Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))

Кафедра інформаційних систем та комп'ютерного модулювання
(повна назва кафедри (предметної циклової комісії))

Пояснювальна записка

до дипломної роботи

перший (бакалаврський)
(рівень вищої освіти)

на тему: «Розроблення системи управління комерційною діяльністю компанії засобами .NET»

Виконав студент 4 курсу, групи ІСТ-41
спеціальності:
126 – "Інформаційні системи та технології"
(шифр і назва напрямку підготовки спеціальності)

Букартик Віталій Ігорович
(прізвище, ініціали)

Керівник: Прусак Ю.В.
(прізвище, ініціали)

Рецензент: Борецька І.Б.
(прізвище, ініціали)

Львів-2025

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій
Кафедра інформаційних систем та комп'ютерного модулювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 – *"Інформаційні системи та технології"*

ЗАТВЕРДЖУЮ:
Завідувач кафедри ІСКМ
_____Сторожук О.Л.
"15" ____11_____2024.

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Букартику Віталію Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема бакалаврської роботи: Розроблення системи управління комерційною діяльністю компанії засобами .NET.

керівник роботи Прусак Юрій Володимирович, канд. техн. наук, доцент ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 15.11.2024 р. № С–884,

2. Термін подання студентом проекту (роботи) 12.06.2025 р.

3. Вихідні дані до проєкту (роботи) Розробити програмну систему для управління продажами. Реалізувати зручний та простий інтерфейс для використання розроблених функцій проєкту. Для розробки використати засоби та можливості крос-платформової технології .NET.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Вступ
1. Стан проблемної області
2. Інформаційне та математичне забезпечення
3. Програмне та технічне забезпечення
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Додаток А, Додаток Б, Додаток В, Додаток Г, Додаток Д.

6. Дата видачі завдання 18 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№, з/п	Етапи бакалаврської роботи	Термін виконання етапів роботи	Примітка
1.	Огляд літератури згідно досліджуваної теми. Збір необхідних матеріалів.	20.11.2024-20.01.2025	Виконано
2.	Постановка задачі і її формалізація	21.01.2025-11.02.2025	Виконано
3.	Виконання вхідного етапу технології	12.02.2025-15.03.2025	Виконано
4.	Реалізація головних класів проєкту	16.03.2025-05.05.2025	Виконано
5.	Виконання етапу відлагодження проєкту	06.05.2025-14.05.2025	Виконано
6.	Виконання етапу впровадження та випуску бета-версії.	15.05.2025-27.05.2025	Виконано
7.	Оформлення записки до дипломного проекту.	28.05.2025-08.06.2025	Виконано

Студент _____ Букартик В.І.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Прусак Ю.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 65 сторінок основного тексту, 46 рисунків, 2 таблиці, 5 додатків, 21 джерело.

Дипломна робота присвячена розробці системи для автоматизації обліку товарів, управлінням запасами та продажами. Для реалізації використати засоби та технології .NET. Серед функціоналу передбачити сторінку логінації, меню з пунктами, можливість зміни паролю користувача та сторінку з описом програми. Система вміє менеджити запаси товарів(додавати товари з детальними описами, кількістю і т.д.), сканувати штрихкоди товарів при покупці товарів, створювати унікальні штрихкоди товарів та друкувати їх, робити звіти проданих товарів та обрухонок отриманої виручки, менеджити категорії товарів. Актуальність вибраного проєкту обгрунтовано бажанням підприємців працювати з персоналізованим програмним забезпеченням.

Ключові слова: програмне забезпечення, .NET, база даних, .NET, XAML, MVVM, автоматизація.

ABSTRACT

The thesis contains 65 pages of main text, 46 figures, 2 tables, 5 appendices, 21 sources.

The thesis is devoted to the development of a system for automating the accounting of goods, inventory and sales management. Use .NET tools and technologies for implementation. The functionality includes a login page, a menu with items, the ability to change the user's password, and a page with a description of the program. The system is able to manage product stocks (add products with detailed descriptions, quantity, etc.), scan product barcodes when purchasing products, create unique product barcodes and print them, make reports on sold products and revenue receipts, manage product categories. The relevance of the selected project is justified by the desire of entrepreneurs to work with personalized software.

Keywords: software, .NET, database, .NET, XAML, MVVM, automation.

ТЕХНІЧНЕ ЗАВДАННЯ

1. Розробити систему для автоматизації обліку товарів, управління запасами та продажами.
2. Використати засоби та технології .NET для реалізації.
3. Реалізувати функціонал:
    a. Сторінка логінації.
    b. Меню з пунктами.
    c. Можливість зміни паролю користувача.
    d. Сторінка з описом програми.
4. Функціонал системи:
    a. Менеджмент запасів товарів (додавання, описи, кількість тощо).
    b. Сканування штрихкодів товарів при покупці.
    c. Створення унікальних штрихкодів товарів та їх друк.
    d. Створення звітів про продані товари та обороти виручки.
    e. Менеджмент категорій товарів.
    f. Створення бекапів файлів системи для відновлення даних.

Розробити систему управління доступами для користувачів (додавання, редагування, зміна прав) та можливість розлогування.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

WPF (Windows Presentation Foundation) – Технологія для створення графічних десктоп-додатків на Windows. Підтримує прив’язку даних, стилі, анімацію, шаблони і ґрунтується на XAML.

DB (Database) – База даних – організована структура для зберігання, управління та отримання інформації. Наприклад: SQL Server, MySQL, PostgreSQL.

.NET – Платформа від Microsoft для створення різних типів застосунків: десктоп, веб, мобільні, хмарні. Підтримує багато мов, зокрема C#.

XAML *(eXtensible Application Markup Language)* – Мова розмітки, що використовується для опису UI в WPF, UWP та MAUI. Візуально схожа на HTML або XML.

C# (C-Sharp) – Основна мова програмування для .NET, об’єктно-орієнтована, потужна та зручна для розробки будь-яких типів застосунків.

MVVM (Model-View-ViewModel) – Архітектурний патерн, часто використовується в WPF. Дозволяє розділити логіку (Model), інтерфейс (View) та прив’язку даних (ViewModel).

ВСТУП

У сучасному світі, де технології стають все більш важливою частиною повсякденного життя, автоматизація процесів управління товарами та обліку набуває значного значення. У зв’язку з цим розробка системи автоматизації відстеження товарів та контролю товарних запасів стала **актуальною задачею**, спрямованою на оптимізацію господарської діяльності та підвищення ефективності управління підприємством.

Крім того, в умовах постійної конкуренції на ринку здатність швидко реагувати на зміни попиту та передбачати потреби в запасах стає критично важливою. Система, що розробляється, пропонує інструменти для аналізу даних про продажі, що дозволяє приймати обґрунтовані рішення щодо управління запасами та планування продажів.

Значну роль у розробці таких систем відіграє вибір технологій, які забезпечують гнучкість, масштабованість та зручність у використанні. Технології платформи .NET, зокрема WPF (Windows Presentation Foundation), надають широкі можливості для створення сучасних настільних додатків з привабливим і функціональним інтерфейсом користувача. Використання мов програмування C# та розмітки XAML дозволяє ефективно реалізовувати як візуальну частину програми, так і складну бізнес-логіку.

Успішне управління товарними запасами вимагає інтеграції різних функцій, таких як сканування штрихкодів, ведення бази даних товарів, формування звітності та обліку продажів, що можливо реалізувати завдяки поєднанню сучасного програмного забезпечення та продуманої архітектури. Застосування шаблону проєктування MVVM (Model-View-ViewModel) у WPF дозволяє чітко розділити відповідальність між компонентами системи, що робить програму легшою для тестування, підтримки та розширення в майбутньому.

Таким чином, розробка програмного забезпечення для автоматизації обліку товарів є не лише актуальною, але й технічно здійсненною задачею за допомогою сучасних інструментів і підходів. Запропоноване рішення дозволяє підвищити

ефективність операційної діяльності підприємства, зменшити ризики, пов’язані з людським фактором, і надати керівництву підприємства актуальну інформацію для прийняття стратегічних рішень.

Об’єктом дослідження є процеси управління та обліку товарів на підприємствах, які потребують автоматизації та оптимізації.

Метою роботи є розробка системи, яка пропонує зручні та ефективні інструменти для управління запасами та обліку товарів.

Предметом дослідження є програмна система, призначена для вирішення завдання з використанням технологій .NET.

Практична цінність розробленої системи полягає в її здатності автоматизувати процеси інвентаризації та обліку товарів на підприємствах. За допомогою цієї системи користувачі зможуть ефективно керувати запасами, створювати звіти про продажі та доходи, а також виконувати інші важливі функції, які оптимізують і покращують діяльність підприємства. Система також включає можливість створювати резервні копії даних, забезпечуючи безпеку та дозволяючи відновлення системи, коли це необхідно.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Огляд проблемної області

Розробка системи автоматизації обліку товарів та управління запасами передбачає вирішення кількох критичних аспектів для забезпечення успішної реалізації проекту:

1. Безпека даних

Підтримання конфіденційності, цілісності та доступності даних є життєво важливим компонентом будь-якої системи управління запасами. Важливо запровадити надійні механізми автентифікації та авторизації користувачів і захистити від несанкціонованого доступу та потенційних кіберзагроз [16].

Безпека даних особливо важлива, оскільки системи інвентаризації часто зберігають конфіденційну інформацію про продукти, клієнтів, фінансові операції тощо. Забезпечення цієї безпеки передбачає низку заходів, призначених для захисту конфіденційних даних від злому та пошкодження.

Основним кроком у цьому процесі є впровадження надійних методів автентифікації користувачів. Вони можуть включати захист паролем, біометричну перевірку та двофакторну автентифікацію – кожна з них пропонує розширену перевірку особи користувача [6].

Після автентифікації необхідно встановити елементи керування авторизацією, щоб визначити, до чого користувачі можуть отримати доступ у системі. Контроль доступу на основі ролей або дозволів допомагає запобігти несанкціонованим операціям і викриттю даних.

Щоб захистити систему від зовнішніх загроз, слід застосовувати комбінацію стратегій захисту, включаючи шифрування даних, багаторівневу автентифікацію, захист на рівні мережі, антивірусне програмне забезпечення та системи моніторингу безпеки в режимі реального часу. Ці інструменти дозволяють

виявляти й реагувати на потенційні вразливості або порушення, щойно вони виникають.

Крім того, корисно використовувати журнали аудиту для відстеження дій користувачів і системних подій. Це допомагає виявити порушення, забезпечити підзвітність і полегшити усунення несправностей у разі виникнення проблем.

Загалом захист даних у системі управління запасами вимагає багаторівневого підходу із залученням передових технологій і комплексних політик безпеки для забезпечення надійного захисту.

2. Інтеграція з пристроями

Система повинна підтримувати сканування штрих-кодів для товарів, які додаються до асортименту або продаються, за допомогою підключених сканерів штрих-кодів. Він також має забезпечувати функціональність для створення та друку унікальних штрих-кодів для щойно доданих продуктів.

Інтеграція пристроїв відіграє вирішальну роль в автоматизації процесів інвентаризації, оскільки спрощує ідентифікацію та відстеження продуктів за допомогою сканування та створення штрих-кодів [16].

Щоб ефективно реалізувати цю можливість, необхідно звернути увагу на кілька ключових міркувань:

- Підтримка різноманітних сканерів штрих-кодів: система має бути сумісною з різними типами сканерів, такими як лазерні пристрої або пристрої для обробки зображень, щоб користувачі могли вибирати пристрої, які найкраще відповідають їхнім вимогам.

- Апаратне підключення: має бути забезпечений безперебійний зв’язок зі сканерами штрих-кодів через такі інтерфейси, як USB, Bluetooth або інші з’єднання.

- Автоматичне розпізнавання продукту: після сканування штрих-коду система має миттєво отримати відповідні деталі продукту, такі як назва, опис і ціна.

- Створення унікального штрих-коду: можливість генерувати унікальні ідентифікатори для нових продуктів є важливою для відстеження запасів і контролю руху товарів.

- Друк штрих-кодів: після створення штрих-коди можна друкувати на етикетках або бірках, які можна фізично прикріпити до продуктів для ідентифікації та відстеження.

- Відстеження товарів у режимі реального часу: інтеграція пристрою також дозволяє відстежувати товари в режимі реального часу, підтримуючи краще управління запасами та прийняття рішень щодо закупівель.

Таким чином, ефективна інтеграція пристроїв значно підвищує точність і ефективність процесів управління запасами. Це мінімізує помилки, підтримує автоматизацію та сприяє більш оптимізованій та оперативній системі інвентаризації.

3. Управління даними про продукт

Ефективна система управління запасами повинна пропонувати інтуїтивно зрозумілі функції для додавання, редагування та видалення інформації про продукт. Це включає зберігання вичерпних відомостей, таких як назва продукту, опис, категорія, кількість на складі, ціна та інформація про постачальника [6].

Основні аспекти включають:

- Введення продукту: зручний інтерфейс має дозволяти безперебійне додавання нових продуктів, фіксуючи такі важливі атрибути, як назва, категорія, рівень запасів і ціна.

- Можливості редагування: користувачі повинні мати можливість оновлювати деталі продукту, наприклад коригувати кількість запасів, змінювати ціни або переглядати описи.

- Видалення продукту: система повинна дозволяти видалення застарілих або вилучених елементів із бази даних.

- Детальні записи: кожен продукт повинен мати повний профіль, включаючи дані про склад, терміни доставки та інформацію про постачальника, що допомагає планувати запаси та поповнювати запаси.

- Категоризація: продукти можна організовувати за категоріями для спрощеного керування та фільтрації.

- Історія змін: слід вести журнал усіх змін даних продукту, щоб забезпечити відстеження та підзвітність.

Загалом надійне керування даними про продукт забезпечує точну, упорядковану та доступну інформацію, що є основою ефективного контролю запасів [8].

4. Аналітика та звітність

Аналітика та звітність відіграють вирішальну роль у дозволі компаніям приймати рішення на основі даних. Система повинна формувати докладні звіти про продажі, оборотність запасів і доходи [7].

Ключові особливості:

- Формування звітів: система повинна підтримувати настроювані звіти про продажі, товарообіг і виручку за різними часовими рамками (щодня, щотижня, щомісяця тощо).
- Візуалізація даних: діаграми, графіки та таблиці допомагають ефективніше інтерпретувати ключові тенденції та показники.
- Порівняльний аналіз. Компанії повинні мати можливість порівнювати різні періоди часу, продукти або категорії, щоб оцінити ефективність і визначити тенденції.
- Прогнозування попиту: аналізуючи минулі продажі, система може передбачити майбутній попит, сприяючи оптимізації запасів і плануванню закупівель.
- Статистика клієнтів: сегментація клієнтів на основі моделей купівлі може допомогти адаптувати стратегії маркетингу та продажів.
- Показники ефективності: система повинна оцінювати ефективність різних каналів продажу, кампаній і рекламних акцій.

Аналітика та звітність надають цінну інформацію, яка підтримує стратегічне планування та ефективність роботи.

5. Резервне копіювання та відновлення даних

Надійні механізми резервного копіювання та відновлення даних необхідні для захисту від втрати даних, спричиненої системними збоями, кібератаками або людськими помилками [6].

Основні міркування включають:

- Регулярне резервне копіювання: резервне копіювання даних слід створювати через постійні проміжки часу – в ідеалі щодня або частіше – залежно від обсягу транзакцій і критичності.
- Методи резервного копіювання: можна використовувати повне, інкрементне або диференціальне резервне копіювання залежно від потреб підприємства та потужності мережі.
- Безпечне зберігання: резервні копії слід зберігати в безпечних місцях – на зовнішніх дисках, хмарних платформах або виділених серверах – щоб запобігти несанкціонованому доступу або пошкодженню.
- Тестування відновлення: регулярне тестування процесу відновлення гарантує швидке й точне відновлення даних у разі надзвичайних ситуацій.
- Автоматизація. Автоматизація резервного копіювання зменшує ризик людської помилки та забезпечує виконання процесу за розкладом.
- Заходи захисту: дані резервного копіювання повинні бути зашифровані та захищені від зловмисного програмного забезпечення для збереження конфіденційності та цілісності.

Підсумовуючи, резервне копіювання та відновлення даних є життєво важливою гарантією, що забезпечує безперервність і мінімізує збої в разі непередбачених інцидентів.

6. Контроль та аудит доступу

Ефективна система управління запасами повинна включати надійний контроль доступу та можливості аудиту для забезпечення безпеки даних і моніторингу активності користувачів. Ці функції необхідні для збереження контролю над конфіденційною інформацією та виявлення потенційних загроз безпеці [16].

Ключові компоненти включають:

- Автентифікація та авторизація користувачів: система повинна перевіряти ідентифікаційні дані користувачів за допомогою механізмів автентифікації та призначати відповідні права доступу на основі попередньо визначених ролей. Авторизація гарантує, що користувачі можуть отримати доступ лише до тих функцій і даних, які дозволені їх роллю.
- Рівні доступу на основі ролей: різні ролі користувачів повинні мати відповідні рівні доступу. Наприклад, системні адміністратори можуть мати необмежений доступ, тоді як звичайні користувачі обмежені окремими модулями або даними, що стосуються їхніх обов'язків.
- Керування правами доступу: адміністратори повинні мати можливість налаштовувати та керувати дозволами користувачів, надаючи або обмежуючи доступ до певних функцій, модулів або наборів даних. Це включає керування правами редагування, перегляду та видалення.
- Аудит доступу: система повинна реєструвати дії користувача, включаючи спроби входу, зміни дозволів і виконання критичних операцій. Ці журнали аудиту необхідні для відстеження дій, діагностики проблем і забезпечення підзвітності.
- Заходи безпеки: щоб запобігти несанкціонованому доступу, система повинна запроваджувати кілька рівнів безпеки, наприклад передачу зашифрованих даних, політику надійних паролів і багатофакторну автентифікацію.
- Моніторинг і аналіз аудиту: система повинна дозволяти адміністраторам регулярно переглядати та аналізувати журнали аудиту, щоб виявити незвичну поведінку або несанкціоновану діяльність. Це підтримує проактивне виявлення загроз і своєчасне реагування на інциденти.

Підсумовуючи, контроль доступу та аудит є життєво важливими компонентами будь-якої системи управління запасами. Вони допомагають підтримувати операційну безпеку, підзвітність і дотримання внутрішньої політики. Реалізація цих функцій вимагає ретельного проектування, щоб збалансувати зручність використання, безпеку та адміністративний контроль.

