

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОПТИМІЗАЦІЇ
ЛОГІСТИЧНИХ ПРОЦЕСІВ У МАГАЗИНІ РОЗДРІБНОЇ
ТОРГІВЛІ
SOFTWARE DEVELOPMENT TO OPTIMIZE LOGISTICS PROCESSES IN A
RETAIL STORE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-130
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ВЛАСЕНКО К.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ДУБРОВІН В.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент КАЗУРОВА А.Є.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

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

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ВЛАСЕНКА Костянтина Вілталійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного забезпечення для оптимізації логістичних процесів у магазині роздрібної торгівлі
Software Development to Optimize Logistics Processes in a Retail Store
керівник проєкту (роботи) к.т.н., професор, ДУБРОВІН Валерій Іванович
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 24 ” квітня 2024 року № 168

2. Строк подання студентом проєкту (роботи) 10 червня 2024 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми. Додатки

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ДУБРОВІН В.І., професор		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання “ 15 ” квітня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

Костянтин ВЛАСЕНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

Валерій ДУБРОВІН
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
96 с., 38 рис., 9 табл., 2 дод., 17 джерел.

ЛОГІСТИЧНІ ПРОЦЕСИ, РОЗДРІБНА ТОРГІВЛЯ, КАСИР, ПРАЦІВНИК
СКЛАДУ, АДМІНІСТРАТОР.

Об'єкт дослідження – процес оптимізації логістичних процесів магазину роздрібною торгівлі.

Предмет дослідження – управління товарними запасами та складськими операціями в роздрібних магазинах і складах.

Мета роботи – розробка автоматизованої системи управління товарними запасами та складськими операціями для роздрібних.

Матеріали, методи та технічні засоби: мова програмування - C#, система керування базами даних – SQL server, середовище розробки - Microsoft Visual Studio, графічна платформа – Windows Forms.

Результати. Створено програмне забезпечення яке оптимізує логістичні процеси магазину роздрібною торгівлі завдяки використанню QR-кодів.

Висновки. Розроблено програмне забезпечення що оптимізує логістичні процеси в магазині роздрібною торгівлі.

Галузь використання – магазини роздрібною торгівлі.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 96 pages, 38 figures, 9 tables, 2 appendixes, 17 sources.

LOGISTICS PROCESSES, RETAIL, CASHIER, WAREHOUSE WORKER, ADMINISTRATOR.

Object of Research: Process of optimizing the logistics processes of a retail store.

Subject of Research: Management of inventory and warehouse operations in retail stores and warehouses.

Purpose: Development of an automated system for managing inventory and warehouse operations in retail.

Materials, Methods, and Technical Means: Programming language - C#, Database Management System - SQL Server, Development Environment - Microsoft Visual Studio, Graphical Platform - Windows Forms.

Results: Developed software that optimizes the logistics processes of a retail store through the use of QR codes.

Conclusions: The developed software optimizes logistics processes in a retail store.

Field of Application: Retail stores.

ЗМІСТ

	С.
1 Аналіз предметної області.....	11
1.1 Опис досліджуваної предметної області.....	11
1.2 Аналіз вимог до програмного забезпечення	12
1.3 Огляд існуючих рішень	13
1.4 Технічне завдання.....	17
1.4.1 Вступ	17
1.4.2 Призначення розробки.....	17
1.4.3 Вимоги до програми	17
1.4.4 Техніко-економічні показники.....	18
1.4.5 Стадії та етапи розробки.....	18
1.4.6 Порядок контролю та приймання.....	19
1.5 Висновки за розділом 1	19
2 Розробка архітектури програми.....	21
2.1 Вибір мови програмування та середовища розробки	21
2.2 Вибір бази даних	23
2.3 Визначення функціональних вимог програмного забезпечення	25
2.4 Визначення структури програми.....	29
2.5 Висновки за розділом 2	32
3 Розробка програми	34
3.1 Алгоритм роботи програми.....	34
3.2 Опис реалізованих таблиць бази даних	34
3.3 Опис реалізованого інтерфейсу	36
3.4 Опис використаних бібліотек	41
3.4 Висновки за розділом 3	42
4 Експлуатація, тестування та експериментальне дослідження програми.	44
4.1 Опис реалізації програми.....	44

	7
4.2 Інструкція по експлуатації програми	52
4.3 Тестування програмного забезпечення.....	53
4.4 Висновки за розділом 4	58
Висновки	59
Перелік джерел посилання	60
ДОДАТОК А Текст програми.....	62
ДОДАТОК Б Слайди презентації	91

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

QR – Quick response;

SQL – Structured Query Language;

БД – База даних;

СКБД – Система керування базами даних;

ПЗ – Програмне забезпечення.

ВСТУП

В сучасному світі роздрібна торгівля відіграє важливу роль у задоволенні потреб споживачів та забезпеченні ефективного обігу товарів. В магазинах роздрібної торгівлі на більшій частині етапів логістичних процесів використовуються штрих-коди. Наприклад, кодування за допомогою штрих-кодів дозволяє автоматизувати процес ідентифікації: палет, контейнерів, ящиків, пакувань [1]. А вже за допомогою штрихового кодування пакувань в магазинах роздрібної торгівлі на касі автоматизується процес продажу товару. Але для сканування штрихових кодів застосовуються термінали збору даних та сканери штрихових кодів, що сповільнює процес та створює додаткові витрати магазину для закупівлі обладнання та його підтримки у робочому стані. Ще одним мінусом штрих-кодів є той факт, що за європейським стандартом він має містити не більше 13 символів, що обмежує кількість інформації яка може знаходитись в ньому. Таким чином, розробка програмного забезпечення, яке використовує QR-коди замість штрих-кодів для ідентифікації товарів, є інноваційним та ефективним рішенням.

Це програмне забезпечення забезпечить інтеграцію з базою даних магазину та використовуватиме QR-коди для ідентифікації товарів. Такий підхід до організації в магазинах роздрібної торгівлі дозволить використовувати звичайну камеру замість спеціального обладнання, що пришвидшить логістичні процеси та дозволить заощадити на обладнанні. Більше того, аналізатор зображень сканує одразу весь QR-код, що відрізняється від лазерного сканера, завдяки чому має більшу універсальність у скануванні різних поверхонь, пошкоджених або погано надрукованих QR-кодів [2]. Також використання динамічних QR кодів дозволить при помилці змінити його, замість того щоб змінювати штрих код на кожній упаковці товару.

Даний підхід до оптимізації логістичних процесів магазину має велику актуальність та потенціал для широкого застосування в різних сферах роздрібної

торгівлі у зв'язку з сучасними тенденціями розвитку технологій та діджиталізації повсякденних процесів.

Тобто, розробка даного програмного забезпечення є актуальною та перспективною задачею, яка відповідає потребам сучасного бізнесу та може мати значний вплив на його ефективність та конкурентоспроможність.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис досліджуваної предметної області

Логістичні процеси у ланцюгу поставок товарів мають виконувати наступні завдання: інтегрувати систему управління ресурсами підприємства, координувати дії всіх учасників ланцюга поставок товарів та забезпечувати ефективне використання ресурсів з мінімізацією витрат. У магазинах роздрібної торгівлі, де кількість товарів може бути значною, ефективне керування логістичними процесами важливо для забезпечення неперервності постачання та задоволення потреб клієнтів.

Логістичний ланцюг - це послідовна група учасників логістичного процесу, які здійснюють різноманітні операції з переміщенням матеріальних потоків від однієї логістичної системи до іншої, призначенням для виробничого або особистого споживання кінцевих користувачів [3].

Штрих-коди, як стандартний метод ідентифікації товарів, допомагають в автоматизації процесів інвентаризації та касової обробки, але вони можуть мати свої обмеження, зокрема, у складності використання на мобільних пристроях та обмеженому обсязі інформації, яку вони можуть містити. Наприклад для реалізації індивідуального кодування потрібні етикетки з кодуванням, друкарське обладнання для друкування кодів товарів, та етикет-пістолети. Штрих-коди можна надрукувати на різноманітних пристроях, таких як принтери етикеток, стрічкові принтери, офісні лазерні та струменеві принтери, а також за допомогою чекодруючих ваг. Крім того, штрих-коди та етикетки можна замовити у друкарнях на відповідних носіях [4].

В той час як QR-коди являються двовимірними, штрих-коди є одновимірними, це означає, що вони можуть бути скановані лише горизонтально. Це призводить до того, що вони містять лише від 20 до 25 символів, залежно від типу штрих-коду. До того ж одновимірні коди не такі безпечні, як двовимірні коди, які можна шифрувати.

Також, штрих-коди зазвичай потрібно сканувати під певними кутами, і деякі підприємства купують висококонтрастні сканери, щоб допомогти в цьому. За цими обставинами, при проблеми з пошкодженням або помилкою під час друку, існує велика ймовірність, що сканери не зможуть прочитати збережену інформацію [5].

QR-код – це графічна мітка, в якій зберігаються певні дані. Інформація кодується таким чином, щоб її можна було зчитувати за допомогою камер, сканерів та інших оптичних датчиків. Тому QR-коди використовують для швидкої передачі даних на смартфони та інші гаджети.

Така мітка є прямокутним контрастним зображенням з рамкою і внутрішнім візерунком, що нагадує лабіринт з пікселів. По кутах розташовані квадрати, які допомагають датчику правильно вибрати напрямок для зчитування даних [6].

Тобто, впровадження програмного забезпечення, яке використовує QR-коди для ідентифікації товарів у магазинах роздрібної торгівлі, може виявитися не лише ефективним методом оптимізації логістичних процесів, але й стратегічним інструментом для розвитку бізнесу та взаємодії з клієнтами.

1.2 Аналіз вимог до програмного забезпечення

Перш ніж описати конкретні вимоги до програмного забезпечення для оптимізації логістичних процесів у роздрібних магазинах, важливо провести глибокий аналіз потреб і вимог самого бізнесу, а також врахувати контекст його діяльності. Від цього залежить успішне впровадження програмного рішення та його подальша ефективність.

Вимоги до програмного забезпечення повинні враховувати не тільки поточні потреби бізнесу, але й його майбутні перспективи та стратегічні цілі.

Під час розробки програмного забезпечення було враховано ряд вимог, які є критичними для успішної реалізації проекту:

- інтеграція з базою даних. Програмне забезпечення має забезпечувати безперервну інтеграцію з існуючою базою даних магазину. Це дозволить забезпечити зручний обмін даними та безперервність логістичних процесів, уникнути дублювання інформації та зберегти цілісність даних;
- підтримка QR-кодів. Однією з ключових вимог є можливість генерування та зчитування QR-кодів. Програмне забезпечення повинно бути здатним ефективно працювати з цим стандартом ідентифікації, що дозволить забезпечити швидкий та надійний обмін даними та відслідковування товарів;
- ефективне керування даними. Важливим аспектом є ефективне керування базою даних, що містить інформацію про товари, їх рух та наявність на складі. Програмне забезпечення повинно забезпечити швидкий доступ до цих даних, їх збереження та актуалізацію, щоб забезпечити оптимальні умови для логістичних операцій;
- інтерфейс користувача. Важливим є також наявність зручного та інтуїтивно зрозумілого інтерфейсу користувача. Програмне забезпечення повинно бути легко засвоюваним персоналом магазину та забезпечувати зручні умови для взаємодії з даними та виконання різних операцій.

1.3 Огляд існуючих рішень

При розробці нового програмного забезпечення для оптимізації логістичних процесів у роздрібних магазинах, важливо переглянути існуючі рішення на ринку. Це дозволить краще зрозуміти поточні тенденції, переваги та недоліки існуючих систем, а також визначити можливості для вдосконалення та інновацій.

Огляд існуючих рішень включає аналіз функціональності, продуктивності, безпеки, простоти використання та інших функцій програмних інструментів, які вже використовуються в галузі роздрібної торгівлі. Для цього можна використовувати такі ресурси, як огляди продуктів, аналітичні звіти, відгуки користувачів та інші джерела інформації.

Метою цього огляду є збір та узагальнення інформації про існуючі рішення, виявлення їх переваг та недоліків, а також визначення можливостей для створення нового програмного забезпечення, яке відповідає потребам і вимогам сучасного ринку роздрібної торгівлі.

RetailOps – це комплексне програмне рішення для управління роздрібною торгівлею, яке допомагає магазинам оптимізувати логістичні процеси та ефективно керувати запасами. Ця система надає широкий спектр інструментів, включаючи інвентаризацію, керування замовленнями та постачанням, відстеження вартості продукції та аналіз продажів. Однією з ключових переваг RetailOps є його здатність інтегруватися з різними системами, що дозволяє компаніям легко адаптувати його до своїх потреб і розширювати функціональні можливості за потреби [7].

Крім того, RetailOps забезпечує високий рівень підтримки клієнтів, що дозволяє швидко вирішити будь-які проблеми та отримати необхідну допомогу. Його інтуїтивно зрозумілий і простий у використанні інтерфейс дозволяє персоналу магазину швидко навчитися користуватися системою та зосередитися на наданні високоякісного обслуговування клієнтів. Інтерфейс продемонстровано на рис. 1.1.

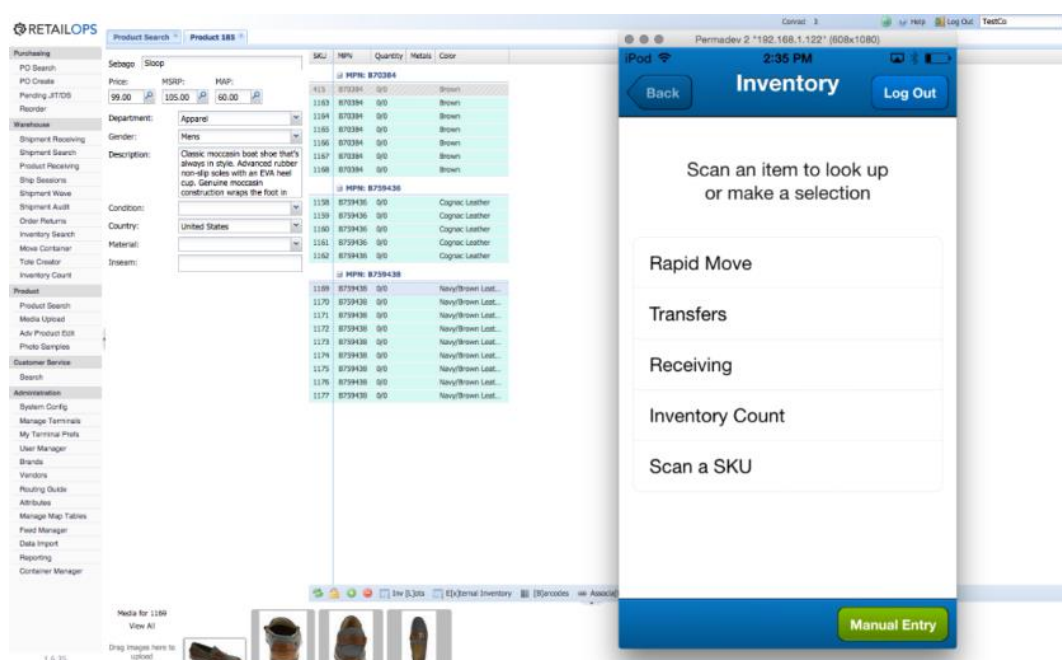


Рисунок 1.1 – Інтерфейс RetailOps

Lightspeed Retail – інтегрована система управління роздрібною торгівлею, яка надає багатофункціональні рішення для магазинів. Однією з головних переваг цієї програми є її інтуїтивно зрозумілий і простий у використанні інтерфейс, який дозволяє персоналу магазину швидко навчитися користуватися системою. За допомогою Lightspeed Retail магазини можуть керувати касовими апаратами, інвентарем, обліковими записами клієнтів тощо [8].

Ця програма також надає зручні інструменти аналізу даних, які дозволяють магазинам розуміти споживчі тенденції та приймати обґрунтовані бізнес-рішення. Крім того, Lightspeed Retail може інтегруватися з різними системами та сервісами, забезпечуючи більшу гнучкість у роботі та можливості розширення функціональності. Інтерфейс продемонстровано на рис. 1.2.

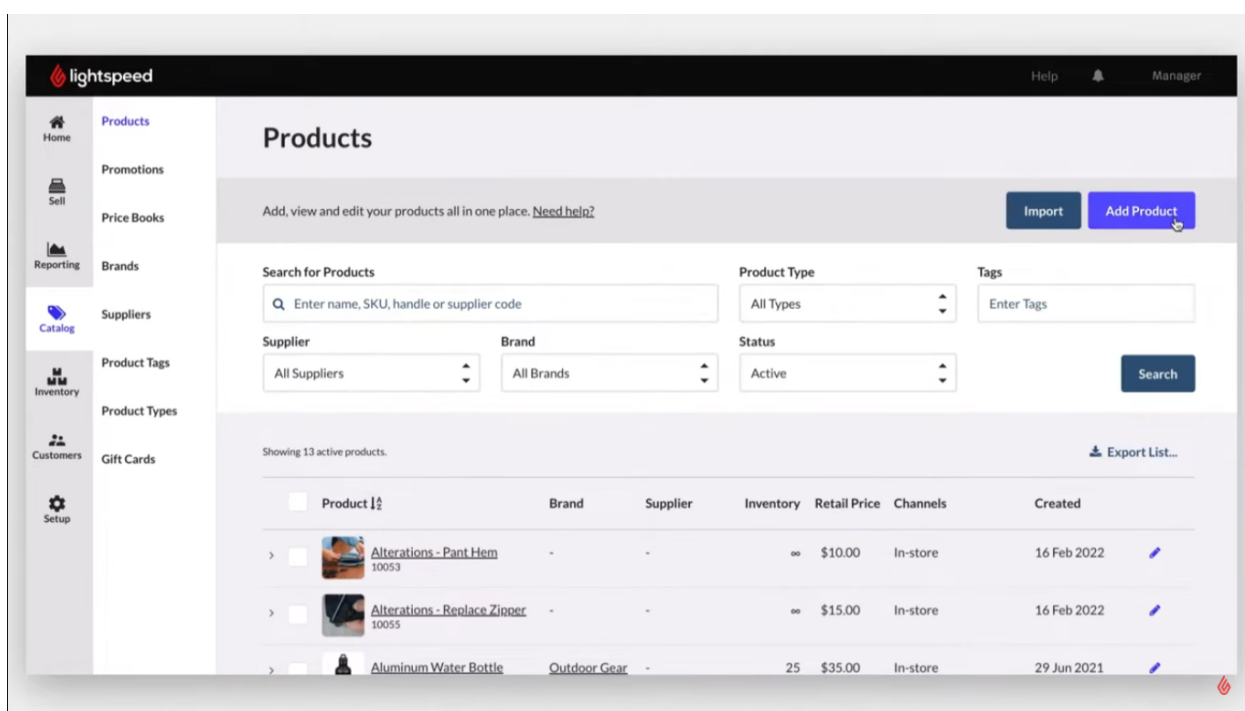


Рисунок 1.2 – Інтерфейс Lightspeed Retail

Vend POS – це програмне забезпечення для управління роздрібною торгівлею, яке відоме своєю ефективністю та простотою використання. Інтерфейс Vend POS дуже інтуїтивно зрозумілий, що дозволяє персоналу магазину легко навчитися користуватися системою. Vend POS дозволяє магазинам керувати

касовими апаратами, управляти запасами, відстежувати клієнтів і виконувати багато інших функцій [9].

