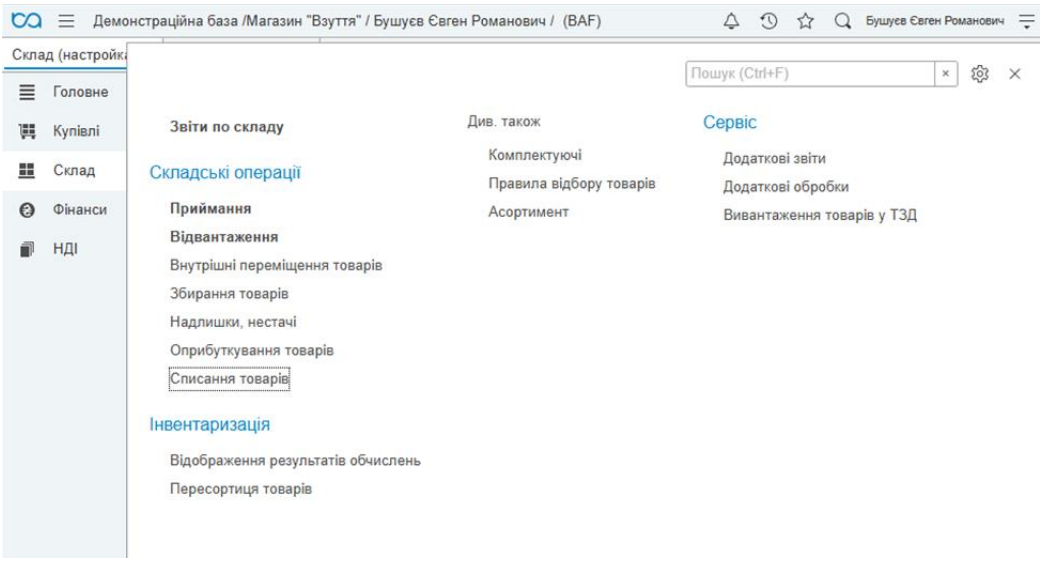


Рис. 1.1 Інтерфейс програми BAS - Складські операції

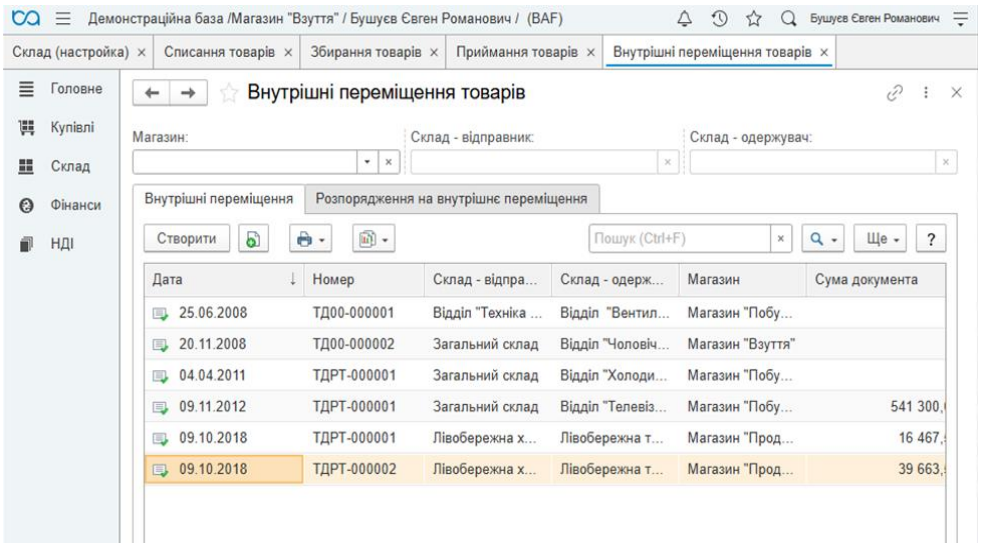


Рис. 1.2 Інтерфейс програми BAS - Внутрішнє переміщення товарів

1.3.2. Torgsoft

Torgsoft - це програмне забезпечення для автоматизації обліку товарів і управління складськими запасами, орієнтоване на підприємства малого та середнього розміру, зокрема в сфері продажу автозапчастин. Програма надає можливість вести облік товарів, контролювати їх рух, а також генерувати звіти для аналізу. База даних у Torgsoft реалізована на SQL, що забезпечує ефективне зберігання та обробку даних про товари.[4]

Функції програми:

Torgsoft дозволяє вести облік товарів, фіксуючи їх кількість, вартість та серійні номери, що дає змогу здійснювати точний контроль за запасами на складі. Програма також дозволяє відслідковувати переміщення товарів між складами та точками зберігання, що дозволяє ефективно керувати рухом товарів і мати актуальну інформацію про їх наявність. Завдяки використанню SQL для зберігання даних, програма забезпечує швидкий доступ і зручну роботу з великими обсягами інформації. Крім того, Torgsoft генерує звіти про запаси, продажі, закупівлі та фінансові витрати, що допомагає підприємствам планувати свою діяльність і контролювати ефективність операцій.

Переваги:

Простота у використанні: простиий інтерфейс спрощує роботу користувачів, навіть для тих, хто не володіє глибокими технічними знаннями.

Підходить для малих підприємств: програма ідеально підходить для малих компаній, що займаються зберіганням та обліком автозапчастин.

Недоліки:

Відсутність інтеграції з іншими бізнес-системами: програма не підтримує інтеграцію з іншими системами, такими як ERP або фінансовими системами.

Не підтримує автоматизацію обліку для великих складів: програма більше підходить для малих і середніх підприємств, оскільки не має функцій для автоматизації обліку на великих складах.

Платні функції:

У платних версіях програми доступні додаткові функції для створення розширених звітів та інтеграції з іншими системами, що робить програму більш ефективною для великих підприємств і мереж.

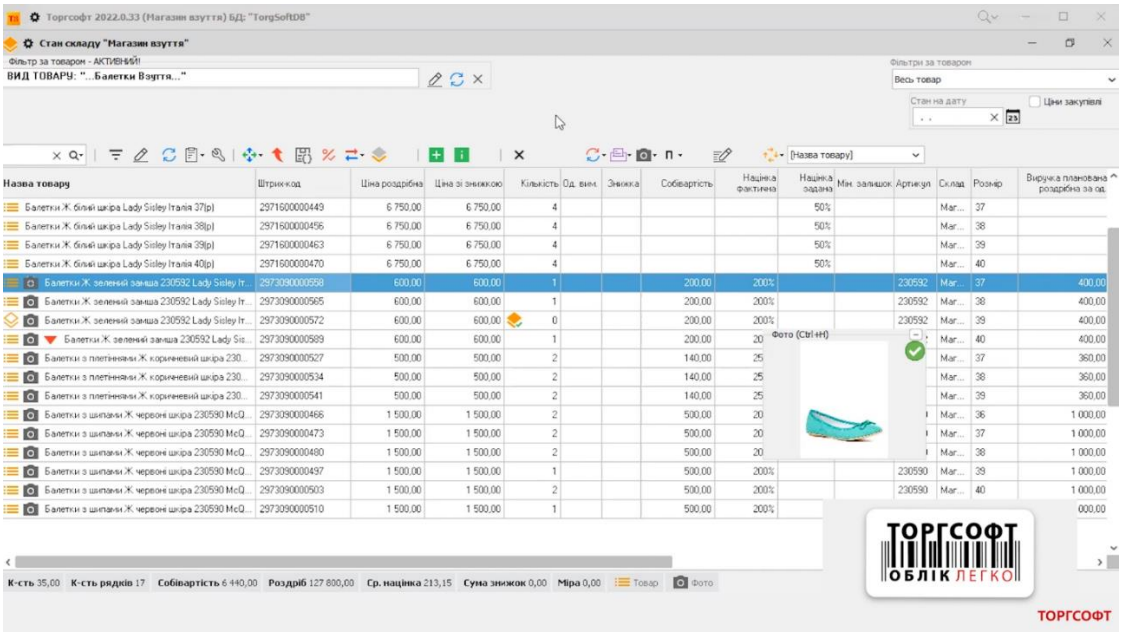


Рис. 1.3 Інтерфейс програми Torgsoft - Облік товарів на складі.

1.3.3. Remonline

Remonline - це програмне забезпечення для управління складом, яке дозволяє здійснювати облік запасів та контролювати їх рух. Це хмарне рішення, що підходить для підприємств, які не займаються автоматизацією продажів, а лише зберігають та обліковують товар. Remonline допомагає відслідковувати товар на складі в реальному часі, управляти переміщенням між точками зберігання та генерувати звіти для подальшого аналізу.[5]

Функції програми:

Remonline дозволяє вести облік запчастин на складі, фіксуючи їх кількість, характеристики, місце зберігання та інші важливі дані. Вона також забезпечує можливість переміщення товарів між складами, що дозволяє зручно відслідковувати рух товару та актуальність запасів у різних точках зберігання. Крім того, програма генерує звіти про запаси на складі, дозволяючи аналізувати дані про

залишки, витрати і переміщення товарів. Це дозволяє підприємствам ефективно планувати свою діяльність та забезпечувати точний облік товарів.

Переваги:

Підтримка мобільних пристроїв: Remonline підтримує мобільні додатки, що дозволяє здійснювати облік товарів та контролювати запаси в реальному часі, незалежно від того, де знаходиться користувач. Це зручне рішення для підприємств, що працюють з великою кількістю товару.

Простота у налаштуванні і введенні даних: Інтерфейс програми є інтуїтивно зрозумілим і простим у використанні, що дозволяє швидко налаштувати систему та почати роботу без великих затрат часу на навчання.

Недоліки:

Відсутність глибокої інтеграції з іншими системами: Програма не підтримує складну інтеграцію з іншими корпоративними системами, такими як ERP або фінансовими платформами, що обмежує її можливості для великих підприємств.

Обмежені можливості для великих підприємств: Програма орієнтована більше на малі та середні підприємства, оскільки не має необхідних функцій для автоматизації складських операцій на великих складах з великим обсягом товару.

Платні функції:

Основні функції доступні в безкоштовній версії, що є великим плюсом для малих підприємств. Однак для інтеграції з іншими бізнес-системами та доступу до більш детальних звітів і аналітики необхідно перейти на платну версію, яка розширює можливості програми для масштабованих підприємств.

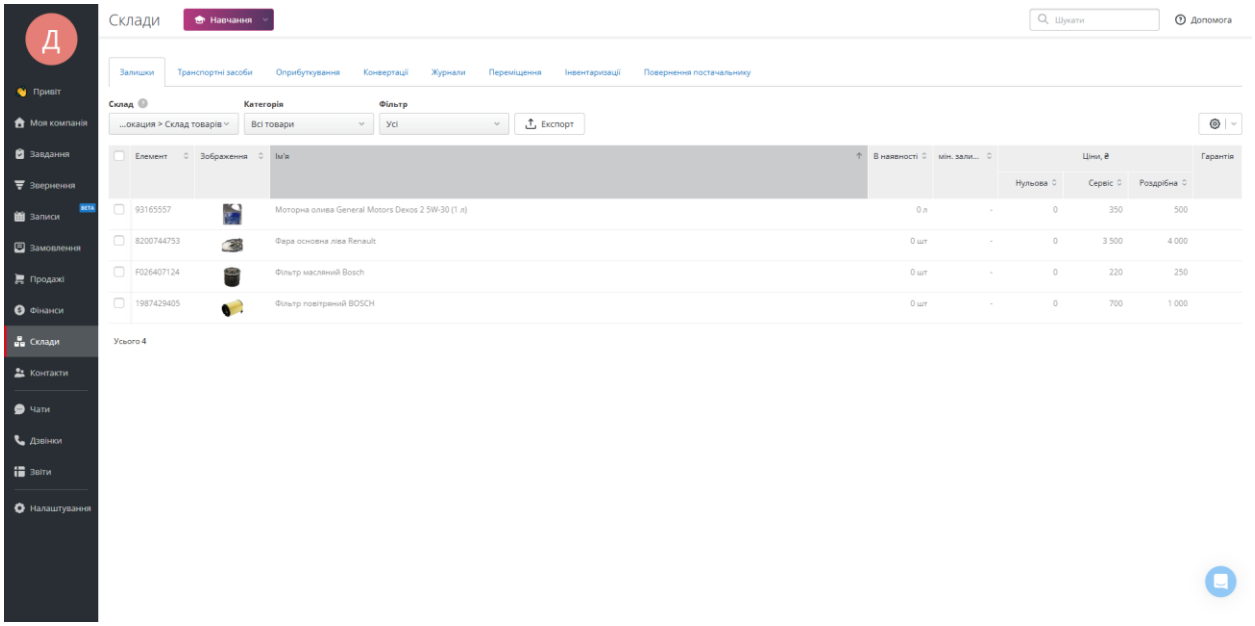


Рис. 1.4 Інтерфейс програми Remonline - Облік товарів на складі

1.3.4. Dilovod

Dilovod - це програмне забезпечення для автоматизації обліку товарів і управління складом, орієнтоване на підприємства малого та середнього бізнесу. Програма дозволяє здійснювати облік товарів на складі, вести фінансові операції та планувати запаси. Dilovod є зручним рішенням для тих підприємств, які потребують простого обліку товарів без використання складних ERP-систем.[6]

Функції програми:

Dilovod дозволяє вести повний облік товарів на складі, включаючи їх характеристики, такі як ціна, кількість, категорії тощо. Користувачі можуть здійснювати переміщення товарів між складами, що дозволяє ефективно відслідковувати їх рух і забезпечувати точність обліку. Програма також включає функціонал для обліку платежів, витрат і доходів підприємства, що дозволяє здійснювати фінансовий контроль. Крім того, Dilovod генерує звіти про запаси, продажі та прибутки, що допомагає бізнесу здійснювати аналіз і планування.

Програма має інструменти для організації інвентаризацій товарів на складах, що дозволяє оперативно перевіряти фактичні залишки товару і порівнювати їх з даними обліку. Створення і обробка замовлень клієнтів і постачальників є ще однією корисною функцією програми.

Переваги:

Підходить для малого бізнесу: Dilovod є ідеальним для малих підприємств, де не потрібні складні ERP-системи.

Легкість у налаштуванні та використанні: інтерфейс програми інтуїтивно зрозумілий і не потребує додаткових IT-ресурсів для налаштування.

Недоліки:

Обмежені можливості для великих підприємств: Dilovod не підходить для великих підприємств із великими обсягами товарів, оскільки не має функцій для автоматизації обліку на великих складах.

Відсутність гнучкої інтеграції з іншими бізнес-системами: програма не підтримує гнучку інтеграцію з іншими програмними продуктами, такими як ERP чи фінансові системи.

Використання:

Dilovod підходить для малих підприємств і магазинів, що потребують простого обліку товарів та управління запасами, але не мають потреби в складних автоматизованих системах або інтеграціях з іншими бізнес-процесами.

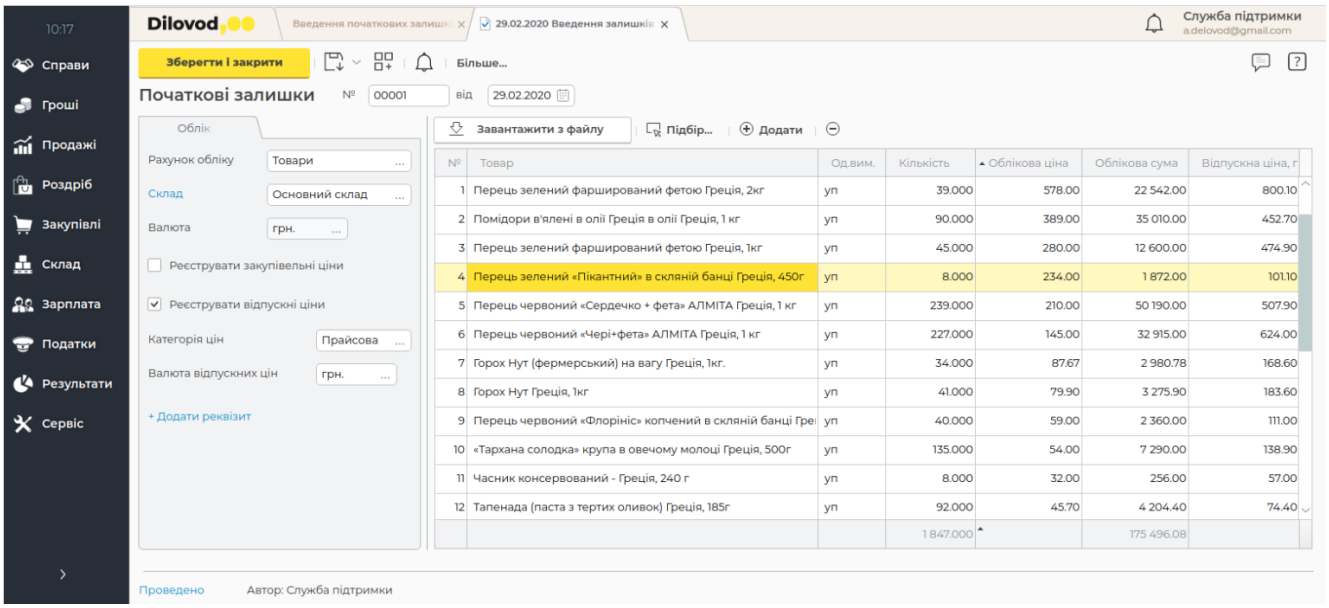


Рис. 1.5 Інтерфейс програми Dilovod - Введення початкових залишків товарів на складі

Таблиця 1.1

Порівняльна таблиця результатів аналізу програмного забезпечення

Показник	BAS Комплексне управління підприємством	Torgsoft	Dilovod	Remonline	Sklad (Мій застосунок)
Ручний облік	-	+	+	+	+
Звітність	-	+	-	-	+
Пошук запчастин	+	+	+	+	+
Фільтр запчастин	+	+	-	+	+
Сортування запчастин	+	+	+	+	+
Сповіщення про нестачу запчастин	-	-	+	-	+
Інвентаризація запчастин	+	+	+	+	+

1.4 Визначення вимог до застосунку

Проаналізувавши специфіку програмного забезпечення для обліку автозапчастин, його цільову аудиторію, а також існуючі аналоги, було визначено наступні вимоги до застосунку.