1.2 Аналіз ринку програм-конкурентів

Для реалізації технічного рішення, яке відповідає зазначеним вимогам, доступно кілька програмних платформ, кожна з яких пропонує ряд функцій, придатних для управління запасами та торгівлею. Нижче наведено кілька відомих варіантів:

Microsoft Dynamics 365 Commerce

Функціональність: ця платформа пропонує комплексні інструменти для автоматизації комерційних операцій, включаючи управління запасами, продажі, звітність, аналітику та управління взаємовідносинами з клієнтами. Це дозволяє централізовано керувати всіма бізнес-діяльністю [17].

Інтеграція .NET: Dynamics 365 Commerce, створена на основі екосистеми Microsoft, бездоганно інтегрується з технологіями .NET, дозволяючи використовувати спеціальні розширення та індивідуальні конфігурації системи [17].

Безпека: корпорація Майкрософт забезпечує високий рівень безпеки всіх своїх продуктів. Dynamics 365 Commerce містить надійні вбудовані функції безпеки та гнучку систему контролю доступу.

SAP Business One

Функціональність: розроблений для малого та середнього бізнесу, SAP Business One надає повністю інтегрований пакет, що охоплює фінансове управління, продажі, запаси, виробництво тощо. Він розроблений для підтримки ефективних наскрізних бізнес-операцій [18].

Інтеграція .NET: хоча SAP Business One не побудований на основі .NET, він підтримує інтеграцію .NET через API та інструменти розробки, що забезпечує взаємодію з іншими платформами та службами.

Аналітика та звітність: пропонує розширені інструменти аналітики та звітності, надаючи ключові відомості, які сприяють прийняттю стратегічних рішень.

Odoo

Функціональність: Odoo – це ERP-система з відкритим вихідним кодом, яка містить широкий набір бізнес-модулів, включаючи інвентаризацію, продажі, фінанси та управління виробництвом. Його гнучкість робить його придатним для різних галузей промисловості та підприємств різних розмірів [19].

Модульність: видатною особливістю Odoo є його модульна архітектура, яка дозволяє організаціям впроваджувати лише ті компоненти, які їм потрібні, адаптуючи систему до конкретних вимог.

Підтримка спільноти: Odoo має активну спільноту користувачів і розробників, які роблять внесок у постійний розвиток, надають підтримку та спільні ресурси.

QuickBooks Commerce

Функціональність: спеціалізуючись на управлінні запасами та продажами, QuickBooks Commerce пропонує інструменти для відстеження рівня запасів, виконання замовлень, моніторингу продажів і створення звітів [20].

Зручний інтерфейс: QuickBooks Commerce, відомий своїм інтуїтивно зрозумілим інтерфейсом, простий у використанні та навігації навіть для нетехнічних користувачів.

Інтеграція .NET: система підтримує інтеграцію з додатками на основі .NET через API, що дозволяє розробляти спеціальні розширення та підключатися до системи.

Fishbowl

Функціональність: Fishbowl зосереджується на вдосконаленому управлінні запасами та виробництвом. Він підтримує такі процеси, як отримання, переміщення на склад, виконання замовлень і коригування запасів [21].

Контроль запасів: надає докладні інструменти для керування запасами в кількох місцях, забезпечуючи точні рівні запасів і ефективний розподіл ресурсів.

Інтеграція .NET: Fishbowl пропонує доступ до API для інтеграції із зовнішніми системами та підтримує розробку на основі .NET.

Звітність і аналітика: Платформа містить широкий вибір звітів і інструментів аналізу даних, які допомагають у прийнятті операційних і стратегічних рішень.

Кожна з цих платформ надає надійні функції для інвентаризації та комерційних операцій і може бути налаштована відповідно до унікальних потреб вашого бізнесу. Рекомендується детально оцінити кожне рішення разом із пілотним тестуванням або пробним розгортанням, щоб визначити найкраще рішення для вашої організації.

1.3 Огляд фреймворку для розробки користувацького інтерфейсу (WPF)

WPF використовує XAML для розробки інтерфейсу користувача та C# для реалізації бізнес-логіки та поведінки. Стандартна програма WPF зазвичай має головне вікно, яке містить вкладені панелі макета та різні елементи інтерфейсу (або елементи керування), як показано нижче [1].

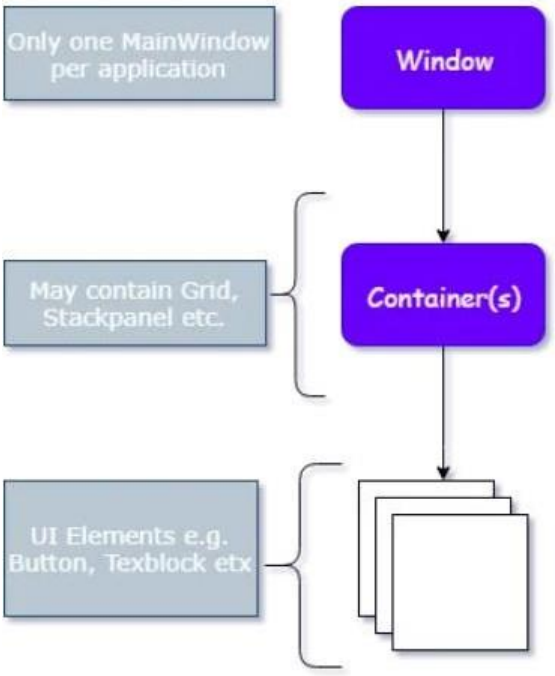


Рисунок 2.1 – Приклад WPF застосунку

MVVM – шаблон ViewModel організовує програму таким чином, щоб представлення отримували свій стан від моделей представлень, а не покладалися

на вбудований код. Дії користувача обробляються за допомогою команд, а не подій, а дані надаються представленням через зв'язування даних. Двостороннє зв'язування даних також дозволяє представленням даних надсилати дані назад до серверної частини. Ця структура робить великі додатки MVVM більш придатними для обслуговування, роблячи шаблон особливо придатним для рішень корпоративного рівня [2].

Окрім гнучкого інтерфейсу користувача, однією з найсильніших сторін WPF є система прив'язки даних (data binding), яка дозволяє створювати динамічні інтерфейси з мінімальним обсягом коду. Через механізм прив'язки можливо легко синхронізувати дані між модельною частиною додатка (Model або ViewModel) та візуальними компонентами (View), що значно спрощує розробку, тестування та підтримку ПЗ [2].

Ще однією важливою особливістю WPF є підтримка шаблонів (Templates) і стилів (Styles), що дозволяє розділяти логіку та візуальне оформлення інтерфейсу. За допомогою шаблонів можна повністю змінювати вигляд елементів керування без зміни їхньої функціональності, що особливо важливо при створенні адаптивних та кросплатформних інтерфейсів або при реалізації фірмового стилю підприємства.

WPF також забезпечує потужну підтримку графіки, включно з 2D і 3D-графікою, анімаціями та мультимедійними можливостями. Це дозволяє створювати не лише функціональні, але й візуально привабливі додатки. Інтеграція з бібліотекою DirectX забезпечує високу продуктивність навіть при складній візуалізації.

Шаблон проектування MVVM є ключовим підходом при розробці WPF-додатків. Його основна ідея полягає у чіткому розділенні відповідальностей: модель (Model) відповідає за доступ до даних і бізнес-логіку, модель представлення (ViewModel) слугує як проміжна ланка між моделлю та уявленням, а представлення (View) відповідає лише за відображення інформації. Завдяки такому підходу зменшується зв'язність між компонентами, що полегшує тестування, повторне використання коду та впровадження змін.

Для реалізації MVVM у WPF часто використовуються такі технічні засоби, як команди (ICommand), властивості з повідомленням про зміну (INotifyPropertyChanged), а також фреймворки на кшталт Prism, MVVM Light або CommunityToolkit.MVVM, які додатково спрощують розробку та підтримку масштабованих застосунків [2].

Таким чином, поєднання потужних можливостей WPF із шаблоном MVVM дозволяє створювати сучасні, гнучкі та масштабовані настільні додатки, які відповідають вимогам бізнес-логіки, зручності використання та візуальної привабливості.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Формування дерева проблем та дерева рішень

У процесі дослідження проблем управління запасами в бізнесі було побудовано дерево проблем, яке ілюструє логічний зв’язок між причинами, наслідками та кінцевими результатами неефективного управління. У центрі цієї моделі знаходиться проблема неефективного управління запасами, яка є критичною для стабільного функціонування підприємства. Вона виникає внаслідок низки внутрішніх недоліків, серед яких ключовими є брак автоматизації та недостатній рівень аналітики.

Брак автоматизації проявляється у використанні ручних процесів, що значно уповільнює обробку інформації, підвищує ризик помилок і ускладнює масштабування операцій. Крім того, застосування застарілих систем обмежує можливості інтеграції з сучасними технологіями, що ще більше ускладнює управління запасами. Паралельно з цим, недостатній рівень аналітики призводить до нечітких прогнозів попиту, що унеможливлює ефективне планування. Відсутність контролю за ключовими показниками, такими як обіговість запасів або рівень обслуговування клієнтів, не дозволяє оперативно реагувати на зміни в ринковому середовищі.

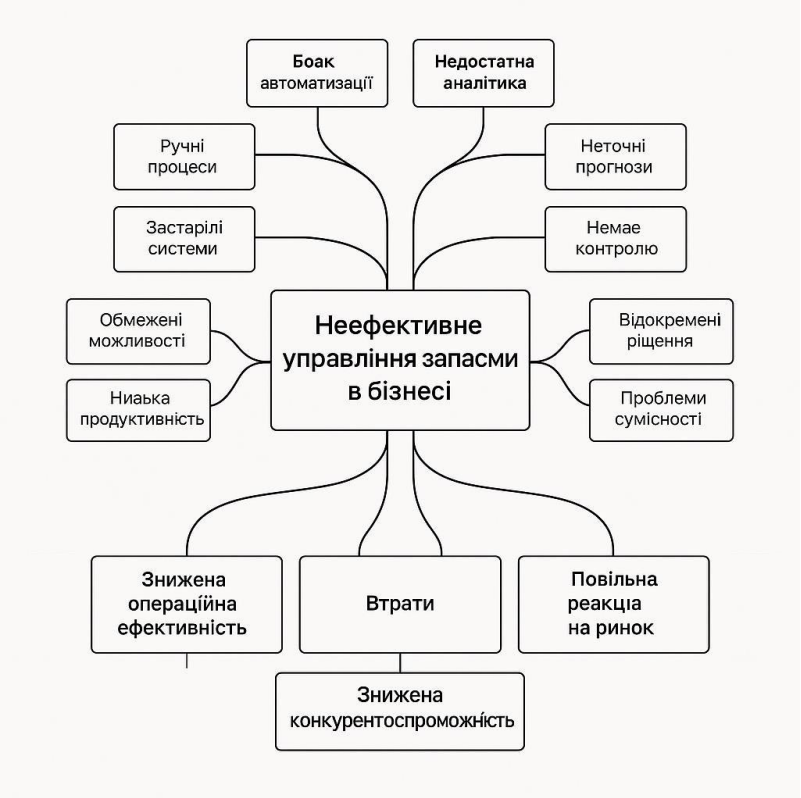


Рисунок 1.1 – Дерево проблем

Ці першопричини породжують низку негативних наслідків. Зокрема, компанія стикається з обмеженими можливостями адаптації до змін, зниженням продуктивності персоналу, який змушений витрачати час на рутинні завдання, а також з фрагментарністю прийняття рішень через відсутність єдиної інформаційної бази. До цього додаються проблеми сумісності між різними підрозділами або програмними системами, що ще більше ускладнює координацію дій.

У сукупності ці фактори призводять до зниження операційної ефективності підприємства. Виникають як прямі втрати, пов’язані з надлишковими або недостатніми запасами, так і непрямі — у вигляді втрати клієнтів, зниження рівня задоволеності споживачів та уповільнення реакції на ринкові зміни. У довгостроковій перспективі це все веде до зниження конкурентоспроможності компанії, оскільки вона втрачає здатність ефективно конкурувати з більш гнучкими та технологічно розвиненими учасниками ринку.

Рисунок 1.2 – Дерево рішень

У рамках дослідження шляхів підвищення ефективності управління запасами було побудовано дерево рішень, яке відображає ключові напрями вдосконалення цієї сфери. Дерево структуровано за чотирма основними напрямами: автоматизація, аналітика, сучасні технології та інтеграція систем. Кожен із цих напрямів відіграє важливу роль у формуванні ефективної системи управління запасами та забезпечує комплексний підхід до вирішення проблем, пов’язаних із обліком, прогнозуванням і контролем товарних залишків.

Першим напрямом є автоматизація, яка передбачає впровадження інструментів, що дозволяють зменшити частку ручної праці, підвищити точність обліку та прискорити обробку інформації. У межах цього напряму виділяються два ключові елементи: сканування штрихкодів та облік запасів. Сканування штрихкодів забезпечує швидке та точне зчитування інформації про товари, що значно знижує ризик помилок при введенні даних. Облік запасів, у свою чергу, дозволяє в режимі реального часу відстежувати наявність товарів, їхнє переміщення та залишки на складах.

Другим напрямом є аналітика, яка охоплює аналіз продажів та прогнозування попиту. Аналіз продажів дозволяє виявити тенденції споживчої поведінки, визначити найбільш популярні товари та оптимізувати асортимент. Прогнозування попиту базується на історичних даних і дозволяє планувати закупівлі таким чином, щоб уникнути як дефіциту, так і надлишкових запасів. Цей підхід сприяє зниженню витрат і підвищенню рівня задоволеності клієнтів.

Третій напрям – це впровадження сучасних технологій. Хоча в дереві рішень цей елемент не деталізується, він охоплює широкий спектр інноваційних рішень, таких як хмарні сервіси, мобільні додатки для управління складом, використання штучного інтелекту для оптимізації логістики тощо. Сучасні технології забезпечують гнучкість, масштабованість і доступність систем управління запасами.

Четвертий напрям – інтеграція систем. Вона включає базу даних, звітність і резервне копіювання. Наявність централізованої бази даних забезпечує єдине джерело достовірної інформації, що є критично важливим для прийняття обґрунтованих управлінських рішень. Система звітності дозволяє оперативно отримувати аналітичну інформацію про стан запасів, ефективність продажів та інші ключові показники. Резервне копіювання гарантує збереження даних у разі технічних збоїв або кібератак, що підвищує надійність усієї системи.

Таким чином, запропоноване дерево рішень демонструє комплексний підхід до побудови ефективної системи управління запасами, де кожен напрям взаємодіє з іншими, створюючи єдину інтегровану інфраструктуру. Такий підхід дозволяє не лише оптимізувати внутрішні процеси, а й підвищити конкурентоспроможність підприємства на ринку.

2.2 Алгоритми управління запасами

Однією з ключових задач розробленої системи автоматизації обліку товарів є ефективне управління товарними запасами. В умовах сучасної торгівлі це питання набуває особливого значення, оскільки неправильне управління запасами може

призводити як до перевантаження складу та замороження оборотного капіталу, так і до нестачі популярних товарів, втрати клієнтів та зниження доходів. У зв'язку з цим у системі було реалізовано два основних алгоритмічних підходи: алгоритм автоматичного формування замовлень на поповнення товарів та алгоритм прогнозування попиту на основі історичних даних [3].

2.2.1 Методи замовлення товарів

Першим кроком до ефективного управління запасами є визначення моменту, коли слід зробити нове замовлення. У нашій системі для цього використовується підхід, відомий як модель точки замовлення (англ. *Reorder Point model*). Суть цього підходу полягає в тому, що для кожного товару визначається такий мінімальний рівень запасів, за досягнення якого автоматично генерується нове замовлення [8].

Цей рівень розраховується на основі трьох ключових параметрів: середнього денного попиту на товар, часу постачання від моменту замовлення до моменту доставки (так званий *lead time*) та страхового запасу, який враховує можливі затримки доставки або непередбачене зростання попиту. Наприклад, якщо товар продається в середньому по 10 одиниць на день, а доставка триває 5 днів, то потрібно мати запас принаймні 50 одиниць, щоб не залишитися без товару до прибуття нової партії. Додаючи страховий запас, ми отримуємо так звану точку замовлення [6].

У рамках системи значення середнього денного попиту розраховується автоматично на основі статистики продажів за останні кілька тижнів або місяців. Кількість днів для вибірки можна задати вручну або визначити динамічно на основі типу товару (наприклад, для сезонних товарів враховуються продажі лише за відповідний сезон). Час постачання встановлюється для кожного товару або групи товарів залежно від постачальника. Страховий запас також може задаватися вручну або обчислюватися як відсоток від середнього попиту. Якщо фактична кількість товару на складі опускається до розрахованої точки замовлення або нижче, система

автоматично генерує внутрішнє повідомлення або замовлення на поповнення, яке можна відправити постачальнику [6].

Таким чином, алгоритм дозволяє уникати ручного контролю залишків і забезпечує безперебійну наявність необхідних товарів у продажу.

2.2.2 Прогнозування попиту

Другим ключовим елементом системи управління запасами є прогнозування майбутнього попиту на товари. У торгівлі, особливо в умовах мінливої ринкової кон'юнктури, важливо не лише фіксувати поточні продажі, а й передбачати, яка кількість товарів буде затребуваною в майбутньому. Це дозволяє не лише забезпечити наявність потрібного асортименту, а й оптимізувати обсяги закупівель, уникати перевищення залишків та ефективніше розподіляти фінансові ресурси підприємства [2].

Для реалізації прогнозування попиту у системі використано метод експоненційного згладжування (англ. *Simple Exponential Smoothing*). Цей метод є одним із найпоширеніших у прикладній статистиці та економетриці через свою простоту, ефективність та малу кількість параметрів. Його сутність полягає у тому, що нове прогнозоване значення обчислюється як зважене середнє між фактичним значенням продажів у попередній період і попереднім прогнозом. При цьому більша вага надається більш актуальним даним [1].

Ключовим параметром методу є коефіцієнт згладжування (позначається як альфа), який може набувати значень від 0 до 1. Чим ближче він до 1, тим більша вага надається останнім продажам, а чим ближче до 0 – тим більше враховується тренд у довгостроковій перспективі. Значення альфа може бути встановлене експериментально або налаштоване адміністратором системи залежно від характеру попиту на різні товари [2].

У системі прогнозування реалізоване автоматично: дані про продажі за попередні періоди зберігаються в базі даних, і для кожного товару розраховується прогноз на наступний період. Цей прогноз потім використовується як вхідне

значення у розрахунку точки замовлення. Таким чином, навіть якщо за останній тиждень продажі були нижчими, але за історичними даними очікується їх збільшення (наприклад, напередодні свят чи у сезон літніх товарів), система враховує цей фактор і формує відповідне замовлення [2].