Однією з особливостей Vend POS є його гнучкість і можливість налаштувати його під конкретні потреби магазину. Крім того, Vend POS має різні тарифні плани для задоволення різних потреб бізнесу, що робить його більш доступним для різних компаній. Інтерфейс продемонстровано на рис. 1.3.

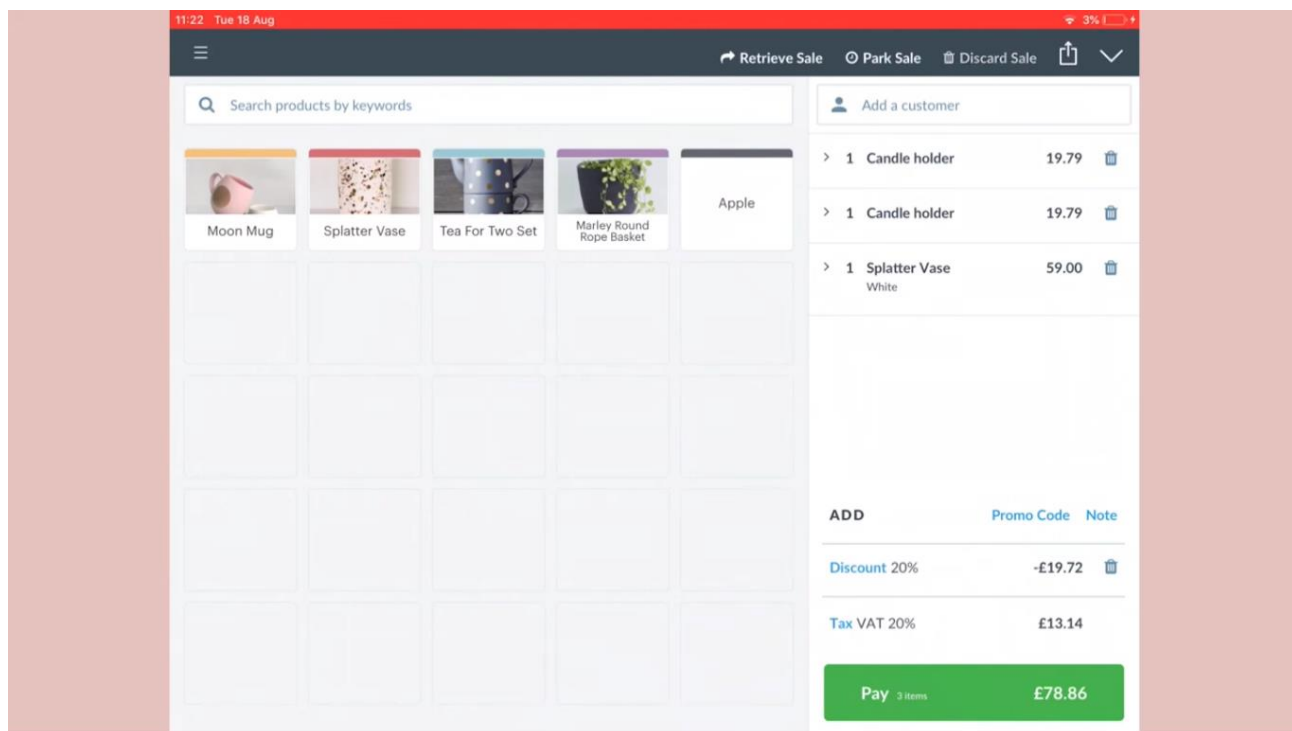


Рисунок 1.3 – Інтерфейс Vend POS

За результатами цього аналізу можна зробити кілька важливих висновків для подальшого розвитку програмного забезпечення. Беручи до уваги переваги та недоліки кожної програми, було обрано кращі функціональні властивості.

Наприклад, RetailOps виділяється своєю високою функціональністю та здатністю інтегруватися з іншими системами, тому розробники можуть спробувати включити подібні функції у своє програмне рішення. З іншого боку, Lightspeed Retail і Vend POS пропонують більш легкі та інтуїтивно зрозумілі інтерфейси, що може бути важливим фактором для користувачів. Тому при

розробці програмного забезпечення було враховано не тільки функціональність, а й зручність використання та наявність підтримки для користувачів.

1.4 Технічне завдання

1.4.1 Вступ

OptShop – це програмне забезпечення створене для оптимізації логістичних процесів магазинів роздрібної торгівлі. Програма створена для використання у магазинах складськими робітниками, касирами та адміністраторами.

1.4.2 Призначення розробки

Програма призначена для автоматизації логістичних процесів магазину роздрібної торгівлі. Вона забезпечує зручний інтерфейс для виконання цих функцій різними категоріями користувачів, що сприяє зниженню людських помилок і підвищенню загальної продуктивності роботи складу.

Функціональним призначенням програми є надання можливості хакеру:

- оптимізувати логістичні процеси замінивши штрих-код на QR-код;
- спростити роботу зі складом та користувачами за допомогою бази даних .

Експлуатаційним призначенням є те, що програма використовується в однокористувацькому режимі та має під'єднана до бази даних.

1.4.3 Вимоги до програми

Програма повинна забезпечувати наступні функції:

- надавання можливості касиру сканувати QR-код з товарів;
- автоматичне обчислювання суми всіх товарів покупця для касира;

- автоматичне введення звіту виторгу за кожен робочий день;
- авторизація користувачів;
- редагування адміністратором всіх таблиць баз даних;
- реєстрування нового товару на складі за допомогою QR-коду;
- проведення інвентаризації за допомогою QR-коду.

Для надійного функціонування програма повинна забезпечувати:

- захист від SQL-ін'єкцій;
- перевірку підключених відео пристроїв;
- перевірку формату QR-коду.

Програма повинна працювати на комп'ютерах з операційною системою Windows 7 або вище, процесором не нижче Intel Core i3, оперативною пам'яттю не менше 2 ГБ та дисковим простором не менше 100 МБ для установки.

Програма повинна бути сумісною з базою даних SQL Server та забезпечувати власний набір функцій для обліку та управління складом.

1.4.4 Техніко-економічні показники

Програма забезпечує підвищення ефективності роботи складу та зменшення кількості помилок, пов'язаних з ручним веденням обліку. Очікувана економічна ефективність полягає в зниженні операційних витрат та скороченні часу на обробку товарних операцій. Багато вітчизняних та закордонних аналогів все ще використовують застарілі штрих-коди в той час як розроблюване програмне забезпечення використовує більш передові QR-коди.

1.4.5 Стадії та етапи розробки

Розробка програми включає наступні стадії та етапи:

- аналіз вимог та планування;
- розробка технічного завдання;
- проектування системи;

- реалізація та кодування;
- тестування;
- впровадження.

1.4.6 Порядок контролю та приймання

Контроль та приймання програми здійснюється шляхом проведення функціонального та навантажувального тестування. Приймальні випробування включають:

- перевірку відповідності функціональних можливостей програмного забезпечення технічному завданню;
- перевірку стабільності роботи програми в різних умовах експлуатації;
- оцінку зручності користування інтерфейсом програми різними категоріями користувачів.

Програма вважається прийнятою після успішного проходження всіх тестів і виправлення виявлених недоліків.

1.5 Висновки за розділом 1

У даному розділі було проведено детальне та порівняльне дослідження програмних рішень для оптимізації логістичних процесів у магазинах роздрібної торгівлі. Результати цього аналізу надають важливу інформацію для розуміння потреб і можливостей у цій сфері.

Під час аналізу було виявлено, що кожна програма має свої переваги та недоліки. RetailOps характеризується високою функціональністю та можливістю інтеграції з іншими системами, що робить його потужним інструментом для управління роздрібною торгівлею. З іншого боку, Lightspeed Retail і Vend POS пропонують простіші та більш інтуїтивні інтерфейси, що може бути важливим для користувачів.

Загальний висновок полягає в тому, що розробка програмного забезпечення для оптимізації логістичних процесів у роздрібних магазинах є важливим і перспективним завданням. Беручи до уваги інформацію, отриману під час аналізу, було створено продукт який відповідає потребам і очікуванням користувачів у цій сфері. Такий продукт може значно підвищити ефективність і конкурентоспроможність магазинів роздрібної торгівлі, сприяючи їх успіху на ринку.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

У процесі розробки програмного забезпечення важливим етапом є вибір засобів проектування та розробки. Це включає в себе вибір мови програмування, середовища розробки, бібліотек та інших інструментів. В цьому розділі наведено аналіз і порівняння інструментів програмування та обґрунтування вибору найкращих інструментів для створення заданого програмного забезпечення.

2.1 Вибір мови програмування та середовища розробки

Найкращими мовами програмування для реалізації створення програмного забезпечення для оптимізації логістичних процесів у роздрібних магазинах будуть:

- Java – одна з найбільш використовуваних мов програмування завдяки своїй портативності та великій кількості бібліотек і фреймворків. Програми, написані на Java, можуть працювати на будь-якій платформі, де встановлено JVM. Однак налаштування середовища розробки для деяких платформ може бути складним, і Java має більше споживання пам'яті та трохи повільніший запуск порівняно з C# [10];
- Python – мова програмування відома своїм простим синтаксисом, який сприяє швидкому розвитку. Має велику кількість бібліотек для різних програм. Однак Python працює повільніше, ніж скомпільовані мови, такі як C#, і має обмежену підтримку багатопоточності через GIL (Global Interpreter Lock) [11];
- C++ – мова програмування, яка забезпечує високу продуктивність і ефективне керування пам'яттю, що робить його чудовим вибором для низькорівневого програмування. Однак складний синтаксис цієї мови ускладнює розробку та обслуговування програм. Відсутність стандартного середовища виконання також створює певні труднощі [12];
- C# — мова програмування, що інтегрується з платформою .NET, яка надає велику кількість бібліотек і платформ. Простота використання порівняно з

C++ і вища продуктивність порівняно з Python роблять C# чудовим вибором для розробки корпоративних програм. Хоча C# спочатку був обмежений платформою Windows, завдяки .NET Core тепер він підтримує крос-платформні рішення [13].

Java та C# обидві дуже добре підходять для реалізації програмного забезпечення, оскільки обидві мови є об'єктно-орієнтованими, мають велику кількість бібліотек та фреймворків для роботи з відеопотоком та базами даних. Java, як і C#, підтримує крос-платформенність, що може бути корисним у певних випадках. Проте, існують певні фактори, що роблять C# більш оптимальним вибором для даної програми, які наведені у табл. 2.1.

Таблиця 2.1 – Порівняння Java та C#

Критерій	C#	Java
Інтеграція з Windows	Відмінна інтеграція з Windows та .NET	Висока інтеграція з Windows
Інструменти розробки	Visual Studio з багатим набором інструментів	IntelliJ IDEA, Eclipse, NetBeans
Робота з відеопотоком	AForge.Video, простота інтеграції	OpenCV, складніша інтеграція
Розпізнавання QR-кодів	ZXing, хороша підтримка бібліотек	ZXing, хороша підтримка бібліотек
Бази даних	Entity Framework, LINQ	Hibernate, JDBC
Продуктивність	Висока продуктивність на Windows	Висока продуктивність на будь-якій платформі

C# було обрано для розробки цієї програми з кількох ключових причин. Перш за все, C# забезпечує глибоку інтеграцію з Windows, що важливо, оскільки програма орієнтована на роботу в цій операційній системі. По-друге, середовище розробки Visual Studio надає найкращі інструменти для налагодження, тестування та розгортання програм, що спрощує процес розробки та підтримки програми. Крім того, бібліотека AForge.Video та ZXing, доступні для C#, дозволяють

ефективно реалізувати захоплення відео та функцію розпізнавання QR-коду, що є ключовими вимогами для нашої програми. Тому, враховуючи всі вищезазначені фактори C# було обрано як основну мову програмування для розробки цього проекту.

Visual Studio було обрано основним середовищем розробки для програмного забезпечення з кількох ключових причин. По-перше, Visual Studio пропонує чудову інтеграцію з мовою програмування C# і платформою .NET, що значно спрощує процес розробки, тестування та розгортання програм. По-друге, Visual Studio забезпечує потужні інструменти відлагодження, що дозволяє швидко знаходити та виправляти помилки у коді, підвищуючи ефективність розробки та якість кінцевого продукту. Це особливо важливо для нашого проекту, де стабільність та надійність програмного забезпечення є критично важливими.

Крім того, Visual Studio має інтегровані інструменти для роботи з базами даних, що дозволяє легко керувати базами даних, такими як SQL Server, прямо з середовища розробки. Це спрощує процес розробки баз даних та забезпечує ефективну взаємодію між додатком та базою даних. Завдяки своїм потужним можливостям, високій продуктивності та зручності використання.

2.2 Вибір бази даних

Для ефективного зберігання та обробки даних у програмі керування оптовим магазином необхідно вибрати відповідну систему керування базами даних (СУБД). Розглянемо кілька популярних СУБД і обґрунтуємо вибір SQL Server.

– MySQL – це програмне забезпечення з відкритим кодом, яке широко підтримується спільнотою. Він забезпечує високу продуктивність читання даних і має велику кількість інструментів управління. Однак MySQL має обмежену підтримку складних транзакцій порівняно з іншими СУБД і має менш потужну інтеграцію з екосистемою Microsoft [14];

– PostgreSQL також є програмним забезпеченням з відкритим кодом, відомим своєю високою відповідністю стандартам SQL і багатим набором функцій. Він підтримує складні транзакції та є розширюваним. Однак PostgreSQL може бути складніше налаштувати та керувати новими користувачами, а його інтеграція з продуктами Microsoft менш надійна [15];

– SQLite – це легка, проста у використанні СУБД, яка не потребує встановлення сервера. Він ідеально підходить для невеликих проєктів або додатків з невеликими обсягами даних. Однак SQLite не підходить для великих проєктів, які потребують багатокористувацького доступу та складних транзакцій [16];

– Microsoft SQL Server – надійне рішення, яке забезпечує високий рівень продуктивності, безпеки та зручності керування даними. Ця база даних інтегрується з іншими продуктами Microsoft, такими як Visual Studio та .NET Framework [17].

З цих варіантів для проєкту було розглянуто два варіанти реляційних баз даних: Microsoft SQL Server і MySQL. Ці системи керування базами даних були обрані для порівняння через їх широку популярність, надійність і підтримку великих обсягів даних. Крім того, обидві системи керування базами даних мають великий досвід роботи в комерційних і корпоративних середовищах, що робить їх майже рівними з точки зору продуктивності, інтеграції та безпеки. Однак все ж було обрано Microsoft SQL Server, за порівнянням наведеним у табл. 2.2.

Таблиця 2.2 – Порівняння SQL Server та MySQL

Критерій	SQL Server	MySQL
Продуктивність	Висока продуктивність	Висока продуктивність
Інтеграція з .NET	Повна інтеграція з .NET і Visual Studio	Менш зручна інтеграція

Продовження таблиці 2.2

Критерій	SQL Server	MySQL
Безпека	Дуже високий рівень безпеки та контроль доступу	Високий рівень безпеки
Підтримка та документація	Широка підтримка від Microsoft	Велика спільнота та документація
Інструменти адміністрування	Дуже хороші інструменти адміністрування	Хороші інструменти
Вартість	Висока	Висока

Microsoft SQL Server було обрано для розробки цієї програми з кількох ключових причин. Перш за все, SQL Server забезпечує високу продуктивність, особливо для великих обсягів даних і складних запитів, що критично важливо для програми, яка обробляє значні обсяги інформації. По-друге, інтеграція з .NET і Visual Studio дозволяє легко налаштувати доступ до даних за допомогою ADO.NET, що спрощує розробку та обслуговування додатків. Це особливо важливо, оскільки обраною мовою програмування являється C#, а тісна інтеграція із середовищем розробки Microsoft забезпечує безперебійну роботу.

Крім того, SQL Server пропонує хороші інструменти адміністрування, які полегшують керування базами даних і моніторинг. Високий рівень безпеки та можливості контролю доступу, які надає SQL Server, забезпечують захист конфіденційних даних, що є критично важливим для будь-якої бізнес-програми.

2.3 Визначення функціональних вимог програмного забезпечення

Програмне забезпечення оптимізації логістичних процесів магазинів роздрібної торгівлі, має виконувати наведені функції:

- авторизація;
- зчитування QR-кодів;

- різні ролі користувачів;
- з'єднання з SQL сервером;
- можливість змінювати, оновлювати, видаляти дані користувачів;
- виконання CRUD (Create, Read, Update, Delete) операцій з базою даних;
- зберігання виторгу магазину за кожен робочий день;
- перевірка коректності даних користувача;
- перевірка наявності доступних відео приладів для зчитування QR-коду;
- вибір відео пристрою для зчитування QR-коду;
- наявність унікального меню для кожної ролі користувача;
- можливість виходу з облікового запису;
- наявність інструментів для проведення інвентаризації;
- можливість перегляду імені авторизованого користувача.

У даному програмному забезпеченні реалізовані 3 ролі, а саме робітник складу, касир і адміністратор. Можливості взаємодії з програмним забезпеченням у ролі касира наведено на рис. 2.1 та у переліку нижче:

- обирати відео прилади для сканування QR-коду;
- сканувати QR-код;
- очищати скановані продукти із списку;
- переглядати інформацію про скановані товари;
- переглядати суму товарів;
- завершити транзакцію.