1. Програма повинна підтримувати вибір користувача зі списку зареєстрованих, з подальшим введенням пароля для входу в систему. Це дозволяє забезпечити захист даних та налаштувати різні рівні доступу, такі як адміністратор, менеджер складу або користувач з обмеженими правами доступу. Тільки адміністратор має можливість реєструвати нових користувачів і змінювати їх права доступу.

2. Управління базою даних автозапчастин:

Система повинна мати можливість зберігати та обробляти дані про автозапчастини, такі як їх бренд, модель, тип, назва, та кількість. Користувач

повинен мати можливість додавати, редагувати, видаляти записи, а також виконувати пошук товарів за різними критеріями (назва, модель, тип, тощо).

3. Система пошуку та фільтрації запчастин:

Необхідно реалізувати функціонал для швидкого пошуку автозапчастин за різними характеристиками, такими як бренд, модель, тип, назва та кількість. Це дасть можливість користувачам оперативно знаходити потрібні запчастини.

4. Облік руху запчастин на складі:

Програма повинна надавати можливість відображати рух запчастин на складі. Користувачі можуть вносити дані про запчастини, включаючи їх назву та кількість, а також обирати тип операції, наприклад, "На склад" або "Зі складу". Це дозволяє точно відслідковувати переміщення запчастин на складі, зберігаючи актуальні дані про залишки на складі. Всі операції відображаються в реальному часі, що дозволяє зберігати точну інформацію про наявність товарів на складі.

5. Формування звітів:

Програма повинна дозволяти генерувати різноманітні звіти про залишки товарів, обороти запчастин, популярні запчастини та запчастини на руках. Звіти повинні бути зручними для подальшого аналізу, а також мати можливість експорту в формат Excel, що дозволяє зберігати інформацію для подальшого використання або для аналізу.

6. Хешування паролю для забезпечення безпеки даних:

Паролі користувачів повинні зберігатися в зашифрованому вигляді за допомогою хешування перед збереженням у базі даних. Для цього використовується сучасний алгоритм хешування, наприклад, SHA-256, що забезпечує високий рівень безпеки даних користувачів.

7. Інтерфейс користувача:

Програма повинна мати зручний і зрозумілий інтерфейс, який дозволяє користувачам швидко орієнтуватися в системі. Всі елементи інтерфейсу (кнопки, поля вводу, меню) повинні бути інтуїтивно зрозумілими, а також доступними для швидкої взаємодії

1.5 Діаграма: Варінтів використання

Діаграма: Варіантів використання (Use Case Diagram)

Адміністратор програми має найвищі права доступу і може виконувати всі операції в системі. Однією з основних функцій адміністратора є реєстрація нових користувачів. Крім того, адміністратор має можливість надавати права доступу різним користувачам, визначаючи, які функції вони можуть виконувати. Адміністратор може надавати права для редагування запчастин, що дозволяє користувачам додавати, редагувати або видаляти запчастини з бази даних. Також адміністратор може дозволити користувачам формувати звіти про рух товарів, залишки на складі або інші важливі дані. Крім того, адміністратор може дозволити користувачам додавати нові запчастини до бази.

Користувач без наданих прав має обмежений функціонал і може лише переглядати список наявних запчастин на складі. Такий користувач не має доступу до редагування даних, формування звітів чи додавання нових запчастин.

Якщо адміністратор надає користувачу певні права доступу, цей користувач може виконувати додаткові функції, залежно від того, які права були йому надані. Наприклад, користувач може отримати можливість редагувати інформацію про запчастини, формувати звіти про залишки на складі або додавати нові запчастини до бази даних.

Водночас лише адміністратор має право реєструвати нових користувачів та змінювати їхні права доступу. Це означає, що адміністратор контролює, хто і до яких функцій має доступ у системі, що забезпечує безпеку та керованість усіх процесів програми.

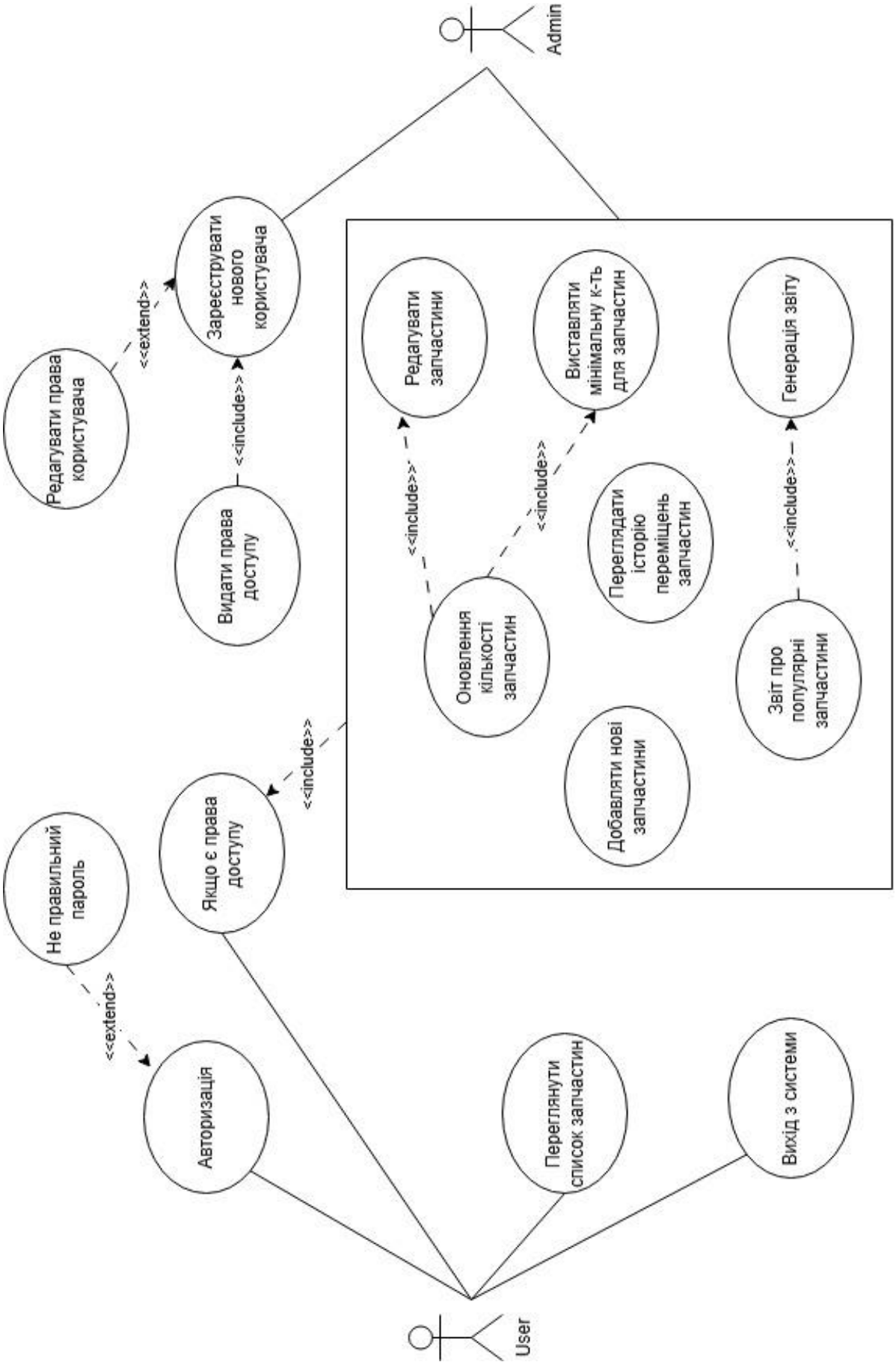


Рис.1.6 Діаграма варіантів використання

2 ЗАСОБИ РЕАЛІЗАЦІЇ

2.1 Середовище розробки

Інтегроване середовище розробки (IDE) є незамінним інструментом для програмістів, яке надає всі необхідні функціональні можливості для ефективної розробки, тестування та налагодження програмного коду. Використання IDE дозволяє значно знизити час, витрачений на розробку програмного забезпечення, а також підвищити продуктивність розробників завдяки інтеграції різних інструментів в єдине середовище. До основних функцій IDE входять текстовий редактор з підтримкою підсвічування синтаксису, інструменти для автоматизації процесу збірки програми, налагоджувач (debugger) для виявлення та усунення помилок, який дпомагає в процесі налагодження програмного коду.[13]

2.1.1 Visual Studio

Для реалізації програмного забезпечення в цьому проєкті було обрано Visual Studio - повнофункціональне інтегроване середовище розробки (IDE), яке розроблено компанією Microsoft і є одним з найпопулярніших інструментів серед розробників.[10] Це середовище має багатий набір функцій і інструментів, що дозволяють ефективно працювати над програмними проектами будь-якої складності. Visual Studio підтримує роботу з різними мовами програмування, зокрема C#, C++, Visual Basic, Python, JavaScript, TypeScript, а також численними бібліотеками та фреймворками. Це забезпечує великий спектр можливостей для розробки різноманітних додатків для різних платформ і під різні вимоги.

Основні можливості Visual Studio:

Підтримка широкого спектру мов програмування: Visual Studio надає підтримку для багатьох мов програмування, зокрема C#, C++, Python, JavaScript, TypeScript і багатьох інших. Це дозволяє розробникам вибирати найкращу мову для конкретного завдання або працювати з декількома мовами в одному проєкті. Для

свого проекту що передбачає собою програмне забезпечення для обліку автозапчастин, була обрана мова C#, оскільки вона є оптимальним вибором для створення надійних і швидких десктопних додатків на платформі .NET.

Інтеграція з .NET: Visual Studio надає середовище для розробки додатків, що використовують .NET Framework, що є основою для створення програм на Windows. Для моєї програми, яка повинна працювати на операційних системах Windows, інтеграція з .NET дозволяє ефективно використовувати всі можливості цієї платформи для управління пам'яттю, обробки подій та взаємодії з користувачем. Це також дозволяє максимально спростити роботу з базами даних і забезпечити високу швидкість виконання програми.

Налагодження та тестування: Visual Studio має вбудовані інструменти для налагодження програмного коду, що дозволяє розробникам ефективно перевіряти свої програми на наявність помилок та багів. Зручний debugger дозволяє відслідковувати виконання коду, значення змінних, виконання умов і навіть виставляти точки зупинки, щоб зупиняти виконання програми в потрібному місці для детального аналізу. Це допомагає знаходити і виправляти помилки, що підвищує якість коду і знижує час на розробку.

Інтеграція з системами контролю версій: Visual Studio має вбудовану підтримку для роботи з системами контролю версій, такими як Git. Це дозволяє команді розробників працювати з різними гілками коду, відслідковувати зміни, повертатися до попередніх версій та ефективно співпрацювати. Завдяки інтеграції з Git, всі зміни зберігаються і можуть бути відновлені, що гарантує стабільність та безпеку в роботі над великими проєктами.

Інтерфейс для роботи з базами даних: Для моєї програми важливим аспектом є робота з базами даних, оскільки програмне забезпечення повинно зберігати дані про автозапчастини, їх кількість, постачальників, категорії тощо. Visual Studio має вбудовані інструменти для взаємодії з реляційними базами даних, такими як SQLite або SQL Server, що спрощує інтеграцію бази даних у додаток. Ці інструменти дозволяють без проблем здійснювати запити до бази даних, виконувати CRUD-

операції (створення, читання, оновлення та видалення даних) і забезпечують високу ефективність роботи з базою.

Переваги використання Visual Studio в моєму проєкті:

Висока інтеграція з іншими продуктами Microsoft: Visual Studio чудово працює в екосистемі Microsoft, що дозволяє без зайвих зусиль інтегрувати застосунок з такими продуктами, як Azure, SQL Server та іншими інструментами для хмарних обчислень та аналізу даних. Це полегшує роботу з великими обсягами даних та дозволяє створювати високопродуктивні додатки.

Підтримка великих проектів та команд: Visual Studio ідеально підходить для роботи з великими проєктами, оскільки має інтерфейс і дозволяє працювати з великою кількістю файлів одночасно. Вона також включає інструменти для налаштування проектів, що дають можливість керувати великою кількістю командних розробок.

Розширені можливості для розробки додатків: Завдяки підтримці різних мов програмування та фреймворків, Visual Studio дозволяє розробляти додатки для різних платформ і операційних систем, включаючи Windows, Android та iOS. Це робить її ідеальним вибором для розробників, які працюють з .NET та потребують інструменту для комплексної розробки програмного забезпечення.

Visual Studio є надзвичайно і багатофункціональним середовищем для розробки програмного забезпечення. Для мого проєкту, який полягає в створенні системи обліку автозапчастин, використання Visual Studio дозволило ефективно розробити програму, що підтримує всі необхідні функції для управління складом, працюючи з базами даних та маючи зручний інтерфейс для користувачів.

Microsoft Excel використовується в програмі для обліку автозапчастин для роботи з таблицями та обробки даних. Це дозволяє ефективно взаємодіяти з даними, які зберігаються в програмі, і забезпечує можливість експорту та імпорту даних безпосередньо з програми в Excel. Важливим інструментом для цього є Microsoft.Office.Interop.Excel, який дозволяє створювати та редагувати Excel-файли з програмного забезпечення, що є необхідним для формування звітів і аналізу даних.

Однією з основних можливостей використання Excel є експорт даних. Програма дозволяє формувати звіти про залишки запчастин, операції на складі у форматі Excel, що забезпечує зручність у подальшому використанні цих даних. Це особливо корисно для ведення бухгалтерії або звітності в інших частинах бізнесу, де потрібна зручна і надійна інформація.

Microsoft Excel має низку переваг, серед яких простота у використанні для кінцевих користувачів. Більшість людей знайомі з базовими функціями Excel, що дозволяє швидко освоїти роботу з програмою без додаткового навчання. Також Excel зручний для роботи з великими обсягами даних, що є важливим для обліку запчастин і забезпечення точності інформації про запчастини.

Завдяки можливості подальшої обробки даних у таких інструментах, як Power BI, користувачі можуть проводити глибокий аналіз і будувати графіки та діаграми, що дає можливість приймати обґрунтовані рішення на основі звітів, що генеруються в Excel.

Отже, Microsoft Excel є інструментом для роботи з даними, що дає користувачам гнучкість, зручність і можливості для подальшого аналізу та обробки інформації про автозапчастини.

2.1.2 SQLite

SQLite - це вбудована реляційна база даних, яка широко використовується для зберігання даних у локальних програмах. Вона є легким і ефективним рішенням для малих і середніх проєктів, оскільки не вимагає окремого серверного програмного забезпечення, що робить її ідеальним варіантом для програм, які не потребують складної інфраструктури бази даних.[8] SQLite зберігає дані у файлі на локальному диску, що дозволяє програмам мати доступ до бази даних без потреби в постійній мережевій взаємодії. Це особливо корисно для програм, які працюють з обмеженими ресурсами або потребують швидкої роботи з невеликими обсягами даних.

SQLite підтримує стандарт SQL для роботи з даними, що дозволяє використовувати можливості запитів, зокрема для вибірки, оновлення та видалення

даних. Вона також підтримує транзакції, індекси для покращення швидкості доступу до даних та інші важливі функції реляційних баз даних. Завдяки своїй простоті та легкості у налаштуванні, SQLite є ідеальним вибором для програм, які потребують зберігання локальних даних без необхідності в складній серверній інфраструктурі. У вашій програмі SQLite використовується для зберігання даних про автозапчастини, постачальників, кількість товарів на складі та інші дані, що необхідні для ефективного управління складом автозапчастин.

У моєму проєкті є кілька важливих файлів і папок, які відповідають за роботу з базою даних, зокрема з SQLite. Ось основні можливості SQLite, які використовуються в моєму застосунку:

В моєму застосунку SQLite використовується для зберігання даних про автозапчастини, таких як їхня назва, категорія, кількість на складі, ціна та постачальник. Це дозволяє ефективно управляти інформацією про запчастини та забезпечує її швидкий доступ.

SQLite інтегрується безпосередньо в моє програмне забезпечення, не потребуючи окремого серверного рішення. Це дає змогу програмі працювати локально, швидко отримуючи доступ до необхідних даних без додаткових налаштувань.

Завдяки вбудованій природі SQLite, моя програма може швидко працювати з великими обсягами даних, що особливо важливо для обліку автозапчастин на складі, де дані повинні оновлюватися в реальному часі.

Підтримка транзакцій:

SQLite дозволяє виконувати транзакції, що забезпечує цілісність даних під час операцій з базою даних, таких як додавання нових запчастин або редагування їх кількості.

SQLite є ідеальним вибором для моєї програми завдяки простоті в налаштуванні та використанні, а також відсутності потреби в сервері для обробки бази даних. Вся інформація зберігається в одному файлі бази даних, що спрощує роботу з даними.

Хоча SQLite є легким рішенням для локальних баз даних, вона здатна підтримувати більші обсяги даних і може бути замінена на системи, якщо це буде необхідно в майбутньому, без зміни основної логіки роботи програми.

Таким чином, SQLite є ефективним інструментом для зберігання даних у моєму застосунку, дозволяючи швидко і зручно управляти даними про автозапчастини та забезпечуючи точність обліку.