У поєднанні з алгоритмом точки замовлення прогнозування попиту дозволяє створити адаптивну систему, яка не лише реагує на поточні зміни, але й готується до майбутніх сценаріїв розвитку попиту [1].

2.2.3 Практичне значення

Обидва алгоритми – і розрахунок точки замовлення, і прогнозування попиту – реалізовані на стороні серверної логіки .NET. Вони інтегровані з базою даних товарів, що забезпечує щоденне оновлення розрахунків без участі оператора. У графічному інтерфейсі користувач бачить рекомендовані замовлення, які може підтвердити одним кліком, а також може переглянути динаміку продажів і прогнозів у вигляді графіків [2].

Ці алгоритми суттєво підвищують ефективність управління торгівлею: зменшують втрати від нестачі товарів, знижують витрати на зберігання надлишкових залишків та покращують обслуговування клієнтів, адже товари рідко зникають із наявності [2].

Мета алгоритму автоматичного заповнення товарів: забезпечити своєчасне поповнення складу на основі:

- мінімального запасу (safety stock);
- середнього попиту на товар за останній період;
- часу доставки (lead time);
- коефіцієнта популярності товару.

Теоретичне обгрунтування: алгоритм базується на моделі точки замовлення (*Reorder Point*), що визначає, коли потрібно створити нове замовлення.

Формула для визначення точки замовлення:

$$ROP = D \cdot L + SS \tag{2.1}$$

де:

- ROP – точка замовлення (reorder point);
- D – середньодобовий попит на товар;
- L – час доставки товару (у днях);
- SS – страховий запас (safety stock), тобто резерв, який компенсує коливання попиту або затримки доставки.

Приклад:

1. Для кожного товару отримати:
   - Середньодобовий попит D за останні N днів;
   - Час доставки L (встановлений адміністратором або постачальником);
   - Поточну кількість товару на складі Q;
   - Страховий запас SS (встановлений вручну або як 20% від середнього попиту).
2. Обчислити $ROP = D \cdot L + SS.$
3. Якщо $Q \leq ROP$, сформувати замовлення у постачальника.

Приклад:

- D=5 одиниць/день,
- L=4 дні,
- SS=10 одиниць,
- Поточний запас Q=27

$$ROP=5 \cdot 4+10=30$$

Оскільки Q=27<ROP=30, система автоматично генерує замовлення на поповнення.

Мета алгоритму прогнозування попиту: прогнозування майбутнього попиту на товари дає змогу точно планувати закупівлі, уникати дефіциту чи надлишку товарів.

Метод експоненційного згладжування - цей метод є простим, швидким і ефективним для короткострокових прогнозів.

Формула експоненційного згладжування:

$$F_t = \alpha \cdot A_{t-1} + (1 - \alpha) \cdot F_{t-1} \quad (2.2)$$

де:

- F_t – прогноз на момент t,
- A_{t-1} – фактичне значення (обсяг продажів) у попередній період,
- F_{t-1} – прогноз на попередній період,
- $\alpha \cdot$ – коефіцієнт згладжування (0 < $\alpha \cdot$ < 1), зазвичай 0.2-0.3.

Алгоритм:

1. Ініціалізувати $F_1 = A_1$ (перший прогноз дорівнює першому фактичному значенню).
2. Для кожного наступного дня t застосувати формулу:

$$F_t = \alpha \cdot A_{t-1} + (1 - \alpha) \cdot F_{t-1} \quad (2.3)$$

3. Використовувати останній F_t як прогноз попиту для майбутніх днів.

Приклад:

Продажі товару за 5 днів:

- День 1: 20 одиниць
- День 2: 22
- День 3: 21
- День 4: 25
- День 5: 24

Нехай α=0.3

$$F_1 = 20$$
$$F_2 = 0.3 \cdot 20 + 0.7 \cdot 20 = 20$$
$$F_3 = 0.3 \cdot 22 + 0.7 \cdot 20 = 20.6$$
$$F_4 = 0.3 \cdot 21 + 0.7 \cdot 20.6 = 20.72$$
$$F_5 = 0.3 \cdot 25 + 0.7 \cdot 20.72 = 22.10$$

Отже, прогноз на день 6 становить:

$$F_6 = 0.3 \cdot 24 + 0.7 \cdot 22.10 = 22.67$$

Візуалізація:

X – дні, Y – кількість продажів. Фактичні дані (At) – синя лінія з кружечками. Прогнозовані значення (Ft) – помаранчева лінія з хрестиками. Точка замовлення – червона пунктирна горизонтальна лінія.

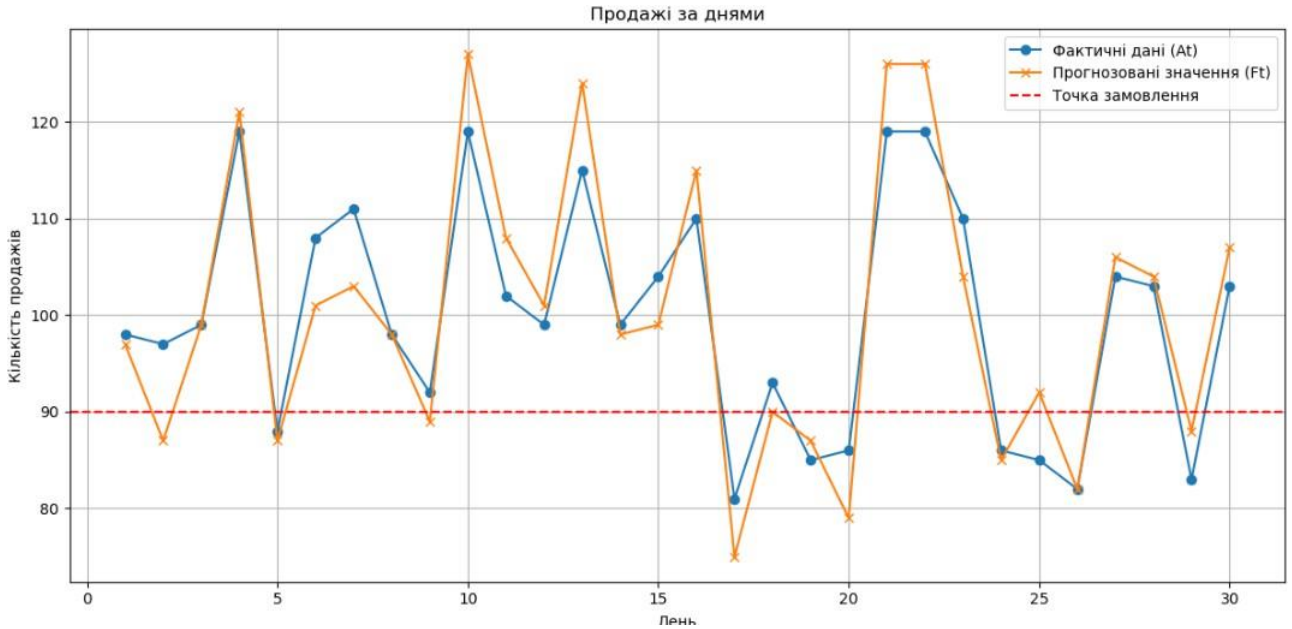


Рисунок. 2.2 – Графік співвідношення продажів за днями з кількістю продажів

Обидва алгоритми – і розрахунок точки замовлення, і прогнозування попиту – реалізовані на стороні серверної логіки .NET. Вони інтегровані з базою даних товарів, що забезпечує щоденне оновлення розрахунків без участі оператора. У графічному інтерфейсі користувач бачить рекомендовані замовлення, які може підтвердити одним кліком, а також може переглянути динаміку продажів і прогнозів у вигляді графіків [5].

Ці алгоритми суттєво підвищують ефективність управління торгівлею: зменшують втрати від нестачі товарів, знижують витрати на зберігання надлишкових залишків та покращують обслуговування клієнтів, адже товари рідко зникають із наявності [5].

У системі облік попиту та запасів відбувається щоденно. Алгоритм прогнозування використовується для розрахунку очікуваного попиту на наступний період, а алгоритм замовлення – для своєчасного поповнення складу.

Комбінування цих алгоритмів забезпечує:

- мінімізацію втрат через відсутність товарів,
- ефективне управління оборотним капіталом,
- зменшення ризиків перенакопичення товарів.

2.3 Алгоритми оптимізації процесів продажів

Одним із найважливіших аспектів ефективної роботи торговельного підприємства є оптимізація процесів продажів. Це дозволяє скоротити витрати, підвищити швидкість обслуговування клієнтів, збільшити конверсію продажів та підвищити загальну ефективність діяльності магазину. У розробленій системі автоматизації були реалізовані два важливих алгоритмічних компоненти, спрямованих на підвищення продуктивності: алгоритм оптимального розміщення товарів у торговельному залі та алгоритм оптимального формування замовлень. Розглянемо їх детально.

2.3.1 Алгоритм оптимального розміщення товарів в магазині

Актуальність задачі полягає в тому, що Розташування товарів у магазині має суттєвий вплив на поведінку покупця. Від того, наскільки зручно покупець може знайти товар, залежить не лише швидкість покупки, а й обсяг спонтанних покупок, рівень задоволеності клієнта та навантаження на персонал. Неефективна схема розміщення може призводити до черг, зайвих пересувань покупців та зниження середнього чеку. Саме тому в системі реалізовано алгоритм оптимізації розміщення товарів з урахуванням частоти їх продажу, категорії, пов’язаності з іншими товарами та загальної логістики магазину [5].

Задача розміщення товарів може бути зведена до оптимізаційної задачі на графі**,** де кожен товар є вершиною, а ребра – частота спільної покупки або логістичні зв'язки. Ціль – мінімізувати середню довжину маршруту покупця або сумарну відстань між товарами, які часто купуються разом [5].

Формулювання:

Нехай $T = \{t_1, t_2, ..., t_n\}$ – множина товарів. Визначимо матрицю сумісності $C = [c_{ij}]$, де c_{ij} – ймовірність того, що товари t_i та t_j будуть куплені разом. Нехай d_{ij} – відстань між позиціями товарів t_i та t_j у плані магазину [5].

Мета мінімізувати функцію:

$$f = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot d_{ij} \tag{2.4}$$

Це класична задача Quadratic Assignment Problem (QAP) – NP-складна, але може бути розв’язана евристичними методами [4].

Алгоритм реалізації полягає у використанні евристичної стратегії:

1. Товари сортуються за популярністю (кількість продажів).
2. Найпопулярніші товари розміщуються у найбільш доступних локаціях – біля входу або в центральних проходах.
3. Товари, які часто купуються разом (на основі історичних чеків), розміщуються поруч.
4. Малопопулярні товари – на периферії магазину.
5. Уникнення кластерів, які створюють скупчення людей (наприклад, хліб, молоко, вода – в різних зонах).

Приклад: У магазині виявлено, що товари «молоко», «кава», «цукор» часто купуються разом. Система на основі чеків сформувала таку матрицю сумісності:

Таблиця 2.1 – Сформована матриця сумісності

Товар	**Молоко**	**Кава**	**Цукор**
Молоко	-	0.8	0.6
Кава	0.8	-	0.7
Цукор	0.6	0.7	-

Система розміщує ці товари на сусідніх полицях, щоб мінімізувати переміщення покупців. Результати оптимізації можна представити графічно у вигляді топології розміщення товарів.

2.3.2 Алгоритм оптимального формування замовлення

Формування замовлень – один із найбільш витратних етапів в логістичному ланцюгу. Занадто часті замовлення призводять до зростання витрат на доставку, тоді як занадто великі – до перевантаження складу. Для вирішення цієї задачі у системі реалізовано алгоритм оптимального формування партій замовлень, що враховує кілька змінних:

- очікуваний попит на товар (на основі прогнозу);
- вартість одиниці доставки;
- фіксована вартість замовлення;
- мінімальний і максимальний розмір партії;
- умови постачальників (наприклад, лише кратні упаковкам по 10 од.).

Задача зводиться до моделі економічного обсягу замовлення (EOQ – Economic Order Quantity), що дозволяє знайти оптимальний розмір партії замовлення [4].

Класична формула EOQ:

$$EOQ = \sqrt{\frac{2 \cdot D \cdot K}{H}} \tag{2.5}$$

де:

- D – річний попит на товар;
- K – фіксована вартість розміщення одного замовлення;
- H – вартість зберігання одиниці товару протягом року.

Ця модель дозволяє зменшити сумарні витрати, поєднуючи економію на масштабі з витратами на зберігання.

У рамках реалізації системи формула EOQ була адаптована для щотижневого періоду. Алгоритм працює таким чином:

1. Для кожного товару обчислюється очікуваний попит на основі прогнозу.
2. Обчислюється EOQ – оптимальний розмір партії.
3. Перевіряються обмеження:
   - Мінімальне та максимальне значення партії,
   - Кратність (наприклад, тільки упаковки по 5 штук).
4. Після обчислення система генерує замовлення з максимально наближеним до EOQ обсягом.

Приклад: припустимо, що прогнозований попит на товар складає 1000 одиниць на рік, фіксована вартість замовлення – 200 грн, а витрати на зберігання – 5 грн/од. на рік.

$$EOQ = \sqrt{\frac{2 \cdot 1000 \cdot 200}{5}} = \sqrt{80000} \approx 283$$

Отже, оптимально замовляти товар партіями по 283 одиниці. Якщо постачальник приймає лише замовлення, кратні 50, система округлить до 300.

2.4 Математичні моделі та аналіз даних

У сучасних торговельних підприємствах, особливо тих, які використовують автоматизовані системи обліку, дедалі важливішим стає ефективне використання накопичених даних про продажі, залишки товарів, поведінку споживачів та інші аспекти діяльності. Просте накопичення інформації без її подальшого аналізу не дає конкурентної переваги. Саме тому в межах цієї дипломної роботи особливу увагу приділено впровадженню математичних моделей та методів аналізу даних, що дозволяють виявити закономірності, оптимізувати процеси управління та приймати обґрунтовані рішення [4].

У цьому розділі буде детально розглянуто два ключові напрямки: аналіз продажів та оптимізація ціноутворення.

2.4.1 Аналіз продажів

Аналіз продажів є основою для прийняття багатьох управлінських рішень – від формування асортименту до закупівельної стратегії. Його головна мета полягає у виявленні трендів (тобто загальних напрямів зміни попиту), сезонних коливань (повторюваних змін у попиті в залежності від пори року, свят, тощо), а також аномалій (раптових змін, які не піддаються звичайній логіці). Результати аналізу дозволяють прогнозувати майбутні обсяги продажів, уникати надмірних або недостатніх запасів, своєчасно реагувати на зміни ринку [3].

Одним із найпоширеніших способів дослідження динаміки продажів є побудова часових рядів**.** Це набір значень певного показника (у даному випадку – кількості реалізованих одиниць товару) у послідовні моменти часу. На основі такого ряду можна застосовувати низку статистичних і аналітичних методів.

Метод ковзного середнього дозволяє згладити випадкові коливання у даних. Суть його полягає у тому, що для кожного моменту часу обчислюється середнє значення продажів за попередні періоди. Таким чином, випадкові стрибки не впливають на загальну картину, і можна простежити загальну тенденцію [3].

Більш точним, і тому більш корисним у практиці, є метод експоненційного згладжування. Його перевага полягає у тому, що він надає більшої ваги останнім спостереженням, що особливо важливо у динамічному середовищі. Формально, кожне нове значення прогнозу розраховується як середньозважене поточного фактичного значення та попереднього прогнозу. Результат – плавна крива, яка відображає реальну динаміку без різких коливань [3].

Особливої уваги потребує сезонна складова, яка характерна для багатьох категорій товарів. Наприклад, шкільне приладдя користується підвищеним попитом у серпні-вересні, тоді як новорічні прикраси – у грудні. Щоб коректно врахувати такі коливання, використовується декомпозиція часових рядів – процес розкладання динамічного ряду на кілька складових: тренд, сезонність та випадкові коливання [3].

Такий підхід дозволяє окремо аналізувати кожен компонент і будувати точніші прогнози. Наприклад, знаючи типовий «сезонний індекс» для певного місяця, можна передбачити, який рівень попиту слід очікувати наступного року в цей же період [3].

Приклад застосування: припустімо, що в системі накопичено дані про щомісячні продажі товару за останні два роки. Було застосовано експоненційне згладжування з коефіцієнтом α=0.3. У результаті побудовано графік, на якому чітко видно загальну тенденцію до зростання попиту у літні місяці. Це дозволяє відповідальному працівнику запланувати збільшення запасів у червні-липні, щоб задовольнити зростаючий попит.

2.4.2 Оптимізація ціноутворення

Установлення ціни на товар – надзвичайно важливий крок, який має прямий вплив на дохід підприємства. З одного боку, висока ціна може знизити кількість покупок, з іншого – надто низька ціна може не покрити витрати або зменшити прибутковість. Тому завдання полягає у знаходженні оптимальної ціни, яка забезпечить максимальний прибуток або дохід [2]**.**

Ключовим поняттям при моделюванні попиту є цінова еластичність – показник, який характеризує, наскільки чутливим є попит до зміни ціни. Якщо невелике зменшення ціни веде до значного зростання кількості покупок, попит вважається еластичним. Якщо ж зміна ціни майже не впливає на обсяг продажів – попит нееластичний [2].

У математичному вигляді еластичність визначається як відношення відсоткової зміни обсягу попиту до відсоткової зміни ціни. Знання цього показника дозволяє моделювати функцію попиту і будувати аналітичну залежність між ціною і доходом [2].

Побудова моделі доходуЖ нехай функція попиту лінійна і має вигляд:

$Q(P)=a–bP$**,** де Q – кількість проданих товарів, P – ціна, a і b – коефіцієнти.

Тоді функція доходу буде: $R(P)=P\times Q(P)=aP-bP^2$. Ця функція є квадратичною, і її максимум досягається у точці: $P=a/(2b)$*.

Це і є оптимальна ціна, яка забезпечує максимальний дохід. У системі можна реалізувати модуль, який автоматично визначатиме параметри a і b за допомогою аналізу історичних даних (наприклад, методом найменших квадратів), а потім обчислюватиме рекомендовану ціну [2].

Приклад застосування: уявімо, що продажі товару відбувалися за такими цінами:

Таблиця 2.2 – Дані співвідношення цін товарів з їхніми продажами.