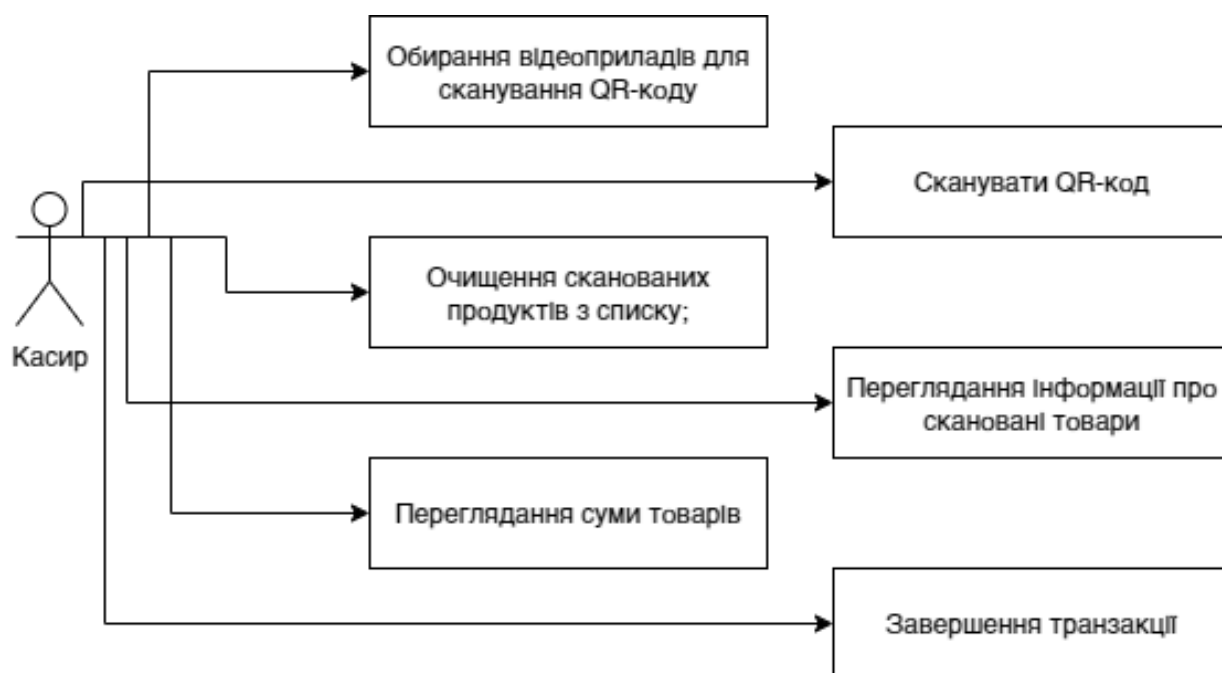


Рисунок 2.1 – Схема можливостей касира

Можливості взаємодії з програмним забезпеченням у ролі робітника складу наведено на рис. 2.2 та у переліку нижче:

- обирати відео прилади для сканування QR-коду;
- сканувати QR-код;
- видалення сканованого товару;
- робити інвентаризацію товарів;
- переглядати прогрес інвентаризації;
- реєструвати новий товар;
- переглядати вміст складу.

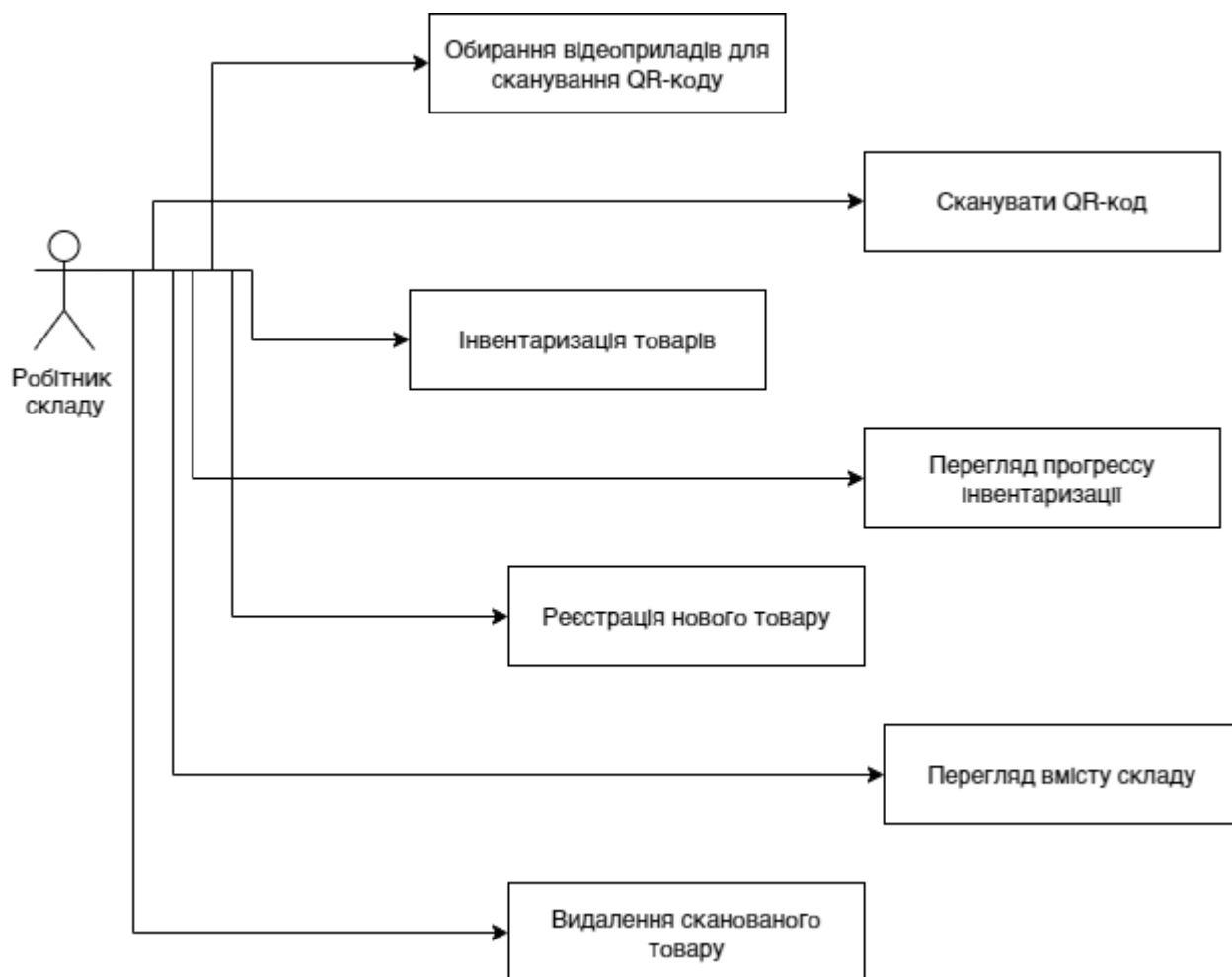


Рисунок 2.2 – Схема можливостей робітника складу

Можливості взаємодії з програмним забезпеченням у ролі адміністратора наведено на рис. 2.3 та у переліку нижче:

- переглядати таблицю користувачів;
- додавати, видаляти, змінювати таблицю користувачів
- переглядати таблицю складу;
- додавати, видаляти, змінювати таблицю складу;
- переглядати таблицю звіту;
- додавати, видаляти, змінювати таблицю звіту;
- зберігати зміни таблиць у базу даних.

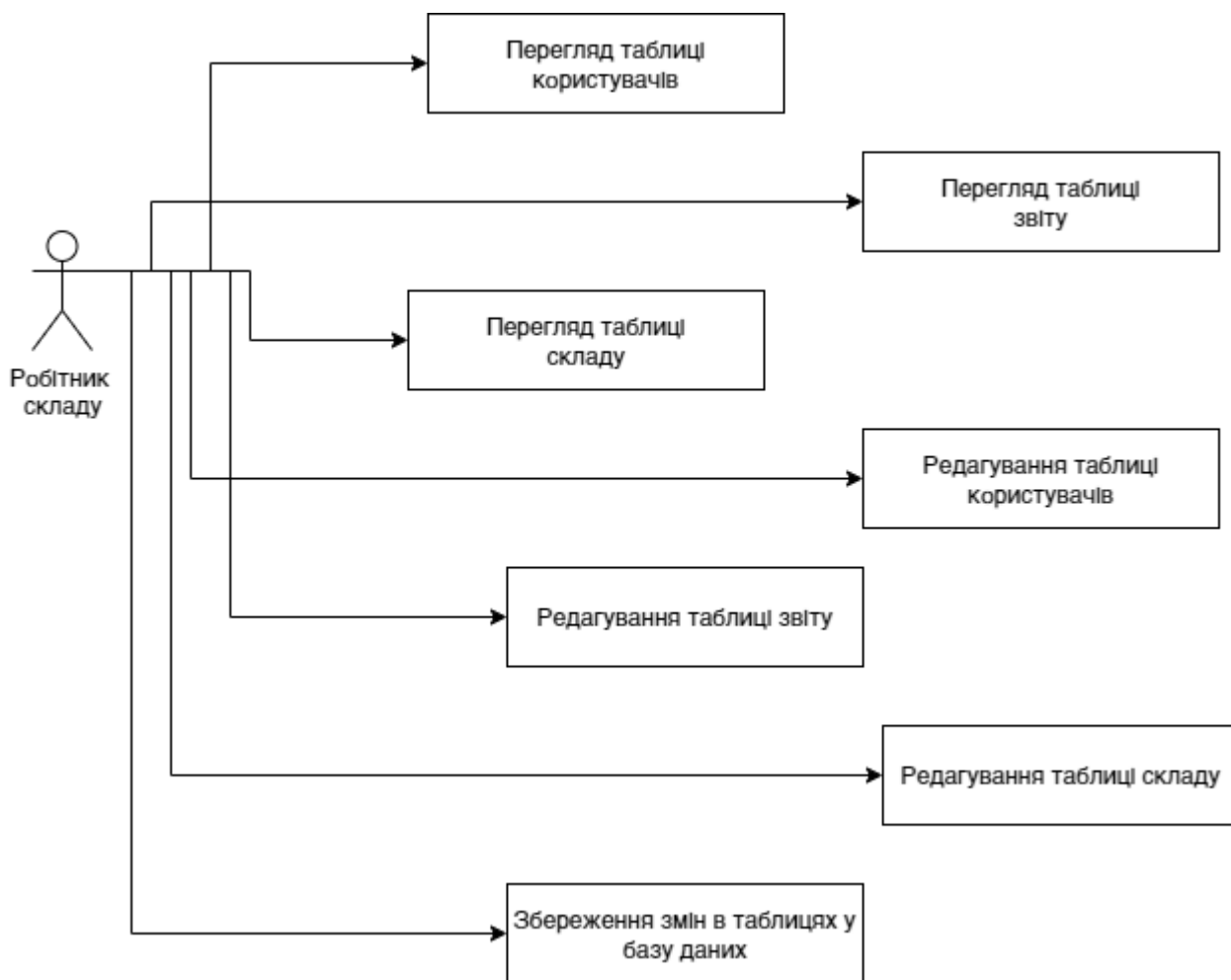


Рисунок 2.3 – Схема можливостей адміністратора

2.4 Визначення структури програми

Програма складається з кількох основних компонентів, кожен з яких виконує певні завдання та взаємодіє з іншими частинами програми. Основними компонентами є форми користувацького інтерфейсу (форма 1, форма 2, форма 3 та форма 4), модулі захоплення відео, обробки QR-кодів, модуль роботи з базою даних та сама база даних SQL Server. Взаємодія між цими компонентами забезпечує функціональність програми.

Форма 1 (Form1) є інтерфейсом для ролі касира. Вона забезпечує доступ до функцій касира, таких як сканування товарів, обробка продажів та оновлення бази даних з інформацією про продажі. Основні функції цієї форми включають сканування товарів, відображення інформації про товар та обробку транзакцій.

Форма взаємодіє з модулем захоплення відео для зчитування QR-кодів та з модулем роботи з базою даних для оновлення інформації про продажі.

Форма 2 (Form2) є формою авторизації. Вона відповідає за ідентифікацію користувачів перед доступом до інших частин програми. Ця форма містить поля для введення логіну та пароля, а також методи для перевірки цих даних у базі даних. Вона взаємодіє з модулем роботи з базою даних для перевірки облікових даних користувача та надання доступу до відповідних форм.

Форма 3 (Form3) є формою адміністратора. Вона надає адміністраторам доступ до функцій управління програмою, таких як додавання нових товарів, редагування існуючих даних, видалення товарів та перегляд звітів. Адміністратор може використовувати цю форму для керування базою даних та виконання інших адміністративних завдань. Форма взаємодіє з модулем роботи з базою даних для виконання CRUD-операцій (створення, читання, оновлення, видалення) над даними.

Форма 4 (Form4) є формою робітника складу. Вона використовується для обробки товарів на складі, включаючи сканування товарів, додавання нових товарів до бази даних та оновлення кількості товарів. Ця форма взаємодіє з модулями захоплення відео та обробки QR-кодів для автоматизації процесу інвентаризації. Вона також взаємодіє з модулем роботи з базою даних для оновлення кількості товарів на складі.

Модуль захоплення відео відповідає за захоплення відео з камери. Цей модуль має властивість пристрою захоплення відео та методи для початку захоплення, зупинки захоплення та обробки нових кадрів. Модуль забезпечує безперервне отримання кадрів з відеопотоку, які потім передаються для подальшої обробки.

Модуль обробки QR-кодів займається обробкою QR-кодів, зчитаних з відеокадрів. Основними властивостями цього модуля є зчитувач штрих-кодів. Методи модуля включають декодування кадрів, отриманих з відео, та обробку результатів декодування. Цей модуль відповідає за ідентифікацію та витяг інформації з QR-кодів, яка потім використовується для оновлення бази даних.

Модуль роботи з базою даних відповідає за взаємодію з базою даних SQL Server. Основними властивостями цього модуля є рядок підключення до бази даних. Методи модуля включають додавання або оновлення інформації про продукт, отримання кількості продуктів, оновлення кількості продуктів та отримання всіх продуктів з бази даних. Цей модуль забезпечує збереження та отримання даних про продукти, а також їх оновлення на основі сканованих QR-кодів.

База даних SQL Server зберігає інформацію про продукти у таблиці Products. Таблиця містить поля, такі як код продукту, назва продукту, показова назва продукту, країна походження, компанія-виробник, ціна та кількість. Ця структура дозволяє ефективно зберігати та отримувати дані про продукти для подальшої обробки програмою.

Взаємодія між компонентами програми здійснюється таким чином, що кожна форма використовує відповідні модулі для виконання своїх функцій. Наприклад, форма 1 для касира використовує модулі захоплення відео та обробки QR-кодів для зчитування товарів, а також модуль роботи з базою даних для оновлення інформації про продажі. Форма 2 для авторизації взаємодіє з модулем роботи з базою даних для перевірки облікових даних користувачів. Форма 3 для адміністратора використовує модуль роботи з базою даних для виконання CRUD-операцій над даними товарів. Форма 4 для робітника складу використовує модулі захоплення відео та обробки QR-кодів для інвентаризації, а також модуль роботи з базою даних для оновлення кількості товарів.

Кожен модуль виконує свої специфічні функції, а разом вони забезпечують виконання загальних завдань програми, таких як сканування продуктів, оновлення бази даних та відображення інформації користувачу.

Для кращого розуміння структури програми, на рис. 2.4 наведено структурну схему програми, яка відображає основні компоненти програми та взаємозв'язки між ними.

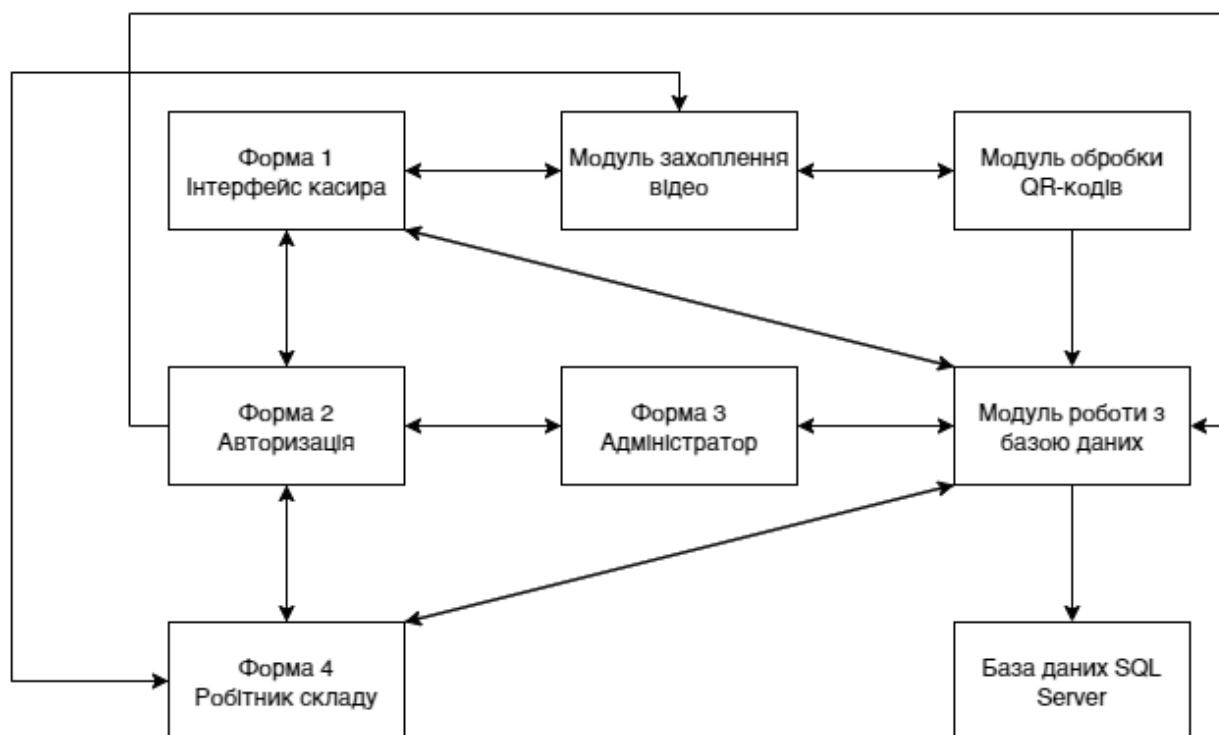


Рисунок 2.4 – Структурна схема програми

Ця схема демонструє взаємодію між формами, модулями та базою даних, підкреслюючи роль кожного компонента в загальній архітектурі програми.

2.5 Висновки за розділом 2

Результати дослідження та розробки програмного забезпечення складського обліку та управління товарообігом підтвердили важливість ретельного аналізу логічної структури програми. Обрані засоби розробки відповідають вимогам проекту, а функціональні вимоги програми відображають реальні потреби користувачів. Мова програмування та середовище розробки були обрані з урахуванням їх ефективності та здатності забезпечити необхідну функціональність. Вибір бази даних базувався на потребах програми та можливостях роботи з даними.

Визначена структура програми враховує взаємозв'язок компонентів і форм, забезпечуючи зручну та ефективну взаємодію між ними. Загалом цей процес

розробки забезпечив створення ефективного та функціонального програмного забезпечення, яке відповідає потребам бізнесу та користувачів.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм роботи програми

Програма розроблена для надання різної функціональності користувачам, в залежності від їх ролі. Нижче наведено детальний опис логіки роботи для кожної з ролей – адміністратора, робітника складу та касира.

Адміністратор має повний доступ до функцій програми. Він може керувати користувачами, товарами і здійснювати інші адміністративні дії. Основні функції, доступні адміністратору, включають додавання, редагування та видалення користувачів та товарів, перегляд та аналіз звітів про продажі, статистику та інші адміністративні дані, а також управління іншими аспектами програми, такими як налаштування та безпека.

Робітник складу відповідає за управління товарами на складі. Його функціональні обов'язки включають сканування QR-кодів товарів для отримання інформації про них, додавання нових товарів на склад та оновлення інформації про наявні товари, перегляд кількості товарів на складі та їх характеристик, а також проведення інвентаризації на складі.