2.2 Мови програмування

Мова програмування - це формальна мова, призначена для створення програм, що виконують конкретні завдання на комп'ютері або іншому пристрої. Вона дозволяє програмістам створювати інструкції, які комп'ютер або система можуть виконати для вирішення проблем або автоматизації задач. Мова програмування має свої правила синтаксису (як мають виглядати інструкції) та семантики (що ці інструкції виконують).[14]

C# (C-Sharp) - це високорівнева об'єктно-орієнтована мова програмування, розроблена компанією Microsoft у рамках платформи .NET. Вона широко використовується для створення різноманітних програм, від веб-додатків і десктопних програм до мобільних і ігрових додатків. C# поєднує в собі можливості, такі як підтримка багатопоточності, безпечна робота з пам'яттю, можливість інтеграції з іншими системами та простота у використанні. Мова має чітку структуру та синтаксис, що робить її легкою для освоєння, водночас вона є достатньо функціональною для вирішення складних завдань.[7]

C# був створений для того, щоб дозволити розробникам швидко створювати стабільні, масштабовані і надійні програми. Це досягається завдяки підтримці принципів об'єктно-орієнтованого програмування (ООП), таких як інкапсуляція, наслідування та поліморфізм. ООП дозволяє моделювати реальний світ у програмному коді, що робить програми більш структурованими і легшими для підтримки.

Однією з ключових особливостей C# є підтримка середовища .NET, яке забезпечує доступ до великої кількості бібліотек і фреймворків, що дозволяє розробникам створювати багатофункціональні програми з мінімальними зусиллями. Це середовище пропонує інструменти для роботи з базами даних, веб-технологіями, графічним інтерфейсом, а також для забезпечення безпеки та масштабованості програм.

C# має гарну інтеграцію з іншими технологіями Microsoft, такими як Azure, SQL Server, Visual Studio, що робить її ідеальним вибором для розробки програм на платформі Windows.

Особливості мови C#:

Простота використання: C# спроектований так, щоб бути зручним для програмістів усіх рівнів. Завдяки чіткій та зрозумілій синтаксичній структурі, розробникам легше писати і підтримувати код.

Підтримка багатопоточності: Завдяки вбудованим можливостям для роботи з багатопоточністю, C# дозволяє ефективно управляти кількома потоками виконання, що є важливим для розробки високопродуктивних і швидкодіючих програм.

Безпека пам'яті: C# забезпечує автоматичне керування пам'яттю через система збору сміття (garbage collection), що дозволяє програмістам не турбуватися про безпечне видалення непотрібних об'єктів і знижує ризик помилок у роботі з пам'яттю.

Інструменти для роботи з базами даних: Завдяки Entity Framework та іншим бібліотекам, C# дозволяє ефективно працювати з реляційними та нереляційними базами даних, виконувати запити, здійснювати зберігання і маніпулювання даними.

Велика екосистема: .NET пропонує велику кількість готових рішень для найрізноманітніших завдань - від створення веб-додатків (через ASP.NET) до роботи з великими даними та інтелектуальними системами. Використання C# у моєму проєкті

У моєму проєкті для управління складом автозапчастин мова програмування C# була обрана через її можливості та зручність у розробці застосунків для Windows. Вона ідеально підходить для створення багатофункціональних та масштабованих програм, що є необхідним для управління складом автозапчастин, де важливо забезпечити точність обліку та швидкий доступ до даних.

Мова C# забезпечує чудову інтеграцію з .NET платформою, що дозволяє створювати програму, яка буде ефективно працювати на операційних системах Windows, а також легко взаємодіяти з іншими технологіями Microsoft, такими як SQLite для зберігання та обробки даних про автозапчастини.

Однією з основних переваг використання C# у моєму проєкті є зручність роботи з Windows Forms, що дає можливість створювати прості, але функціональні графічні інтерфейси користувача для взаємодії з програмою. Це дає змогу користувачам, навіть без глибоких технічних знань, легко додавати, редагувати або шукати запчастини на складі.

Використання Entity Framework в поєднанні з C# дозволяє мені ефективно працювати з SQLite, що використовуються для зберігання даних про автозапчастини. Це забезпечує простоту та ефективність в обробці великих обсягів даних, забезпечуючи високу продуктивність і швидкість роботи програми.

Для створення звітів і аналізу даних я використовую інструменти C# для інтеграції з Microsoft Excel, що дозволяє експортувати та імпортувати дані з Excel безпосередньо з програми. Це є важливим інструментом для формування звітів про залишки товарів на складі, а також для аналізу операцій, таких як надходження і списання товарів.

Таким чином, C# дозволяє створити програмне забезпечення для управління складом автозапчастин, яке відповідає всім вимогам щодо зручності, продуктивності та точності обліку для ефективної роботи складу.

2.3 Інструменти для роботи з базами даних

SQLite - це вбудована реляційна база даних, яка використовується для зберігання та обробки даних у локальних застосунках. Вона є легким інструментом для малих і середніх проектів, оскільки вона не потребує окремого серверного рішення для функціонування. SQLite зберігає всі дані в одному файлі, що знаходиться на локальному диску. Це дозволяє програмі працювати швидко, зберігаючи дані локально, і не витрачаючи ресурси на обслуговування сервера, що є важливим для застосунків, які не потребують великих масштабів обробки даних [8].

Основною особливістю SQLite є те, що вона інтегрується безпосередньо в програму, що робить її ідеальним вибором для використання в програмах, які потребують локального зберігання даних. Всі операції з базою даних виконуються безпосередньо в межах програми, що дозволяє значно знизити затримки та підвищити швидкість обробки запитів. Завдяки цьому SQLite є відмінним вибором для програм, які мають обмежені вимоги до обсягу даних і потребують швидкого доступу до них. Це робить SQLite ідеальним рішенням для локальних баз даних у таких програмах, як облік складських запасів або системи управління інвентарем.

У моєму проєкті SQLite використовується як основне рішення для зберігання всіх даних про автозапчастини на складі. Вона дозволяє зберігати всю необхідну інформацію про кожну запчастину, таку як її найменування, тип, модель, кількість на складі, а також забезпечує збереження історії операцій з запчастинами, таких як надходження або списання. Всі ці дані зберігаються в базі даних, і доступ до них здійснюється через програми для управління складом, що дозволяє користувачам ефективно працювати з інформацією про запчастини.

SQLite дає змогу ефективно зберігати і обробляти великі обсяги даних, що є особливо важливим для програм, які повинні виконувати операції з багатьма запчастинами. Всі операції з додавання, редагування, видалення та пошуку запчастин відбуваються через базу даних, що дозволяє підтримувати актуальність і точність обліку.

Завдяки своєму компактному формату та високій швидкості доступу до даних, SQLite ідеально підходить для використання в програмі для обліку складських запасів.

Використання SQLite в моєму застосунку також дозволяє забезпечити простоту в інтеграції з іншими компонентами системи, що робить її чудовим вибором для розробки програм, які потребують локального зберігання даних без додаткових складнощів з налаштуванням серверів баз даних. Вона забезпечує достатньо високу продуктивність і при цьому є дуже легким і зручним рішенням для зберігання даних на складі.

Це також дає можливість розробити програму, яка буде працювати навіть без підключення до мережі, зберігаючи всі дані локально. У випадку потреби в масштабуванні програми в майбутньому можна буде перейти на більш високоефективні рішення для зберігання та обробки даних, при цьому базова логіка програми залишиться незмінною. Таким чином, SQLite у моєму застосунку дозволяє досягти ідеального балансу між простотою використання, продуктивністю та надійністю зберігання даних.

2.4 Використовувані фреймворки

Фреймворк - це набір інструментів, бібліотек і компонентів, які полегшують процес розробки програмного забезпечення. Він автоматизує багато аспектів створення додатків, зокрема надає структуру для організації коду, забезпечує стандартні рішення для часто використовуваних задач, таких як взаємодія з базами даних, обробка запитів або створення інтерфейсу користувача. Використання фреймворків дозволяє розробникам зосередитися на реалізації унікальних функцій додатку, замість того, щоб витрачати час на вирішення стандартних завдань, які вже були вирішені фреймворком.

У моїй програмі для обліку автозапчастин використовуються кілька фреймворків, що значно спрощують процес розробки та забезпечують ефективну

роботу програми. Кожен з цих фреймворків має свою роль і вносить свій вклад у загальну архітектуру програми.

2.4.1 Entity Framework

Entity Framework (EF) - це об'єктно-реляційний мапер (ORM), що є частиною платформи .NET. Він дозволяє працювати з базою даних як з об'єктами .NET, що значно спрощує процес збереження та доступу до даних.[11] Завдяки Entity Framework розробники можуть працювати з базою даних, не пишучи складні SQL-запити вручну, оскільки EF автоматично генерує необхідні запити для виконання операцій з даними. Це дозволяє зосередитись на логіці програми замість того, щоб писати код для взаємодії з базою.

Однією з ключових особливостей Entity Framework є підтримка моделей даних, де кожен об'єкт програми відображається на таблицю в базі даних. Це дозволяє легко працювати з даними, використовуючи об'єктно-орієнтовані концепції, замість того щоб безпосередньо писати SQL-код. Міграції EF дозволяють автоматично оновлювати структуру бази даних, зберігаючи актуальність даних при змінах у моделях програми. Ця функція робить процес підтримки бази даних простішим і більш зручним для розробника.

Entity Framework підтримує автоматичне відображення класів на таблиці бази даних, що зменшує кількість коду, необхідного для виконання операцій з даними. Це знижує ризик помилок і прискорює процес розробки, оскільки значну частину роботи з базою даних виконує сам фреймворк.

У моїй програмі, Entity Framework використовується для взаємодії з базою даних SQLite. Це дозволяє автоматизувати створення запитів до бази даних, спрощує роботу з даними і забезпечує зручне зберігання інформації про автозапчастини, постачальників та залишки на складі в структурованому вигляді.[9] Таким чином, Entity Framework полегшує роботу з базою даних і допомагає підтримувати її актуальність і цілісність без необхідності писати SQL-запити вручну.

2.4.2 .NET Framework

Один з основних фреймворків, що використовуються в моєму проєкті, це .NET Framework, який є основною платформою для розробки програм на мові C#. .NET Framework забезпечує широкий набір бібліотек для роботи з різними компонентами програми, такими як графічний інтерфейс користувача, обробка даних, доступ до баз даних, мережеві з'єднання та багато іншого.[7] Завдяки цьому фреймворку я можу використовувати готові рішення для роботи з базами даних (через Entity Framework), створення користувацького інтерфейсу (через Windows Forms), а також взаємодії з іншими частинами програми. Це дозволяє зосередитися на розробці специфічних функцій для обліку автозапчастин, таких як додавання та редагування товарів на складі, генерування звітів та інше.

2.4.3 Windows Forms

Windows Forms (WinForms) - це технологія для створення графічних інтерфейсів користувача (GUI) в десктопних додатках на платформі .NET. WinForms дозволяє швидко створювати зручні інтерфейси для користувачів завдяки великій кількості вбудованих елементів управління, таких як кнопки, текстові поля, таблиці та панелі. Ця технологія є однією з найстаріших для розробки додатків для операційної системи Windows, але й донині залишається популярною завдяки своїй простоті використання та гнучкості.

Однією з ключових особливостей Windows Forms є можливість швидко і зручно створювати інтерфейси користувача за допомогою візуальних елементів, які можна просто перетягувати на форму. Це дозволяє значно зменшити час на розробку графічного інтерфейсу для додатка, особливо для тих користувачів, які не мають глибоких знань у дизайні інтерфейсів.

Windows Forms використовує модель подій для обробки взаємодії користувача з елементами інтерфейсу. Це означає, що кожен елемент інтерфейсу, наприклад, кнопка або текстове поле, може бути налаштований для виконання певних дій (наприклад, відкриття вікна, обробка введених даних) під час натискання або інших подій.

У контексті вашої програми Windows Forms використовується для створення інтерфейсу користувача, через який користувачі можуть взаємодіяти з даними про автозапчастини, додавати нові записи, редагувати існуючі та переглядати звіти про залишки на складі. Ця технологія надає простий і інтуїтивно зрозумілий інтерфейс для введення та відображення інформації, що робить взаємодію з програмою зручною і швидкою.

2.5 Інструменти проєктування інтерфейсу користувача

Для створення інтерфейсу користувача в програмі для обліку автозапчастин використовуються інструменти, які дозволяють швидко й ефективно розробляти графічні інтерфейси для десктопних додатків. У моєму проєкті основним інструментом для створення інтерфейсу є Windows Forms, що є частиною популярної платформи .NET від компанії Microsoft. Windows Forms є технологією для створення графічних інтерфейсів користувача (GUI), яка дозволяє розробникам створювати прості та функціональні інтерфейси для застосунків, які працюють на операційних системах Windows.[12]

Windows Forms дозволяє використовувати широкий набір стандартних елементів управління, таких як кнопки, текстові поля, списки, таблиці, панелі та інші компоненти, необхідні для створення повноцінного інтерфейсу користувача. Всі ці елементи можна використовувати для розробки інтерфейсу, через який користувачі взаємодіють з програмою. Зокрема, в моєму проєкті Windows Forms використовується для того, щоб користувачі могли переглядати, додавати та редагувати дані про автозапчастини, а також генерувати звіти про рух товарів і залишки на складі.

Основною перевагою використання Windows Forms є швидкість і зручність розробки. Завдяки великій кількості готових елементів управління та інтеграції з іншими інструментами для розробки, створення інтерфейсу в Windows Forms займає мінімум часу, дозволяючи фокусуватися на функціональності додатку.

Користувач отримує інтуїтивно зрозумілий інтерфейс, що дозволяє швидко працювати з програмою і виконувати необхідні операції без зайвих труднощів.

Для обліку автозапчастин в моєму застосунку інтерфейс включає кілька ключових компонентів: таблицю для перегляду даних про запчастини, форми для додавання нових запчастин, а також кнопки для редагування та видалення записів. У таблиці користувачі можуть переглядати основні характеристики запчастин, такі як їхня назва, категорія, кількість на складі. Усі ці дані зберігаються у базі даних, а інтерфейс Windows Forms забезпечує зручний доступ до них, що дозволяє користувачам легко знаходити потрібну запчастину і виконувати необхідні операції.

Visual Studio Designer - це інструмент, що дозволяє швидко та зручно розробляти інтерфейси користувача без необхідності писати багато коду вручну. У цьому середовищі розробник може просто перетягувати елементи керування, такі як кнопки, поля вводу, списки та таблиці, на форму, налаштовувати їхні властивості та визначати, як вони будуть взаємодіяти з користувачем. Це значно прискорює процес розробки інтерфейсу і дає можливість швидко створювати робочі макети без великих витрат часу.

За допомогою Visual Studio Designer можна легко додавати нові елементи на форму, налаштовувати їхні параметри, такі як розмір, колір, шрифт і навіть прив'язку до бази даних або інших об'єктів програми. Кожен елемент керування має інтерфейс для налаштування подій, таких як натискання кнопки, введення тексту або вибір зі списку. Це дозволяє створювати інтерактивні форми, які можуть взаємодіяти з користувачем, наприклад, для додавання нових запчастин на склад, редагування інформації або формування звітів.

Основна мета при розробці інтерфейсу - це створити зручний і інтуїтивно зрозумілий інтерфейс, який би полегшив користування програмою, забезпечуючи легкість у виконанні основних операцій. Windows Forms дозволяє реалізувати такі інтерфейси, де кожен елемент має чітке призначення, а користувачам не доводиться витрачати час на пошук необхідних функцій.

За допомогою Visual Studio Designer процес створення таких інтерфейсів став ще простішим, оскільки можна зосередитися на бізнес-логіці програми, залишаючи частину роботи з дизайном автоматичною.

Windows Forms є інструментом для створення графічних інтерфейсів для десктопних програм на платформі .NET, що дає змогу швидко і зручно створювати інтуїтивно зрозумілі інтерфейси. Завдяки використанню Visual Studio Designer процес проєктування інтерфейсу стає простим та ефективним, дозволяючи розробникам зосередитися на основній логіці програми, а не на написанні складного коду для побудови інтерфейсу.

Це дозволяє значно прискорити розробку та підвищити якість кінцевого продукту.

3. ПРОЄКТУВАННЯ

3.1 Проєктування архітектури програмного забезпечення

Проєктування архітектури програмного забезпечення для управління складом автозапчастин є важливим етапом у розробці системи, оскільки саме від архітектури залежить ефективність роботи програми, її масштабованість і зручність для користувача. В даному випадку, програма повинна забезпечити надійний облік автозапчастин, контроль за їх рухом, а також можливість генерації звітів для подальшого аналізу та планування. Тому архітектура повинна бути побудована таким чином, щоб забезпечити гнучкість у додаванні нових функцій, зручність у використанні та надійність у роботі з даними.

Архітектура системи базується на модульному підході. Всі основні компоненти системи чітко розподілені: інтерфейс користувача (UI), бізнес-логіка (контролери) та база даних. Такий підхід дозволяє незалежно оновлювати чи змінювати кожен з компонентів без впливу на інші частини системи. Основними складовими є інтерфейс користувача, бізнес-логіка та взаємодія з базою даних, що забезпечує зберігання всіх необхідних даних про автозапчастини.

Інтерфейс користувача (UI) є основним компонентом, через який здійснюється взаємодія користувача з програмою. Він створюється на основі Windows Forms, що дозволяє легко реалізувати форми для введення та редагування даних. Користувач має змогу переглядати наявні автозапчастини на складі, здійснювати пошук по товару, додавати нові запчастини та створювати звіти. Інтерфейс має бути простим та інтуїтивно зрозумілим, що дає змогу швидко освоїти програму.[12]

Бізнес-логіка (контролери) є основною частиною програми, що відповідає за обробку даних. Контролери отримують запити від інтерфейсу користувача, обробляють їх і передають відповідні дані в базу даних або виводять їх на екран. Вони здійснюють перевірку правильності введених даних, управління обліком

товарів на складі, а також виконують функції для генерації звітів. Наприклад, коли користувач додає нову запчастину, контролер перевіряє введені дані, створює новий об'єкт запчастини і передає його в базу даних для зберігання.