Ціна (грн)	Кількість продажів
100	50
90	60
80	70
70	80
60	90

На основі цих даних будується лінійна регресія, яка дозволяє наближено описати залежність між ціною і кількістю продажів. Отримавши значення a=140 та b=0.8, можна обчислити оптимальну ціну:

$P=140/(2\times0.8)=87.5$ *грн**

Отже, за даними моделі, найвигідніше продавати товар за ціною близько 87.5 грн. Це дозволить збалансувати обсяг продажів і рівень доходу.

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Послідовний опис розробки програмного забезпечення

При розробці програмного забезпечення на платформі WPF ключовим є використання шаблону Model-View-ViewModel (MVVM). Наш проєкт повністю побудований навколо цього шаблону, що спрощує розширення функціональності і швидке адаптування до потреб користувачів. Важливо забезпечити можливість авторизації для обмеження доступу лише для довірених осіб (директора, менеджерів, продавців). Кожен користувач має свої особисті облікові дані для входу до програми. При запуску додатку користувач відкриває MainWindow.xaml, де доступні два поля для введення логіна та пароля, а також кнопка Login. Після введення цих даних і натискання кнопки користувач проходить процес автентифікації [3].

Рисунок 3.1 – Вікно логінації на етапі розробки системи

```
<Grid>
    <StackPanel Orientation="Vertical"
                Margin="0,20,0,0">
        <Label Content="Simple Inventory Login"
               FontWeight="Bold"
               FontSize="20"
               HorizontalAlignment="Center" />
        <TextBox Width="300"
                 mahapps:TextBoxHelper.Watermark="Username"
                 Margin="10"
                 Text="{Binding Username, UpdateSourceTrigger=PropertyChanged}"
                 KeyDown="UsernameTextbox_KeyDown"
                 Name="UsernameInput" />
        <PasswordBox Name="PasswordInput"
                     Width="300"
                     mahapps:TextBoxHelper.Watermark="Password"
                     Margin="10"
                     PasswordChanged="PasswordInput_PasswordChanged"
                     KeyDown="PasswordInput_KeyDown" />
        <Button Content="Login"
                Width="150"
                Command="{Binding AttemptLogin}" />
        <Label Content="{Binding Error}"
               Foreground="IndianRed"
               FontWeight="Bold"
               FontSize="16"
               FontStyle="Italic"
               HorizontalAlignment="Center" />
    </StackPanel>
</Grid>
```

Рисунок 3.2 – Реалізація UI сторінки логінації

Після успішного входу користувач потрапляє на головну сторінку системи (App.xaml), де представлене меню з великим вибором опцій. Приклад реалізації інтерфейсу користувача наведено в Додатку А [3].

У проєкті використовується база даних SQLite, що дає змогу запускати програму локально без необхідності підключення до мережі [4].

Основними методами для роботи з базою даних є CreateDatabase та PerformMigrationsAsNecessary.

Метод CreateDatabase відповідає за створення та початкову ініціалізацію бази даних SQLite, створення необхідних таблиць і встановлення базових даних для забезпечення роботи системи. Зокрема, він виконує такі дії:

- Перевіряє наявність каталогу для зберігання бази даних і створює його, якщо потрібно.
- Створює файл бази даних SQLite у визначеній директорії.
- Встановлює з’єднання з базою даних і створює таблиці для товарів, користувачів, валют, інвентаризації, проданих товарів та штрих-кодів.
- Додає початкові дані, включно з обліковим записом адміністратора та валютами за замовчуванням (долар США і камбоджійський рієль).
- Визначає початкову версію бази даних.

Реалізацію методу CreateDatabase наведено в Додатку Б.

Метод PerformMigrationsAsNecessary забезпечує проведення міграцій бази даних SQLite, тобто внесення змін у її структуру відповідно до нових вимог.

Основна логіка міграцій зосереджена у цьому методі: він приймає об'єкт SQLiteCommand для виконання SQL-запитів [4].

У методі використовується оператор switch, де кожен «кейс» відповідає певній версії бази даних. Наприклад, якщо поточна версія дорівнює 1, додається нова таблиця QuantityAdjustments, після чого версію оновлюють до 2. За аналогічною схемою виконуються міграційні дії для наступних версій: створення нових таблиць, додавання стовпців, видалення чи перейменування таблиць тощо [4].

Загалом така реалізація гарантує, що структура бази даних завжди узгоджується з актуальною версією програми. Це дозволяє безпечно і автоматично оновлювати базу даних разом із розвитком системи [4].

Код методу PerformMigrationsAsNecessary наведено в Додатку В.

Окрім цих двох основних методів, у системі передбачені також додаткові методи та поля.

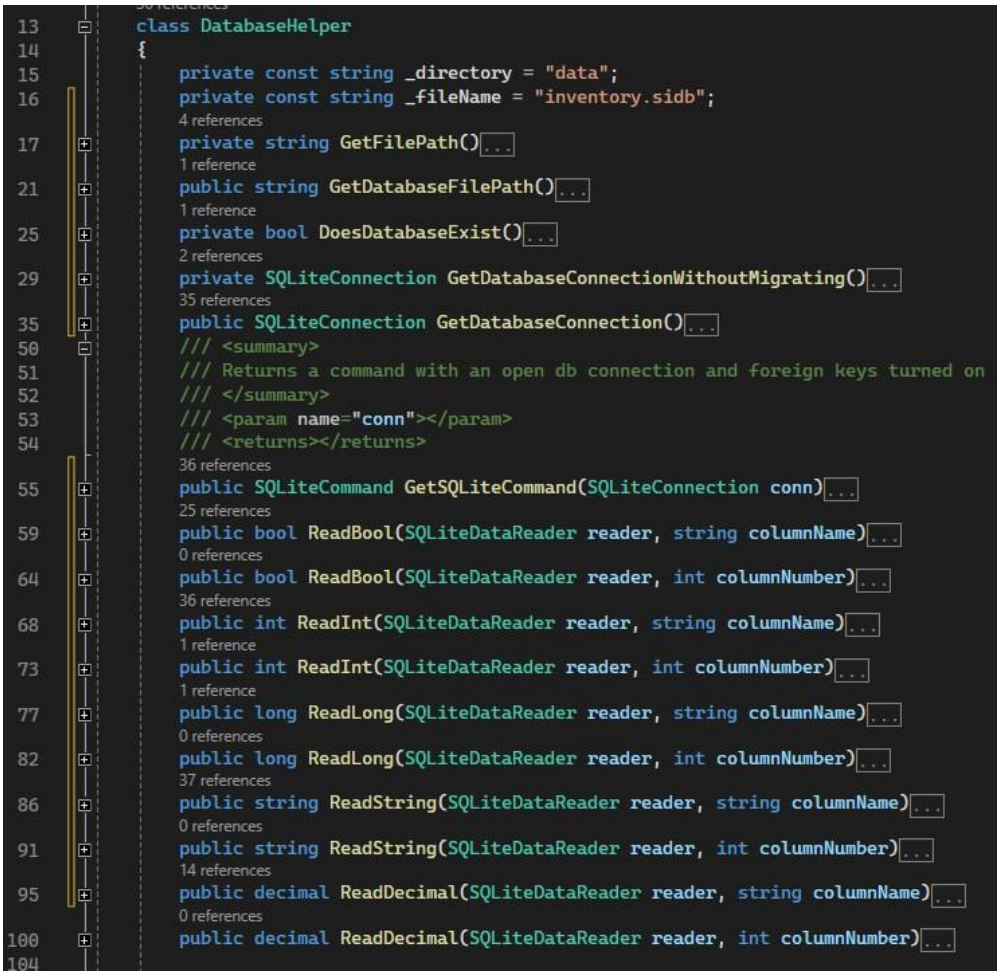


Рисунок 3.3 – Додаткові методи та поля у класі DatabaseHelper

Кожна система обліку товарів повинна мати можливість створювати звіти про товари. Такі звіти повинні містити як інформацію про наявні товари, так і про продані. Для реалізації цієї функції був розроблений клас StockInfoExcelGenerator, який відповідає за створення Excel-файлів із даними про запаси. Основним методом цього класу є ExportStockInfo, який реалізує процес генерації звіту. Деталі його роботи такі:

- Метод приймає список елементів типу List<DetailedStockReportInfo>, дату початку (startDate), дату завершення (endDate) та шлях до файлу (path), куди буде збережено результат. На першому етапі елементи списку сортуються за іменем та описом товару [5].
- Створюється новий Excel-файл (об’єкт XLWorkbook).
- Додається новий аркуш під назвою «Stock Info».
- У рядках 4 та 5 створюються заголовки таблиці.
- Далі проходить цикл заповнення інформації про кожен товар у відповідні клітинки Excel.
- Для кожного товару додаються формули для розрахунку різниці в кількості та вартості.
- У разі додавання нових колонок необхідно відповідно скоригувати область друку.
- Після заповнення обчислюється загальна вартість різниці товарів.
- Здійснюється форматування колонок і рядків для покращення читабельності.
- У фіналі створений файл зберігається за вказаним шляхом і відкривається для перегляду.

Реалізацію коду класу StockInfoExcelGenerator наведено в Додатку Г.

Окрім цього, був створений клас BarcodePDFGenerator, призначений для формування PDF-файлу зі штрих-кодами. Для цього використовуються бібліотеки PDFSharp і BarcodeLib [5].

Основні атрибути та методи класу:

- IsDryRun – булевий параметр, який визначає, чи потрібно фактично створювати PDF-файл і оновлювати базу даних. Якщо встановлено в true, створення файлу не виконується, лише розраховується кількість штрих-кодів.
- BarcodeType – визначає тип штрих-коду через BarcodeLib.TYPE, за замовчуванням встановлений CODE128.

- PageSize – задає розмір сторінки PDF, за замовчуванням використовується формат A4.
- NumberOfPages – визначає кількість сторінок у PDF-файлі, стандартне значення – одна сторінка.
- GenerateBarcodes(string outputPath) – метод для безпосередньої генерації PDF-файлу зі штрих-кодами. Приймає шлях збереження файлу (outputPath) і повертає кількість створених штрих-кодів. Він створює штрих-коди та розміщує їх на сторінках відповідно до заданих параметрів, обробляючи переходи на нові сторінки при необхідності.

Додатково клас має приватні методи для конвертації та зміни розміру зображень, що допомагає коректно розміщувати штрих-коди в PDF-файлі [5].

Детальний код реалізації класу BarcodePDFGenerator наведено у Додатку Д.

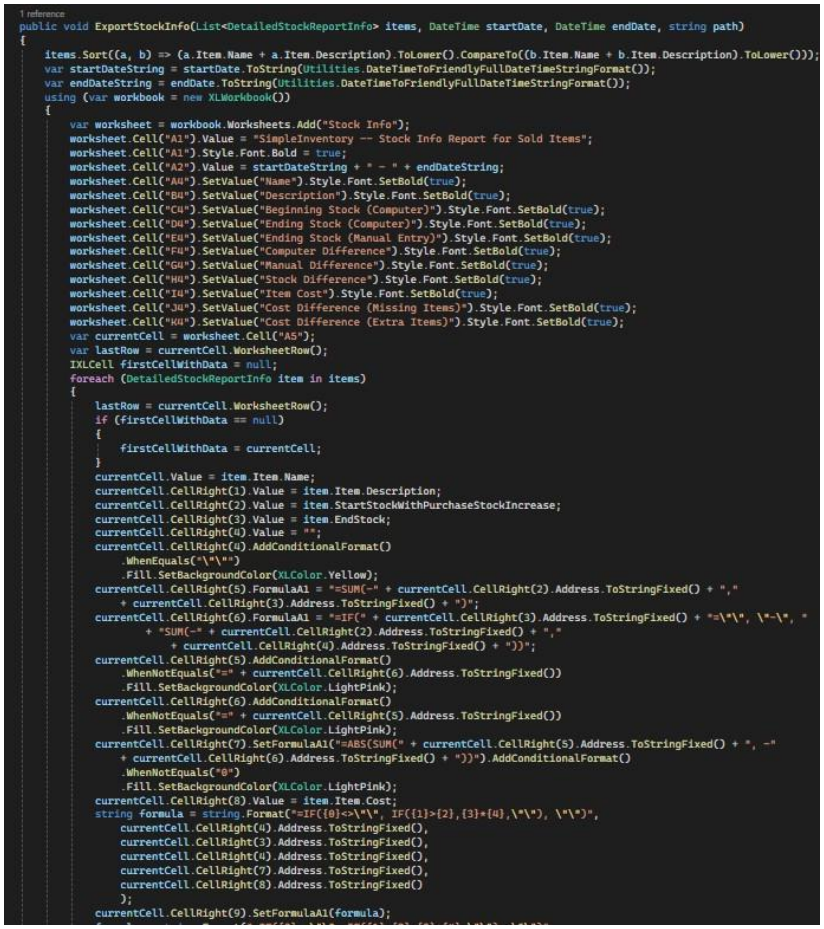
У будь-якому програмному забезпеченні, яке обробляє запаси товарів, повинна бути функція створення звітів про продані товари та наявні залишки. У нашій системі за це відповідає клас StockInfoExcelGenerator, основна функція якого – метод ExportStockInfo, що експортує список деталізованої інформації List<DetailedStockReportInfo> у форматі Excel.

Основні етапи роботи методу:

- Сортування елементів за іменем та описом товарів.
- Створення нового Excel-файлу та додавання аркуша для звіту.
- Заповнення заголовків таблиці.
- Занесення інформації про запаси товарів та встановлення відповідних формул.
- Застосування кольорового виділення до окремих клітинок за певними умовами.
- Автоматичне налаштування ширини стовпців та встановлення області друку.
- Збереження Excel-файлу на заданому шляху та відкриття його для перегляду.

Клас побудований таким чином, що легко піддається модифікаціям і доповненням, що робить процес створення звітів більш автоматизованим і зручним для бізнес-аналізу [6].

Рисунок 3.5 ілюструє реалізацію класу для генерації Excel-файлу із запасами товарів [6].



```
1 reference
public void ExportStockInfo(List<DetailedStockReportInfo> items, DateTime startDate, DateTime endDate, string path)
{
    items.Sort((a, b) => (a.Item.Name + a.Item.Description).ToLower().CompareTo((b.Item.Name + b.Item.Description).ToLower()));
    var startDateString = startDate.ToString(Utilities.DateTimeToFriendlyFullDateTimeStringFormat());
    var endDateString = endDate.ToString(Utilities.DateTimeToFriendlyFullDateTimeStringFormat());
    using (var workbook = new XLWorkbook())
    {
        var worksheet = workbook.Worksheets.Add("Stock Info");
        worksheet.Cell("A1").Value = "SimpleInventory -- Stock Info Report for Sold Items";
        worksheet.Cell("A1").Style.Font.Bold = true;
        worksheet.Cell("A2").Value = startDateString + " - " + endDateString;
        worksheet.Cell("A4").SetValue("Name").Style.Font.SetBold(true);
        worksheet.Cell("B4").SetValue("Description").Style.Font.SetBold(true);
        worksheet.Cell("C4").SetValue("Beginning Stock (Computer)").Style.Font.SetBold(true);
        worksheet.Cell("D4").SetValue("Ending Stock (Computer)").Style.Font.SetBold(true);
        worksheet.Cell("E4").SetValue("Ending Stock (Manual Entry)").Style.Font.SetBold(true);
        worksheet.Cell("F4").SetValue("Computer Difference").Style.Font.SetBold(true);
        worksheet.Cell("G4").SetValue("Manual Difference").Style.Font.SetBold(true);
        worksheet.Cell("H4").SetValue("Stock Difference").Style.Font.SetBold(true);
        worksheet.Cell("I4").SetValue("Item Cost").Style.Font.SetBold(true);
        worksheet.Cell("J4").SetValue("Cost Difference (Missing Items)").Style.Font.SetBold(true);
        worksheet.Cell("K4").SetValue("Cost Difference (Extra Items)").Style.Font.SetBold(true);
        var currentCell = worksheet.Cell("A5");
        var lastRow = currentCell.WorksheetRow();
        IXLCell firstCellWithData = null;
        foreach (DetailedStockReportInfo item in items)
        {
            lastRow = currentCell.WorksheetRow();
            if (firstCellWithData == null)
            {
                firstCellWithData = currentCell;
            }
            currentCell.Value = item.Item.Name;
            currentCell.CellRight(1).Value = item.Item.Description;
            currentCell.CellRight(2).Value = item.StartStockWithPurchaseStockIncrease;
            currentCell.CellRight(3).Value = item.EndStock;
            currentCell.CellRight(4).Value = "";
            currentCell.CellRight(4).AddConditionalFormat()
                .WhenEquals("\"\"")
                .Fill.SetBackgroundColor(XLColor.Yellow);
            currentCell.CellRight(5).FormulaA1 = "=SUM(-" + currentCell.CellRight(2).Address.ToStringFixed() + ","
                + currentCell.CellRight(3).Address.ToStringFixed() + ")";
            currentCell.CellRight(6).FormulaA1 = "=IF(" + currentCell.CellRight(3).Address.ToStringFixed() + "=\"\", \"-\", "
                    + "SUM(-" + currentCell.CellRight(2).Address.ToStringFixed() + ","
                        + currentCell.CellRight(4).Address.ToStringFixed() + "))";
            currentCell.CellRight(5).AddConditionalFormat()
                .WhenNotEquals("=" + currentCell.CellRight(6).Address.ToStringFixed())
                .Fill.SetBackgroundColor(XLColor.LightPink);
            currentCell.CellRight(6).AddConditionalFormat()
                .WhenNotEquals("=" + currentCell.CellRight(5).Address.ToStringFixed())
                .Fill.SetBackgroundColor(XLColor.LightPink);
            currentCell.CellRight(7).SetFormulaA1("=ABS(SUM(" + currentCell.CellRight(5).Address.ToStringFixed() + ", -"
                + currentCell.CellRight(6).Address.ToStringFixed() + "))").AddConditionalFormat()
                .WhenNotEquals("0")
                .Fill.SetBackgroundColor(XLColor.LightPink);
            currentCell.CellRight(8).Value = item.Item.Cost;
            string formula = string.Format("=IF({0}<>\"\", IF({1}>{2},{3}*{4},\"\"), \"\")",
                currentCell.CellRight(4).Address.ToStringFixed(),
                currentCell.CellRight(3).Address.ToStringFixed(),
                currentCell.CellRight(4).Address.ToStringFixed(),
                currentCell.CellRight(7).Address.ToStringFixed(),
                currentCell.CellRight(8).Address.ToStringFixed()
                );
            currentCell.CellRight(9).SetFormulaA1(formula);
```

```
            formula = string.Format("=IF({0}<>\"\", IF({1}<{2},{3}*{4},\"\"), \"\")",
                currentCell.CellRight(4).Address.ToStringFixed(),
                currentCell.CellRight(3).Address.ToStringFixed(),
                currentCell.CellRight(4).Address.ToStringFixed(),
                currentCell.CellRight(7).Address.ToStringFixed(),
                currentCell.CellRight(8).Address.ToStringFixed()
                );
            currentCell.CellRight(10).SetFormulaA1(formula);
            if (currentCell.WorksheetRow().RowNumber() % 2 == 0)
            {
                currentCell.WorksheetRow().Style.Fill.BackgroundColor = XLColor.LightGray;
            }
            currentCell = currentCell.CellBelow();
        }
        if (items.Count > 0)
        {
            currentCell.CellRight(8).SetValue("Cost Discrepency").Style.Font.SetBold(true);
            currentCell.CellRight(9).SetFormulaA1("=SUM(" + firstCellWithData.CellRight(9).Address.ToStringFixed()
                + ":" + currentCell.CellAbove(1).CellRight(9).Address.ToStringFixed() + ")").Style.Font.SetBold(true);
            currentCell.CellRight(10).SetFormulaA1("=SUM(" + firstCellWithData.CellRight(10).Address.ToStringFixed()
                + ":" + currentCell.CellAbove(1).CellRight(10).Address.ToStringFixed() + ")").Style.Font.SetBold(true);
        }
        worksheet.Columns().AdjustToContents(4, 4, 10, 25);
        worksheet.PageSetup.PrintAreas.Clear();
        var firstCellForPrinting = worksheet.Cell("A1");
        var lastCellForPrinting = items.Count > 0 ? currentCell.CellRight(10) : worksheet.Cell("J4");
        worksheet.PageSetup.PrintAreas.Add(firstCellForPrinting.Address.ToStringRelative() + ":" + lastCellForPrinting.Address.ToStringRelative());
        worksheet.PageSetup.SetRowsToRepeatAtTop("4:4");
        worksheet.PageSetup.PagesWide = 1;
        worksheet.PageSetup.PageOrientation = XLPageOrientation.Landscape;
        workbook.SaveAs(path);
        Process.Start(path);
    }
}
}
```