Касир використовує програму для обробки продажів та видачі товарів покупцям. Його головні обов'язки включають сканування товарів та їх QR-кодів для оформлення продажу, а також обробку скасування операцій.

Ця логіка роботи програми забезпечує кожній з ролей зручні інструменти для виконання її завдань у рамках програмного забезпечення. Кожен користувач отримує доступ до функцій, необхідних для ефективного виконання своїх обов'язків у системі.

3.2 Опис реалізованих таблиць бази даних

Програмне забезпечення для оптимізації логістичних процесів магазинів роздрібної торгівлі має комплексну структуру таблиць у базі даних. Нижче наведено більш детальний розгляд таблиць бази даних.

Таблиця “Users” містить інформацію про користувачів системи, їх повне ім’я, логін, пароль та роль. Більш детальний опис наведено у табл. 3.1.

Таблиця 3.1 – Структура таблиці “Users”

Назва поля	Тип даних	Опис
Login	nvarchar(50)	Логін користувача
Password	nvarchar(50)	Пароль користувача
PIB	nvarchar(50)	Повне ім’я користувача
Role	nvarchar(50)	Роль користувача

Таблиця “Products” містить інформацію про продукти які знаходяться на складі, їх повне ім’я, код, країну, виробника, ціну та кількість. Більш детальний опис наведено у табл. 3.2.

Таблиця 3.2 – Структура таблиці “Products”

Назва поля	Тип даних	Опис
ProductCode	int	Код продукту
ProductName	nvarchar(50)	Скорочене ім’я продукту
ProductShowName	nvarchar(50)	Повне ім’я продукту
OriginCountry	nvarchar(50)	Країна походження
CreatorCompany	nvarchar(50)	Виробник товару
Price	int	Ціна товару
Quantity	int	Кількість товару

Таблиця “Reports” містить інформацію про виторг за певний день. Більш детальний опис наведено у табл. 3.3.

Таблиця 3.3 – Структура таблиці “Reports”

Назва поля	Тип даних	Опис
Date	date	Дата робочого дня
SumMon	int	Виторг за певний день

3.3 Опис реалізованого інтерфейсу

Форма 1 є основним інтерфейсом для ролі касира. Вона забезпечує функціональність для обслуговування клієнтів та здійснення продажів. Елементи інтерфейсу форми 1 наведено в табл. 3.4.

Таблиця 3.4 – Елементи інтерфейсу форми “Form1”

Назва елемента	Тип	Опис
PibLabel	Label	Напис, що показує повне ім'я користувача
TotalCostLabel	Label	Напис, що показує розраховану суму всіх товарів
dataGridPrice	DataGridView	Сітка даних, що показує про скановані товари
pictureBoxCamera	PictureBox	Відображення відео потоку з камери
comboBoxDevices	ComboBox	Вибір доступного відеоприладу
buttonCL	Button	Кнопка відміни транзакції
buttonCLG	Button	Кнопка виходу з облікового запису

Продовження таблиці 3.4

Назва елемента	Тип	Опис
buttonTR	Button	Кнопка завершення транзакції
buttonStart	Button	Кнопка, що починає відображення відео потоку з камери
buttonEnd	Button	Кнопка, що завершує відображення відео потоку з камери

Форма 2 забезпечує функціональність авторизації користувачів. Користувачі вводять свої облікові дані для доступу до системи. Елементи інтерфейсу форми 1 наведено в табл. 3.5.

Таблиця 3.5 – Елементи інтерфейсу форми “Form2”

Назва елемента	Тип	Опис
AuthLabel	Label	Напис, що показує розраховану суму всіх товарів
LogintextBox	TextBox	Сітка даних, що показує про скановані товари
PasswordtextBox	TextBox	Відображення відео потоку з камери
AuthButton	Button	Кнопка, що починає відображення відео потоку з камери

Форма 3 є інтерфейсом для адміністратора. Вона дозволяє адміністратору керувати користувачами, товарами та звітами. Елементи інтерфейсу форми 3

наведено в табл. 3.6.

Таблиця 3.6 – Елементи інтерфейсу форми “Form3”

Назва елемента	Тип	Опис
ANameTable	Label	Напис, що показує повне ім'я користувача
button1	Button	Кнопка для переходу на сторінку складу
button2	Button	Кнопка для переходу на сторінку користувачів
button3	Button	Кнопка виходу з облікового запису
button4	Button	Кнопка для переходу на сторінку звіту
tabPage1	TabControl	Сторінка складу
tabPage2	TabControl	Сторінка користувачів
tabPage3	TabControl	Сторінка звіту
dataGridProducts	DataGrid	Таблиця продуктів на складі з бази даних
SaveButton1	Button	Кнопка, що зберігає інформацію з сітки даних у таблицю продуктів в базі даних
SaveButton2	Button	Кнопка, що зберігає інформацію з сітки даних у таблицю користувачів в базі даних

Продовження таблиці 3.6

Назва елемента	Тип	Опис
dataGridUsers	DataGrid	Таблиця користувачів з бази даних
dataGridReports	DataGrid	Таблиця звітів з бази даних
buttonSaveR	Button	Кнопка, що зберігає інформацію з сітки даних у таблицю звітів в базі даних
splitter1	Splitter	Розділяє область форми для створення меню

Форма 4 забезпечує функціональність для роботи на складі. Вона включає інструменти для сканування QR-кодів та додавання товарів на склад і проведенням інвентаризації. Елементи інтерфейсу форми 4 наведено в табл. 3.7.

Таблиця 3.7 – Елементи інтерфейсу форми “Form4”

Назва елемента	Тип	Опис
WNameTable	Label	Напис, що показує повне ім'я користувача
buttonTab1	Button	Кнопка для переходу на сторінку реєстрації нового товару
buttonTab2	Button	Кнопка для переходу на сторінку інвентаризації

Продовження таблиці 3.7

Назва елемента	Тип	Опис
buttonTab3	Button	Кнопка для переходу на сторінку перегляду складу
Button3	Button	Кнопка виходу з облікового запису
tabPage1	TabControl	Сторінка реєстрації нового товару
tabPage2	TabControl	Сторінка інвентаризації товару
tabPage3	TabControl	Сторінка перегляду вмісту складу
splitter1	Splitter	Розділяє область форми для створення меню
dataGridWare	DataGrid	Таблиця продуктів на складі з бази даних
pictureBoxCamera	PictureBox	Відображення відео потоку з камери
pictureBoxCamera2	PictureBox	Відображення відео потоку з камери
comboBoxDevices	ComboBox	Вибір доступного відеоприладу
comboBoxDevices2	ComboBox	Вибір доступного відеоприладу
richInvBox	RichTextBox	Поле для виводу прогресу інвентаризації
StartInvButton	Button	Кнопка початку інвентаризації

Продовження таблиці 3.7

Назва елемента	Тип	Опис
EndInvButton	Button	Кнопка кінця інвентаризації
buttonStart	Button	Кнопка початку реєстрації нового товару
ClearButton	Button	Кнопка очистки останніх просканиваних даних QR-коду
AddButton	Button	Додати останній просканий QR-код до бази даних
QRData	Label	Напис, що показує останній просканий QR-код

Ці форми та їх елементи забезпечують функціональність, необхідну для ефективної роботи кожної ролі в системі. Кожна форма містить специфічні елементи інтерфейсу, що дозволяють користувачам виконувати їхні завдання швидко та зручно.

3.4 Опис використаних бібліотек

Для роботи з відеопотоками використовується бібліотека AForge.Video, яка дозволяє отримувати зображення з камери в реальному часі. Цей функціонал є основою для впровадження сканування QR-кодів, яке є ключовою частиною програми. AForge.Video пропонує можливість підключатися до відеопристроїв, отримувати з них кадри та обробляти їх у реальному часі.

Бібліотека AForge.Video.DirectShow розширює AForge.Video, надаючи підтримку DirectShow для відеопристроїв. Це дозволяє отримувати відеопотоки з

підключених до комп'ютера камер і використовувати їх для подальшої обробки. У даному програмному забезпеченні ця бібліотека використовується для вибору певного відеопристрою та отримання з нього кадрів.

Бібліотека ZXing використовується для читання та декодування QR-кодів. Надає інструменти для розпізнавання різних типів штрих-кодів, зокрема QR-кодів. У розроблюваному програмному забезпеченні бібліотека ZXing використовується для обробки отриманих з відеопотоку кадрів і виявлення в них QR-кодів. Після виявлення QR-коду бібліотека декодує його вміст, дозволяючи відновити закодовані в ньому дані.

Бібліотека System.Data.SqlClient забезпечує взаємодію з базою даних SQL Server. Надає класи та методи для виконання запитів SQL, отримання даних із бази даних, вставки, оновлення та видалення записів. Програма використовує цю бібліотеку для роботи з таблицею продуктів, звітів та користувачів.

Бібліотека System.Drawing є частиною .NET Framework і надає засоби для роботи з графікою. Вона дозволяє створювати, маніпулювати і відображати зображення та текст на формах, а також виконувати різні графічні операції. У даному програмному забезпеченні ця бібліотека використовується для відображення відеопотоків з відеопристрою та обробки зображень, зокрема для сканування QR-кодів.

3.4 Висновки за розділом 3

У цьому розділі детально розібрано розробку програми для автоматизації логістичних процесів в магазині роздрібної торгівлі. Було описано алгоритм роботи програми, особливості реалізації різних компонентів, а також інтерфейс користувача для кожної з ролей.

Була розглянута структура бази даних, яка включає таблиці для зберігання інформації про користувачів, продуктів та звітів. Також розглядалося використання бібліотек AForge.Video, AForge.Video.DirectShow, ZXing,

System.Data.SqlClient і System.Drawing. Ці бібліотеки забезпечують потокове відео, сканування QR-кодів, взаємодію з базою даних і графіку.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис реалізації програми

Програма складається з декількох основних форм, кожна з яких реалізує специфічні функції для різних ролей: касир, адміністратор та робітник складу. Нижче наведено опис основних класів, методів та алгоритмів.

Форма 1 відповідає за інтерфейс касира. Її основні методи забезпечують функціональність для обробки продажів та управління чеками. В розробленій програмі декілька форм мають в собі функціонал сканування QR-коду, але для цього спочатку треба підключити відео пристрій для сканування QR-коду. Після реалізації вибору відео пристрою, користувач натисканням кнопки запускає трансляцію з відео пристрою на форму. Код представлено на рис. 4.1.

```
private void buttonStart_Click(object sender, EventArgs e)
{
    if (comboBoxDevices.SelectedIndex >= 0)
    {
        videoCaptureDevice = new VideoCaptureDevice(filterInfoCollection[comboBoxDevices.SelectedIndex].MonikerString);
        videoCaptureDevice.NewFrame += VideoCaptureDevice_NewFrame;
        videoCaptureDevice.Start();
    }
    else
    {
        MessageBox.Show("Виберіть відеоприлад для початку захвата відео.");
    }
}
```

Рисунок 4.1 – Реалізація трансляції з відео пристрою

Для трансляції зображення з відео пристрою, використовується метод VideoCaptureDevice_NewFrame. Цей метод використовується для обробки нових кадрів, отриманих з відео пристрою. Метод є обробником події NewFrame об'єкта VideoCaptureDevice, який належить до бібліотеки AForge.Video. Також цей метод реалізовує сканування QR-коду. Сканування робиться тільки після 5 секундної паузи яку запроваджує метод CanScan, щоб уникнути багаторазового сканування одного і того ж самого коду. Код представлено на рис. 4.2.

```

private void VideoCaptureDevice_NewFrame(object sender, NewFrameEventArgs eventArgs)
{
    pictureBoxCamera.Image = (Bitmap)eventArgs.Frame.Clone();

    try
    {
        Result result = barcodeReader.Decode((Bitmap)eventArgs.Frame.Clone());
        if (result != null && CanScan())
        {
            ProductInfo product = new ProductInfo(result.Text);
            AddProductToGrid(product);
            lastScanTime = DateTime.Now;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Помилка сканування QR-коду: " + ex.Message);
    }
}

//COMMAND 1
private bool CanScan()
{
    return (DateTime.Now - lastScanTime).TotalSeconds >= 5;
}

```

Рисунок 4.2 – Реалізація трансляції з відео пристрою

Для використання даних з QR-коду використовується метод AddProductToGrid. Цей метод звіряє дані про продукт з базою даних і якщо не знаходить помилок, надсилає дані з QR-коду до сітки даних. Та рахує суму всіх товарів із сітки даних. Код представлено на рис. 4.3.

```

private void AddProductToGrid(ProductInfo product)
{
    if (InvokeRequired)
    {
        Invoke(new Action<ProductInfo>(AddProductToGrid), product);
        return;
    }
    bool productExists = CheckProductExistsInDatabase(product.Code);

    if (productExists)
    {
        DataGridViewRow existingRow = null;
        foreach (DataGridViewRow row in dataGridPrice.Rows)
        {
            if (row.Cells[6].Value != null && row.Cells[6].Value.ToString() == product.Code)
            {
                existingRow = row;
                break;
            }
        }

        if (existingRow != null)
        {
            existingRow.Cells[6].Value = Convert.ToInt32(existingRow.Cells[6].Value) + product.Quantity;
        }
        else
        {
            DataGridViewRow row = new DataGridViewRow();
            row.CreateCells(dataGridPrice, product.Code, product.Name, product.Description, product.Country, product.Brand, product.Price, product.Quantity);
            dataGridPrice.Rows.Add(row);
        }

        totalCost += product.Price * product.Quantity;
        UpdateTotalCost();
    }
    else
    {
        MessageBox.Show("Продукт не знайдено в базі даних.", "Помилка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

Рисунок 4.3 – Реалізація додання даних з QR-коду в сітку даних

Після натискання кнопки закінчення трансляції касир та за допомогою методу `UpdateProductsTable` оновлює кількість проданого товару. Код представлено на рис. 4.4.

```

private void UpdateProductsTable()
{
    foreach (DataGridViewRow row in dataGridPrice.Rows)
    {
        if (row.Cells[0].Value != null)
        {
            string productCode = row.Cells[0].Value.ToString();
            int quantity = Convert.ToInt32(row.Cells[6].Value);

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                string query = "UPDATE Products SET Quantity = Quantity - @Quantity WHERE ProductCode = @ProductCode";

                using (SqlCommand command = new SqlCommand(query, connection))
                {
                    command.Parameters.AddWithValue("@ProductCode", productCode);
                    command.Parameters.AddWithValue("@Quantity", quantity);

                    connection.Open();
                    command.ExecuteNonQuery();
                }
            }
        }
    }
}

```

Рисунок 4.4 – Оновлення таблиці продуктів в базі даних

Також кнопка закінчення трансляції очищує сітку даних та за допомогою методу `UpdateReportsTable` оновлює виторг за відповідний день. Код представлено на рис. 4.5.

```

private void UpdateReportsTable(DateTime currentDate, decimal totalSum)
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string query = "IF EXISTS (SELECT 1 FROM Reports WHERE Date = @Date) " +
            "BEGIN " +
            "    UPDATE Reports SET SumMon = SumMon + @TotalSum WHERE Date = @Date " +
            "END " +
            "ELSE " +
            "BEGIN " +
            "    INSERT INTO Reports (Date, SumMon) VALUES (@Date, @TotalSum) " +
            "END";

        currentDate = currentDate.Date;

        using (SqlCommand command = new SqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@Date", currentDate);
            command.Parameters.AddWithValue("@TotalSum", totalSum);

            connection.Open();
            command.ExecuteNonQuery();
        }
    }
}

```

Рисунок 4.5 – Оновлення таблиці звітів в базі даних

Форма 2 забезпечує функції авторизації користувачів, перевіряючи їхні облікові дані. При натисканні на кнопку входу, виконується аналіз введених

користувачем облікових даних. При вводі неправильних даних користувач отримає повідомлення про помилку, а в зворотному випадку почнеться перевірка ролі користувача. Код представлено на рис. 4.6.

```
private void AuthButton_Click(object sender, EventArgs e)
{
    string login = LoginTextBox.Text;
    string password = PasswordtextBox.Text;

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        try
        {
            connection.Open();
            string query = "SELECT Role, PIB FROM Users WHERE Login = @Login AND Password = @Password";
            SqlCommand command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Login", login);
            command.Parameters.AddWithValue("@Password", password);

            SqlDataReader reader = command.ExecuteReader();
            if (reader.Read())
            {
                string role = reader["Role"].ToString();
                string pib = reader["PIB"].ToString();
                MessageBox.Show($"Ви успішно авторизувались з роллю: {role}");

                OpenNextForm(role, pib);
            }
            else
            {
                MessageBox.Show("Неправильні облікові дані");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Помилка: " + ex.Message);
        }
    }
}
```

Рисунок 4.6 – Перевірка введення облікових даних користувача

Перевірка ролі користувача виконується методом OpenNextForm який перевіряє роль користувача та відкриває відповідну форму. Код представлено на рис. 4.7.

```
private void OpenNextForm(string role, string pib)
{
    switch (role)
    {
        case "Cashier":
            Form1 form1 = new Form1(pib);
            form1.Show();
            this.Hide();
            break;
        case "Admin":
            Form3 form3 = new Form3(pib);
            form3.Show();
            this.Hide();
            break;
        case "Warehouse":
            Form4 form4 = new Form4(pib);
            form4.Show();
            this.Hide();
            break;
        default:
            MessageBox.Show("Неприпустима роль користувача");
            break;
    }
}
```

Рисунок 4.7 – Відкриття нової форми відповідно ролі

Форма 3 відповідає за управління обліковими записами користувачів, а також за коректування продуктів на складі та звітів. Оскільки реалізація редагування кожної з таблиць бази даних реалізоване майже однаково за основу буде взято таблицю користувачів. Також редагування не буде виконуватися при вводі некоректної ролі користувача, тобто яка не відповідає ролі касира, робітника складу та адміністратора. Редагування виконується після редагування сітки даних та натискання кнопки збереження. Код представлено на рис. 4.8.

```
private void SaveButton2_Click(object sender, EventArgs e)
{
    DataTable changedDataTable = ((DataTable)dataGridUsers.DataSource).GetChanges();

    if (changedDataTable != null)
    {
        foreach (DataRow row in changedDataTable.Rows)
        {
            if (!IsValidRole(row["Role"].ToString()))
            {
                MessageBox.Show("Неприпустима роль. Роль повинна бути 'Cashier', 'Admin' або 'Warehouse'.");
                return;
            }
        }

        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            connection.Open();

            SqlTransaction transaction = connection.BeginTransaction();

            try
            {
                UpdateUsers(connection, transaction, changedDataTable);

                transaction.Commit();
                ((DataTable)dataGridUsers.DataSource).AcceptChanges();

                RefreshUserDataGrid();
            }
            catch (Exception ex)
            {
                transaction.Rollback();
                MessageBox.Show("Виникла помилка: " + ex.Message);
            }
        }
    }
}
```

Рисунок 4.8 – Редагування таблиці користувачів в базі даних

При натисканні кнопки збереження використовується метод UpdateUsers, який покроково виконує методи InsertUser, UpdateUser та DeleteUser. Метод InsertUser додає нових користувачів до таблиці. Метод UpdateUser оновлює дані вже створеного користувача. Метод DeleteUser видаляє вже існуючого користувача. Код представлено на рис. 4.9.