Для зберігання даних програма використовує базу даних, яку можна реалізувати за допомогою SQLite або іншої реляційної бази даних. У базі зберігається вся інформація про запчастини, постачальників, рух товарів на складі, а також історія операцій (додавання, списання, переміщення товарів). Взаємодія з базою даних здійснюється через Entity Framework або інші ORM (об'єктно-реляційні мапери), що спрощують роботу з даними і зменшують потребу в написанні складних SQL запитів.

Крім того, важливим етапом є інтеграція програми з іншими системами підприємства. Це може бути бухгалтерська система або система управління відносинами з клієнтами (CRM), яка дозволяє автоматично обмінювати дані між різними частинами бізнесу, що значно підвищує ефективність роботи програми та зменшує ймовірність помилок.

Для забезпечення розширюваності системи передбачена можливість додавання нових функцій, таких як облік автозапчастин через штрих-коди або інтеграція з іншими базами даних для покращення роботи складу.

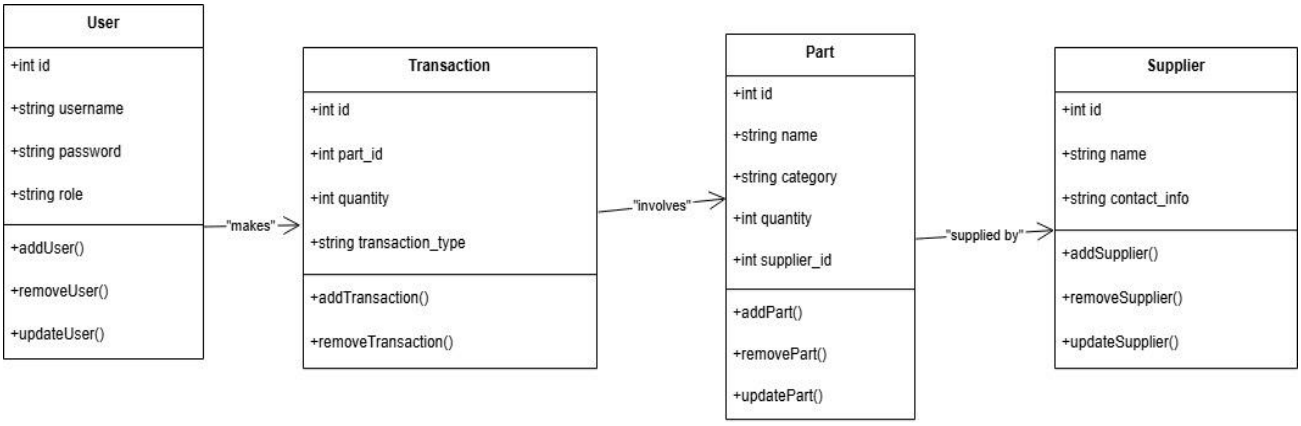


Рис. 3.1 Діаграма класів

3.2 Архітектура бази даних

Архітектура бази даних є ключовим компонентом програмного забезпечення для управління складом автозапчастин. Вона повинна забезпечити ефективне зберігання, обробку та доступ до даних про запчастини, рух товарів і користувачів системи.[14] База даних повинна підтримувати всі необхідні операції, такі як додавання, редагування, видалення даних, а також забезпечити точність обліку товарів на складі.

1. Структура бази даних

Для зберігання даних у програмі використовуються реляційні таблиці, кожна з яких відповідає за окрему сутність (наприклад, запчастини, користувачі, документи, історія руху товарів тощо). Усі таблиці бази даних взаємопов'язані за допомогою зовнішніх ключів (FK), що дозволяє забезпечити цілісність даних і коректну взаємодію між таблицями.

Основні таблиці бази даних:

Users (Користувачі):

Таблиця для зберігання інформації про користувачів, таких як ім'я, пароль, права доступу, роль.

Зв'язок: кожен користувач може створювати документи, що фіксуються в таблиці Document.

Parts (Запчастини):

Зберігаються дані про автозапчастини: їхні назви, типи, кількість на складі.

Зв'язок: кожна запчастина може належати певній категорії (таблиця Part_Type) і автомобілю (таблиця Car).

Transactions (Транзакції):

Таблиця для зберігання операцій з запчастинами, таких як їх надходження на склад або списання.

Зв'язок: кожна транзакція пов'язана з певною запчастиною та користувачем, що її здійснив.

Notifications (Сповіщення):

Сповіщення про необхідність поповнення запасів на складі (наприклад, коли кількість товару падає нижче мінімального значення).

Зв'язок: сповіщення пов'язані з конкретними запчастинами, і вони містять мінімальну кількість, при досягненні якої спрацьовує сповіщення.

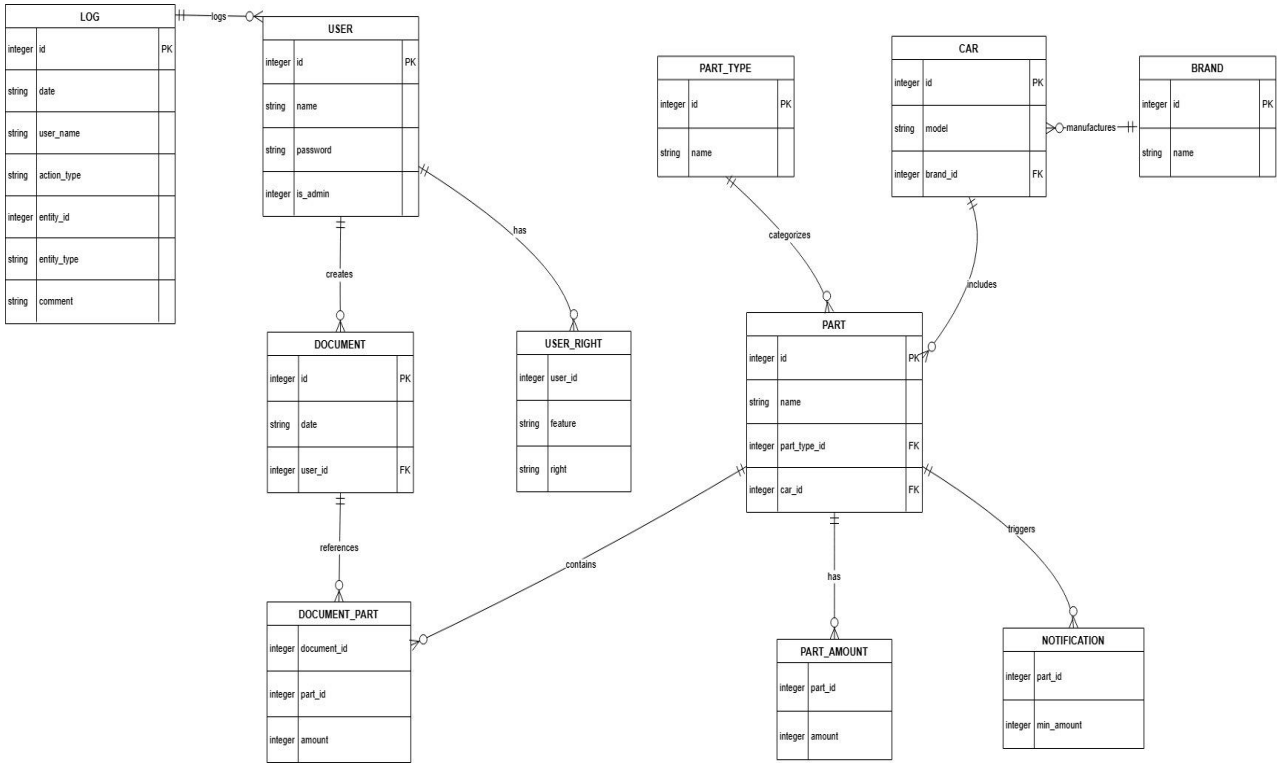


Рис3.2 Архітектура бази даних

На діаграмі зображена структура бази даних з основними таблицями: Users, Parts, Transactions, Notifications, Part_Type, Car та зв'язки між ними. Зовнішні ключі в таблицях забезпечують зв'язок між даними про запчастини, користувачів, транзакції та сповіщення.

Зв'язки між таблицями:

Таблиця Users містить дані про користувачів, а також інформацію про їхні права доступу (через зв'язок з таблицею User_Right).

Зв'язок з таблицею Documents: кожен користувач може створювати документи на склад, що відображають рух товарів.

Таблиця Parts є основною для зберігання даних про автозапчастини, а також містить зв'язок з таблицею Part_Type для визначення типу кожної запчастини.

Зв'язок з таблицею Car: кожна запчастина може бути частиною конкретного автомобіля.

Таблиця Transactions містить інформацію про всі операції з товарами. Кожна транзакція відображає зміну кількості запчастини на складі.

Зв'язок з таблицею Part: транзакція пов'язана з конкретною запчастиною, що дозволяє відслідковувати її рух.

Таблиця Notifications інформує про необхідність поповнення запасів. Вона відслідковує мінімальну кількість для кожної запчастини.

Зв'язок з таблицею Part: кожне сповіщення відноситься до певної запчастини і містить дані про мінімальний запас.

3.3 Архітектура застосунку

Програма для обліку автозапчастин побудована на основі платформи .NET, що забезпечує зручну і ефективну розробку додатків для Windows. Архітектура застосунку включає кілька основних компонентів, що працюють разом для забезпечення необхідного функціоналу обліку і управління запасами автозапчастин.

1. Інтерфейс користувача (UI):

Для створення графічного інтерфейсу користувача використовується технологія Windows Forms. Ця технологія дозволяє швидко розробляти інтуїтивно зрозумілі інтерфейси, де користувачі можуть взаємодіяти з системою. Програма надає можливість:

Реєструвати нових користувачів та керувати їх доступом.

Додавати, редагувати та видаляти записи про автозапчастини.

Переміщати запчастини між складами.

Генерувати звіти про наявність товарів на складі.

2. Логіка бізнесу:

Основна логіка програми, що обробляє введені дані та взаємодіє з базою даних, реалізована через класи Inventory, Part та Document.

Вони відповідають за:

Додавання нових запчастин до бази даних.

Оновлення кількості запчастин.

Переміщення товарів між користувачами.

3. База даних (SQLite):

Програма використовує SQLite для зберігання інформації про автозапчастини, їх кількість, постачальників, категорії тощо. SQLite дозволяє зберігати всі дані локально, що є зручним для малих і середніх підприємств, оскільки не вимагає налаштування окремого серверу бази даних. Взаємодія з базою даних здійснюється через Entity Framework, що спрощує доступ до даних і дозволяє виконувати запити на основі об'єктів.

4. Модуль обліку користувачів:

Програма містить функціонал для авторизації користувачів, що дозволяє обмежити доступ до певних частин програми в залежності від рівня доступу користувача (наприклад, адміністратор або звичайний користувач). Адміністратор може керувати правами доступу та реєстрацією нових користувачів.

5. Модуль звітності:

Програма включає можливість формувати звіти про запчастини, рух товарів та залишки на складах. Звіти можна експортувати у популярні формати, такі як Excel, для подальшого аналізу і зберігання.

Тож архітектура застосунку побудована таким чином, щоб забезпечити ефективне управління складом автозапчастин, зручний інтерфейс для користувачів і швидкий доступ до даних. Використання SQLite і Windows Forms дозволяє створити рішення для малих і середніх підприємств з обліку запасів без складних налаштувань.

4 ДИЗАЙН ІНТЕРФЕЙСУ КОРИСТУВАЧА ТА ТЕСТУВАННЯ ЗАСТОСУНКУ

4.1 Дизайн інтерфейсу користувача

Інтерфейс користувача для програми управління складом автозапчастин розроблений з акцентом на зручність, швидкість виконання операцій і інтуїтивну зрозумілість для користувачів. Програма використовує Windows Forms, що дозволяє створювати ефективний і функціональний графічний інтерфейс для взаємодії користувача з даними.

Основні елементи інтерфейсу включають такі компоненти, як форми для введення, редагування та перегляду даних, таблиці для відображення списку запчастин, а також кнопки і меню для навігації між функціями програми.

1. Головна форма

Головна форма програми є точкою входу для користувача. Вона містить панель навігації, де користувач може вибрати один із розділів програми:

Запчастини - для перегляду і керування списком автозапчастин.

Звіти - для генерації звітів про рух товарів, залишки, надходження.

Пошук - для швидкого пошуку потрібної запчастини за різними параметрами (назва, категорія, кількість).

Ця форма забезпечує користувачу легкий доступ до всіх основних функцій програми, при цьому всі елементи інтерфейсу логічно організовані для швидкої навігації.

2. Форма для додавання та редагування запчастини

Ця форма дозволяє користувачеві вводити нові дані або редагувати існуючі. У формі є кілька основних полів:

Назва запчастини - текстове поле для введення назви автозапчастини.

Категорія - випадаючий список для вибору категорії запчастини (наприклад, "двигун", "гальма", "коробка передач").

Кількість - поле для введення кількості запчастин на складі.

Користувач може заповнити ці поля і натискати кнопку "Зберегти" для збереження даних у базі або "Скасувати" для скасування змін.

3. Таблиця для перегляду запчастин

Таблиця є основним елементом інтерфейсу для перегляду даних про автозапчастини, що зберігаються в базі. Вона містить кілька стовпців, що дозволяють швидко знайти потрібну інформацію:

Бренд

Модель

Тип

Назва

Кількість

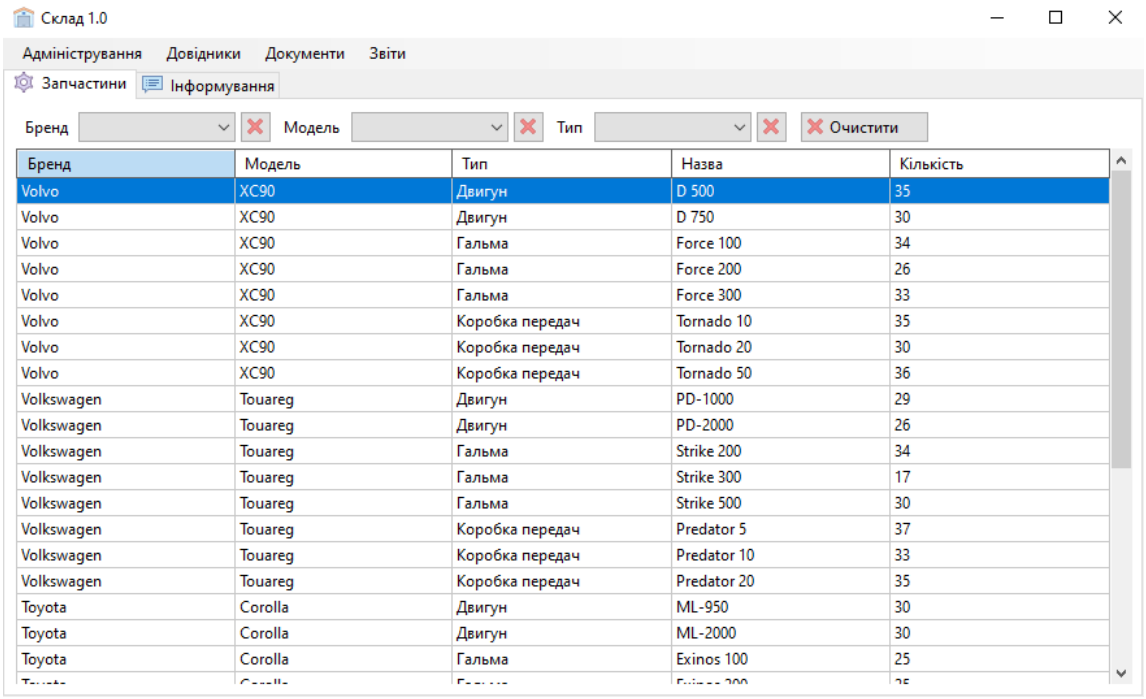
Таблиця також дозволяє фільтрувати та сортувати дані по будь-якому з цих стовпців, що значно спрощує пошук необхідної запчастини. Користувач може вибирати записи, редагувати їх або видаляти.

4. Форма для генерації звітів

Цей компонент дозволяє користувачу генерувати звіти на основі даних складу. Користувач може вибрати тип звіту (наприклад, звіт про залишки запчастин, обороти запчастин, популярні запчастини). Після натискання кнопки звіта, система формує звіт, який можна переглянути на екрані у форматі Excel.

5. Навігація та зручність користування

Для зручності користувачів у програмі використовується панель навігації, де всі основні функції програми згруповані за категоріями. Панель дозволяє легко перемикатися між різними розділами програми. Кожен розділ має свою вкладку, що дозволяє користувачеві зручно взаємодіяти з різними аспектами програми без необхідності постійно перемикатися між вікнами.