Рисунок 3.4 – Реалізація генерації Excel файлу з інформацією про запаси

На рис 3.5 представлено програмну реалізацію створення та зберігання щоденного звіту у форматі PDF.

```
1 reference
private void CreateAndSaveDayReport()
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    saveFileDialog.Filter = "PDF file (*.pdf)|*.pdf";
    saveFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
    saveFileDialog.FileName = "Daily-Inventory-Report-" + SelectedDailyReportDate.ToString("yyyy-MM-dd");
    var lastDayReportLocation = Properties.Settings.Default.LastDayReportSaveFolder;
    if (!string.IsNullOrWhiteSpace(lastDayReportLocation) && Directory.Exists(Path.GetDirectoryName(lastDayReportLocation)))
    {
        saveFileDialog.RestoreDirectory = true;
        saveFileDialog.InitialDirectory = lastDayReportLocation;
    }
    if (saveFileDialog.ShowDialog() == true)
    {
        try
        {
            var generator = new ReportPDFGenerator();
            generator.GeneratePDF(CurrentDaySalesReport, saveFileDialog.FileName);
            Properties.Settings.Default.LastDayReportSaveFolder = Path.GetDirectoryName(saveFileDialog.FileName);
        }
        catch (Exception)
        {
            MessageBox.Show("Error generating PDF! Please make sure to close the PDF with the same name" +
                " if it is open in Adobe or other software before generating a PDF report.", "Error!", MessageBoxButton.OK);
        }
    }
}
```

Рисунок 3.5 – Реалізація створення і зберігання щоденного звіту у форматі PDF

Далі детально розглянемо, що виконує код:

Створюється об’єкт класу SaveFileDialog, який дає змогу користувачу обрати місце для збереження файлу. Встановлюється фільтр для типу файлів – у даному випадку дозволено зберігати лише PDF-файли. Початковою папкою для діалогового вікна встановлюється робочий стіл користувача. Ім’я файлу формується за заданим шаблоном, що містить вибрану дату для звіту. Перевіряється, чи збережений попередній шлях для збереження звітів і чи існує відповідний каталог; якщо так, то він встановлюється як стартова папка для діалогу.

Далі відкривається діалогове вікно збереження файлу та очікується вибір користувача. Якщо користувач обирає шлях для збереження, створюється екземпляр класу ReportPDFGenerator, який відповідає за створення PDF-звіту. Викликається метод GeneratePDF, якому передаються дані звіту за день (CurrentDaySalesReport) та шлях збереження. Після успішного створення файлу шлях збереження оновлюється у налаштуваннях програми.

Якщо під час створення PDF-файлу виникає помилка, користувачу виводиться повідомлення із рекомендацією закрити відкритий PDF-файл перед повторною спробою.

Завдяки цьому коду користувач має можливість зберігати щоденні звіти у форматі PDF, вибираючи місце збереження, а система автоматично запам'ятовує останній вибраний шлях для подальшої роботи.

На рис. 3.6 зображено реалізацію процесу створення та збереження тижневих звітів у форматі PDF.

```
1 reference
private void CreateAndSaveWeeklyReport()
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    saveFileDialog.Filter = "PDF file (*.pdf)|*.pdf";
    saveFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
    saveFileDialog.FileName = "Weekly-Inventory-Report-" + SelectedWeeklyReportDate.ToString("yyyy-MM-dd");
    var lastWeekReportLocation = Properties.Settings.Default.LastWeekReportSaveFolder;
    if (!string.IsNullOrWhiteSpace(lastWeekReportLocation) && Directory.Exists(Path.GetDirectoryName(lastWeekReportLocation)))
    {
        saveFileDialog.RestoreDirectory = true;
        saveFileDialog.InitialDirectory = lastWeekReportLocation;
    }
    if (saveFileDialog.ShowDialog() == true)
    {
        try
        {
            var generator = new ReportPDFGenerator();
            generator.GeneratePDF(CurrentWeeklySalesReport, saveFileDialog.FileName);
            Properties.Settings.Default.LastWeekReportSaveFolder = Path.GetDirectoryName(saveFileDialog.FileName);
        }
        catch (Exception)
        {
            MessageBox.Show("Error generating PDF! Please make sure to close the PDF with the same name" +
                " if it is open in Adobe or other software before generating a PDF report.", "Error!", MessageBoxButton.OK);
        }
    }
}
```

Рисунок 3.6 – Реалізація створення та зберігання тижневих звітів у форматі PDF

Метод private void CreateAndSaveWeeklyReport() відповідає за формування та збереження щотижневого звіту.

SaveFileDialog saveFileDialog = new SaveFileDialog(); – створюється об'єкт класу SaveFileDialog, що дозволяє користувачу обрати місце для збереження файлу.

saveFileDialog.Filter = "PDF file (*.pdf)|*.pdf"; – встановлюється фільтр для вибору лише PDF-файлів.

saveFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Desktop); – початковою текою для збереження встановлюється робочий стіл користувача.

saveFileDialog.FileName = "Weekly-Inventory-Report-" + SelectedWeeklyReportDate.ToString("yyyy-MM-dd"); – задається шаблон імені файлу за замовчуванням із зазначенням дати тижневого звіту.

var lastWeekReportLocation = Properties.Settings.Default.LastWeekReportSaveFolder; – отримується шлях до останньої теки збереження звіту з налаштувань програми.

if (!string.IsNullOrWhiteSpace(lastWeekReportLocation) && Directory.Exists(Path.GetDirectoryName(lastWeekReportLocation))) – перевіряється наявність попередньої теки збереження та її існування.

saveFileDialog.RestoreDirectory = true; – вмикається автоматичне відновлення теки для діалогового вікна.

saveFileDialog.InitialDirectory = lastWeekReportLocation; – якщо попередня тека існує, вона встановлюється як стартова для діалогу збереження.

if (saveFileDialog.ShowDialog() == true) { ... } – відкривається вікно вибору місця збереження файлу; обробка продовжується у випадку підтвердження вибору користувачем.

var generator = new ReportPDFGenerator(); – створюється екземпляр генератора PDF-звітів.

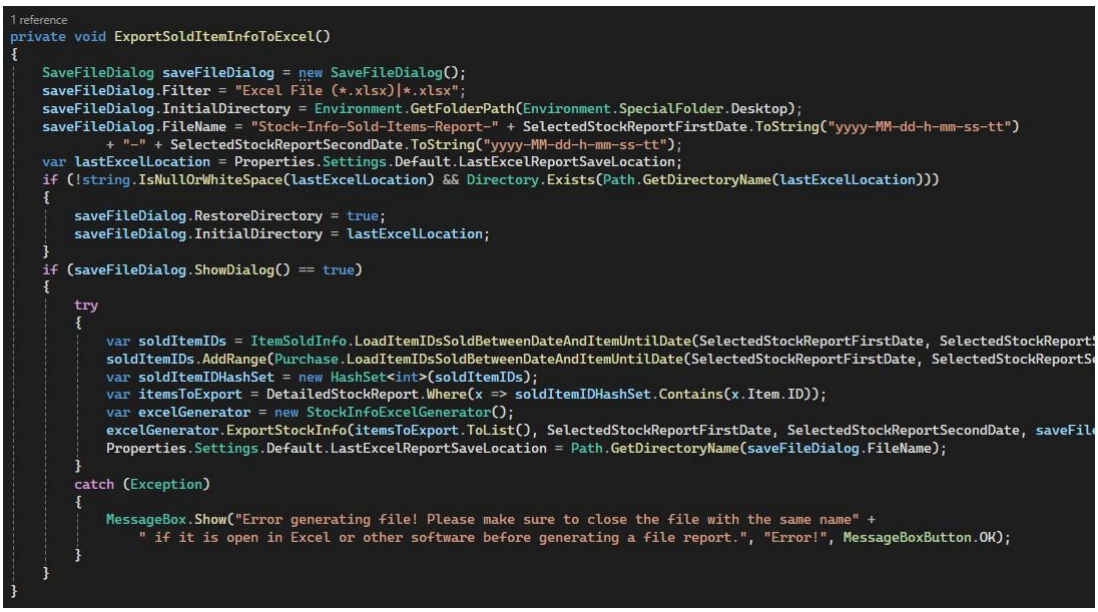
generator.GeneratePDF(CurrentWeeklySalesReport, saveFileDialog.FileName); – виконується генерація PDF-файлу на основі даних щотижневого звіту та обраного шляху збереження.

Properties.Settings.Default.LastWeekReportSaveFolder = Path.GetDirectoryName(saveFileDialog.FileName); – шлях останнього збереження оновлюється у налаштуваннях програми.

У випадку помилки користувач отримує відповідне повідомлення.

У цілому, цей код дає можливість користувачу створювати щотижневі звіти у форматі PDF та самостійно обирати місце для їх збереження.

Також передбачена можливість експортувати дані про продані товари за певний період у файл Excel.



```
1 reference
private void ExportSoldItemInfoToExcel()
{
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    saveFileDialog.Filter = "Excel File (*.xlsx)|*.xlsx";
    saveFileDialog.InitialDirectory = Environment.GetFolderPath(Environment.SpecialFolder.Desktop);
    saveFileDialog.FileName = "Stock-Info-Sold-Items-Report-" + SelectedStockReportFirstDate.ToString("yyyy-MM-dd-h-mm-ss-tt")
            + "-" + SelectedStockReportSecondDate.ToString("yyyy-MM-dd-h-mm-ss-tt");
    var lastExcelLocation = Properties.Settings.Default.LastExcelReportSaveLocation;
    if (!string.IsNullOrWhiteSpace(lastExcelLocation) && Directory.Exists(Path.GetDirectoryName(lastExcelLocation)))
    {
        saveFileDialog.RestoreDirectory = true;
        saveFileDialog.InitialDirectory = lastExcelLocation;
    }
    if (saveFileDialog.ShowDialog() == true)
    {
        try
        {
            var soldItemIDs = ItemSoldInfo.LoadItemIDsSoldBetweenDateAndItemUntilDate(SelectedStockReportFirstDate, SelectedStockReport
            soldItemIDs.AddRange(Purchase.LoadItemIDsSoldBetweenDateAndItemUntilDate(SelectedStockReportFirstDate, SelectedStockReportS
            var soldItemIDHashSet = new HashSet<int>(soldItemIDs);
            var itemsToExport = DetailedStockReport.Where(x => soldItemIDHashSet.Contains(x.Item.ID));
            var excelGenerator = new StockInfoExcelGenerator();
            excelGenerator.ExportStockInfo(itemsToExport.ToList(), SelectedStockReportFirstDate, SelectedStockReportSecondDate, saveFil
            Properties.Settings.Default.LastExcelReportSaveLocation = Path.GetDirectoryName(saveFileDialog.FileName);
        }
        catch (Exception)
        {
            MessageBox.Show("Error generating file! Please make sure to close the file with the same name" +
                " if it is open in Excel or other software before generating a file report.", "Error!", MessageBoxButton.OK);
        }
    }
}
```

Рисунок 3.7 – Представлення реалізації експорту інформації про продані товари між певними датами в Excel

Детальний опис коду для експорту інформації про продані товари у файл Excel починається зі створення екземпляру класу SaveFileDialog, що дає змогу користувачеві обрати місце збереження файлу. Діалогове вікно налаштовується таким чином, щоб дозволити збереження лише у форматі Excel-файлів (.xlsx). Початковим каталогом для вибору файлу встановлюється робочий стіл користувача. Ім’я файлу формується на основі дат, що зберігаються у змінних SelectedStockReportFirstDate та SelectedStockReportSecondDate [8].

Перевіряється наявність у налаштуваннях програми шляху до попереднього місця збереження Excel-файлу. Якщо користувач обирає місце збереження і підтверджує вибір (натискає «ОК»), виконується подальший блок коду:

Завантажується список ідентифікаторів товарів, які були продані у вказаний період.

Додаються також ідентифікатори придбаних товарів за той самий період.

Створюється множина (HashSet) для збереження унікальних ідентифікаторів проданих товарів.

Із загального списку DetailedStockReport обираються лише ті товари, чиї ідентифікатори присутні в цій множині.

Далі створюється об’єкт класу StockInfoExcelGenerator, що відповідає за створення Excel-файлу, і через нього здійснюється експорт обраних даних.

На завершення шлях до щойно збереженого Excel-файлу оновлюється у налаштуваннях програми [7].

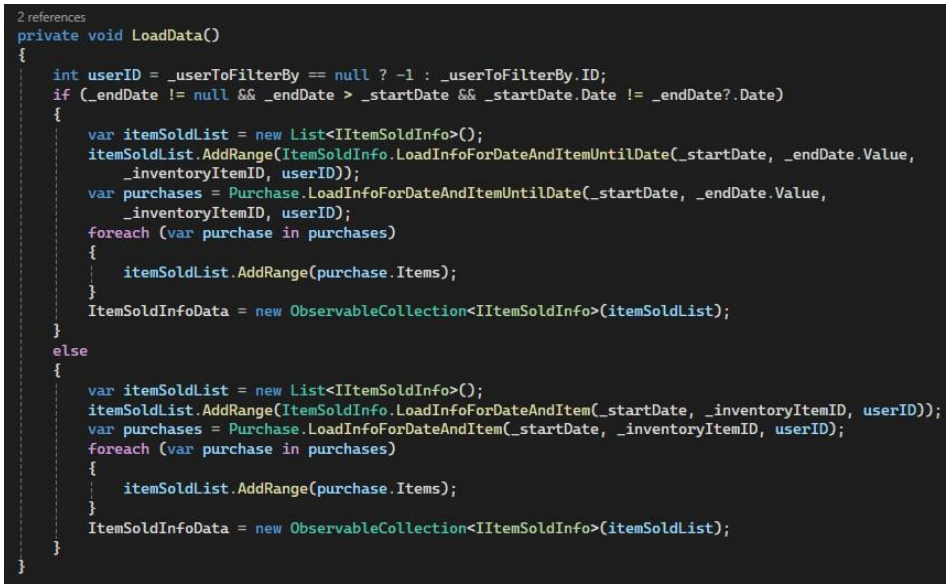


Рисунок 3.8 – Реалізація завантаження даних про продажі з певного інвентарю за певний період часу

Основні етапи роботи методу виглядають так:

Спочатку визначається змінна userID, яка представляє ідентифікатор користувача для фільтрації даних. Значення присвоюється на основі _userToFilterBy, і якщо воно дорівнює null, userID встановлюється як -1.

Далі перевіряється, чи обидві дати – _startDate і _endDate – задані і чи є вони коректними для фільтрації. Якщо дата початку менша за дату завершення і вони не збігаються, виконується наступне:

- Створюється список itemSoldList для зберігання інформації про продажі.
- Завантажуються дані про продажі за заданий період за допомогою методу LoadInfoForDateAndItemUntilDate() класу ItemSoldInfo, а також дані про покупки.
- Для кожної знайденої покупки її елементи додаються до списку itemSoldList.

- Після цього створюється об’єкт типу ObservableCollection<IItemSoldInfo> на основі заповненого списку і присвоюється властивості ItemSoldInfoData.

Якщо ж умова щодо дат не виконується (тобто _startDate більша або дорівнює _endDate, або _endDate не задана), аналогічний процес відбувається без обмеження на дату завершення.

Окремо важливим є створення автоматичної резервної копії бази даних. Для цього в системі реалізовано метод BackupDatabase, який має такі основні етапи:

- Метод BackupDatabase є приватним (private void) і не повертає значення.
- Створюється екземпляр SaveFileDialog, що дозволяє користувачеві обрати місце для збереження файлу.
- Встановлюється фільтр для вибору лише файлів формату .sidb.
- Початковою директорією обирається робочий стіл або остання тека, у якій зберігалася резервна копія.
- Формується стандартне ім'я файлу у форматі "inventory-backup-<дата-час>".
- Зчитується шлях до останньої резервної копії із налаштувань програми і, якщо такий шлях існує, встановлюється початковою директорією.
- Відкривається діалогове вікно збереження файлу. Якщо користувач підтверджує вибір, створюється екземпляр DatabaseHelper і копіюється поточний файл бази даних до обраного місця.
- Оновлюється шлях до останньої резервної копії в налаштуваннях.

У підсумку цей код забезпечує зручне створення резервної копії бази даних через інтерфейс користувача.

3.2 Тестування проєкту

У процесі тестування проєкту буде продемонстровано реалізацію інтерфейсу та описано функціональні можливості системи.

Головне вікно програми поділене на чотири основні блоки:

- Перший блок відповідає за управління товарами – тут можна додавати, редагувати та видаляти товари. Під час продажу або видачі товарів зі складу

є можливість сканувати штрихкоди окремих товарів або цілих груп товарів для формування замовлень.