```

private void InsertUser(SqlConnection connection, SqlTransaction transaction, DataRow row)
{
    string query = "INSERT INTO Users (Login, Password, PID, Role) * +
        "VALUES (@Login, @Password, @PID, @Role)";

    using (SqlCommand command = new SqlCommand(query, connection, transaction))
    {
        command.Parameters.AddWithValue("@Login", row["Login"]);
        command.Parameters.AddWithValue("@Password", row["Password"]);
        command.Parameters.AddWithValue("@PID", row["PID"]);
        command.Parameters.AddWithValue("@Role", row["Role"]);

        command.ExecuteNonQuery();
    }
}

//SOURCE 1
private void UpdateUser(SqlConnection connection, SqlTransaction transaction, DataRow row)
{
    string query = "UPDATE Users SET Login = @Login, Password = @Password, " +
        "PID = @PID, Role = @Role WHERE Login = @OriginalLogin";

    using (SqlCommand command = new SqlCommand(query, connection, transaction))
    {
        command.Parameters.AddWithValue("@Login", row["Login"]);
        command.Parameters.AddWithValue("@Password", row["Password"]);
        command.Parameters.AddWithValue("@PID", row["PID"]);
        command.Parameters.AddWithValue("@Role", row["Role"]);
        command.Parameters.AddWithValue("@OriginalLogin", row["Login", DataRowVersion.Original]);

        command.ExecuteNonQuery();
    }
}

//SOURCE 1
private void DeleteUser(SqlConnection connection, SqlTransaction transaction, DataRow row)
{
    string query = "DELETE FROM Users WHERE Login = @Login";

    using (SqlCommand command = new SqlCommand(query, connection, transaction))
    {
        command.Parameters.AddWithValue("@Login", row["Login", DataRowVersion.Original]);

        command.ExecuteNonQuery();
    }
}

```

Рисунок 4.9 – Методи додавання, оновлення та видалення даних в таблиці користувачів

Форма 4 призначена для керування товарними запасами на складі. Вона включає методи для додавання нових товарів на склад, а також проведення інвентаризації.

Після натискання кнопки початку проведення інвентаризації виконується метод UpdateInventoryTextBox, який додає інформацію про товари на складі у текстовий бокс через який користувач може бачити прогрес інвентаризації. Код представлено на рис. 4.10.

```

//SOURCE 1
private void UpdateInventoryTextBox()
{
    string query = "SELECT ProductShowName, Quantity FROM Products";

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        SqlCommand command = new SqlCommand(query, connection);
        SqlDataAdapter adapter = new SqlDataAdapter(command);
        DataTable dataTable = new DataTable();

        adapter.Fill(dataTable);

        richInvBox.Clear();

        foreach (DataRow row in dataTable.Rows)
        {
            richInvBox.AppendText(row["ProductShowName"].ToString() + " : " + row["Quantity"].ToString() + Environment.NewLine);
        }
    }
}

```

Рисунок 4.10 – Початок процесу інвентаризації

Далі після кожного сканованого товару, інформація в текст боксі оновлюється показуючи скільки ще товару повинно бути на складі беручи інформацію з таблиці в базі даних. Це виконується за допомогою методу ProcessQRCodeINV, який до того ж перевіряє правильність інформації в QR-коді. Код представлено на рис. 4.11.

```
private void ProcessQRCodeINV(string qrData)
{
    string[] qrValues = qrData.Split(',');

    if (qrValues.Length != 7)
    {
        MessageBox.Show("Некоректна кількість значень QR-коду.");
        return;
    }

    string scannedProductName = qrValues[2];

    richInvBox.Invoke((MethodInvoker)delegate
    {
        string[] lines = richInvBox.Lines;
        bool productFound = false;
        for (int i = 0; i < lines.Length; i++)
        {
            string[] lineParts = lines[i].Split(':');
            if (lineParts.Length == 2)
            {
                string productName = lineParts[0].Trim();
                int quantity = int.Parse(lineParts[1].Trim());
                if (scannedProductName == productName)
                {
                    productFound = true;

                    quantity -= int.Parse(qrValues[6]);

                    lines[i] = $"{productName} : {quantity}";

                    richInvBox.Lines = lines;

                    MessageBox.Show($"Продукт {scannedProductName} успішно відскановано.");

                    break;
                }
            }
        }

        if (!productFound)
        {
            MessageBox.Show($"Продукт {scannedProductName} не знайдено.");
        }
    });
}
```

Рисунок 4.11 – Сканування QR-коду під час інвентаризації

За допомогою методу CalculateInventoryResult підводяться підсумки інвентаризації. Після натискання на кнопку завершення інвентаризації виводиться інформація про її підсумки, а саме те, якого товару було забагато або недостатньо на складі відносно інформації в базі даних. Код представлено на рис. 4.12.

```

private string CalculateInventoryResult()
{
    string result = "Підсумки інвентаризації:" + Environment.NewLine;

    string[] lines = richInvBox.Lines;
    foreach (string line in lines)
    {
        string[] lineParts = line.Split(':');
        if (lineParts.Length == 2)
        {
            string productName = lineParts[0].Trim();
            int actualQuantity = int.Parse(lineParts[1].Trim());

            int expectedQuantity = 8;

            if (actualQuantity < expectedQuantity)
            {
                result += $"{productName} - Забарано товару: {expectedQuantity - actualQuantity}" + Environment.NewLine;
            }
            else if (actualQuantity > expectedQuantity)
            {
                result += $"{productName} - Недостатньо товару: {actualQuantity - expectedQuantity}" + Environment.NewLine;
            }
            else
            {
                result += $"{productName} - В нормі" + Environment.NewLine;
            }
        }
    }

    return result;
}

```

Рисунок 4.12 – Підсумки інвентаризації

Кнопка додавання інформації до бази даних, додає інформацію з щойно просканованого QR-коду до таблиці складу в базі даних. Вона також перевіряє чи вже існує даний товар в базі даних, у випадку чого просто збільшує кількість даного товару. Код представлено на рис. 4.13.

```

private void AddButton_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(QData.Text))
    {
        MessageBox.Show("QR-код не відскановано або дані не відповідають потрібному формату.");
        return;
    }

    string[] qrValues = QDataFull.Text.Split(',');

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();

        string query = "SELECT Quantity FROM Products WHERE ProductShowName = @ProductShowName";
        SqlCommand command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@ProductShowName", qrValues[2].Substring(1));

        object result = command.ExecuteScalar();
        if (result != null)
        {
            int count = (int)command.ExecuteScalar();