Бренд	Модель	Тип	Назва	Кількість
Volvo	XC90	Двигун	D 500	35
Volvo	XC90	Двигун	D 750	30
Volvo	XC90	Гальма	Force 100	34
Volvo	XC90	Гальма	Force 200	26
Volvo	XC90	Гальма	Force 300	33
Volvo	XC90	Коробка передач	Tornado 10	35
Volvo	XC90	Коробка передач	Tornado 20	30
Volvo	XC90	Коробка передач	Tornado 50	36
Volkswagen	Touareg	Двигун	PD-1000	29
Volkswagen	Touareg	Двигун	PD-2000	26
Volkswagen	Touareg	Гальма	Strike 200	34
Volkswagen	Touareg	Гальма	Strike 300	17
Volkswagen	Touareg	Гальма	Strike 500	30
Volkswagen	Touareg	Коробка передач	Predator 5	37
Volkswagen	Touareg	Коробка передач	Predator 10	33
Volkswagen	Touareg	Коробка передач	Predator 20	35
Toyota	Corolla	Двигун	ML-950	30
Toyota	Corolla	Двигун	ML-2000	30
Toyota	Corolla	Гальма	Exinos 100	25

Рис 4.1 Екрана форма головного вікна програми

4.2 Реалізація функціональних можливостей

Конструктор EditBrandsController ініціалізує контролер, форму та список брендів. Він отримує всі бренди з бази даних і додає їх до відображення на формі.

```
using Database;
using Database.Entities;
using Sklad.Extensions;

namespace Sklad.Forms.EditBrands
{
    2 references
    internal class EditBrandsController
    {
        private readonly Db _db;
        private readonly EditBrandsForm _form;
        private readonly List<Brand> _brands;

        1 reference
        public EditBrandsController(EditBrandsForm form)
        {
            _db = Factory.GetDb();
            _form = form;
            _brands = _db.Brands.GetAll();
            _form.BrandGrid.AddBrands(_brands);
        }
```

Рис. 4.2 Ініціалізація та отримання даних

Метод CreateBrand створює новий бренд, перевіряє наявність вже існуючих назв і відкриває форму для редагування. Після успішного введення даних, новий бренд додається до бази та оновлюється на формі.

```
1 reference
public void CreateBrand()
{
    var brand = new Brand();
    var existingNames = GetExistingNames();

    using var editBrandForm = new EditBrandForm(brand, existingNames);
    if (editBrandForm.ShowDialog() == DialogResult.OK)
        AddBrand(editBrandForm.GetBrand());
}
```

Рис. 4.3 Створення нового бренду

Редагування існуючого бренду, метод ChangeBrand отримує вибраний бренд, перевіряє наявність інших брендів з тією ж назвою, і відкриває форму для редагування. Після внесення змін, бренд оновлюється в базі даних та відображається на формі.

```
2 references
public void ChangeBrand()
{
    var brand = _form.BrandGrid.GetSelectedBrand();
    var existingNames = GetExistingNames(brand.Id);

    using var editBrandForm = new EditBrandForm(brand, existingNames);
    if (editBrandForm.ShowDialog() == DialogResult.OK)
        UdateBrand(editBrandForm.GetBrand());
}
```

Рис. 4.4 Редагування існуючого бренду

Видалення бренду, метод DeleteBrand перевіряє, чи використовується бренд у базі даних. Якщо так, користувач отримує повідомлення, що не можна видалити цей бренд. Якщо бренд не використовується, з'являється підтвердження видалення, після чого бренд видаляється з бази і форми.

```
1 reference
public void DeleteBrand()
{
    var brand = _form.BrandGrid.GetSelectedBrand();
    var isInUse = _db.Brands.IsInUse(brand.Id);

    if (isInUse)
    {
        _form.InformBrandIsInUse(brand.Name);
        return;
    }

    if (!_form.ConfirmDeleteBrand(brand.Name))
        return;

    _db.Brands.Delete(brand.Id);
    _brands.RemoveAll(x => x.Id == brand.Id);
    _form.BrandGrid.DeleteBrand(brand.Id);
}
```

Рис. 4.5 Видалення бренду

Ця частина коду визначає три методи: AddBrand, UpdateBrand і GetExistingNames.

AddBrand(Brand brand): Метод додає новий бренд до бази даних, отримує його ID та оновлює список брендів у пам'яті, після чого оновлює відображення бренду на формі.

UpdateBrand(Brand brand): Метод оновлює дані існуючого бренду в базі, отримує оновлену інформацію та замінює стару версію бренду в пам'яті, а також оновлює його на формі.

GetExistingNames(long idToExclude = Entity.NotAssignedId): Метод отримує список усіх назв брендів, за винятком тієї, що має зазначений ID. Це забезпечує уникнення дублювання назв при додаванні або редагуванні брендів.

```
1 reference
private void AddBrand(Brand brand)
{
    var brandId = _db.Brands.Add(brand);
    brand = _db.Brands.Get(brandId);

    _brands.Add(brand);
    _form.BrandGrid.AddBrand(brand);
}

1 reference
private void UpdateBrand(Brand brand)
{
    _db.Brands.Update(brand);
    brand = _db.Brands.Get(brand.Id);

    _brands.Replace(brand);
    _form.BrandGrid.UpdateBrand(brand);
}

2 references
private HashSet<string> GetExistingNames(long idToExclude = Entity.NotAssignedId)
{
    return _brands
        .Where(x => x.Id != idToExclude)
        .Select(x => x.Name)
        .ToHashSet();
}
```

Рис. 4.6 Методи додавання та оновлення брендів.

Модель, тип і назва працюють аналогічно.

Клас LoginController відповідає за управління процесом входу в систему. У його конструкторі ініціалізується підключення до бази даних через метод Factory.GetDb(). Метод GetUsers отримує список користувачів з бази даних, сортує їх за ім'ям та повертає результат у вигляді списку.

```
using Database;
using Database.Entities;

namespace Sklad.Forms.Login
{
    3 references
    internal class LoginController
    {
        private readonly Db _db;

        1 reference
        public LoginController()
        {
            _db = Factory.GetDb();
        }

        1 reference
        public List<User> GetUsers()
        {
            return _db.Users.GetAll()
                .OrderBy(x => x.Name)
                .ToList();
        }
    }
}
```

Рис.4.7 Клас LoginController

У конструкторі контролера ініціалізуються підключення до бази даних, форма та отримуються повні назви частин (через _db.GetPartFullNames()). Також завантажуються всі існуючі повідомлення та додаються до таблиці в формі.

```
using Database;
using Database.Entities;
using Sklad.Extensions;
using Sklad.Forms.EditNotification;

namespace Sklad.Forms.EditNotifications
{
    2 references
    internal class EditNotificationsController
    {
        private readonly Db _db;
        private readonly EditNotificationsForm _form;
        private readonly Dictionary<long, string> _partFullNames;

        1 reference
        public EditNotificationsController(EditNotificationsForm form)
        {
            _db = Factory.GetDb();
            _form = form;
            _partFullNames = _db.GetPartFullNames();

            var notifications = _db.Notifications.GetAll();
            _form.NotificationGrid.AddNotifications(notifications);
        }
```

Рис. 4.8 Ініціалізація та отримання даних

Створення нового повідомлення, метод CreateNotification створює нове повідомлення, отримує ідентифікатори частин для виключення та відкриває форму для редагування. Якщо форма закрита успішно, нове повідомлення додається до бази даних та відображається на формі.

```
1 reference
public void CreateNotification()
{
    var notification = new Notification();
    var partIdsToExclude = GetPartIdsToExclude();

    using var editNotificationForm = new EditNotificationForm(notification, partIdsToExclude);
    if (editNotificationForm.ShowDialog() == DialogResult.OK)
        AddNotification(editNotificationForm.GetNotification());
}
```

Рис. 4.9 Створення нового повідомлення

Редагування існуючого повідомлення, метод ChangeNotification отримує вибране повідомлення, виключає його частину з можливих для редагування, і відкриває форму для внесення змін. Після редагування інформація оновлюється в базі даних, а зміни відображаються на формі.

```
2 references
public void ChangeNotification()
{
    var notification = _form.NotificationGrid.GetSelectedNotification();
    var partIdsToExclude = GetPartIdsToExclude(notification.PartId);

    using var editNotificationForm = new EditNotificationForm(notification, partIdsToExclude);
    if (editNotificationForm.ShowDialog() == DialogResult.OK)
        UdateNotification(notification.PartId, editNotificationForm.GetNotification());
}
```

Рис. 4.10 Редагування існуючого повідомлення

Видалення повідомлення:

Метод DeleteNotification перевіряє, чи підтвердив користувач видалення повідомлення, після чого видаляє його з бази даних і оновлює інтерфейс.

```
1 reference
public void DeleteNotification()
{
    var notification = _form.NotificationGrid.GetSelectedNotification();
    var partFullName = _partFullNames[notification.PartId];

    if (!_form.ConfirmDeleteNotification(partFullName))
        return;

    _db.Notifications.Delete(notification.PartId);
    _form.NotificationGrid.DeleteNotification(notification.PartId);
}
```

Рис. 4.11 Видалення повідомлення

Метод AddNotification: Додає нове повідомлення в базу даних та оновлює відображення.

Метод UdateNotification: Оновлює існуюче повідомлення в базі та на формі.

Метод GetPartIdsToExclude: Отримує набір ідентифікаторів частин, які виключаються з можливих варіантів для вибору при редагуванні.

```
1 reference
private void AddNotification(Notification notification)
{
    _db.Notifications.Add(notification);
    _form.NotificationGrid.AddNotification(notification);
}

1 reference
private void UdateNotification(long previousPartId, Notification notification)
{
    _db.Notifications.Update(notification);
    _form.NotificationGrid.UpdateNotification(previousPartId, notification);
}

2 references
private HashSet<long> GetPartIdsToExclude(long partIdToInclude = Entity.NotAssignedId)
{
    return _form.NotificationGrid.GetNotifications()
        .Select(x => x.PartId)
        .Where(x => x != partIdToInclude)
        .ToHashSet();
}
```

Рис. 4.12 Методи керування сповіщеннями

Заповнення контенту звіту, Метод FillContent викликається для заповнення таблиці Excel даними. Спочатку задаються заголовки для стовпців: "Назва запчастини" та "Кількість". Потім створюється об'єкт для зберігання даних про частини та їх кількість (partAmounts), які фільтруються таким чином, щоб включати тільки ті частини, у яких кількість більше за нуль.

Запис даних у таблицю, для кожної частини в списку partAmounts, дані про її назву та кількість записуються в таблицю Excel в відповідні клітинки. Кількість рядків для запису збільшується з кожним циклом.

Метод також включає очищення діапазону клітинок за допомогою Marshal.ReleaseComObject, що забезпечує правильне управління пам'яттю після заповнення звіту.

В результаті цього методу створюється Excel-звіт, який містить список частин і їх кількості, відсортовані за іменами частин.

```
using System.Runtime.InteropServices;
using Microsoft.Office.Interop.Excel;
using Sklad.Extensions;

namespace Sklad.Reports
{
    1 reference
    internal class PartAmountsReport : Report
    {
        2 references
        protected override void FillContent(dynamic worksheet, dynamic range)
        {
            HeaderColumn(worksheet, range, 1, 70, "Назва запчастини");
            HeaderColumn(worksheet, range, 2, 15, "Кількість", XlHAlign.xlHAlignRight);

            var rowRange = worksheet.Rows["1:1"];
            rowRange.Font.Bold = true;
            Marshal.ReleaseComObject(rowRange);

            var partAmounts = Db.PartAmounts.GetAll().Where(x => x.Amount > 0).ToList();
            var partFullNames = Db.GetPartFullNames();
            var rowNumber = 2;

            foreach (var partAmount in partAmounts.OrderBy(x => partFullNames[x.PartId]))
            {
                range.Cells[rowNumber, 1].Value2 = partFullNames[partAmount.PartId];
                range.Cells[rowNumber, 2].Value2 = partAmount.Amount;
                rowNumber++;
            }
        }
    }
```

Рис. 4.13 Клас для генерування звітів

Цей код представляє абстрактний клас Report, який використовується для генерації звітів у форматі Excel через бібліотеку Microsoft.Office.Interop.Excel. Ось що робить кожен його компонент:

У конструкторі ініціалізується доступ до бази даних за допомогою Factory.GetDb() і зберігається в властивості Db, що дозволяє працювати з даними з бази в усіх методах класу.

Метод FillContent є абстрактним, тобто він має бути реалізований у похідних класах. Його задача - заповнити вміст звіту на переданих параметрах worksheet та range, що є об'єктами для роботи з листом Excel.

Метод Generate створює новий файл Excel, ініціалізує додаток Excel, створює нову книгу та лист, після чого викликає метод FillContent для заповнення вмісту цього листа. Після завершення заповнення звіт відображається на екрані, і об'єкти

Excel звільняються з пам'яті через Marshal.ReleaseComObject, щоб уникнути витоків пам'яті.

Метод HeaderColumn відповідає за створення заголовка для стовпця в Excel. Він приймає параметри для вказівки номера стовпця, ширини стовпця, заголовка та вирівнювання тексту. Заголовок встановлюється в першу клітинку стовпця, а також налаштовується ширина стовпця та вирівнювання тексту.

Загалом, клас Report є основою для створення звітів у Excel, де реалізація конкретного вмісту звіту здійснюється в похідних класах.

```
using Microsoft.Office.Interop.Excel;
using System.Runtime.InteropServices;
using Database;

namespace Sklad.Reports
{
    5 references
    internal abstract class Report
    {
        0 references
        protected Report()
        {
            Db = Factory.GetDb();
        }

        11 references
        protected Db Db { get; }
        5 references
        protected abstract void FillContent(dynamic worksheet, dynamic range);

        4 references
        public void Generate()
        {
            var application = new Microsoft.Office.Interop.Excel.Application();
            var workbook = application.Workbooks.Add();
            var worksheet = workbook.Sheets[1];
            var range = worksheet.UsedRange;

            FillContent(worksheet, range);

            application.Visible = true;
            Marshal.ReleaseComObject(range);
            Marshal.ReleaseComObject(worksheet);
            Marshal.ReleaseComObject(workbook);
            Marshal.ReleaseComObject(application);
        }

        13 references
        protected void HeaderColumn(dynamic worksheet, dynamic range, int columnNumber, int width, string title, XlHAlign align = XlHAlign.xlHAlignLeft)
        {
            range.Cells[1, columnNumber] = title;
            worksheet.Columns[columnNumber].ColumnWidth = width;
            worksheet.Columns[columnNumber].HorizontalAlignment = align;
        }
    }
```

Рис. 4.14 Клас “Звіт”

Цей код є частиною контролера, який працює з формою для перегляду деталей частин у програмі. Ось що робить кожна його частина:

Ініціалізація змінних, у класі зберігаються змінні для доступу до бази даних (_db), до форми для перегляду частин (_partView), а також до колекцій, що містять бренди (_brands), автомобілі (_cars), типи частин (_partTypes) та список частин (_parts).

Конструктор ініціалізує з'єднання з базою даних через Factory.GetDb(), зберігає передану форму для перегляду частин і викликає методи для відстеження змін та оновлення даних.

Метод GetBrands повертає список брендів, які відповідають певним критеріям. Він фільтрує частини, застосовуючи допоміжний метод PartHelper.IsMatch, який перевіряє, чи відповідає автомобіль і тип частини. Потім метод вибирає унікальні бренди, сортує їх за назвою та повертає результат як список.

```
namespace Sklad.Forms.Main.Parts
{
    3 references
    internal class PartViewController
    {
        private readonly Db _db;
        private readonly PartView _partView;
        private Dictionary<long, Brand> _brands;
        private Dictionary<long, Car> _cars;
        private Dictionary<long, PartType> _partTypes;
        private List<Part> _parts;

        1 reference
        public PartViewController(PartView partView)
        {
            _db = Factory.GetDb();
            _partView = partView;
            TraceDataChange();
            UpdateData();
        }

        1 reference
        public List<Brand> GetBrands()
        {
            return _parts
                .Where(x => PartHelper.IsMatch(x, _cars[x.CarId], null, _partView.PartGrid.CarId, _partView.PartGrid.PartTypeId))
                .Select(x => _brands[x.CarId])
                .Distinct()
                .OrderBy(x => x.Name)
                .ToList();
        }
```

Рис.4.15 Клас форми для перегляду деталей

Клас Factory відповідає за створення та ініціалізацію основних компонентів програми. Він відкриває підключення до бази даних SQLite за допомогою Db.Open("database.sqlite") та створює об'єкти для обробки подій і зберігання деталей програми. Статичні методи GetDb(), GetApplicationEvents() та GetApplicationDetails() надають доступ до цих компонентів для використання в інших частинах програми.

```
namespace Sklad
{
    41 references
    public class Factory
    {
        private static readonly Factory _factory = new Factory();
        private readonly Db _db;
        private readonly ApplicationEvents _applicationEvents;
        private readonly ApplicationDetails _applicationDetails;

        1 reference
        private Factory()
        {
            _db = Db.Open("database.sqlite");
            _applicationEvents = new ApplicationEvents();
            _applicationDetails = new ApplicationDetails();
        }

        26 references
        public static Db GetDb()
        {
            return _factory._db;
        }

        4 references
        public static ApplicationEvents GetApplicationEvents()
        {
            return _factory._applicationEvents;
        }

        8 references
        public static ApplicationDetails GetApplicationDetails()
        {
            return _factory._applicationDetails;
        }
    }
```

Рис.4.16 Підключення до баз даних

Цей код реалізує клас RightCheckBoxes, який управляє правами доступу користувачів через два CheckBox: для перегляду (view) і редагування (edit) даних. Метод Bind синхронізує два CheckBox, де вибір одного впливає на стан іншого, забезпечуючи логіку, що дозволяє або знімати, або встановлювати права доступу. Метод GetRight визначає тип права доступу (перегляд, редагування або відсутність прав) на основі стану CheckBox. Метод SetRight встановлює відповідні значення для CheckBox, залежно від переданого права доступу (UserRight). Цей клас дозволяє зручно обробляти права користувача в інтерфейсі, керуючи вибором прав через візуальні елементи.

```
namespace Sklad.Forms.EditUser
{
    13 references
    internal static class RightCheckBoxes
    {
        3 references
        public static void Bind(CheckBox view, CheckBox edit)
        {
            view.CheckedChanged += (_, _) =>
            {
                if (!view.Checked)
                    edit.Checked = false;
            };

            edit.CheckedChanged += (_, _) =>
            {
                if (edit.Checked)
                    view.Checked = true;
            };
        }

        3 references
        public static UserRight GetRight(CheckBox view, CheckBox edit)
        {
            if (edit.Checked)
                return UserRight.Edit;

            if (view.Checked)
                return UserRight.View;

            return UserRight.None;
        }

        2 references
        public static UserRight GetRight(CheckBox view)
        {
            if (view.Checked)
                return UserRight.View;

            return UserRight.None;
        }
```

```
        3 references
        public static void SetRight(UserRight right, CheckBox view, CheckBox edit)
        {
            if (right == UserRight.Edit)
            {
                view.Checked = true;
                edit.Checked = true;
            }
            else if (right == UserRight.View)
            {
                view.Checked = true;
                edit.Checked = false;
            }
            else
            {
                view.Checked = false;
                edit.Checked = false;
            }
        }

        2 references
        public static void SetRight(UserRight right, CheckBox view)
        {
            view.Checked = right != UserRight.None;
        }
    }
}
```

Рис. 4.17 Керування правами доступа

4.3 Тестування програмного забезпечення

Тестування програмного забезпечення є важливою частиною процесу його розробки, оскільки дозволяє перевірити коректність роботи всіх функцій та переконатися, що система працює належним чином. Важливо забезпечити, щоб усі компоненти програми функціонували відповідно до вимог, а також щоб користувачі могли зручно і безпечно взаємодіяти з програмою.