- Другий блок надає інструменти для генерації штрихкодів для товарів, які можна надалі використовувати для маркування продукції. Також тут доступна генерація звітів щодо залишків на складі та продажів за різні часові періоди.
- У третьому блоці здійснюється управління категоріями товарів. Наприклад, можна редагувати наявні категорії, такі як «крупи» або «хлібобулочні вироби». Крім того, передбачена можливість створення резервної копії бази даних для відновлення інформації у разі збою програми.

Останній, четвертий блок призначений для керування користувачами, які мають доступ до системи, а також забезпечує можливість виходу з програми.

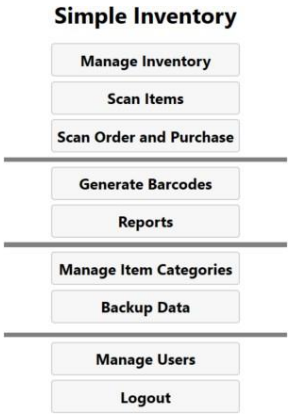


Рисунок 3.9 – Меню програми

Вікно зміни пароля для поточного користувача має такий вигляд: воно містить кнопку для повернення до головного меню, поле для введення нового пароля, поле для підтвердження введеного пароля та кнопку для збереження змін.

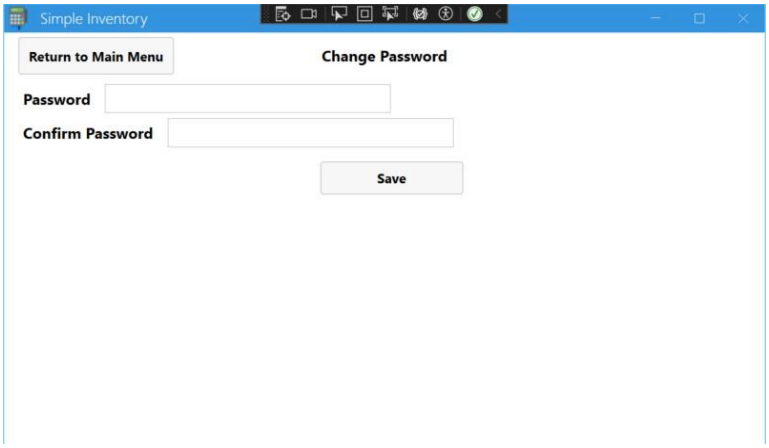


Рисунок 3.10 – Вікно зміни паролю

Кожна програма має містити вікно, яке інформує користувачів про розробників програмного забезпечення та надає контактні дані для співпраці чи пропозицій щодо покращення. У нашій програмі таке вікно також реалізоване – в ньому відображається інформація про авторів проєкту.

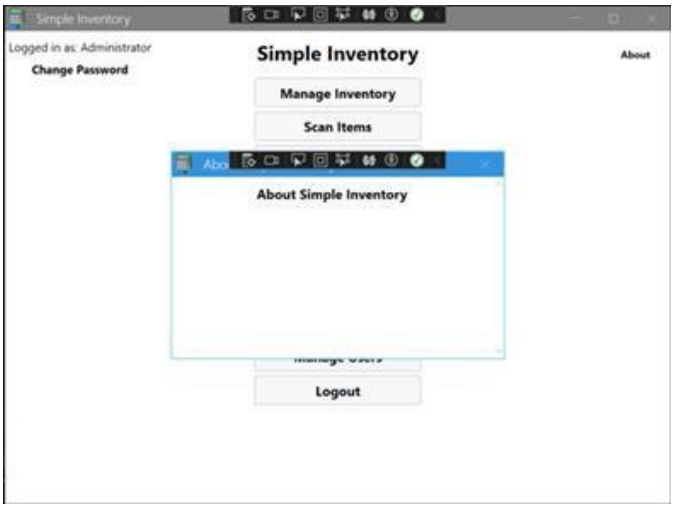


Рисунок 3.11 – Інформація про розробників програми

Так виглядає вікно зі списком усіх товарів, коли на склад ще не було додано жодного товару.

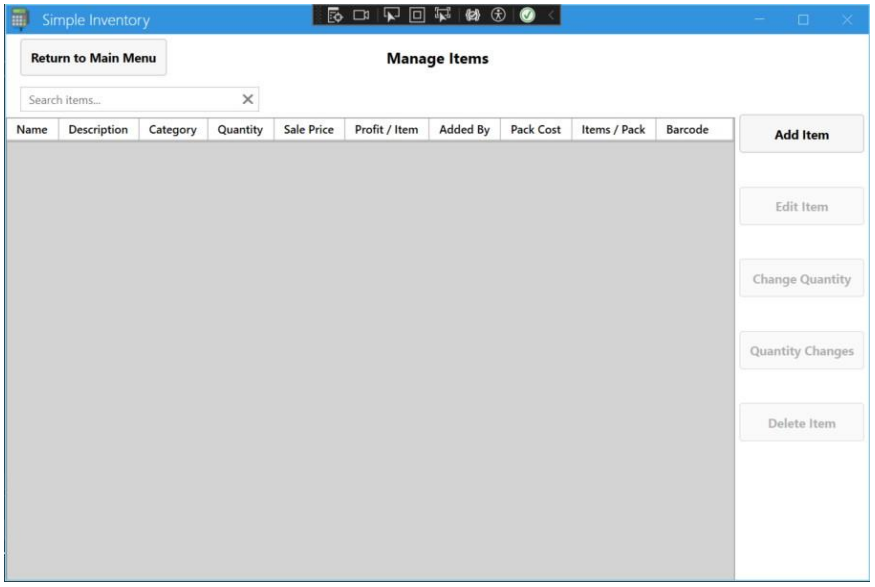


Рисунок 3.12 – Пустий список коли немає доданого жодного товару

При спробі видалення товару зі списку з’являтиметься діалогове вікно, яке запитуватиме у користувача, чи справді він бажає видалити цей товар. Це необхідно, щоб уникнути випадкових натискань на кнопку видалення і випадкового видалення товару.

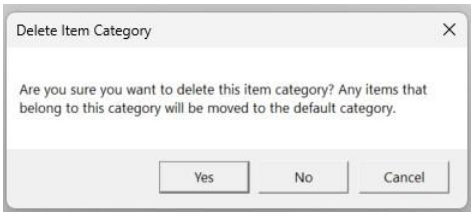


Рисунок 3.13 – Діалогове вікно при видаленні пункту зі списку

Кожному товару, який ми додаємо, необхідно призначити певну категорію. Це може бути категорія хлібобулочних виробів, молочних продуктів або м'ясної продукції. На цьому зображенні показано, що можна вказати назву категорії, її опис і зробити її категорією за замовчуванням. Якщо категорія встановлена за замовчуванням, вона автоматично присвоюється новому товару при його додаванні.

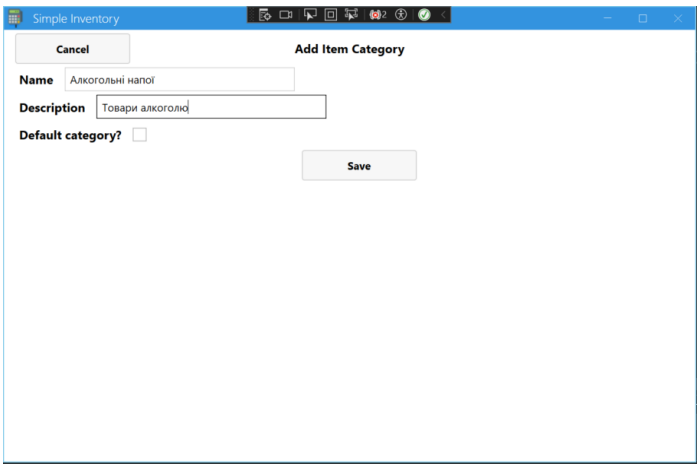


Рисунок 3.14 – Додавання категорій

На рисунку нижче є представлено вікно з доданими категоріями:

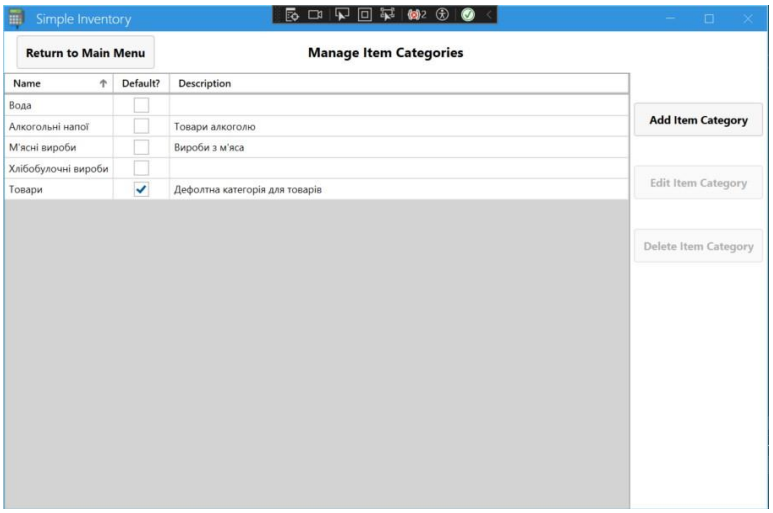


Рисунок 3.15 – Список з доданими категоріями товарів

Ми маємо три доступні опції для управління нашими категоріями: додавання нової категорії, редагування існуючої категорії зі списку та видалення категорії з цього списку.

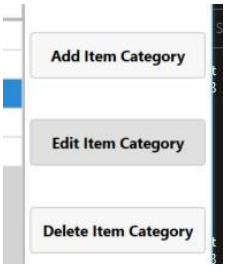


Рисунок 3.16 – Функціонал для оперування об’єктами зі списку

Під час редагування категорії зі списку будуть доступні ті ж самі поля, що і при додаванні нової категорії.

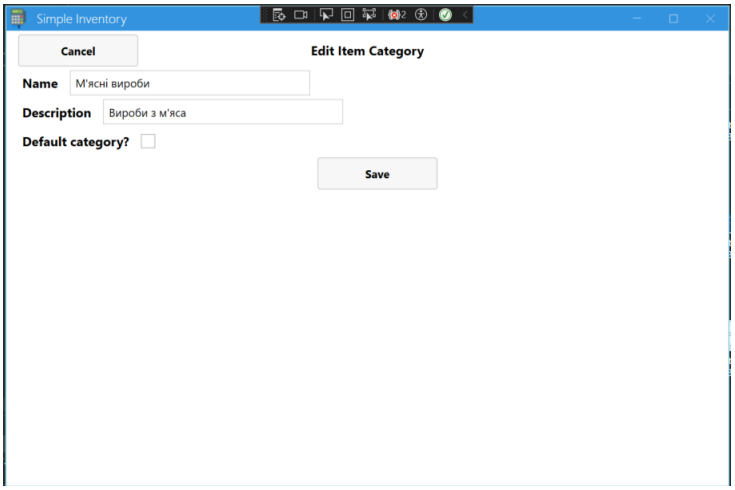


Рисунок 3.17 – Вікно редагування категорії товарів

Вікно для додавання нового товару містить такі поля: назва товару, опис, кількість, категорія та ціна за одиницю. Для бухгалтерії, коли товар замовляється на склад, передбачені додаткові поля: ціна продажу та прибуток від продажу однієї одиниці. Також є можливість створити індивідуальний номер штрих-коду.

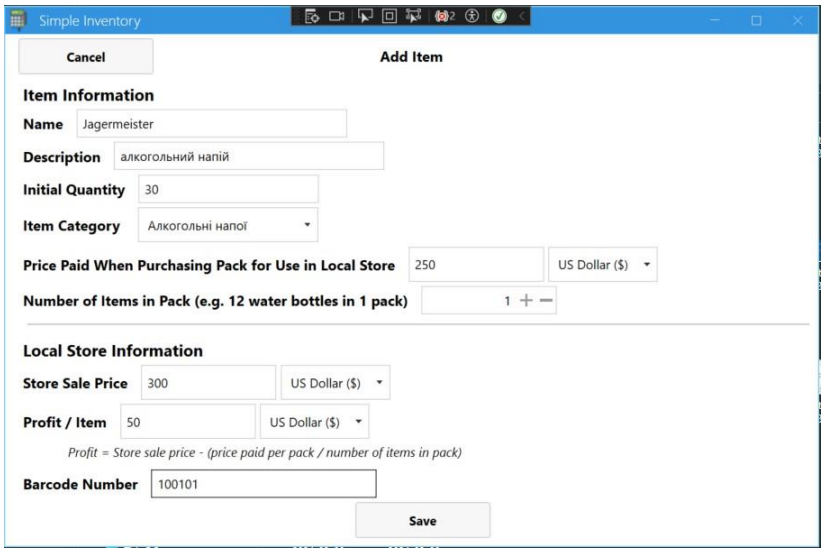


Рисунок 3.18 – Вікно додавання нового товару

Під час додавання товару ми також можемо вибрати валюту, в якій буде здійснюватися продаж цього товару. У тестовій версії доступні рієль та американський долар.

Рисунок 3.19 – Вибір валюти товару

Процес редагування товару на складі майже не відрізняється від додавання нового товару. На наведеному рисунку показано, як виглядає редагування товару.

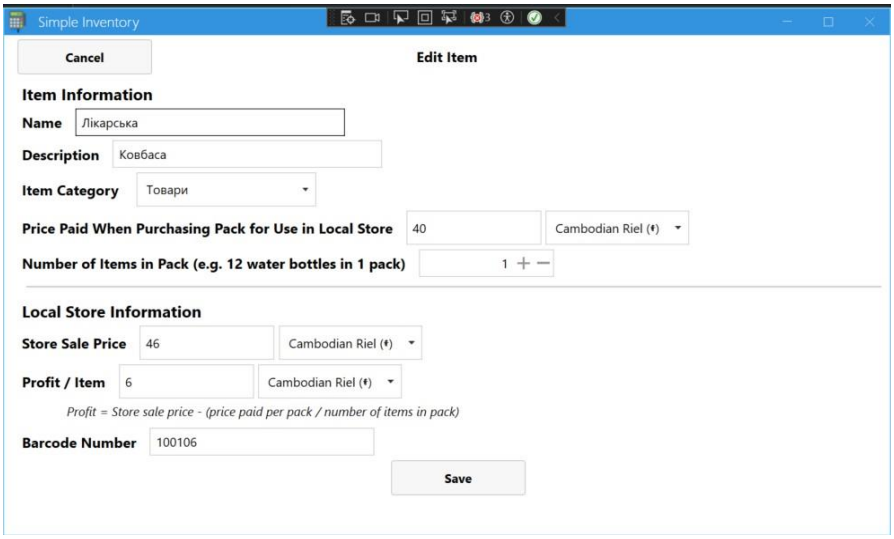


Рисунок 3.20 – Редагування товару зі списку

Після додавання товарів вони з'являються у списку всіх товарів. На наступному рисунку можна побачити цей список.

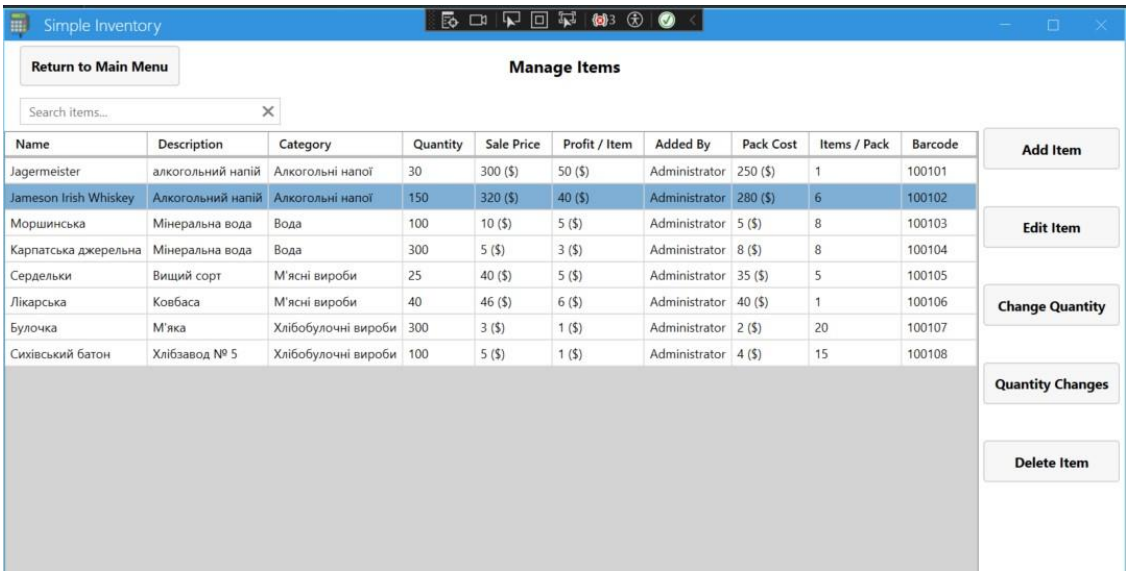


Рисунок 3.21 – Список з товарами на складі

Якщо відбувається зміна кількості товарів, ми можемо оновити кількість для конкретного товару. На наступному рисунку показано, як це можна зробити.

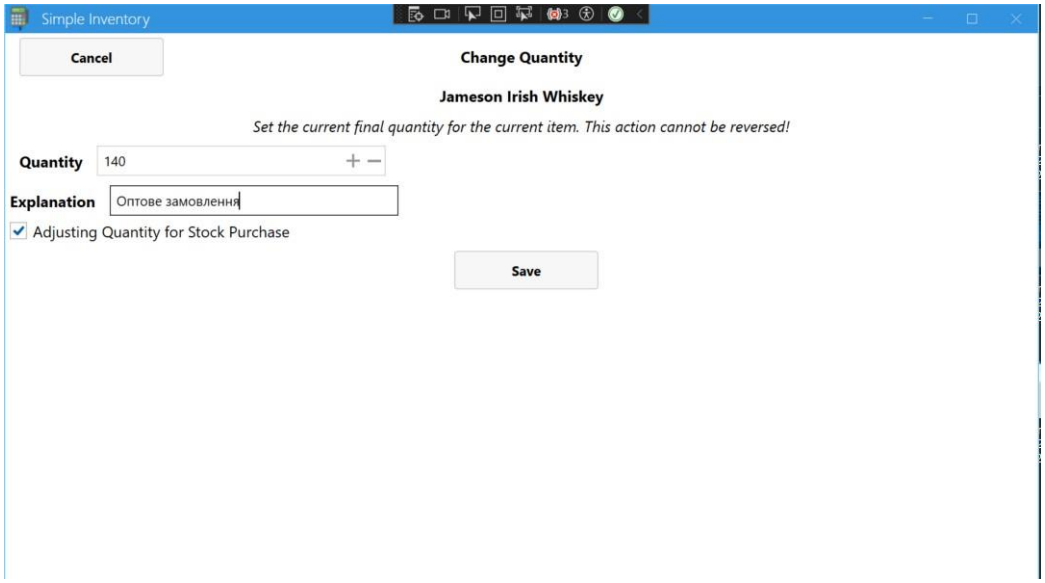


Рисунок 3.22 – Зміна кількості певного товару

Якщо кількість товарів змінюється, ми зберігаємо список, у якому фіксується кожна зміна та час її здійснення.