            if (result != null && int.TryParse(result.ToString(), out int existingQuantity))
            {
                int newQuantity = existingQuantity + int.Parse(qrValues[0]);
                query = "UPDATE Products SET Quantity = @Quantity WHERE ProductShowName = @ProductShowName";
                command = new SqlCommand(query, connection);
                command.Parameters.AddWithValue("@Quantity", newQuantity);
                command.Parameters.AddWithValue("@ProductShowName", qrValues[2].Substring(1));
                command.ExecuteNonQuery();
                MessageBox.Show("Кількість продукту змінено.");
            }
            else
            {
                query = "INSERT INTO Products (ProductCode, ProductName, ProductShowName, OriginCountry, CreatorCompany, Price, Quantity) * + VALUES (@ProductCode, @ProductName, @ProductShowName, @OriginCountry, @CreatorCompany, @Price, @Quantity)";
                command = new SqlCommand(query, connection);
                command.Parameters.AddWithValue("@ProductCode", int.Parse(qrValues[0]));
                command.Parameters.AddWithValue("@ProductName", qrValues[1].Substring(1));
                command.Parameters.AddWithValue("@ProductShowName", qrValues[2].Substring(1));
                command.Parameters.AddWithValue("@OriginCountry", qrValues[3].Substring(1));
                command.Parameters.AddWithValue("@CreatorCompany", qrValues[4].Substring(1));
                command.Parameters.AddWithValue("@Price", int.Parse(qrValues[5]));
                command.Parameters.AddWithValue("@Quantity", int.Parse(qrValues[6]));
                command.ExecuteNonQuery();
                MessageBox.Show("Продукт додано.");
            }
        }
    }
}

```

Рисунок 4.13 – Реєстрація нового товару на склад

4.2 Інструкція по експлуатації програми

Створене програмне забезпечення має клієнт-серверну архітектуру. Це видно з того, що клієнтська частина програми (форми для касира, адміністратора та робітника складу) взаємодіє з серверною частиною, зокрема з базою даних SQL Server, для зберігання та отримання інформації про товари, користувачів та інші дані.

Клієнтська частина реалізована у вигляді Windows Forms додатків, які надають різні функції в залежності від ролі користувача. Вона включає форми для касира, адміністратора та робітника складу, що дозволяє виконувати різні операції, такі як сканування товарів, управління базою даних товарів, перегляд звітів тощо.

Серверна частина реалізована за допомогою SQL Server, який зберігає всі необхідні дані (товари, користувачі, операції). Клієнтські додатки взаємодіють з сервером через SQL-запити, виконуючи необхідні операції з даними.

Для забезпечення функціонування програмного забезпечення треба щоб комп'ютер відповідав системним вимогам:

- операційна система: Windows 7 x64;
- процесор: Intel Core i3;
- оперативна пам'ять: 2 GB;
- відеопам'ять: 3 GB;
- місце на диску: 50 MB для встановлення програми та додаткове місце

для бази даних.

Для успішного виконання програми потрібне підключення до бази даних, встановлений відео пристрій з оновленими драйверами, та комп'ютер який задовольняє системні вимоги.

Вхідними даними програмного забезпечення являються логін і пароль для авторизації користувача у системі.

Вихідними даними користувача є сторінка користувача, де він може взаємодіяти з базою даних та виконувати роботу яка передбачена його роллю в магазині.

4.3 Тестування програмного забезпечення

Перша форма яку побачить користувач, це форма авторизації. Першим тестом програмного забезпечення буде тестування захисту від SQL ін'єкцій, для захисту даних магазину. Тестування представлено на рис. 4.14.

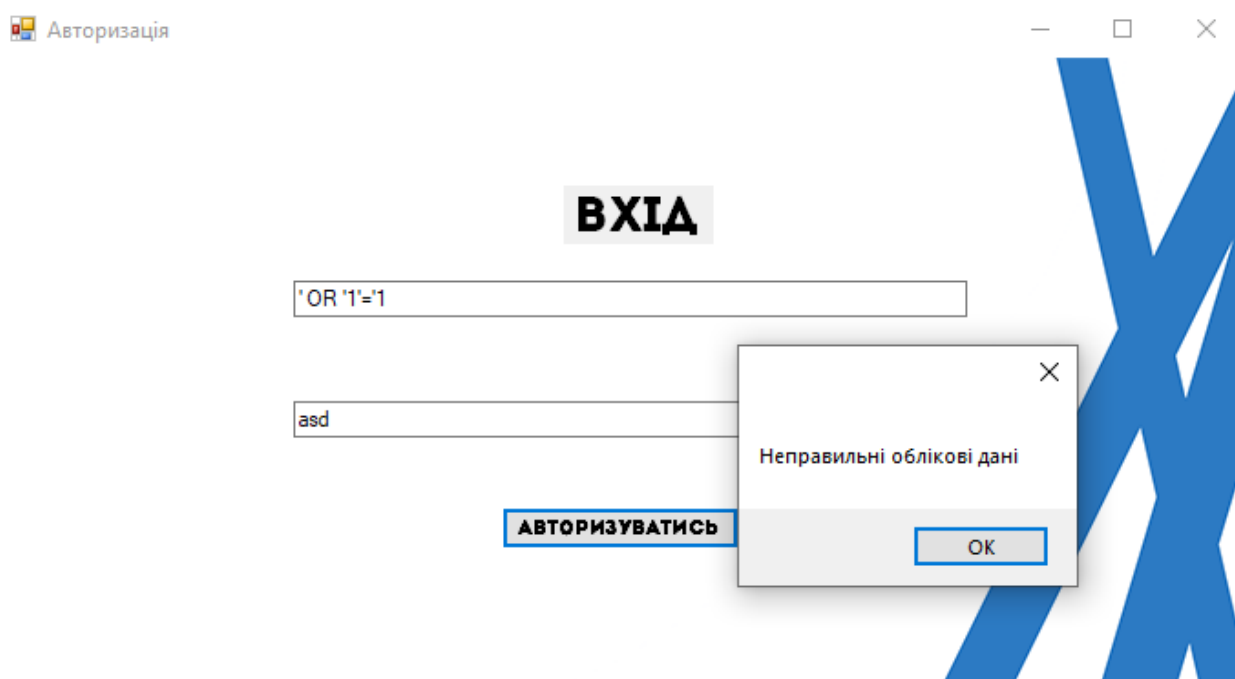


Рисунок 4.14 – Тестування захисту від SQL ін'єкцій

Роль касира та робітника складу потребують підключеного відео пристрою для сканування QR-коду, тому програмне забезпечення повинно перевіряти доступність хоча б одного відео пристрою після авторизації. Тестування представлено на рис. 4.15.

Авторизація

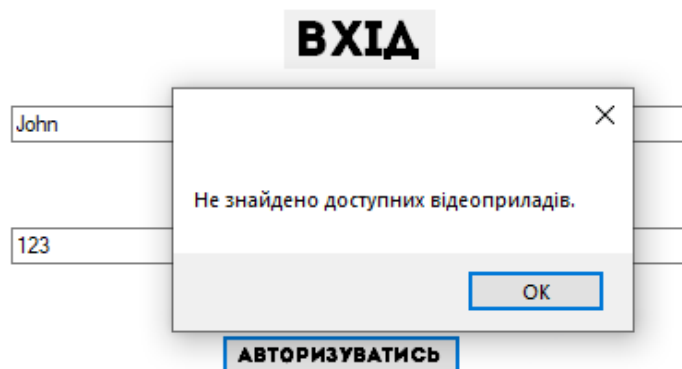


Рисунок 4.15 – Тестування доступних відео пристроїв

Сканований код повинен відповідати формату таблиці бази даних, тож програмне забезпечення має мати перевірку яка не допустить додання до таблиці даних неправильного формату. Тестування представлено на рис. 4.16.

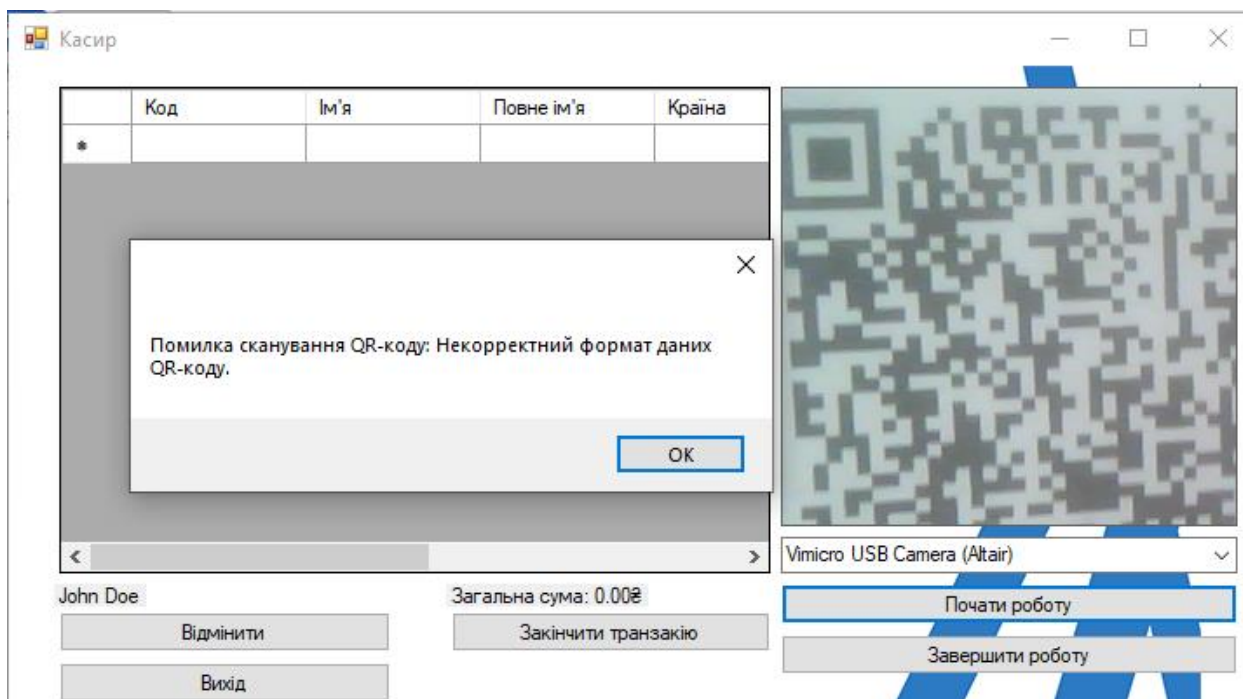


Рисунок 4.16 – Тестування формату QR-коду

Роль касира у програмному забезпеченні закладається в продажі існуючого товару, тобто навіть якщо формат QR-коду задовільний, але просканованого продукту немає в базі даних, просканований товар не може додатися у сітку даних. Тестування представлено на рис. 4.17.

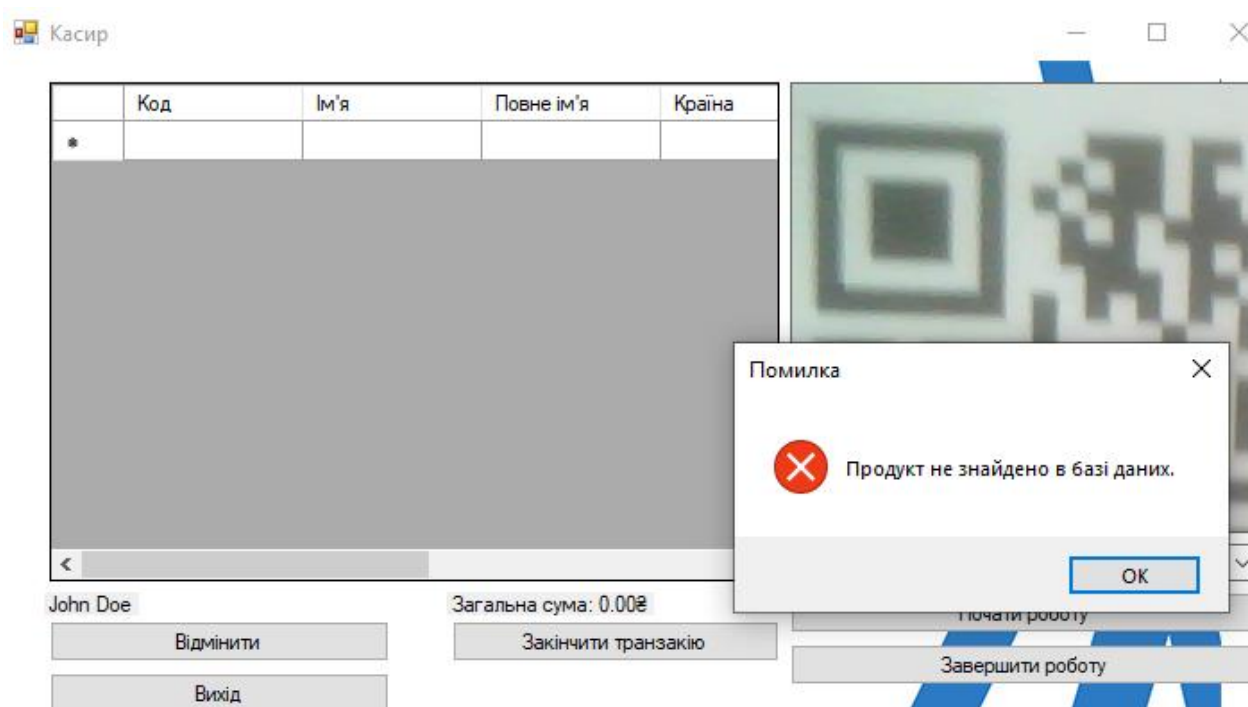


Рисунок 4.17 – Тестування сканування неіснуючого продукту

Після закінчення транзакцій, таблиця звіту в базі даних повинна оновлюватись та додавати суму транзакції до вже існуючої дати або у випадку коли це перша транзакція за день, створювати новий рядок з датою. Тестування представлено на рис. 4.18.

	Date	SumMon
	14.05.2024	18600
	15.05.2024	8305
	23.05.2024	3100

Рисунок 4.18 – Додання нового рядка в таблицю звітів

Користувач з роллю адміністратора може редагувати всі таблиці в базі даних, але в таблиці користувачів некоректна зміна ролі може призвести до того що користувач не зможе авторизуватися. У зв'язку з цим повинна бути реалізована перевірка ролі користувача, а також недоступність для адміністратора змінювати роль користувача на ту, яка не буде співпадати з трьома реалізованими ролями користувача. Тестування представлено на рис. 4.19.

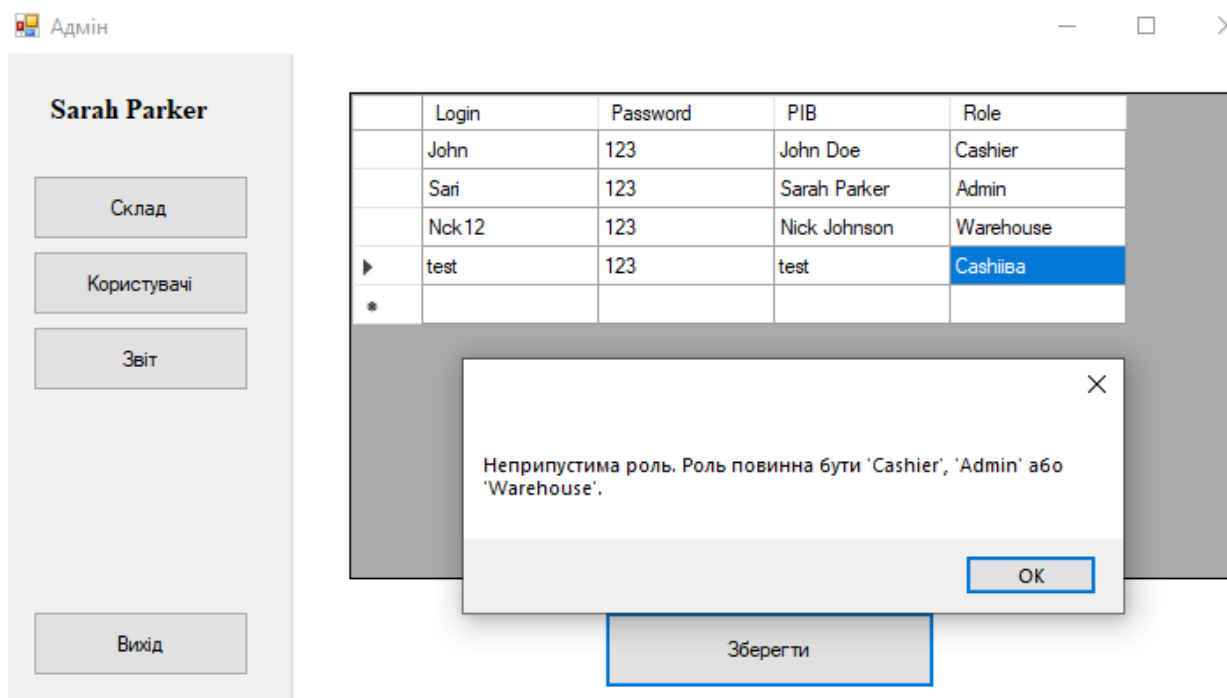


Рисунок 4.19 – Перевірка ролі користувача

Користувач з роллю робітника складу має можливість додання нового товару на склад, тому програмне забезпечення повинно показувати загальну інформацію про товар перед доданням щоб не допустити помилок. Таким чином робітник складу зможе побачити якщо у товару буде невідповідність з даними QR-коду. Тестування представлено на рис. 4.20.

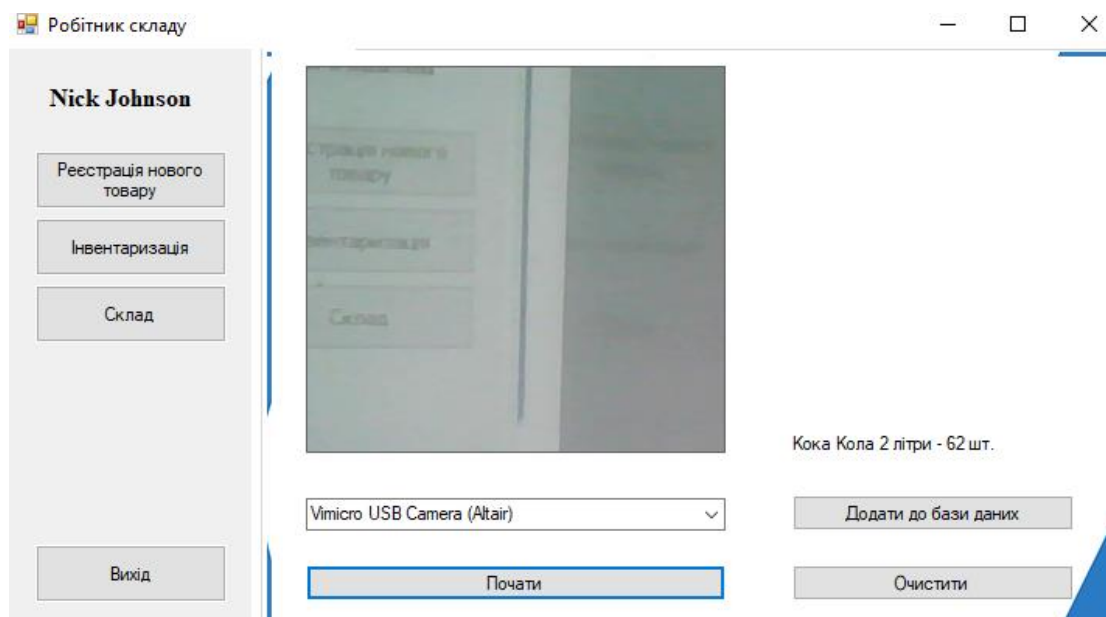


Рисунок 4.20 – Показ загальної інформації про товар перед скануванням

Робітник складу також має обов'язок проводити інвентаризацію, але після сканування QR-коду кожного продукту передивлятися результати кожного продукту буде займати забагато часу і до того ж робітник складу може прогледіти важливу інформацію. Таким чином програмне забезпечення має показувати тільки ті товари, де дані не відповідають тим, що знаходяться в базі даних. Тестування представлено на рис. 4.21.

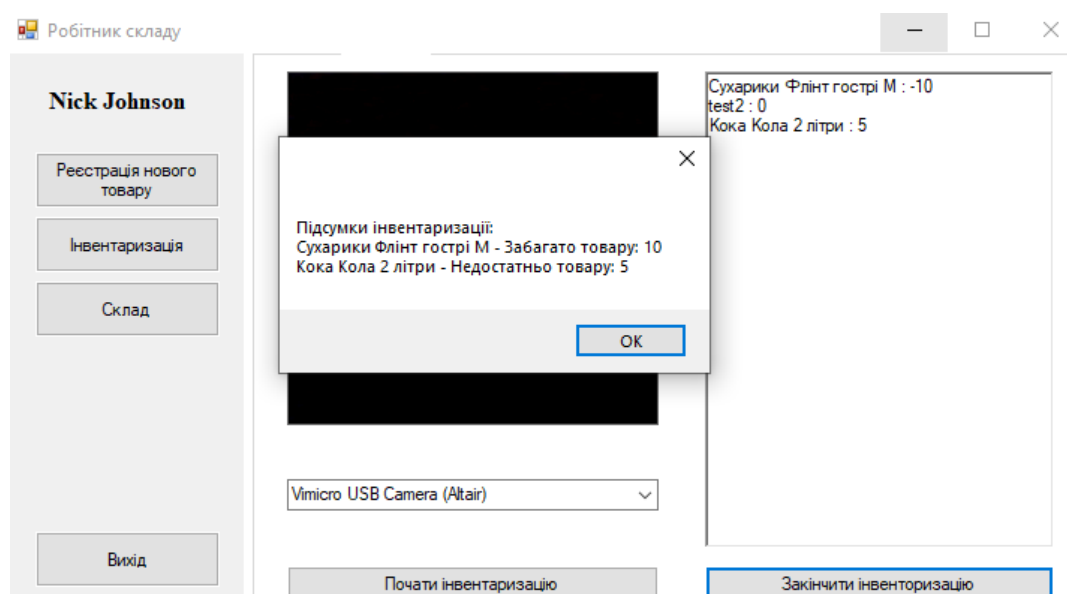


Рисунок 4.21 – Підсумки інвентаризації

4.4 Висновки за розділом 4

Даний розділ демонструє, як розроблена програма досягає поставлених цілей і вирішує завдання, визначені у вступі. Опис застосування програми показує її призначення та області застосування, включаючи мінімальні вимоги до технічних засобів. Проведена методика тестування підтверджує надійність і коректність роботи програми, виявляючи та усуваючи можливі дефекти. Експериментальні дослідження підтверджують ефективність програми в реальних умовах, демонструючи її здатність оптимізувати процеси обліку товарів та керування складом.

ВИСНОВКИ

У дипломній роботі проведено комплексне дослідження та впровадження програми автоматизації логістичних процесів в магазині роздрібної торгівлі. На початковому етапі роботи було проведено аналіз предметної області, визначено основні функціональні вимоги до програмного забезпечення, що дозволило чітко окреслити рамки та цілі проекту. Згодом архітектура програми була розроблена з використанням сучасних підходів і методологій, що забезпечило її надійність і масштабованість.

Для реалізації програми було обрано мову програмування C# та середовище розробки Visual Studio, що забезпечило високу продуктивність та простоту розробки. Також було порівняно кілька систем керування базами даних, і було обрано SQL Server, який найкраще відповідає вимогам безпеки даних і продуктивності проекту.

Розробка інтерфейсу програми враховувала потреби різних ролей користувачів: касира, адміністратора та працівника складу. Для кожної з цих ролей реалізовано низку функцій, що забезпечує зручність та ефективність їх роботи. Програма пройшла комплексне тестування, результати якого підтвердили її точність і надійність.

Загалом проведені дослідження та розробки програми досягли поставлених цілей і завдань. Робота продемонструвала важливість використання сучасних технологій і підходів у магазинах роздрібної торгівлі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. SSCC — ідентифікація у постачальницьких та транспортних процесах [Електронний ресурс]. – Режим доступу : <https://www.vostok.dp.ua/ukr/infa1/shtrih-kod/sscc/>
2. Barcodes and Logistics: Going Beyond Traditional Scanning Methods [Electronic resource]. – Access mode : <https://packagex.io/blog/barcodes-and-logistics>
3. Логістичні стратегії в торгівлі [Електронний ресурс]. – Режим доступу : <https://knute.edu.ua/file/MjExMzA=/ebee6b4b3d12d4cd18180f0f18338a1a.pdf>
4. Логістика [Електронний ресурс]. – Режим доступу : <https://ela.kpi.ua/server/api/core/bitstreams/831e5c8a-1f83-4888-adf7-35d8a0183df6/content>
5. Barcodes vs Qr Codes for Inventory Management [Electronic resource]. – Access mode : <https://www.sortly.com/blog/barcodes-vs-qr-codes-for-inventory-management/>
6. Що таке QR-код та як його створити [Електронний ресурс]. – Режим доступу : <https://apix-drive.com/ua/blog/useful/schto-take-qr-kod>
7. A complete Warehouse Management system built for growth [Electronic resource]. – Access mode : <https://www.retailops.com/products/warehouse-management/>
8. Inventory tracking software built to empower retailers [Electronic resource]. – Access mode : <https://www.lightspeedhq.com/pos/retail/inventory-management-software/>
9. Vend POS [Electronic resource]. – Access mode : <https://www.rtechcenters.com/vend-pos/>
10. Java (programming language) [Electronic resource]. – Access mode : [https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))
11. The Python Tutorial [Electronic resource]. – Access mode : <https://docs.python.org/3/tutorial/index.html>

12. C++ [Electronic resource]. – Access mode :
<https://en.wikipedia.org/wiki/C%2B%2B>
13. A tour of the C# language [Электронный ресурс]. – Режим доступа:
<https://learn.microsoft.com/uk-ua/dotnet/csharp/tour-of-csharp/overview>
14. What is MySQL? [Electronic resource]. – Access mode :
<https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>
15. About PostgreSQL [Electronic resource]. – Access mode :
<https://www.postgresql.org/about/>
16. About SQLite [Electronic resource]. – Access mode :
<https://www.sqlite.org/about.html>
17. What is SQL Server? [Electronic resource]. – Access mode :
<https://learn.microsoft.com/en-us/sql/sql-server/what-is-sql-server?view=sql-server-ver16>

ДОДАТОК А
Текст програми

Form1.cs:

```

using System;
using System.Data.SqlClient;
using System.Drawing;
using System.Threading;
using System.Windows.Forms;
using AForge.Video;
using AForge.Video.DirectShow;
using ZXing;

namespace OptShop
{
    public partial class Form1 : Form
    {
        private FilterInfoCollection filterInfoCollection;
        private VideoCaptureDevice videoCaptureDevice;
        private BarcodeReader barcodeReader;
        private DateTime lastScanTime;
        private decimal totalCost = 0;
        private string userPIB;
        private string connectionString = @"Data Source=DESKTOP-199NT0R;Initial
Catalog=OptShop;Integrated Security=True;";

        public Form1(string pib)
        {
            InitializeComponent();
            PibLabel.Text = pib;
        }

        private void VideoCaptureDevice_NewFrame(object sender, NewFrameEventArgs
eventArgs)
        {
            pictureBoxCamera.Image = (Bitmap)eventArgs.Frame.Clone();

            try
            {
                Result result = barcodeReader.Decode((Bitmap)eventArgs.Frame.Clone());
                if (result != null && CanScan())
                {
                    ProductInfo product = new ProductInfo(result.Text);
                    AddProductToGrid(product);
                    lastScanTime = DateTime.Now;
                }
            }
        }
    }
}

```

```

        catch (Exception ex)
        {
            MessageBox.Show("Помилка сканування QR-коду: " + ex.Message);
        }
    }

    private bool CanScan()
    {
        return (DateTime.Now - lastScanTime).TotalSeconds >= 5;
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        filterInfoCollection = new FilterInfoCollection(FilterCategory.VideoInputDevice);
        foreach (FilterInfo filterInfo in filterInfoCollection)
        {
            comboBoxDevices.Items.Add(filterInfo.Name);
        }
        if (comboBoxDevices.Items.Count > 0)
        {
            comboBoxDevices.SelectedIndex = 0;
        }
        else
        {
            MessageBox.Show("Не знайдено доступних відеоприладів.");
            buttonStart.Enabled = false;
        }

        barcodeReader = new BarcodeReader();
        lastScanTime = DateTime.MinValue;
    }

    private void buttonStart_Click(object sender, EventArgs e)
    {
        if (comboBoxDevices.SelectedIndex >= 0)
        {
            videoCaptureDevice = new VideoCaptureDevice(filterInfoCollection[comboBoxDevices.SelectedIndex].MonikerString);
            videoCaptureDevice.NewFrame += VideoCaptureDevice_NewFrame;
            videoCaptureDevice.Start();
        }
        else
        {

```

```

        MessageBox.Show("Виберіть відеоприлад для початку захвата відео.");
    }
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
    if (videoCaptureDevice != null && videoCaptureDevice.IsRunning)
    {
        videoCaptureDevice.Stop();
    }
}

private void AddProductToGrid(ProductInfo product)
{
    if (InvokeRequired)
    {
        Invoke(new Action<ProductInfo>(AddProductToGrid), product);
        return;
    }
    bool productExists = CheckProductExistsInDatabase(product.Code);

    if (productExists)
    {
        DataGridViewRow existingRow = null;
        foreach (DataGridViewRow row in dataGridPrice.Rows)
        {
            if (row.Cells[0].Value != null && row.Cells[0].Value.ToString() ==
product.Code)
            {
                existingRow = row;
                break;
            }
        }

        if (existingRow != null)
        {
            existingRow.Cells[6].Value =
Convert.ToInt32(existingRow.Cells[6].Value) + product.Quantity;
        }
        else
        {
            DataGridViewRow row = new DataGridViewRow();
            row.CreateCells(dataGridPrice, product.Code, product.Name,
product.Description, product.Country, product.Brand, product.Price, product.Quantity);
            dataGridPrice.Rows.Add(row);
        }
    }
}

```

```

    }

    totalCost += product.Price * product.Quantity;
    UpdateTotalCost();
}
else
{
    MessageBox.Show("Продукт не знайдено в базі даних.", "Помилка",
    MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

private bool CheckProductExistsInDatabase(string productCode)
{
    bool exists = false;
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        string query = "SELECT COUNT(*) FROM Products WHERE ProductCode
= @ProductCode";
        using (SqlCommand command = new SqlCommand(query, connection))
        {
            command.Parameters.AddWithValue("@ProductCode", productCode);
            connection.Open();
            int count = Convert.ToInt32(command.ExecuteScalar());
            exists = (count > 0);
        }
    }
    return exists;
}

private void UpdateTotalCost()
{
    TotalCostLabel.Text = "Загальна сума: " + totalCost.ToString("C");
}

private void buttonTR_Click(object sender, EventArgs e)
{
    DateTime currentDate = DateTime.Today;

    decimal totalSum = CalculateTotalSum();

    UpdateReportsTable(currentDate, totalSum);