У таблиці 4.1 представлені основні тест-кейси для перевірки різних функціональних можливостей програми. Це включає авторизацію користувачів, роботу фільтрів запчастин, створення та редагування користувачів, інформування про закінчення запасів, перевірку логів роботи, додавання нових деталей, накладні, а також формування звітів. Кожен з цих тестів дозволяє перевірити, чи відповідає програмне забезпечення очікуваним результатам і чи працює система без помилок. Тестування допомагає виявити будь-які недоліки чи проблеми на ранніх етапах, що дозволяє зробити програму більш стабільною і надійною.

Нижче наведена таблиця з детальним описом кожного тест-кейсу, результатів тестування та перевірки функціональних можливостей програми.

Таблиця 4.1

Тестування основних функцій

№	Тест-кейс	OK/ NOK	Дії	Очікуваний результат
1	Авторизація	OK	Вибір користувача "Адміністратор", введення правильного пароля, натискання OK.	Система успішно авторизує користувача та перенаправить на головний екран програми.
2	Фільтр запчастин	OK	Вибір фільтрів "Бренд" та "Модель", наприклад, "Volvo" та "XC90". Натискання "Очисити".	Система відфільтрує запчастини за вибраними критеріями, при скиданні фільтрів усі записи будуть відображені.
3	Створення та редагування користувачів	OK	Натискання "Створити", введення даних користувача, натискання OK. Зміна даних користувача через "Редагувати".	Користувач буде успішно доданий до системи, зміни в даних користувача будуть збережені.

Продовження таблиці 4.1

Тестування функціональності

№	Тест-кейс	OK/ NOK	Дії	Очікуваний результат
4	Інформування про закінчення запчастин	OK	Введення запчастини та мінімальної кількості, перевірка сповіщення про запчастини з недостатньою кількістю.	Система відобразить правильну інформацію про запчастини, де кількість менша за мінімальну допустиму.
5	Логи роботи	OK	Вибір періоду (04.04.2025 - 15.05.2025) у фільтрах. Перевірка записів подій.	Логи будуть відфільтровані коректно, всі події відображаються з вказаними даними (дата, користувач, подія).
6	Довідник по додаванню нових деталей	OK	Натискання "Створити", заповнення даних для брендів, моделей, типів запчастин.	Нові записи будуть успішно додані, зміни будуть збережені після редагування, видалення працює коректно.
7	Накладні (з складу на склад)	OK	Додавання нової накладної з вибором дати, користувача, типу операції та запчастини.	Нова накладна буде успішно додана, операції з редагуванням і

Продовження таблиці 4.1

Тестування функціональності

№	Тест-кейс	OK/ NOK	Дії	Очікуваний результат
5	Логи роботи	OK	Вибір періоду (04.04.2025 - 15.05.2025) у фільтрах. Перевірка записів подій.	Логи будуть відфільтровані коректно, всі події відображаються з вказаними даними.
6	Довідник по додаванню нових деталей	OK	Натискання "Створити", заповнення даних для брендів, моделей, типів запчастин.	Нові записи будуть успішно додані, зміни будуть збережені після редагування, видалення працює коректно.
7	Накладні (з складу на склад)	OK	Додавання нової накладної з вибором дати, користувача, типу операції та запчастини.	Нова накладна буде успішно додана, операції з редагуванням і видаленням накладних працюють коректно.
8	Звіти	OK	Вибір звіту "Залишки на складах" і перевірка коректності даних в Excel.	Звіт генерується коректно, всі стовпці заповнені вірно, відображається актуальна інформація про запчастини.

ВИСНОВКИ

У процесі виконання даної кваліфікаційної роботи була розроблена программа для управління складом автозапчастин. Під час реалізації були розглянуті та проаналізовані існуючі програмні рішення, на основі яких були сформовані вимоги до нового застосунку. Основною метою роботи було створення програмного забезпечення, яке дозволяє точний облік запасів на складі, зручне введення даних, швидкий доступ до інформації та можливість формування звітів.

Використання таких технологій, як C#, .NET, Entity Framework і SQLite, дозволило створити надійне, швидке та зручне для користувачів рішення. Програма підтримує всі необхідні функції для обліку автозапчастин, такі як додавання, редагування та видалення даних, формування звітів, а також перевірка залишків товарів на складі.

Розроблена система дозволяє значно знизити ймовірність помилок при веденні обліку, покращити організацію роботи складу, зберігати точні дані про наявність товарів і забезпечити зручний доступ до цієї інформації. За допомогою цієї програми підприємства можуть значно зменшити час, необхідний для виконання операцій на складі, підвищити ефективність роботи складу і мінімізувати операційні витрати.

Робота є практично корисною для підприємств, що займаються продажем автозапчастин або обслуговуванням автотранспортних засобів, адже вона сприяє оптимізації процесів управління складом і підвищенню точності обліку запасів.

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Кривоносенко Д.М., Забезпечення безпеки даних у програмі для управління складом автозапчастин. Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.2025, ДУІКТ, Київ, Україна, с. 409.

2. Кривоносенко Д.М., Яскевич В. О. Розробка програмного забезпечення «Підтримки управління складом автозапчастин» з використанням Python та SQLite. Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.2025, ДУІКТ, Київ, Україна, с. 438-439.

ПЕРЕЛІК ПОСИЛАНЬ

1. Особливості методики аналізу торговельних підприємств та його інформаційне забезпечення. Випуск # 71 / 2025 https://economyandsociety.in.ua

2. Управління розвитком економічного середовища в умовах глобальних трансформацій: колективна монографія / за ред. В. В. Прохорової. Харків: Видавництво Іванченка, 2023. 436 ст.

3. BAS. BAS Програми Онлайн | Для Навчальних Закладів І Курсів | Edu.iBuh.Online. URL: https://nz.ibuh.online/solutions/edukup (дата звернення 02.05.2025)

4. Torgsoft. Що таке торгсофт? URL: https://torgsoft.ua/ (Дата звернення 03.04.2025)

5. Remonline. Програма для обліку та автоматизації бізнесу. URL: https://remonline.ua/ (Дата звернення 04.05.2025)

6. Dilovod. Онлайн-бухгалтерія Dilovod. URL: https://dilovod.ua/ (Дата звернення 5.05.2025)

7. C# та .NET 8.0. C# Guide - .NET managed language. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (Дата звернення 09.05.2025)

8. SQLite. SQLite Documentation. URL: https://www.sqlite.org/docs.html (Дата звернення 12.05.2025)

9. Windows Forms. Introduction to C# Windows Forms Applications – GeeksforGeeks. URL: https://www.geeksforgeeks.org/introduction-to-c-sharp-windows-forms-applications/ (Дата звернення 14.05.2025)

10. Entity Framework. qalight Центр підготовки IT фахівців. URL: https://qalight.ua/baza-znaniy/frejmvork-v-programuvanni/ (Дата звернення 15.05.2025)

11. Visual Studio. Introduction to Visual Studio – GeeksforGeeks. URL: https://www.geeksforgeeks.org/introduction-to-visual-studio/ (Дата зверненн 16.05.2025)

12. UI. What is UX / UI Design? URL: https://flatironschool.com/blog/what-is-ux-ui-design/ (Дата звернення 16.05.2025)

13. Codecademy. What Is an IDE? | Codecademy. URL: https://www.codecademy.com/article/what-is-an-ide (Дата звернення 13.05.2025)

14. Coursera. What Is Programming? And How To Get Started. URL: https://www.coursera.org/articles/what-is-programming?msockid=36e7ca1022ca68db2f7adec823d26939 (Дата звернення 07.05.2025)

15. FoxmindEd. Схеми бази даних: компоненти, типи та проєктування. URL: https://foxminded.ua/skhemy-bazy-danyh/ (Дата звернення 11.05.2025)

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Розробка програмного забезпечення підтримки управління складом автозапчастин

Виконав студент 4 курсу
групи ПД-43
Кривоносенко Денис Миколайович
керівник роботи
к.т.н., доц., доцент кафедри ІПЗ Яскевич Владислав Олександрович

Київ – 2025

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: полегшити процес управління складом автозапчастин.

Об’єкт дослідження: процес управління складськими запасами автозапчастин у малому та середньому бізнесі.

Предмет дослідження: програмні засоби та технології створення прикладного програмного забезпечення для локального збереження та обробки складських даних.

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз літературних джерел з проблеми управління складом автозапчастин, вивчити існуючі рішення для обліку товарів на складі.
2. Оцінити та проаналізувати існуючі програмні продукти для управління складом автозапчастин, визначити їх переваги та недоліки.
3. Визначити вимоги до програмного забезпечення для управління складом автозапчастин, описати необхідний функціонал та можливості.
4. Розробити програмне забезпечення для обліку автозапчастин, управління запасами, надходженням та витратами товарів зі складу.
5. Реалізувати функціональність для генерування звітів про рух товарів на складі, залишки, надходження та витрати.

3

АНАЛІЗ АНАЛОГІВ

	BAS Комплексне управління підприємством	Torgsoft	Dilovod	Remonline	Sklad
Ручний облік	-	+	+	+	+
Звітність	-	+	-	-	+
Пошук запчастин	+	+	+	+	+
Фільтр запчастин	+	+	-	+	+
Сортування запчастин	+	+	+	+	+
Сповіщення про нестачу запчастин	-	-	+	-	+
Інвентиризація запчастин	+	+	+	+	+

4

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

1. Додавання, редагування та видалення інформації про автозапчастини.
2. Пошук та фільтрація запчастин за параметрами (назва, тип, модель, кількість).
3. Формування звітів про залишки та рух запчастин.
4. Реєстрація користувачів з вибором доступу (додавання запчастин, звіти).
5. Оповіщення про недостатню кількість запчастин.

Нефункціональні вимоги:

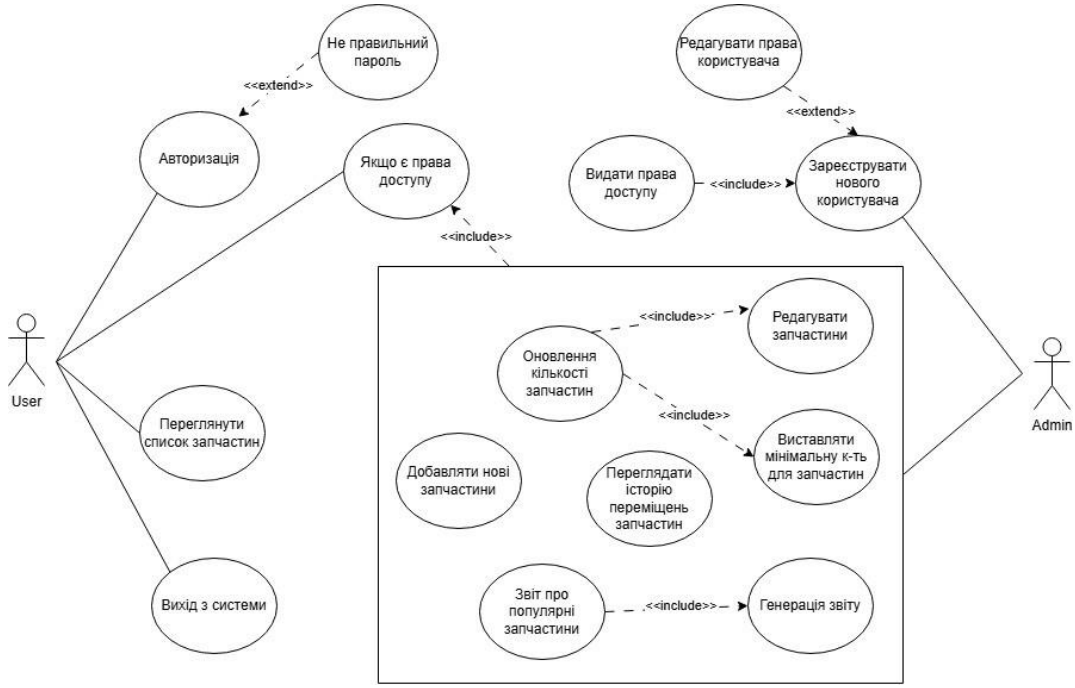
1. Інтерфейс на основі Windows Forms.
2. Збереження даних у локальній базі SQLite.
3. Хешування паролю для безпеки даних.
4. Авторизація

5

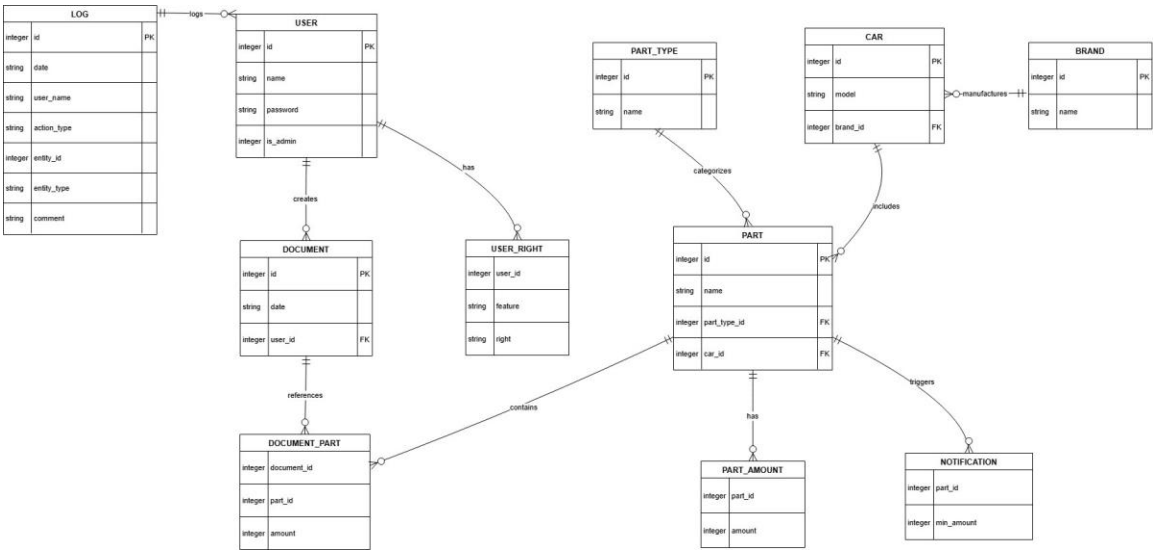
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

6

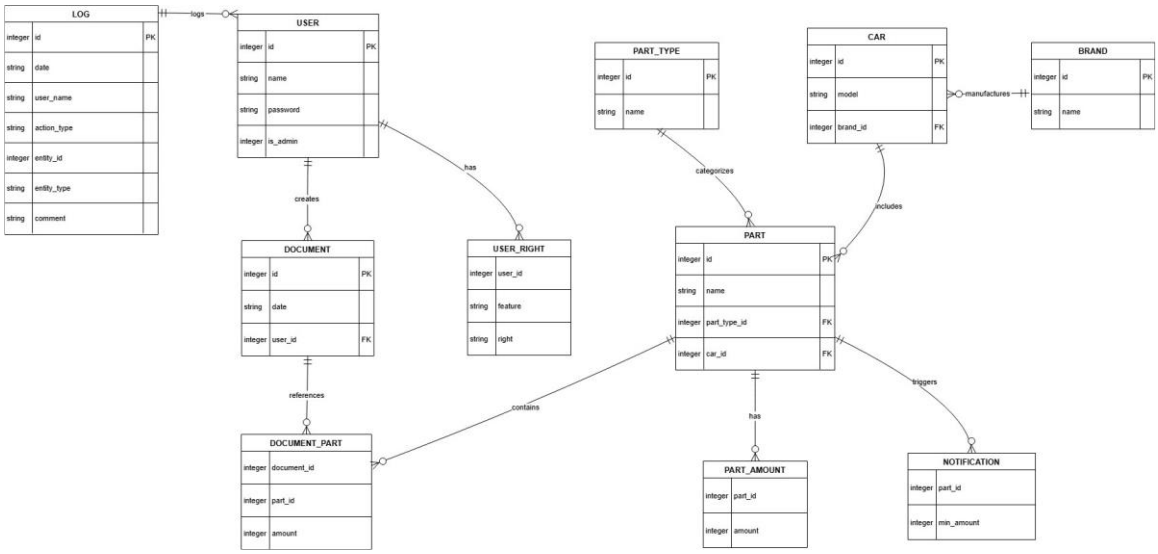
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