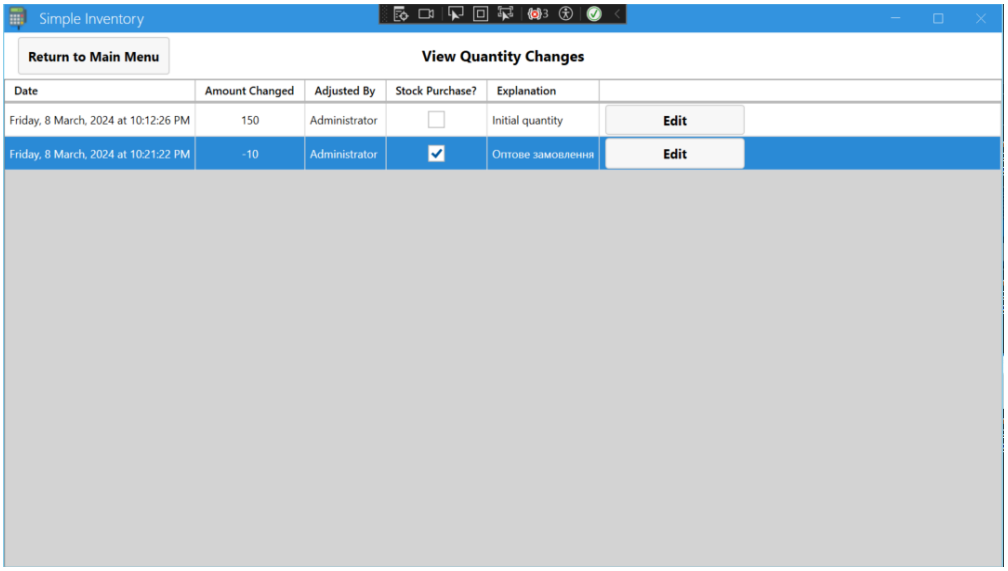


Рисунок 3.23 – Список зі змінами кількості певного товару

Під час продажу товарів ми можемо сканувати їх, для чого достатньо ввести номер штрих-коду, після чого з’являються деталі про цей товар.

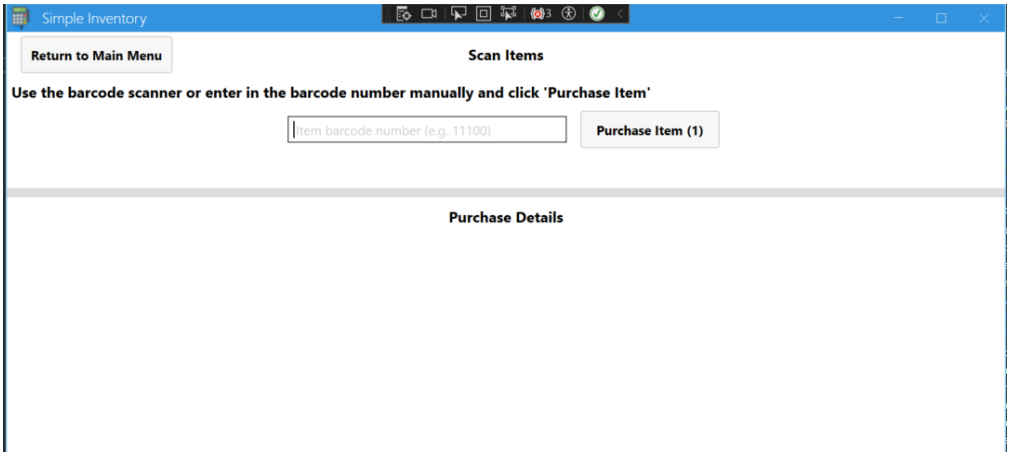


Рисунок 3.24 – Сканування товарів при продажі

Ось як виглядає інтерфейс після того, як були завантажені дані для введеного штрих-коду:

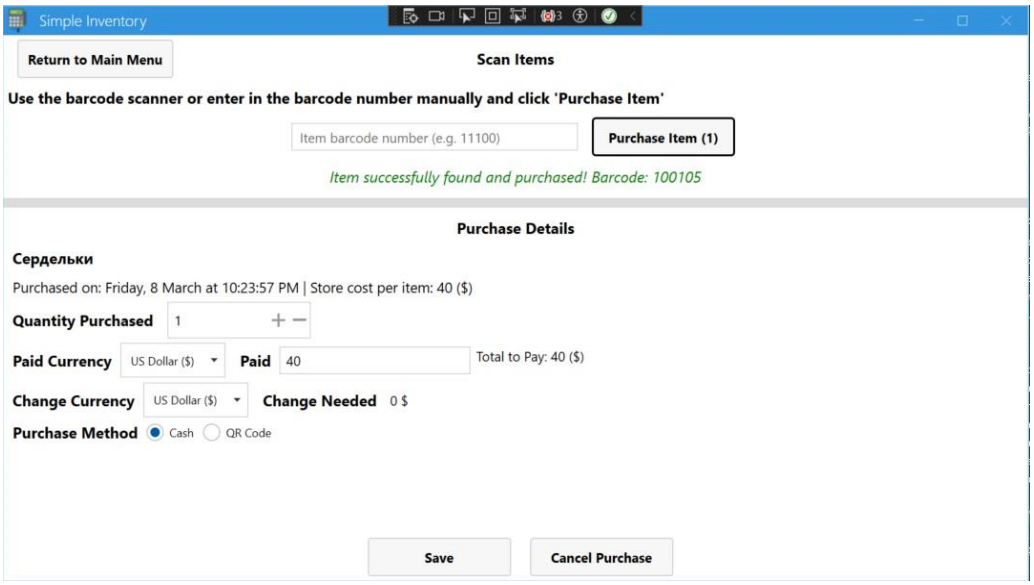


Рисунок 3.25 – Інформація про відсканований товар

У нашій програмі також реалізовано інтерактивне виведення повідомлень для користувача, які інформують його про результат дій при натисканні кнопок збереження – чи були вони успішними, чи ні.

Рисунок 3.26 – Інформаційний статус виконання операції

Після здійснення покупки кількість цього товару в загальному списку товарів буде оновлено.

Моршинська	Мінеральна вода	Вода	100	10 ($)	5 ($)	Administrator	5 ($)	8	100103
Сердельки	Вищий сорт	М'ясні вироби	22	40 ($)	5 ($)	Administrator	35 ($)	5	100105
Сихівський батон	Хлібзавод № 5	Хлібобулочні вироби	100	5 ($)	1 ($)	Administrator	4 ($)	15	100108

Рисунок 3.27 – Змінена кількість товару після його покупки клієнтом

Також можна вибирати кілька товарів за їхніми штрих-кодами під час проведення покупки.

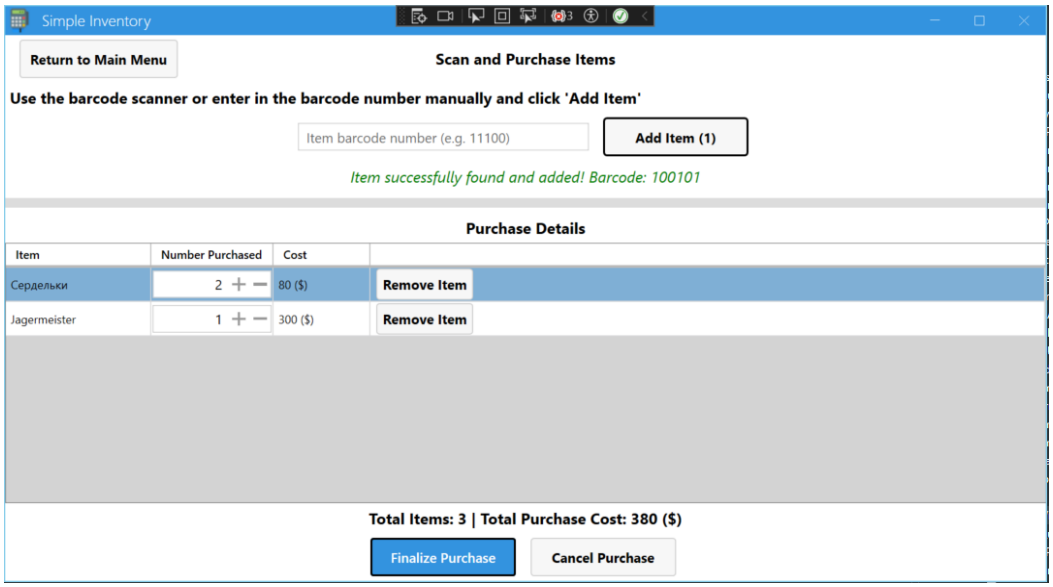


Рисунок 3.28 – Сканування штрихкодів декількох товарів при покупці

Після того, як усі товари, які клієнт бажає придбати, будуть додані, відбувається завершення покупки, і на сторінці фіналізації відображаються підсумкові суми.

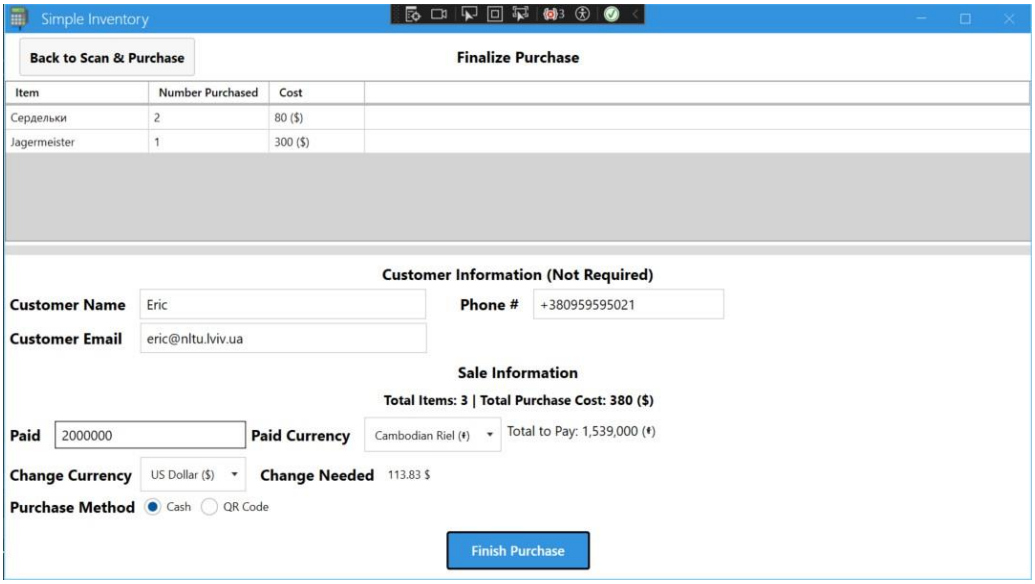


Рисунок 3.29 – Підбиття суми замовлених товарів для оформлення замовлення

Ми маємо можливість створювати штрих-коди, які можна приклеїти на товари. Для цього обираємо розмір паперу, кількість сторінок для друку та тип штрихкоду.

Рисунок 3.30 – Вікно генерації штрихкодів

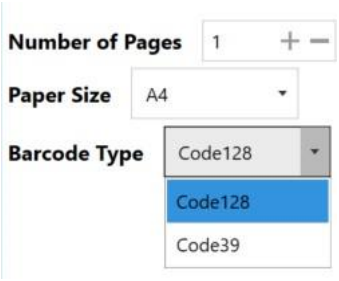


Рисунок 3.31 – Вибір типу штрихкоду

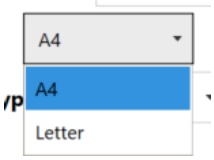


Рисунок 3.32 – Вибір розміру листа на якому будуть згенеровані штрихкоди

Згенеровані штрихкоди матимуть такий вигляд:

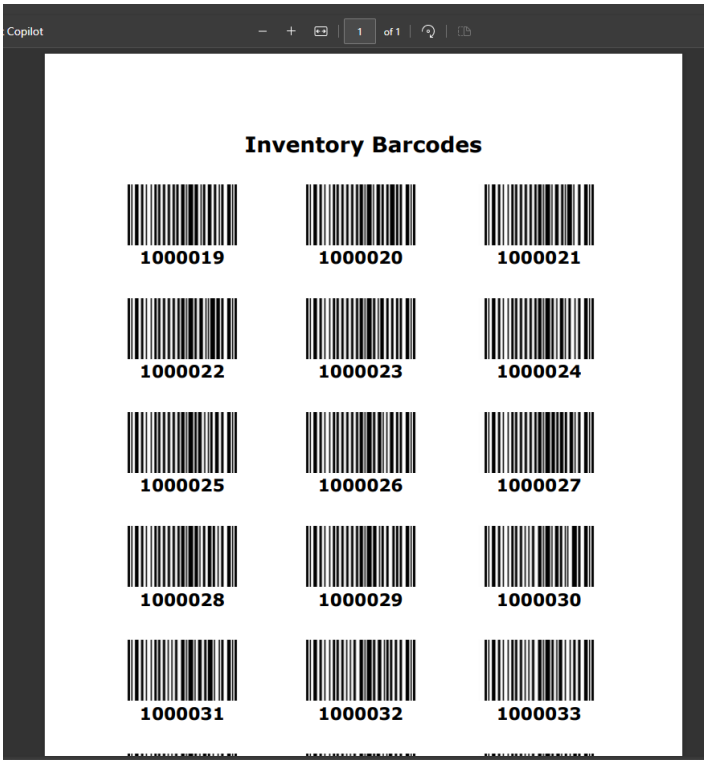


Рисунок 3.33 – Штрихкоди типу Code 128

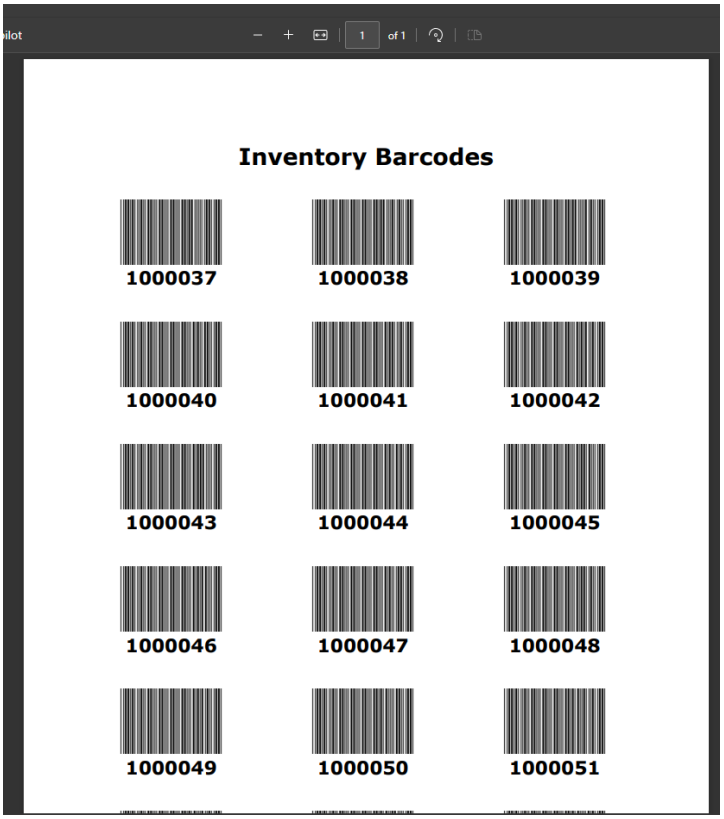


Рисунок 3.34 – Штрихкоди типу Code 39

Програма дозволяє генерувати звіти про продані товари за день або тиждень, а також можна отримати звіт про кількість товарів, що залишаються на складі.