    UpdateProductsTable();
}

```

```

        ClearDataGrid();
    }

    private decimal CalculateTotalSum()
    {
        decimal totalSum = 0;
        foreach (DataGridViewRow row in dataGridPrice.Rows)
        {
            if (row.Cells[6].Value != null)
            {
                totalSum += Convert.ToDecimal(row.Cells[6].Value) *
Convert.ToDecimal(row.Cells[5].Value);
            }
        }
        return totalSum;
    }

    private void UpdateReportsTable(DateTime currentDate, decimal totalSum)
    {
        using (SqlConnection connection = new SqlConnection(connectionString))
        {
            string query = "IF EXISTS (SELECT 1 FROM Reports WHERE Date =
@Date) " +
                "BEGIN " +
                "    UPDATE Reports SET SumMon = SumMon + @TotalSum
WHERE Date = @Date " +
                "END " +
                "ELSE " +
                "BEGIN " +
                "    INSERT INTO Reports (Date, SumMon) VALUES (@Date,
@TotalSum) " +
                "END";

            currentDate = currentDate.Date;

            using (SqlCommand command = new SqlCommand(query, connection))
            {
                command.Parameters.AddWithValue("@Date", currentDate);
                command.Parameters.AddWithValue("@TotalSum", totalSum);

                connection.Open();
                command.ExecuteNonQuery();
            }
        }
    }
}

```

```

private void UpdateProductsTable()
{
    foreach (DataGridViewRow row in dataGridPrice.Rows)
    {
        if (row.Cells[0].Value != null)
        {
            string productCode = row.Cells[0].Value.ToString();
            int quantity = Convert.ToInt32(row.Cells[6].Value);

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                string query = "UPDATE Products SET Quantity = Quantity -
@Quantity WHERE ProductCode = @ProductCode";

                using (SqlCommand command = new SqlCommand(query, connection))
                {
                    command.Parameters.AddWithValue("@ProductCode",
productCode);
                    command.Parameters.AddWithValue("@Quantity", quantity);

                    connection.Open();
                    command.ExecuteNonQuery();
                }
            }
        }
    }
}

private void buttonCL_Click(object sender, EventArgs e)
{
    ClearDataGrid();
}

private void ClearDataGrid()
{
    if (InvokeRequired)
    {
        Invoke(new Action(ClearDataGrid));
        return;
    }
}

```

```

        dataGridPrice.Rows.Clear();
        totalCost = 0;
        UpdateTotalCost();
    }

    private void buttonEnd_Click(object sender, EventArgs e)
    {
        if (videoCaptureDevice != null && videoCaptureDevice.IsRunning)
        {
            videoCaptureDevice.Stop();
        }
    }

    private void Form1_FormClosing_1(object sender, FormClosingEventArgs e)
    {
        Environment.Exit(1);
    }

    private void buttonCLG_Click(object sender, EventArgs e)
    {
        Form2 form2 = new Form2();
        form2.Show();
        this.Hide();
    }
}

public class ProductInfo
{
    public string Code { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public string Country { get; set; }
    public string Brand { get; set; }
    public decimal Price { get; set; }
    public int Quantity { get; set; }

    public ProductInfo(string qrData)
    {
        string[] parts = qrData.Split(',');
        if (parts.Length == 7)
        {
            Code = parts[0];
            Name = parts[1];
            Description = parts[2];

```

```

        Country = parts[3];
        Brand = parts[4];
        Price = Convert.ToDecimal(parts[5]);
        Quantity = Convert.ToInt32(parts[6]);
    }
    else
    {
        throw new ArgumentException("Некорректный формат данных QR-кода.");
    }
}
}
}

```

Form2.cs:

```

using System;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace OptShop
{
    public partial class Form2 : Form
    {
        string connectionString = @"Data Source=DESKTOP-199NT0R;Initial
Catalog=OptShop;Integrated Security=True;";

        public Form2()
        {
            InitializeComponent();
        }

        private void AuthButton_Click(object sender, EventArgs e)
        {
            string login = LogintextBox.Text;
            string password = PasswordtextBox.Text;

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                try
                {
                    connection.Open();
                    string query = "SELECT Role, PIB FROM Users WHERE Login =
@Login AND Password = @Password";
                    SqlCommand command = new SqlCommand(query, connection);
                    command.Parameters.AddWithValue("@Login", login);

```

```

command.Parameters.AddWithValue("@Password", password);

SqlDataReader reader = command.ExecuteReader();
if (reader.Read())
{
    string role = reader["Role"].ToString();
    string pib = reader["PIB"].ToString();
    MessageBox.Show($"Ви успішно авторизувались з роллю: {role}");

    OpenNextForm(role, pib);
}
else
{
    MessageBox.Show("Неправильні облікові дані");
}
}
catch (Exception ex)
{
    MessageBox.Show("Помилка: " + ex.Message);
}
}
}

```

```

private void OpenNextForm(string role, string pib)
{
    switch (role)
    {
        case "Cashier":
            Form1 form1 = new Form1(pib);
            form1.Show();
            this.Hide();
            break;
        case "Admin":
            Form3 form3 = new Form3(pib);
            form3.Show();
            this.Hide();
            break;
        case "Warehouse":
            Form4 form4 = new Form4(pib);
            form4.Show();
            this.Hide();
            break;
        default:
            MessageBox.Show("Неприпустима роль користувача");
    }
}

```

```

        break;
    }
}

private void Form2_FormClosing(object sender, FormClosingEventArgs e)
{
    Environment.Exit(1);
}
}
}

```

Form3.cs:

```

using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;

namespace OptShop
{
    public partial class Form3 : Form
    {
        private string userPIB;
        private string connectionString = @"Data Source=DESKTOP-199NT0R;Initial
Catalog=OptShop;Integrated Security=True;";

        public Form3(string pib)
        {
            InitializeComponent();
            userPIB = pib;
            ANameTable.Text = userPIB;

            LoadData();
        }

        private void LoadData()
        {
            string query = "SELECT * FROM Products";

            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                SqlCommand command = new SqlCommand(query, connection);
                SqlDataAdapter adapter = new SqlDataAdapter(command);
                DataTable dataTable = new DataTable();
            }
        }
    }
}

```

```

        adapter.Fill(dataTable);
        dataGridProducts.DataSource = dataTable;
    }

    query = "SELECT * FROM Users";

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        SqlCommand command = new SqlCommand(query, connection);
        SqlDataAdapter adapter = new SqlDataAdapter(command);
        DataTable dataTable = new DataTable();

        adapter.Fill(dataTable);
        dataGridUsers.DataSource = dataTable;
    }

    query = "SELECT * FROM Reports";

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        SqlCommand command = new SqlCommand(query, connection);
        SqlDataAdapter adapter = new SqlDataAdapter(command);
        DataTable dataTable = new DataTable();

        adapter.Fill(dataTable);
        dataGridReports.DataSource = dataTable;
    }
}

private void button3_Click(object sender, EventArgs e)
{
    Form2 form2 = new Form2();
    form2.Show();
    this.Hide();
}

private void button1_Click(object sender, EventArgs e)
{
    tabWarehouse.SelectedIndex = 0;
}

private void button2_Click(object sender, EventArgs e)
{
    tabWarehouse.SelectedIndex = 1;
}

```

```
private void Form3_FormClosing(object sender, FormClosingEventArgs e)
{
    Environment.Exit(1);
}
```

```
private void SaveButton1_Click(object sender, EventArgs e)
{
    DataTable modifiedDataTable = (DataTable)dataGridProducts.DataSource;

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();

        SqlTransaction transaction = connection.BeginTransaction();

        try
        {
            UpdateProducts(connection, transaction, modifiedDataTable);

            transaction.Commit();
        }
        catch (Exception ex)
        {
            transaction.Rollback();
            MessageBox.Show("Виникла помилка: " + ex.Message);
        }
    }
    RefreshUserDataGrid();
}
```

```
private void UpdateProducts(SqlConnection connection, SqlTransaction
transaction, DataTable dataTable)
{
    foreach (DataRow row in dataTable.Rows)
    {
        if (row.RowState == DataRowState.Added)
        {
            InsertProduct(connection, transaction, row);
        }
        else if (row.RowState == DataRowState.Modified)
        {
            UpdateProduct(connection, transaction, row);
        }
    }
}
```

```

        else if (row.RowState == DataRowState.Deleted)
        {
            DeleteProduct(connection, transaction, row);
        }
    }
}

```

```

private void InsertProduct(SqlConnection connection, SqlTransaction transaction,
DataRow row)
{
    string query = "INSERT INTO Products (ProductCode, ProductName,
ProductShowName, OriginCountry, CreatorCompany, Price, Quantity) " +
        "VALUES (@ProductCode, @ProductName, @ProductShowName,
@OriginCountry, @CreatorCompany, @Price, @Quantity)";

    using (SqlCommand command = new SqlCommand(query, connection,
transaction))
    {
        command.Parameters.AddWithValue("@ProductCode",
row["ProductCode"]);
        command.Parameters.AddWithValue("@ProductName",
row["ProductName"]);
        command.Parameters.AddWithValue("@ProductShowName",
row["ProductShowName"]);
        command.Parameters.AddWithValue("@OriginCountry",
row["OriginCountry"]);
        command.Parameters.AddWithValue("@CreatorCompany",
row["CreatorCompany"]);
        command.Parameters.AddWithValue("@Price", row["Price"]);
        command.Parameters.AddWithValue("@Quantity", row["Quantity"]);

        command.ExecuteNonQuery();
    }
}

```

```

private void UpdateProduct(SqlConnection connection, SqlTransaction transaction,
DataRow row)
{
    string query = "UPDATE Products SET ProductName = @ProductName,
ProductShowName = @ProductShowName, " +
        "OriginCountry = @OriginCountry, CreatorCompany =
@CreatorCompany, Price = @Price, " +
        "Quantity = @Quantity WHERE ProductCode = @ProductCode";
}

```

```

        using (SqlCommand command = new SqlCommand(query, connection,
transaction))
        {
            command.Parameters.AddWithValue("@ProductName",
row["ProductName"]);
            command.Parameters.AddWithValue("@ProductShowName",
row["ProductShowName"]);
            command.Parameters.AddWithValue("@OriginCountry",
row["OriginCountry"]);
            command.Parameters.AddWithValue("@CreatorCompany",
row["CreatorCompany"]);
            command.Parameters.AddWithValue("@Price", row["Price"]);
            command.Parameters.AddWithValue("@Quantity", row["Quantity"]);
            command.Parameters.AddWithValue("@ProductCode",
row["ProductCode"]);

            command.ExecuteNonQuery();
        }
    }

```

```

private void DeleteProduct(SqlConnection connection, SqlTransaction transaction,
DataRow row)
{
    string query = "DELETE FROM Products WHERE ProductCode =
@ProductCode";

```

```

        using (SqlCommand command = new SqlCommand(query, connection,
transaction))
        {
            command.Parameters.AddWithValue("@ProductCode", row["ProductCode",
DataRowVersion.Original]);

            command.ExecuteNonQuery();
        }
    }

```

```

private void SaveButton2_Click(object sender, EventArgs e)
{
    DataTable changedDataTable =
((DataTable)dataGridUsers.DataSource).GetChanges();

    if (changedDataTable != null)
    {
        foreach (DataRow row in changedDataTable.Rows)
        {

```

```

        if (!IsValidRole(row["Role"].ToString()))
        {
            MessageBox.Show("Неприпустима роль. Роль повинна бути 'Cashier',
'Admin' або 'Warehouse'.");
            return;
        }
    }

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();

        SqlTransaction transaction = connection.BeginTransaction();

        try
        {
            UpdateUsers(connection, transaction, changedDataTable);

            transaction.Commit();
            ((DataTable)dataGridUsers.DataSource).AcceptChanges();

            RefreshUserDataGrid();
        }
        catch (Exception ex)
        {
            transaction.Rollback();
            MessageBox.Show("Виникла помилка: " + ex.Message);
        }
    }
}

private bool IsValidRole(string role)
{
    return role == "Cashier" || role == "Admin" || role == "Warehouse";
}

private void RefreshUserDataGrid()
{
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        SqlDataAdapter adapter = new SqlDataAdapter("SELECT * FROM Users",
connection);
        DataTable dataTable = new DataTable();
        adapter.Fill(dataTable);
    }
}

```

```

        dataGridUsers.DataSource = dataTable;
    }
}

private void UpdateUsers(SqlConnection connection, SqlTransaction transaction,
DataTable dataTable)
{
    foreach (DataRow row in dataTable.Rows)
    {
        if (row.RowState == DataRowState.Added)
        {
            InsertUser(connection, transaction, row);
        }
        else if (row.RowState == DataRowState.Modified)
        {
            UpdateUser(connection, transaction, row);
        }
        else if (row.RowState == DataRowState.Deleted)
        {
            DeleteUser(connection, transaction, row);
        }
    }
}

private void InsertUser(SqlConnection connection, SqlTransaction transaction,
DataRow row)
{
    string query = "INSERT INTO Users (Login, Password, PIB, Role) " +
        "VALUES (@Login, @Password, @PIB, @Role)";

    using (SqlCommand command = new SqlCommand(query, connection,
transaction))
    {
        command.Parameters.AddWithValue("@Login", row["Login"]);
        command.Parameters.AddWithValue("@Password", row["Password"]);
        command.Parameters.AddWithValue("@PIB", row["PIB"]);
        command.Parameters.AddWithValue("@Role", row["Role"]);

        command.ExecuteNonQuery();
    }
}

private void UpdateUser(SqlConnection connection, SqlTransaction transaction,
DataRow row)
{

```

```

string query = "UPDATE Users SET Login = @Login, Password = @Password,
" +
                "PIB = @PIB, Role = @Role WHERE Login = @OriginalLogin";

using (SqlCommand command = new SqlCommand(query, connection,
transaction))
{
    command.Parameters.AddWithValue("@Login", row["Login"]);
    command.Parameters.AddWithValue("@Password", row["Password"]);
    command.Parameters.AddWithValue("@PIB", row["PIB"]);
    command.Parameters.AddWithValue("@Role", row["Role"]);
    command.Parameters.AddWithValue("@OriginalLogin", row["Login",
DataRowVersion.Original]);

    command.ExecuteNonQuery();
}

private void DeleteUser(SqlConnection connection, SqlTransaction transaction,
DataRow row)
{
    string query = "DELETE FROM Users WHERE Login = @Login";

    using (SqlCommand command = new SqlCommand(query, connection,
transaction))
    {
        command.Parameters.AddWithValue("@Login", row["Login",
DataRowVersion.Original]);

        command.ExecuteNonQuery();
    }
}

private void button4_Click(object sender, EventArgs e)
{
    tabWarehouse.SelectedIndex = 2;
}

private void buttonSaveR_Click(object sender, EventArgs e)
{
    DataTable modifiedDataTable = (DataTable)dataGridReports.DataSource;

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();
    }
}

```

```

        SqlTransaction transaction = connection.BeginTransaction();

        try
        {
            UpdateReports(connection, transaction, modifiedDataTable);
            transaction.Commit();
        }
        catch (Exception ex)
        {
            transaction.Rollback();
            MessageBox.Show("Виникла помилка: " + ex.Message);
        }
    }

    private void UpdateReports(SqlConnection connection, SqlTransaction transaction,
        DataTable dataTable)
    {
        foreach (DataRow row in dataTable.Rows)
        {
            if (row.RowState == DataRowState.Added)
            {
                InsertReport(connection, transaction, row);
            }
            else if (row.RowState == DataRowState.Modified)
            {
                UpdateReport(connection, transaction, row);
            }
            else if (row.RowState == DataRowState.Deleted)
            {
                DeleteReport(connection, transaction, row);
            }
        }
    }

    private void InsertReport(SqlConnection connection, SqlTransaction transaction,
        DataRow row)
    {
        string query = "INSERT INTO Reports (Date, SumMon) VALUES (@Date, @SumMon)";

        using (SqlCommand command = new SqlCommand(query, connection,
            transaction))
        {
            command.Parameters.AddWithValue("@Date", row["Date"]);

```

```

        command.Parameters.AddWithValue("@SumMon", row["SumMon"]);

        command.ExecuteNonQuery();
    }
}

private void UpdateReport(SqlConnection connection, SqlTransaction transaction,
DataRow row)
{
    string query = "UPDATE Reports SET SumMon = @SumMon WHERE Date =
@Date";

    using (SqlCommand command = new SqlCommand(query, connection,
transaction))
    {
        command.Parameters.AddWithValue("@SumMon", row["SumMon"]);
        command.Parameters.AddWithValue("@Date", row["Date"]);

        command.ExecuteNonQuery();
    }
}

private void DeleteReport(SqlConnection connection, SqlTransaction transaction,
DataRow row)
{
    string query = "DELETE FROM Reports WHERE Date = @Date";

    using (SqlCommand command = new SqlCommand(query, connection,
transaction))
    {
        command.Parameters.AddWithValue("@Date", row["Date",
DataRowVersion.Original]);

        command.ExecuteNonQuery();
    }
}
}

```

Form4.cs:

```

using System;
using System.Data;
using System.Data.SqlClient;
using System.Linq;

```

```

using System.Windows.Forms;
using AForge.Video;
using AForge.Video.DirectShow;
using ZXing;
using System.Drawing;

namespace OptShop
{
    public partial class Form4 : Form
    {
        private FilterInfoCollection filterInfoCollection;
        private VideoCaptureDevice videoCaptureDevice;
        private BarcodeReader barcodeReader;
        private DateTime lastScanTime;

        private string connectionString = @"Data Source=DESKTOP-199NT0R;Initial
Catalog=OptShop;Integrated Security=True;";

        public Form4(string pib)
        {
            InitializeComponent();
            WNameTable.Text = pib;
            QRData.Text = "";
            QRDataFull.Text = "";
        }

        private void VideoCaptureDevice_NewFrame(object sender, NewFrameEventArgs
eventArgs)
        {
            pictureBoxCamera.Image = (Bitmap)eventArgs.Frame.Clone();

            try
            {
                Result result = barcodeReader.Decode((Bitmap)eventArgs.Frame.Clone());
                if (result != null && CanScan())
                {
                    ProcessQRCode(result.Text);
                    lastScanTime = DateTime.Now;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Помилка сканування QR-коду: " + ex.Message);
            }
        }
    }
}

```

```

private bool CanScan()
{
    return (DateTime.Now - lastScanTime).TotalSeconds >= 5;
}

private void ProcessQRCode(string qrData)
{
    string[] qrValues = qrData.Split(',');

    if (qrValues.Length != 7)
    {
        MessageBox.Show("Некоректна кількість значень QR-коду.");
        return;
    }

    if (IsValidProduct(qrValues))
    {
        QRData.Invoke((MethodInvoker)(() => QRData.Text = qrValues[2] + " - " +
qrValues[6] + " шт.));
        QRDataFull.Invoke((MethodInvoker)(() => QRDataFull.Text = qrValues[0] +
", " + qrValues[1] + ", " + qrValues[2] + ", " + qrValues[3] + ", " + qrValues[4] + ", " +
qrValues[5] + ", " + qrValues[6]));
    }
    else
    {
        MessageBox.Show("QR-код не відповідає структурі таблиці Products.");
    }
}

private bool IsValidProduct(string[] qrValues)
{
    return int.TryParse(qrValues[0], out int productCode)
        && int.TryParse(qrValues[5], out int price)
        && int.TryParse(qrValues[6], out int quantity);
}

private void Form4_Load(object sender, EventArgs e)
{
    filterInfoCollection = new
FilterInfoCollection(FilterCategory.VideoInputDevice);
    foreach (FilterInfo filterInfo in filterInfoCollection)
    {
        comboBoxDevices.Items.Add(filterInfo.Name);
        comboBoxDevices2.Items.Add(filterInfo.Name);
    }
}

```