ДІАГРАМА СТРУКТУРИ БАЗ ДАНИХ

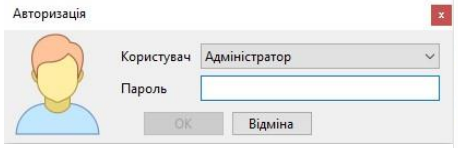


ДІАГРАМА СТРУКТУРИ БАЗ ДАНИХ

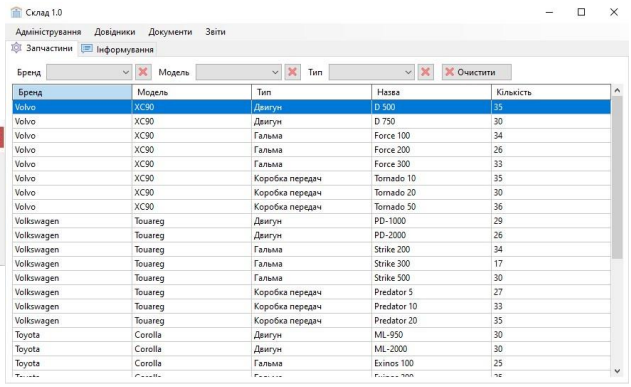


8

ЕКРАННІ ФОРМИ



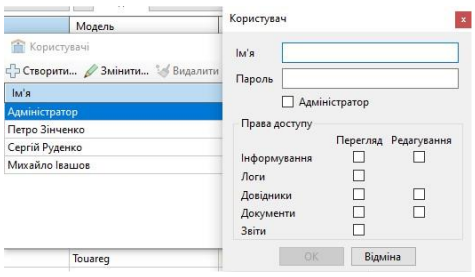
Авторизація



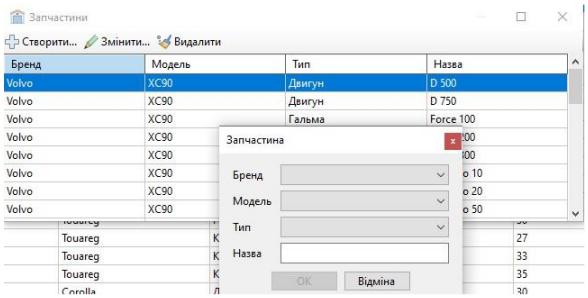
Основна сторінка складу

11

ЕКРАННІ ФОРМИ



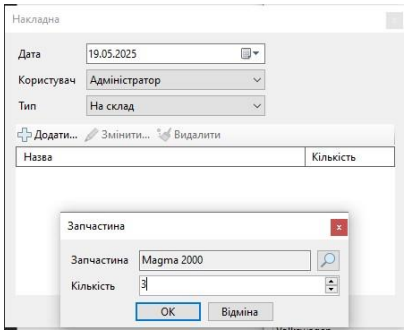
Реєстрація користувача



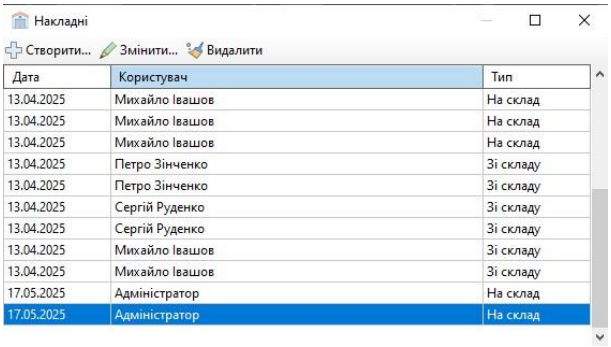
Створення нової запчастини

12

ЕКРАННІ ФОРМИ



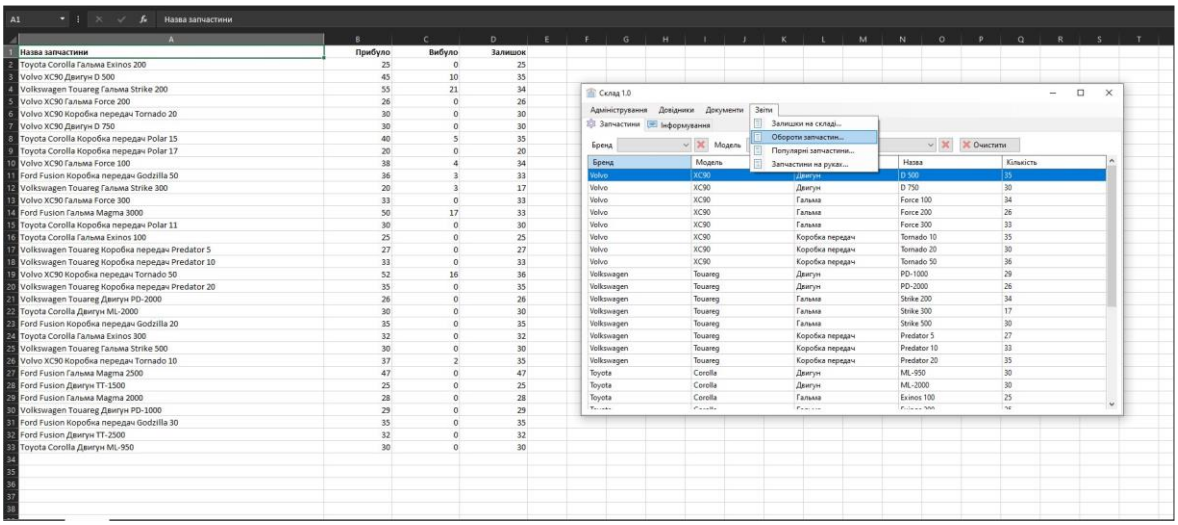
Створення накладної



Історія накладних

13

ЕКРАННІ ФОРМИ



Назва запчастини	Прибуло	Вибуло	Залишок
Toyota Corolla Гальма Exinos 200	25	0	25
Volvo XC90 Двигун D 500	45	10	35
Volkswagen Touareg Гальма Strike 200	55	21	34
Volvo XC90 Гальма Force 200	26	0	26
Volvo XC90 Коробка передач Tornado 20	30	0	30
Volvo XC90 Двигун D 750	30	0	30
Toyota Corolla Коробка передач Polar 15	40	5	35
Toyota Corolla Коробка передач Polar 17	20	0	20
Volvo XC90 Гальма Force 100	38	4	34
Ford Fusion Коробка передач Godzilla 50	36	3	33
Volkswagen Touareg Гальма Strike 300	20	3	17
Volvo XC90 Гальма Force 300	33	0	33
Ford Fusion Гальма Magma 3000	50	17	33
Toyota Corolla Коробка передач Polar 11	30	0	30
Toyota Corolla Гальма Exinos 100	25	0	25
Volkswagen Touareg Коробка передач Predator 5	27	0	27
Volkswagen Touareg Коробка передач Predator 10	33	0	33
Volvo XC90 Коробка передач Tornado 50	52	16	36
Volkswagen Touareg Коробка передач Predator 20	35	0	35
Volkswagen Touareg Двигун PD-2000	26	0	26
Toyota Corolla Двигун ML-2000	30	0	30
Ford Fusion Коробка передач Godzilla 20	35	0	35
Toyota Corolla Гальма Exinos 300	32	0	32
Volkswagen Touareg Гальма Strike 500	30	0	30
Volvo XC90 Коробка передач Tornado 10	37	2	35
Ford Fusion Гальма Magma 2500	47	0	47
Ford Fusion Двигун TT-1500	25	0	25
Ford Fusion Гальма Magma 2000	28	0	28
Volkswagen Touareg Двигун PD-1000	29	0	29
Ford Fusion Коробка передач Godzilla 30	35	0	35
Ford Fusion Двигун TT-2500	32	0	32
Toyota Corolla Двигун ML-950	30	0	30

Звіт “Обороти запчастин”

14

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Кривоносенко Д.М., Забезпечення безпеки даних у програмі для управління складом автозапчастин. Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25, ДУІКТ, Київ, Україна, с. 409.
2. Кривоносенко Д.М., Яскевич В. О. Розробка програмного забезпечення «Підтримки управління складом автозапчастин» з використанням Python та SQLite. Матеріали VI Всеукраїнської науково-технічної конференції «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24.04.25, ДУІКТ, Київ, Україна, с. 438-439.

15

ВИСНОВКИ

1. Проведено аналіз джерел з проблеми управління складом автозапчастин.
2. Вивчено існуючі програмні рішення для підтримки обліку запчатин на складі, зокрема їхні переваги та недоліки.
3. Сформовано вимоги до розробки програмного забезпечення для обліку автозапчастин, яке забезпечить обробку запчастин на складі, облік надходжень та витрат.
4. Спроєктовано архітектуру програмного забезпечення для підтримки управління складом автозапчастин, визначено оптимальну структуру бази даних та інтерфейс користувача.
5. Розроблено програмне забезпечення для підтримки управління складом автозапчастин, що включає функції обліку запчастин, оновлення запасів та генерацію звітів.

ДОДАТОК Б. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ

class Program.cs

```
using Sklad.Forms.Main;
namespace Sklad
{
    internal static class Program
    {
      [STAThread]
       static void Main()
       {
          ApplicationConfiguration.Initialize();
          Application.Run(new MainForm());
       }
    }
}
```

MainController.cs

```
using System.Diagnostics.CodeAnalysis;
using Database;
using Database.Entities;
using Sklad.Forms.EditBrands;
using Sklad.Forms.EditCars;
using Sklad.Forms.EditDocuments;
using Sklad.Forms.EditNotifications;
using Sklad.Forms.EditParts;
using Sklad.Forms.EditPartTypes;
using Sklad.Forms.EditUsers;
using Sklad.Forms.Login;
using Sklad.Forms.Logs;
using Sklad.Reports;

namespace Sklad.Forms.Main
{
	internal class MainController
	{
	private readonly Db _db;
	private readonly ApplicationDetails
	_applicationDetails;
	private readonly ApplicationEvents
	_applicationEvents;
	private readonly MainForm _form;

	public MainController(MainForm form)
	{
		_db = Factory.GetDb();
		_applicationDetails =
	Factory.GetApplicationDetails();
		_applicationEvents =
	Factory.GetApplicationEvents();
		_form = form;
		_form.Load += OnFormLoad;
		_form.FormClosing +=
	OnFormFormClosing;
	}

	public void EditUsers()
	{
		using var editUsersForm = new
	EditUsersForm();
		editUsersForm.ShowDialog();
	}

	public void EditNotifications()
	{
		using var editNotificationsForm = new
	EditNotificationsForm();
		editNotificationsForm.ShowDialog();

	_applicationEvents.InformNotificationsModifie
	d();
	}

	public void EditBrands()
	{
		using var editBrandsForm = new
	EditBrandsForm();
		editBrandsForm.ShowDialog();

	_applicationEvents.InformHandbooksModified(
	);
	}

	public void EditCars()
	{
		using var editCarsForm = new
EditCarsForm();
		editCarsForm.ShowDialog();

_applicationEvents.InformHandbooksModified(
);
}

public void EditPartTypes()
{
		using var editPartTypesForm = new
EditPartTypesForm();
		editPartTypesForm.ShowDialog();

_applicationEvents.InformHandbooksModified(
);
}

public void EditParts()
{
		using var editPartsForm = new
EditPartsForm();
		editPartsForm.ShowDialog();

_applicationEvents.InformHandbooksModified(
);
}

public void EditDocuments()
{
		using var editDocumentsForm = new
EditDocumentsForm();
		editDocumentsForm.ShowDialog();

_applicationEvents.InformDocumentsModified(
);
}

public void GeneratePartAmountsReport()
{
		var report = new
PartAmountsReport();
		report.Generate();
}

public void GeneratePartsTurnoverReport()
{
		var report = new
PartsTurnoverReport();
		report.Generate();
}

public void GeneratePopularPartsReport()
{
		var report = new PopularPartsReport();
		report.Generate();
}

public void GeneratePartsOnHandReport()
{
		var report = new
PartsOnHandsReport();
		report.Generate();
}

public void Logout()
{
		Application.Restart();
}

private bool Authorize([NotNullWhen(true)]
out User? user)
{
		user = null;
		if (!EnsureUserExists())
			return false;

		using var loginForm = new
LoginForm();
		if (loginForm.ShowDialog() !=
DialogResult.OK)
			return false;

		user = loginForm.GetUser();
		return true;
}
```

```
        private bool EnsureUserExists()
        {
            var users = _db.Users.GetAll();
            if (users.Any())
                return true;

            using var adminPasswordForm = new
AdminPasswordForm();
            if (adminPasswordForm.ShowDialog()
!= DialogResult.OK)
                return false;

            var adminUser = new
User("Адміністратор",
adminPasswordForm.Password, true,
UserRights.Full);
            _db.Users.Add(adminUser);
            return true;
        }

        public void ViewLogs()
        {
            using var logsForm = new
LogsForm();
            logsForm.ShowDialog();
        }

        private void OnFormLoad(object? sender,
EventArgs e)
        {
            if (!Authorize(out var user))
            {
                _form.Close();
                return;
            }

            _applicationDetails.SetUser(user);

_form.RestrictFeatureAccess(user.IsAdmin,
user.Rights);


_db.ActivityLogging.SetUser(user.Name);
            _db.ActivityLogging.Login();
        }

        private void OnFormFormClosing(object?
sender, FormClosingEventArgs e)
        {
            _db.ActivityLogging.Logout();
            _db.Close();
        }
        }
}
```

```
        class ComboBoxExtensions
using Database.Entities;

namespace Sklad.Extensions
{
        internal static class ComboBoxExtensions
        {
        public static void SetItems<T>(this ComboBox
comboBox, List<T> items, bool
preserveSelectedItem = false) where T : Entity
        {
            var selectedItemId =
comboBox.TryGetSelectedItemId<T>();
            comboBox.Items.Clear();

            foreach (var item in items)
                comboBox.Items.Add(item);

            if (preserveSelectedItem &&
selectedItemId.HasValue)
                comboBox.SelectedItem =
items.FirstOrDefault(x => x.Id ==
selectedItemId);
        }

        public static T? TryFindItem<T>(this
ComboBox comboBox, long id) where T :
Entity
        {
            foreach (T item in comboBox.Items)
                if (item.Id == id)
                    return item;
            return default;
        }

        public static void SelectItem<T>(this
ComboBox comboBox, long id) where T :
Entity
        {
            comboBox.SelectedItem =
comboBox.TryFindItem<T>(id);
        }

        public static long GetSelectedItemId<T>(this
ComboBox comboBox) where T : Entity
        {
            if (comboBox.SelectedItem == null)
                return Entity.NotAssignedId;

            var item = (T)comboBox.SelectedItem;
            return item.Id;
        }

        public static long?
TryGetSelectedItemId<T>(this ComboBox
comboBox) where T : Entity
        {
            if (comboBox.SelectedItem == null)
                return null;

            var item = (T)comboBox.SelectedItem;
            return item.Id;
        }

        public static void
OnSelectedEntityChanged<T>(this ComboBox
comboBox, Action action) where T : Entity
        {
            var lastSelectedId =
comboBox.GetSelectedItemId<T>();

            comboBox.SelectedIndexChanged +=
(_, _) =>
            {
                var selectedId =
comboBox.GetSelectedItemId<T>();
                if (selectedId ==
lastSelectedId)
                    return;

                action();
                lastSelectedId = selectedId;
            };
        }
        }
}
```

```
        class EditBrandsController
using Database;
using Database.Entities;
using Sklad.Extensions;

namespace Sklad.Forms.EditBrands
{
        internal class EditBrandsController
        {
        private readonly Db _db;
        private readonly EditBrandsForm _form;
        private readonly List<Brand> _brands;

        public EditBrandsController(EditBrandsForm
        form)
        {
            _db = Factory.GetDb();
            _form = form;
            _brands = _db.Brands.GetAll();

        _form.BrandGrid.AddBrands(_brands);
        }

        public void CreateBrand()
        {
            var brand = new Brand();
            var existingNames =
GetExistingNames();

            using var editBrandForm = new
EditBrandForm(brand, existingNames);
            if (editBrandForm.ShowDialog() ==
DialogResult.OK)

        AddBrand(editBrandForm.GetBrand());
```

```
        }

        public void ChangeBrand()
        {
                var brand =
_form.BrandGrid.GetSelectedBrand();
                var existingNames =
GetExistingNames(brand.Id);

                using var editBrandForm = new
EditBrandForm(brand, existingNames);
                if (editBrandForm.ShowDialog() ==
DialogResult.OK)

UdateBrand(editBrandForm.GetBrand());
        }

        public void DeleteBrand()
        {
                var brand =
_form.BrandGrid.GetSelectedBrand();
                var isInUse =
_db.Brands.IsInUse(brand.Id);

                if (isInUse)
                {

_form.InformBrandIsInUse(brand.Name);
                        return;
                }

                if
(!_form.ConfirmDeleteBrand(brand.Name))
                        return;

                _db.Brands.Delete(brand.Id);
                _brands.RemoveAll(x => x.Id ==
brand.Id);

_form.BrandGrid.DeleteBrand(brand.Id);
        }

        private void AddBrand(Brand brand)
        {
                var brandId = _db.Brands.Add(brand);
                brand = _db.Brands.Get(brandId);

                _brands.Add(brand);
                _form.BrandGrid.AddBrand(brand);
        }

        private void UdateBrand(Brand brand)
        {
                _db.Brands.Update(brand);
                brand = _db.Brands.Get(brand.Id);

                _brands.Replace(brand);
                _form.BrandGrid.UpdateBrand(brand);
        }

        private HashSet<string>
GetExistingNames(long idToExclude =
Entity.NotAssignedId)
        {
                return _brands
                        .Where(x => x.Id !=
idToExclude)
                        .Select(x => x.Name)
                        .ToHashSet();
        }
        }
}
```

EditCarController

```
using Database;
using Database.Entities;

namespace Sklad.Forms.EditCar
{
        internal class EditCarController
        {
        private readonly Db _db;

        public EditCarController()
        {
                _db = Factory.GetDb();
        }

        public List<Brand> GetBrands()
        {
                return _db.Brands.GetAll();
        }

        public bool IsCarExist(string model, long
brandId)
        {
                return _db.Cars.IsExist(model,
brandId);
        }
        }
}
```

EditNotificationsController

```
using Database;
using Database.Entities;
using Sklad.Extensions;
using Sklad.Forms.EditNotification;

namespace Sklad.Forms.EditNotifications
{
        internal class EditNotificationsController
        {
        private readonly Db _db;
        private readonly EditNotificationsForm _form;
        private readonly Dictionary<long, string>
        _partFullNames;

        public
EditNotificationsController(EditNotificationsFo
rm form)
        {
                _db = Factory.GetDb();
                _form = form;
                _partFullNames =
_db.GetPartFullNames();

                var notifications =
_db.Notifications.GetAll();

_form.NotificationGrid.AddNotifications(notifi
cations);
        }

        public void CreateNotification()
        {
                var notification = new Notification();
                var partIdsToExclude =
GetPartIdsToExclude();

                using var editNotificationForm = new
EditNotificationForm(notification,
partIdsToExclude);
                if (editNotificationForm.ShowDialog()
== DialogResult.OK)

AddNotification(editNotificationForm.GetNotif
ication());
        }

        public void ChangeNotification()
        {
                var notification =
_form.NotificationGrid.GetSelectedNotification
();
                var partIdsToExclude =
GetPartIdsToExclude(notification.PartId);

                using var editNotificationForm = new
EditNotificationForm(notification,
partIdsToExclude);
                if (editNotificationForm.ShowDialog()
== DialogResult.OK)

UdateNotification(notification.PartId,
editNotificationForm.GetNotification());
        }

        public void DeleteNotification()
        {
                var notification =
_form.NotificationGrid.GetSelectedNotification
();
                var partFullName =
_partFullNames[notification.PartId];

                if
(!_form.ConfirmDeleteNotification(partFullNa
me))
                        return;
```

```
            _db.Notifications.Delete(notification.PartId);

            _form.NotificationGrid.DeleteNotification(notif
            ication.PartId);
            }

            private void AddNotification(Notification
            notification)
            {
                        _db.Notifications.Add(notification);

            _form.NotificationGrid.AddNotification(notific
            ation);
            }

            private void UdateNotification(long
            previousPartId, Notification notification)
            {
                        _db.Notifications.Update(notification);

            _form.NotificationGrid.UpdateNotification(pre
            viousPartId, notification);
            }

            private HashSet<long>
            GetPartIdsToExclude(long partIdToInclude =
            Entity.NotAssignedId)
            {
                        return
            _form.NotificationGrid.GetNotifications()
                                    .Select(x => x.PartId)
                                    .Where(x => x !=
            partIdToInclude)
                                    .ToHashSet();
            }
            }
}
```

EditPartController

```
using Database;
using Database.Entities;

namespace Sklad.Forms.EditPart
{
            internal class EditPartController
            {
            private readonly Db _db;
            private readonly List<Car> _cars;

            public EditPartController()
            {
                        _db = Factory.GetDb();
                        _cars = _db.Cars.GetAll();
            }

            public List<Brand> GetBrands()
            {
                        return _db.Brands.GetAll();
            }

            public List<Car> GetCars(long brandId)
            {
                        return _cars
                                    .Where(x => x.BrandId ==
            brandId)
                                    .ToList();
            }

            public List<PartType> GetPartTypes()
            {
                        return _db.PartTypes.GetAll();
            }

            public long GetBrandIdByCarId(long id)
            {
                        var car = _db.Cars.Get(id);
                        return car.BrandId;
            }
            }
}
```

EditPartsController

```
using Database;
using Database.Entities;
using Sklad.Extensions;
using Sklad.Forms.EditPart;

namespace Sklad.Forms.EditParts
{
            internal class EditPartsController
            {
            private readonly Db _db;
            private readonly EditPartsForm _form;
            private readonly List<Part> _parts;

            public EditPartsController(EditPartsForm form)
            {
                        _db = Factory.GetDb();
                        _form = form;
                        _parts = _db.Parts.GetAll();
                        _form.PartGrid.AddParts(_parts);
            }

            public void CreatePart()
            {
                        var part = new Part();
                        using var editPartForm = new
            EditPartForm(part);

                        if (editPartForm.ShowDialog() ==
            DialogResult.OK)

            AddPart(editPartForm.GetPart());
            }

            public void ChangePart()
            {
                        var part =
            _form.PartGrid.GetSelectedPart();
                        using var editPartForm = new
            EditPartForm(part);

                        if (editPartForm.ShowDialog() ==
            DialogResult.OK)

            UdatePart(editPartForm.GetPart());
            }

            public void DeletePart()
            {
                        var part =
            _form.PartGrid.GetSelectedPart();
                        var isInUse =
            _db.Parts.IsInUse(part.Id);

                        if (isInUse)
                        {

            _form.InformPartIsInUse(part.Name);
                                    return;
                        }

                        if
            (!_form.ConfirmDeletePart(part.Name))
                                    return;

                        _db.Parts.Delete(part.Id);
                        _parts.RemoveAll(x => x.Id ==
            part.Id);
                        _form.PartGrid.DeletePart(part.Id);
            }

            private void AddPart(Part part)
            {
                        var partId = _db.Parts.Add(part);
                        part = _db.Parts.Get(partId);

                        _parts.Add(part);
                        _form.PartGrid.AddPart(part);
            }

            private void UdatePart(Part part)
            {
                        _db.Parts.Update(part);
                        part = _db.Parts.Get(part.Id);

                        _parts.Replace(part);
                        _form.PartGrid.UpdatePart(part);
            }
            }
}
```

DescriptionProvider

```
namespace Sklad.Forms.Main.Notifications
{
            internal static class DescriptionProvider
            {
            public static string GetDescription(int
            entryCount)
```

```
        {
            if (entryCount == 0)
                return "Усі запчастини є в
наявності у необхідній кількості.";

            var foundWord =
GetFoundWord(entryCount);
            var partWord =
GetPartWord(entryCount);
            return $"{foundWord} {entryCount}
{partWord} з недостатньою кількістю.";
        }

        private static string GetFoundWord(int
entryCount)
        {
            return entryCount == 1 ? "Знайдена" :
"Знайдено";
        }

        private static string GetPartWord(int
entryCount)
        {
            return
GetRemainderBelow10(entryCount) switch
            {
                1 => "запчастина",
                2 or 3 or 4 => "запчастини",
                _ => "запчастин"
            };
        }

        private static int GetRemainderBelow10(int
number)
        {
            while (number > 10)
                number %= 10;

            return number;
        }
    }
}
                    class Entry
namespace Sklad.Forms.Main.Notifications
{
    public class Entry
    {
        public Entry(string partFullName, long
minAmount, long actualAmount)
        {
            PartFullName = partFullName;
            MinAmount = minAmount;
            ActualAmount = actualAmount;
        }

        public string PartFullName { get; }
        public long MinAmount { get; }
        public long ActualAmount { get; }
    }
}
                PartGridController
using Database;
using Database.Entities;
using Sklad.Extensions;

namespace Sklad.Forms.SelectPart
{
    internal class PartGridController
    {
        private readonly Db _db;
        private readonly Dictionary<long, string>
_partFullNames;
        private HashSet<long> _partIdsToExclude;

        public PartGridController()
        {
            _db = Factory.GetDb();
            _partFullNames =
_db.GetPartFullNames();
            _partIdsToExclude = new
HashSet<long>();
        }

        public void
SetPartIdsToExclude(HashSet<long>
partIdsToExclude)
        {
            _partIdsToExclude =
partIdsToExclude;
        }

        public List<PartAmount> GetPartAmounts()
        {
            return _db.PartAmounts.GetAll()
                .Where(x =>
!_partIdsToExclude.Contains(x.PartId))
                .ToList();
        }

        public Part GetPart(long partId)
        {
            return _db.Parts.Get(partId);
        }

        public string GetPartFullName(long partId)
        {
            return _partFullNames[partId];
        }
    }
}
                PartAmountsReport
using System.Runtime.InteropServices;
using Microsoft.Office.Interop.Excel;
using Sklad.Extensions;

namespace Sklad.Reports
{
    internal class PartAmountsReport : Report
    {
        protected override void FillContent(dynamic
worksheet, dynamic range)
        {
            HeaderColumn(worksheet, range, 1,
70, "Назва запчастини");
            HeaderColumn(worksheet, range, 2,
15, "Кількість", XlHAlign.xlHAlignRight);

            var rowRange =
worksheet.Rows["1:1"];
            rowRange.Font.Bold = true;

            Marshal.ReleaseComObject(rowRange);

            var partAmounts =
Db.PartAmounts.GetAll().Where(x =>
x.Amount > 0).ToList();
            var partFullNames =
Db.GetPartFullNames();
            var rowNumber = 2;

            foreach (var partAmount in
partAmounts.OrderBy(x =>
partFullNames[x.PartId]))
            {
                range.Cells[rowNumber,
1].Value2 = partFullNames[partAmount.PartId];
                range.Cells[rowNumber,
2].Value2 = partAmount.Amount;
                rowNumber++;
            }
        }
    }
}
                PartsOnHandsReport
using System.Runtime.InteropServices;
using Database.Entities;
using Microsoft.Office.Interop.Excel;
using Sklad.Extensions;

namespace Sklad.Reports
{
    internal class PartsOnHandsReport : Report
    {
        protected override void FillContent(dynamic
worksheet, dynamic range)
        {
            HeaderColumn(worksheet, range, 1,
20, "Користувач");
            HeaderColumn(worksheet, range, 2,
48, "Назва запчастини");
            HeaderColumn(worksheet, range, 3,
11, "Узяв", XlHAlign.xlHAlignRight);
            HeaderColumn(worksheet, range, 4,
11, "Повернув", XlHAlign.xlHAlignRight);
            HeaderColumn(worksheet, range, 5,
11, "На руках", XlHAlign.xlHAlignRight);
```

```
                var rowRange =
        worksheet.Rows["1:1"];
                rowRange.Font.Bold = true;

        Marshal.ReleaseComObject(rowRange);

                var users = Db.Users.GetAll();
                var partFullNames =
        Db.GetPartFullNames();
                var rowNumber = 2;

                foreach (var user in users)
                {
                        var documents =
        Db.Documents.GetAll().Where(x => x.UserId
        == user.Id).ToList();
                        if (documents.Count == 0)
                                continue;

                        var partIds =
        GetPartIds(documents);
                        var incomeAmounts =
        GetDocumentAmounts(documents, true);
                        var outcomeAmounts =
        GetDocumentAmounts(documents, false);

                        foreach (var partId in partIds)
                        {
                                if
        (!incomeAmounts.TryGetValue(partId, out var
        income))
                                        income = 0;

                                if
        (!outcomeAmounts.TryGetValue(partId, out var
        outcome))
                                        outcome =
        0;

                                var amount =
        outcome - income;
                                if (amount <= 0)
                                        continue;


        range.Cells[rowNumber, 1].Value2 =
        user.Name;

        range.Cells[rowNumber, 2].Value2 =
        partFullNames[partId];

        range.Cells[rowNumber, 3].Value2 = outcome;

        range.Cells[rowNumber, 4].Value2 = income;

        range.Cells[rowNumber, 5].Value2 = amount;
                                rowNumber++;
                        }
                }
        }

        private List<long> GetPartIds(List<Document>
        documents)
        {
                return documents
                        .SelectMany(x => x.Parts)
                        .Select(x => x.PartId)
                        .Distinct()
                        .ToList();
        }

        private Dictionary<long, long>
        GetDocumentAmounts(List<Document>
        documents, bool incoming)
        {
                return documents
                        .Where(x => x.Incoming ==
        incoming)
                        .SelectMany(x => x.Parts)
                        .GroupBy(x => x.PartId)
                        .ToDictionary(x => x.Key, x
        => x.Sum(y => y.Amount));
        }
        }
}
                PartsTurnoverReport
using System.Runtime.InteropServices;
using Database.Entities;
using Microsoft.Office.Interop.Excel;
using Sklad.Extensions;

namespace Sklad.Reports
{
        internal class PartsTurnoverReport : Report
        {
        protected override void FillContent(dynamic
        worksheet, dynamic range)
        {
                HeaderColumn(worksheet, range, 1,
        70, "Назва запчастини");
                HeaderColumn(worksheet, range, 2,
        15, "Прибуло", XlHAlign.xlHAlignRight);
                HeaderColumn(worksheet, range, 3,
        15, "Вибуло", XlHAlign.xlHAlignRight);
                HeaderColumn(worksheet, range, 4,
        15, "Залишок", XlHAlign.xlHAlignRight);

                var rowRange =
        worksheet.Rows["1:1"];
                rowRange.Font.Bold = true;

        Marshal.ReleaseComObject(rowRange);

                var documents =
        Db.Documents.GetAll();
                var partFullNames =
        Db.GetPartFullNames();
                var partAmounts =
        Db.PartAmounts.GetAll().ToDictionary(x =>
        x.PartId, x => x.Amount);
                var rowNumber = 2;

                var partIds = GetPartIds(documents);
                var incomAmounts =
        GetDocumentAmounts(documents, true);
                var outcomeAmounts =
        GetDocumentAmounts(documents, false);

                foreach (var partId in partIds)
                {
                        range.Cells[rowNumber,
        1].Value2 = partFullNames[partId];
                        range.Cells[rowNumber,
        2].Value2 =
        incomAmounts.TryGetValue(partId, out var
        income) ? income : 0;
                        range.Cells[rowNumber,
        3].Value2 =
        outcomeAmounts.TryGetValue(partId, out var
        outcome) ? outcome : 0;
                        range.Cells[rowNumber,
        4].Value2 = partAmounts[partId];
                        rowNumber++;
                }
        }

        private List<long> GetPartIds(List<Document>
        documents)
        {
                return documents
                        .SelectMany(x => x.Parts)
                        .Select(x => x.PartId)
                        .Distinct()
                        .ToList();
        }

        private Dictionary<long, long>
        GetDocumentAmounts(List<Document>
        documents, bool incoming)
        {
                return documents
                        .Where(x => x.Incoming ==
        incoming)
                        .SelectMany(x => x.Parts)
                        .GroupBy(x => x.PartId)
                        .ToDictionary(x => x.Key, x
        => x.Sum(y => y.Amount));
        }
        }
}
                PopularPartsReport
using System.Runtime.InteropServices;
using Database.Entities;
using Microsoft.Office.Interop.Excel;
using Sklad.Extensions;

namespace Sklad.Reports
{
```

```
	internal class PopularPartsReport : Report
	{
	protected override void FillContent(dynamic
	worksheet, dynamic range)
	{
		HeaderColumn(worksheet, range, 1,
	70, "Назва запчастини");
		HeaderColumn(worksheet, range, 2,
	23, "Кількість переміщень",
	XlHAlign.xlHAlignRight);

		var rowRange =
	worksheet.Rows["1:1"];
		rowRange.Font.Bold = true;

	Marshal.ReleaseComObject(rowRange);

		var documents =
	Db.Documents.GetAll();
		var partFullNames =
	Db.GetPartFullNames();
		var rowNumber = 2;

		var partIds = GetPartIds(documents);
		var movements =
	GetMovements(documents);

		foreach (var partId in
	partIds.OrderByDescending(x =>
	movements[x]))
		{
			range.Cells[rowNumber,
	1].Value2 = partFullNames[partId];
			range.Cells[rowNumber,
	2].Value2 = movements[partId];
			rowNumber++;
		}
	}

	private List<long> GetPartIds(List<Document>
	documents)
	{
		return documents
			.SelectMany(x => x.Parts)
			.Select(x => x.PartId)
			.Distinct()
			.ToList();
	}

	private Dictionary<long, long>
	GetMovements(List<Document> documents)
	{
		return documents
			.SelectMany(x => x.Parts)
			.GroupBy(x => x.PartId)
			.ToDictionary(x => x.Key, x
	=> x.Sum(y => y.Amount));
	}
	}
}
```

Report

```
using Microsoft.Office.Interop.Excel;
using System.Runtime.InteropServices;
using Database;

namespace Sklad.Reports
{
	internal abstract class Report
	{
	protected Report()
	{
		Db = Factory.GetDb();
	}

	protected Db Db { get; }
	protected abstract void FillContent(dynamic
	worksheet, dynamic range);

	public void Generate()
	{
		var application = new
	Microsoft.Office.Interop.Excel.Application();
		var workbook =
	application.Workbooks.Add();
		var worksheet = workbook.Sheets[1];
		var range = worksheet.UsedRange;

		FillContent(worksheet, range);

		application.Visible = true;
		Marshal.ReleaseComObject(range);
		Marshal.ReleaseComObject(worksheet);
		Marshal.ReleaseComObject(workbook);
		Marshal.ReleaseComObject(application);
	}

	protected void HeaderColumn(dynamic worksheet,
	dynamic range, int columnNumber, int width, string
	title, XlHAlign align = XlHAlign.xlHAlignLeft)
	{
		range.Cells[1, columnNumber] = title;

	worksheet.Columns[columnNumber].ColumnWidth
	= width;

	worksheet.Columns[columnNumber].HorizontalAlig
	nment = align;
	}
	}
}
```

Factory

```
using Database;

namespace Sklad
{
	public class Factory
	{
	private static readonly Factory _factory = new
	Factory();
	private readonly Db _db;
	private readonly ApplicationEvents
	_applicationEvents;
	private readonly ApplicationDetails
	_applicationDetails;

	private Factory()
	{
		_db = Db.Open("database.sqlite");
		_applicationEvents = new
	ApplicationEvents();
		_applicationDetails = new
	ApplicationDetails();
	}

	public static Db GetDb()
	{
		return _factory._db;
	}

	public static ApplicationEvents
	GetApplicationEvents()
	{
		return _factory._applicationEvents;
	}
	public static ApplicationDetails
	GetApplicationDetails()
	{
		return _factory._applicationDetails;
	}
	}
}
```