```

    }
    if (comboBoxDevices.Items.Count > 0 && comboBoxDevices2.Items.Count >
0)
    {
        comboBoxDevices.SelectedIndex = 0;
        comboBoxDevices2.SelectedIndex = 0;
    }
    else
    {
        MessageBox.Show("Не виявлено доступних відеопристроїв.");
        buttonStart.Enabled = false;
    }

    barcodeReader = new BarcodeReader();
    lastScanTime = DateTime.MinValue;
}

private void Form4_FormClosing(object sender, FormClosingEventArgs e)
{
    Environment.Exit(1);
}

private void buttonStart_Click(object sender, EventArgs e)
{
    if (comboBoxDevices.SelectedIndex >= 0)
    {
        videoCaptureDevice = new
VideoCaptureDevice(filterInfoCollection[comboBoxDevices.SelectedIndex].MonikerSt
ring);
        videoCaptureDevice.NewFrame += VideoCaptureDevice_NewFrame;
        videoCaptureDevice.Start();
    }
    else
    {
        MessageBox.Show("Виберіть відеопристрій, щоб розпочати захоплення
відео.");
    }
}

private void button3_Click(object sender, EventArgs e)
{
    Form2 form2 = new Form2();
    form2.Show();
    this.Hide();
}

```

```

private void AddButton_Click(object sender, EventArgs e)
{
    if (string.IsNullOrEmpty(QRData.Text))
    {
        MessageBox.Show("QR-код не відскановано або дані не відповідають  
потрібному формату.");
        return;
    }

    string[] qrValues = QRDataFull.Text.Split(',');

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        connection.Open();

        string query = "SELECT Quantity FROM Products WHERE  
ProductShowName = @ProductShowName";
        SqlCommand command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@ProductShowName",  
qrValues[2].Substring(1));

        object result = command.ExecuteScalar();
        if (result != null)
        {
            int count = (int)command.ExecuteScalar();
        }

        if (result != null && int.TryParse(result.ToString(), out int  
existingQuantity))
        {
            int newQuantity = existingQuantity + int.Parse(qrValues[6]);
            query = "UPDATE Products SET Quantity = @Quantity WHERE  
ProductShowName = @ProductShowName";
            command = new SqlCommand(query, connection);
            command.Parameters.AddWithValue("@Quantity", newQuantity);
            command.Parameters.AddWithValue("@ProductShowName",  
qrValues[2].Substring(1));
            command.ExecuteNonQuery();
            MessageBox.Show("Кількість продукту успішно оновлено.");
        }
        else
        {

```

```

        query = "INSERT INTO Products (ProductCode, ProductName,
ProductShowName, OriginCountry, CreatorCompany, Price, Quantity) " +
                "VALUES (@ProductCode, @ProductName, @ProductShowName,
@OriginCountry, @CreatorCompany, @Price, @Quantity)";
        command = new SqlCommand(query, connection);
        command.Parameters.AddWithValue("@ProductCode",
int.Parse(qrValues[0]));
        command.Parameters.AddWithValue("@ProductName",
qrValues[1].Substring(1));
        command.Parameters.AddWithValue("@ProductShowName",
qrValues[2].Substring(1));
        command.Parameters.AddWithValue("@OriginCountry",
qrValues[3].Substring(1));
        command.Parameters.AddWithValue("@CreatorCompany",
qrValues[4].Substring(1));
        command.Parameters.AddWithValue("@Price", int.Parse(qrValues[5]));
        command.Parameters.AddWithValue("@Quantity",
int.Parse(qrValues[6]));
        command.ExecuteNonQuery();
        MessageBox.Show("Продукт додано.");
    }
}

QRData.Text = string.Empty;
}

private void ClearButton_Click(object sender, EventArgs e)
{
    QRDataFull.Text = string.Empty;
    QRData.Text = string.Empty;
}

private void buttonTab1_Click(object sender, EventArgs e)
{
    WareControl.SelectedIndex = 0;
    if (videoCaptureDevice != null && videoCaptureDevice.IsRunning)
    {
        videoCaptureDevice.Stop();
    }
}

private void buttonTab2_Click(object sender, EventArgs e)
{
    WareControl.SelectedIndex = 1;
    if (videoCaptureDevice != null && videoCaptureDevice.IsRunning)

```

```

    {
        videoCaptureDevice.Stop();
    }
}

private void buttonTab3_Click(object sender, EventArgs e)
{
    WareControl.SelectedIndex = 2;
    string query = "SELECT * FROM Products";

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        SqlCommand command = new SqlCommand(query, connection);
        SqlDataAdapter adapter = new SqlDataAdapter(command);
        DataTable dataTable = new DataTable();

        adapter.Fill(dataTable);
        dataGridWare.DataSource = dataTable;
    }
}

private void StartInvButton_Click(object sender, EventArgs e)
{
    if (comboBoxDevices2.SelectedIndex >= 0)
    {
        videoCaptureDevice = new
VideoCaptureDevice(filterInfoCollection[comboBoxDevices2.SelectedIndex].Moniker
String);
        videoCaptureDevice.NewFrame += VideoCaptureDevice_NewFrameINV;
        videoCaptureDevice.Start();
        UpdateInventoryTextBox();
    }
    else
    {
        MessageBox.Show("Виберіть відеопристрій, щоб розпочати захоплення
відео.");
    }
}

private void EndInvButton_Click(object sender, EventArgs e)
{
    if (videoCaptureDevice != null && videoCaptureDevice.IsRunning)
    {
        videoCaptureDevice.SignalToStop();
        videoCaptureDevice = null;
    }
}

```

```

    }

    string inventoryResult = CalculateInventoryResult();

    MessageBox.Show(inventoryResult);
}

private void VideoCaptureDevice_NewFrameINV(object sender,
NewFrameEventArgs eventArgs)
{
    pictureBoxCamera2.Image = (Bitmap)eventArgs.Frame.Clone();

    try
    {
        Result result = barcodeReader.Decode((Bitmap)eventArgs.Frame.Clone());
        if (result != null && CanScan())
        {
            ProcessQRCodeINV(result.Text);
            lastScanTime = DateTime.Now;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("Помилка сканування QR-коду: " + ex.Message);
    }
}

private void UpdateInventoryTextBox()
{
    string query = "SELECT ProductShowName, Quantity FROM Products";

    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        SqlCommand command = new SqlCommand(query, connection);
        SqlDataAdapter adapter = new SqlDataAdapter(command);
        DataTable dataTable = new DataTable();

        adapter.Fill(dataTable);

        richInvBox.Clear();

        foreach (DataRow row in dataTable.Rows)
        {
            richInvBox.AppendText(row["ProductShowName"].ToString() + " : " +
row["Quantity"].ToString() + Environment.NewLine);
        }
    }
}

```

```

    }
}

private void ProcessQRCodeINV(string qrData)
{
    string[] qrValues = qrData.Split(',');

    if (qrValues.Length != 7)
    {
        MessageBox.Show("Некоректна кількість значень QR-коду.");
        return;
    }

    string scannedProductName = qrValues[2];

    richInvBox.Invoke((MethodInvoker)delegate
    {
        string[] lines = richInvBox.Lines;
        bool productFound = false;
        for (int i = 0; i < lines.Length; i++)
        {
            string[] lineParts = lines[i].Split(':');
            if (lineParts.Length == 2)
            {
                string productName = lineParts[0].Trim();
                int quantity = int.Parse(lineParts[1].Trim());
                if (scannedProductName == productName)
                {
                    productFound = true;

                    quantity -= int.Parse(qrValues[6]);

                    lines[i] = $"{productName} : {quantity}";

                    richInvBox.Lines = lines;

                    MessageBox.Show($"Продукт {scannedProductName} успішно
відскановано.");

                    break;
                }
            }
        }

        if (!productFound)

```

```

        {
            MessageBox.Show($"Продукт {scannedProductName} не знайдено.");
        }
    });
}

private string CalculateInventoryResult()
{
    string result = "Підсумки інвентаризації:" + Environment.NewLine;

    string[] lines = richInvBox.Lines;
    foreach (string line in lines)
    {
        string[] lineParts = line.Split(':');
        if (lineParts.Length == 2)
        {
            string productName = lineParts[0].Trim();
            int actualQuantity = int.Parse(lineParts[1].Trim());

            int expectedQuantity = 0;

            if (actualQuantity < expectedQuantity)
            {
                result += $"{productName} - Забарато товари: {expectedQuantity -
actualQuantity}" + Environment.NewLine;
            }
            else if (actualQuantity > expectedQuantity)
            {
                result += $"{productName} - Недостатньо товари: {actualQuantity -
expectedQuantity}" + Environment.NewLine;
            }
        }
    }

    return result;
}
}

```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»

Кафедра програмних засобів

Дипломна кваліфікаційна робота бакалавра

Розробка програмного забезпечення для оптимізації логістичних процесів у магазині роздрібної торгівлі

Виконав:

ст. групи КНТ-130

Костянтин ВЛАСЕНКО

Керівники:

професор, к.т.н

Валерій ДУБРОВІН

Рисунок Б.1 – Слайд № 1

Мета, об'єкт, предмет і задачі дослідження

- Мета роботи – розробка автоматизованої системи управління товарними запасами та складськими операціями для магазинів роздрібної торгівлі;
- Об'єкт дослідження – процес оптимізації логістичних процесів магазину роздрібної торгівлі;
- Предмет дослідження – управління товарними запасами та складськими операціями в роздрібних магазинах і складах;
- У даній роботі було поставлено завдання створити програмне забезпечення яке оптимізує логістичні процеси магазину роздрібної торгівлі завдяки використанню QR-кодів.

Рисунок Б.2 – Слайд № 2

Аналіз предметної області

Основні аспекти логістичних процесів;

Проблеми керування товарними запасами;

Ролі користувачів у системі;

Сучасні підходи.

Рисунок Б.3 – Слайд № 3

Схема структури розробленої програми

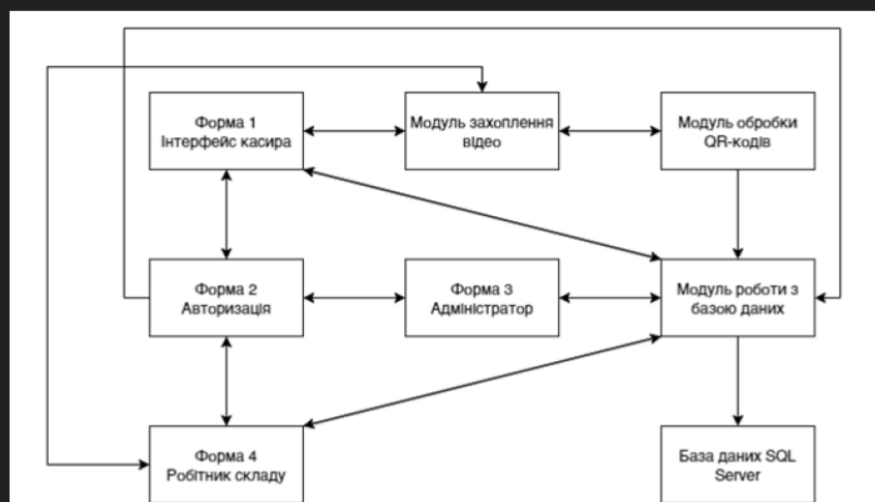


Рисунок Б.4 – Слайд № 4

Схема функціонування розробленої програми

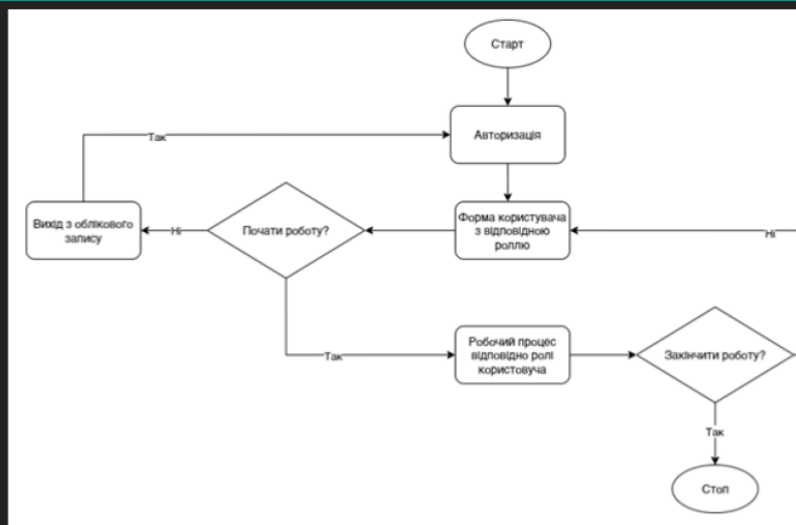


Рисунок Б.5 – Слайд № 5

Схема структури бази даних

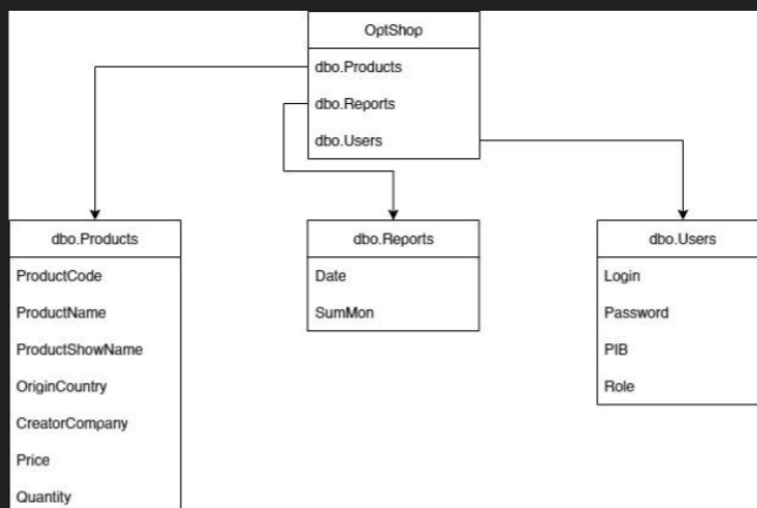


Рисунок Б.6 – Слайд № 6

Вибір мови програмування

Критерій	C#	Java
Інтеграція з Windows	Відмінна інтеграція з Windows та .NET	Висока інтеграція з Windows
Інструменти розробки	Visual Studio з багатим набором інструментів	IntelliJ IDEA, Eclipse, NetBeans
Робота з відеопотоком	AForge.Video, простота інтеграції	OpenCV, складніша інтеграція
Розпізнавання QR-кодів	ZXing, хороша підтримка бібліотек	ZXing, хороша підтримка бібліотек
Бази даних	Entity Framework, LINQ	Hibernate, JDBC
Продуктивність	Висока продуктивність на Windows	Висока продуктивність на будь-якій платформі

Рисунок Б.7 – Слайд № 7

Інтерфейс роботи робітника складу

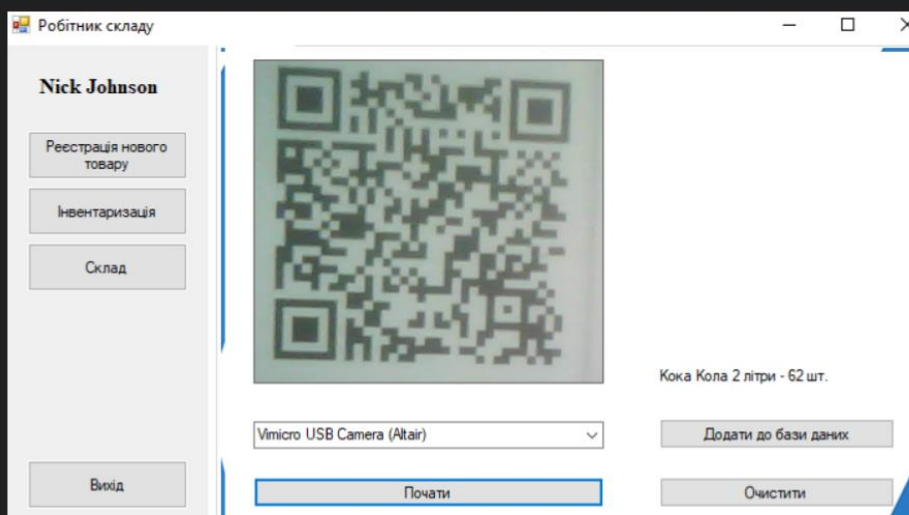


Рисунок Б.8 – Слайд № 8

Інтерфейс роботи касира

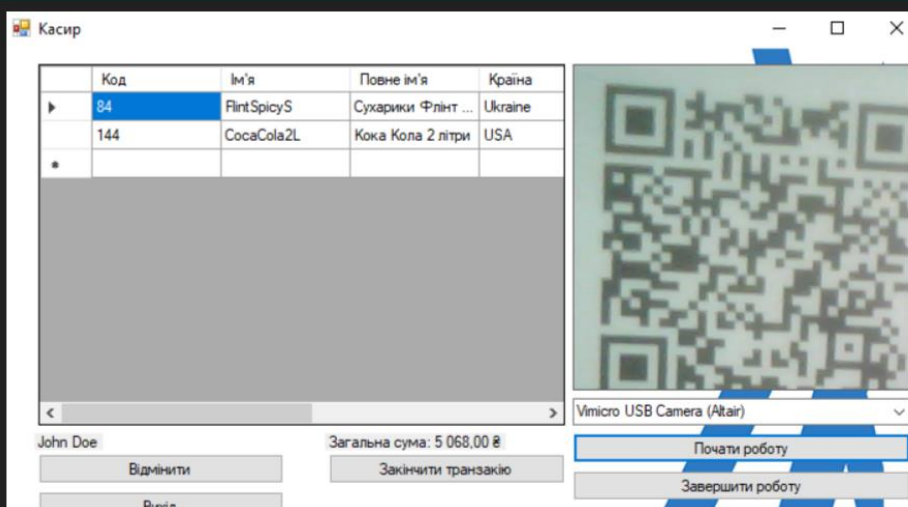


Рисунок Б.9 – Слайд № 9

Висновки

Розроблене програмне забезпечення виконує поставлені задачі. А саме, такі задачі як автоматизація інвентаризації та підвищення точності обліку товарів. Новизна роботи полягає у застосуванні QR-кодів для оптимізації складських операцій. Практична цінність полягає у зниженні витрат на управління товарними запасами та покращенні обслуговування клієнтів.

Рисунок Б.10 – Слайд № 10