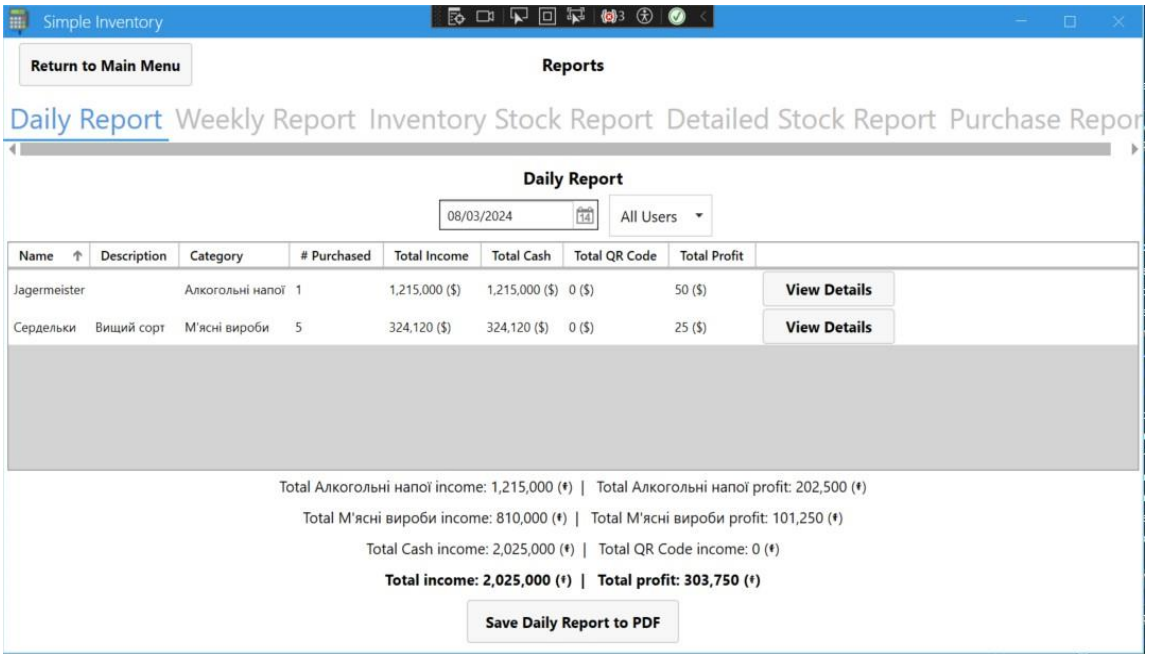


Рисунок 3.35 – Процес створення звіту проданих товарів за день або тиждень

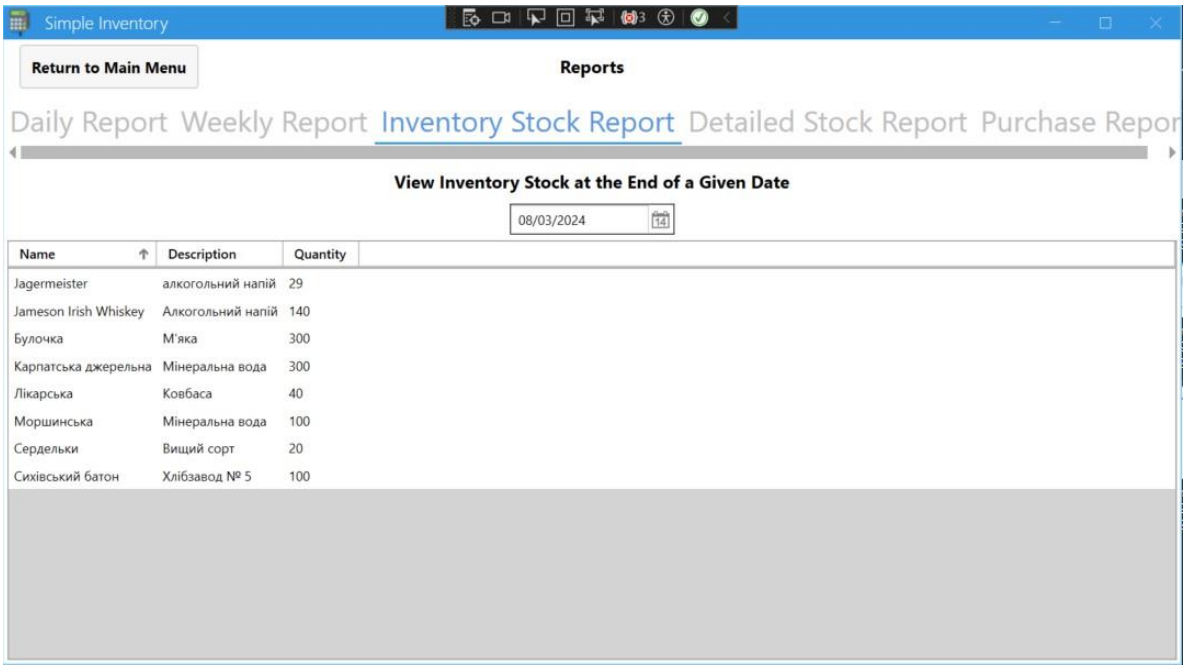


Рисунок 3.36 – Процес створення звіту залишків товарів на складі

Доступна можливість отримання деталізованого звіту зі складу:

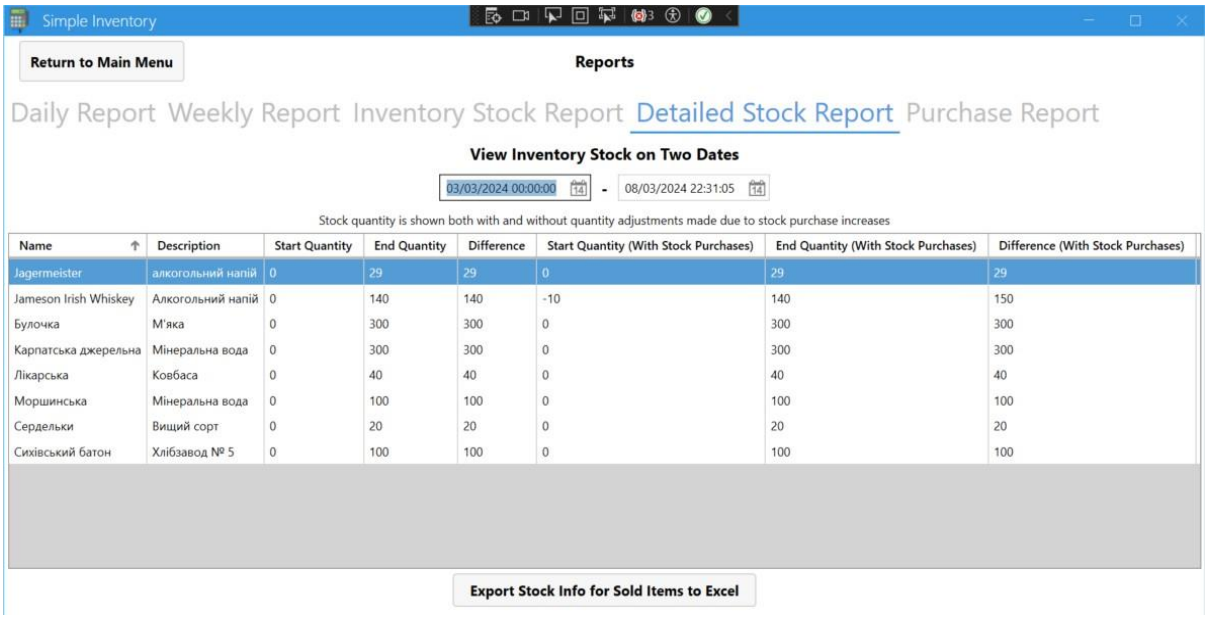


Рисунок 3.37 – Процес створення детального звіту залишків товарів на складі

Менеджери зможуть створювати звіти про покупки за конкретний день або за визначений період тижнів.

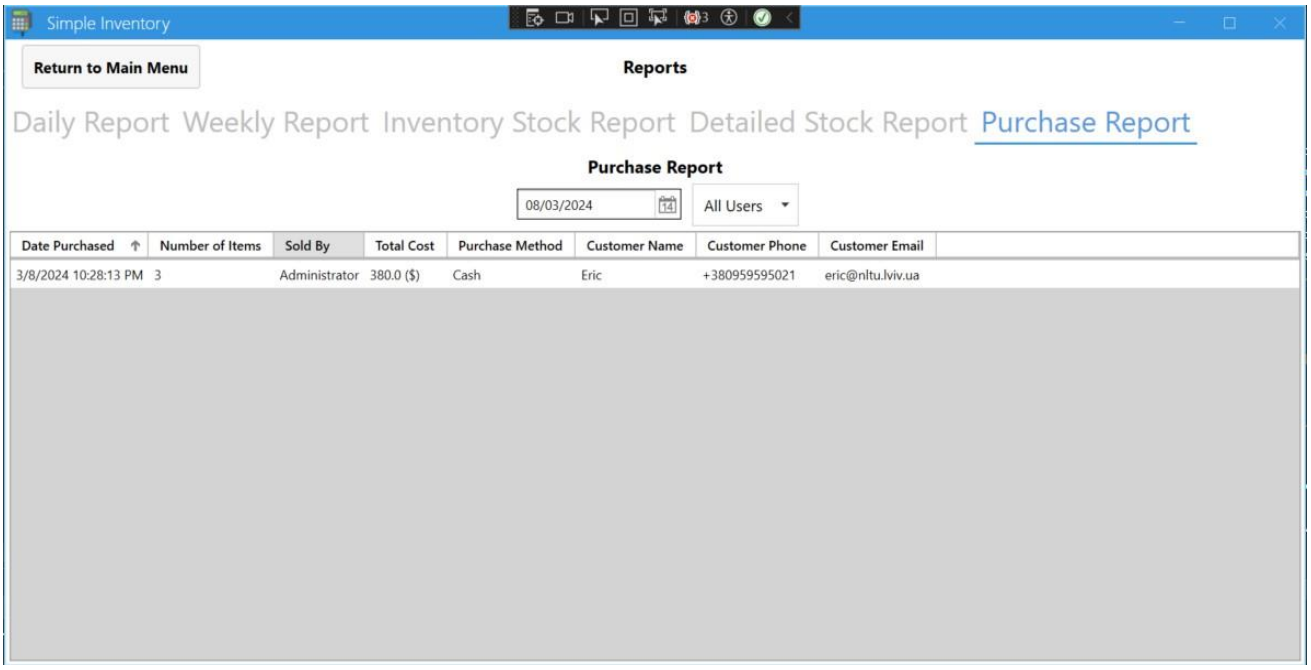
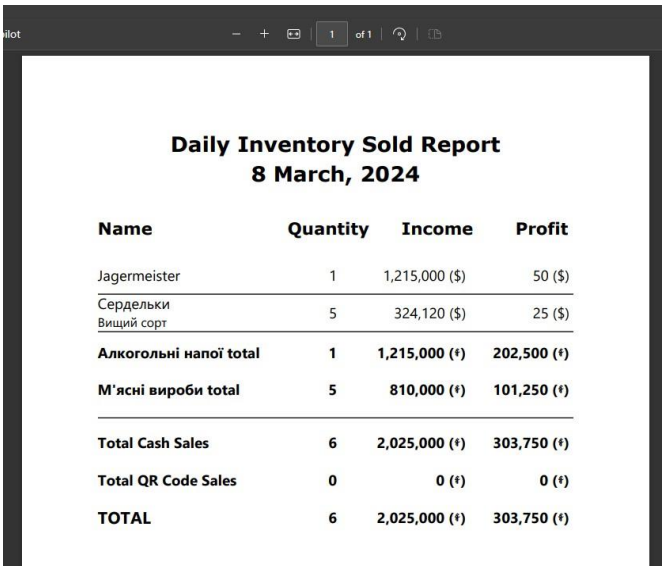


Рисунок 3.38 – Процес генерування звіту покупок за певний період часу

Згенеровані звіти у форматі PDF:



Daily Inventory Sold Report
8 March, 2024

Name	Quantity	Income	Profit
Jagermeister	1	1,215,000 ($)	50 ($)
Сердельки Вищий сорт	5	324,120 ($)	25 ($)
Алкогольні напої total	**1**	**1,215,000 (₹)**	**202,500 (₹)**
М'ясні вироби total	**5**	**810,000 (₹)**	**101,250 (₹)**
Total Cash Sales	**6**	**2,025,000 (₹)**	**303,750 (₹)**
Total QR Code Sales	**0**	**0 (₹)**	**0 (₹)**
TOTAL	**6**	**2,025,000 (₹)**	**303,750 (₹)**

Рисунок 3.39 – Приклад згенерованого звіту у форматі PDF

Під час створення бекапу програми на робочому столі буде згенерований файл з відповідним розширенням, який можна буде імпортувати для відновлення стану програми.

Рисунок 3.40 – Створена бекап файл на робочому столі

Ця програма також дозволяє додавати нових користувачів та редагувати інформацію про існуючих.

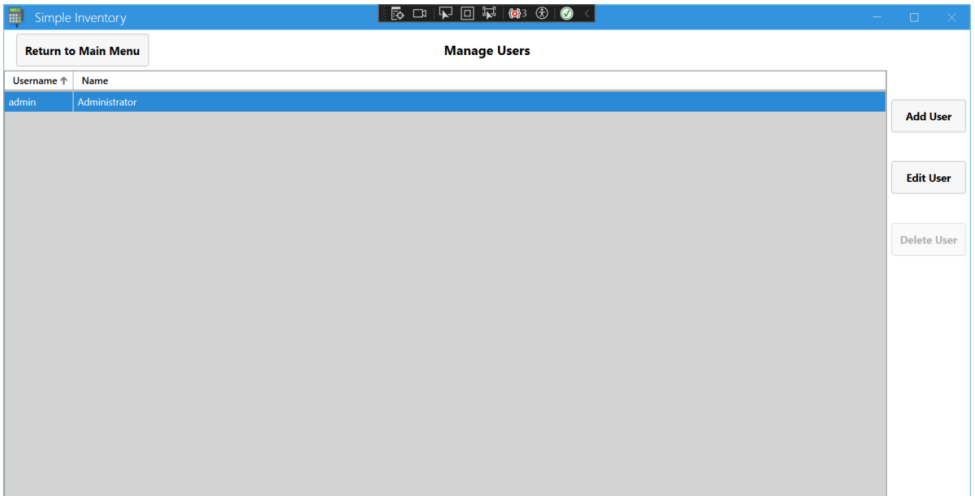


Рисунок 3.41 – Панель менеджменту користувачів

Кожен користувач матиме визначені права, які дозволяють редагувати, додавати та виконувати інші доступні функції в програмі.

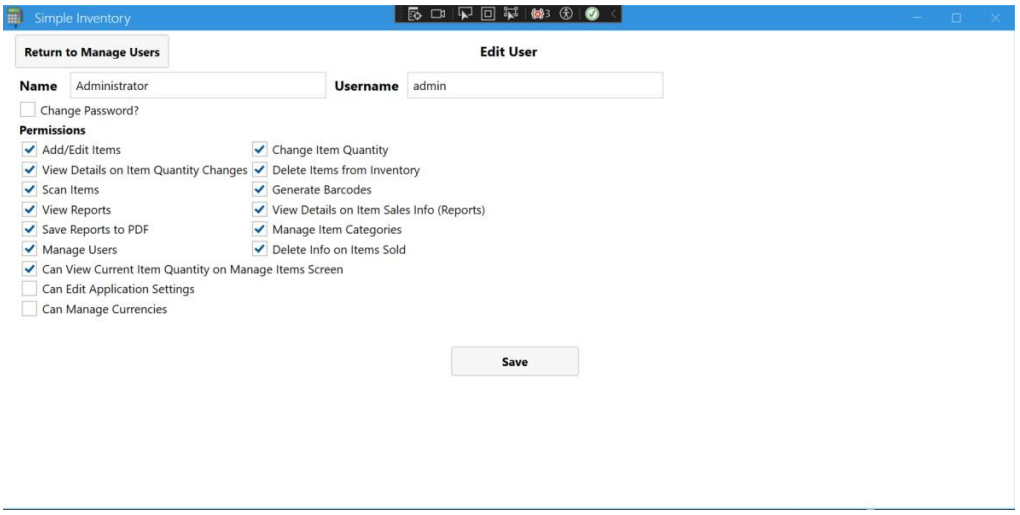


Рисунок 3.42 – Редагування доступів для користувача

ВИСНОВКИ

У результаті виконання технічного завдання була розроблена повнофункціональна система автоматизації обліку товарів, управління запасами та продажами із застосуванням сучасних засобів та технологій платформи .NET. Під час реалізації проєкту було повністю виконано всі поставлені вимоги.

По-перше, створено захищену сторінку логінації, яка дозволяє авторизувати користувачів із відповідною перевіркою облікових даних. Реалізовано головне меню з основними пунктами для зручної навігації в системі. Також передбачена можливість зміни паролю, що підвищує безпеку користувацьких облікових записів, і створено окрему інформаційну сторінку з описом програми, яка містить загальні відомості про систему та її призначення.

Основний функціонал системи реалізований згідно з поставленими завданнями:

- Модуль управління товарними запасами дозволяє додавати нові товари, редагувати їх описи, встановлювати кількість та інші характеристики.
- Впроваджено функцію сканування штрихкодів товарів при здійсненні покупки, що значно пришвидшує процес продажу та мінімізує ризики помилок.
- Реалізовано генерацію унікальних штрихкодів для кожного товару, а також функцію їх друку, що дозволяє оперативно маркувати продукцію.
- Створено модуль звітування, який забезпечує формування звітів щодо проданих товарів, фінансових оборотів та статистики продажів.
- Здійснено менеджмент категорій товарів, що дозволяє структурувати асортимент та спростити навігацію по базі.
- Для забезпечення надійного збереження даних реалізовано механізм створення резервних копій (бекапів) з можливістю подальшого відновлення.

Окрему увагу приділено системі управління доступами. Реалізовано можливість додавання нових користувачів, редагування їхніх даних, надання або

зміни прав доступу, а також функцію розлогування, що гарантує безпечну роботу з системою.

У результаті дослідження та практичної реалізації можна зробити висновок, що розроблена система повністю відповідає сучасним вимогам до автоматизації бізнес-процесів. Вона забезпечує надійний контроль за товарними запасами, ефективне управління продажами та гнучке адміністрування користувачів. Система має інтуїтивно зрозумілий інтерфейс, зручну навігацію та високу продуктивність, що дозволяє інтегрувати її у будь-який торговельний або складський процес.

Її впровадження сприятиме оптимізації внутрішніх операцій підприємства, зменшенню витрат часу на облік, підвищенню точності даних, а також забезпечить конфіденційність та цілісність інформації. Таким чином, дана система є стратегічно важливим рішенням, яке посилить конкурентоспроможність підприємства та сприятиме його успішному розвитку в умовах цифрової трансформації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Mark J. Price. Modern C# and .NET Guidance. Торонто : Manning Publications, 2023 . – 350 с.

2. Andrew Troelsen, Philip Japikse. Pro C# 11 and .NET 7: Deep Dive into Modern C# Development. Нью-Йорк : Apress, 2023 . – 600 с.

3. Joseph Albahari, Ben Albahari. C# 12.0 in a Nutshell. O'Рейлі, 2023 . – 480 с.

4. Tanay Parikh. Practical .NET for Beginners: From Novice to Professional. Нью-Делі : BPB Publications, 2023 . – 320 с.

5. Claudio Silva. Mastering .NET Performance: Building Scalable and Responsive Applications. Лондон : Packt Publishing, 2023 . – 240 с.

6. Mark Collins. WPF 2023: Comprehensive Beginner's Guide to Mastering Windows Presentation Foundation. Сан-Франциско : No Starch Press, 2023 . – 300 с.

7. Matthew MacDonald. Pro WPF in C# 11: Windows Presentation Foundation in .NET. O'Рейлі, 2023 . – 360 с.

8. Allen Jones. WPF Recipes in C# 2023: A Problem-Solution Approach. Нью-Йорк : Apress, 2023 . – 280 с.

9. Alex Kroll. Building Modern WPF Applications. Лондон : Packt Publishing, 2023 . – 270 с.

10. Amanda White. Effective WPF: Building Robust, Maintainable Windows Applications. CRC Press, 2023 . – 310 с.

11. Peter Goldsmith. Modern Desktop Application Development: Advanced Techniques for Building Powerful and User-Friendly Applications. Apress, 2023 . – 400 с.

12. Nicholas Makarov. Data Structures and Algorithms for Desktop Applications. Packt Publishing, 2022 . – 240 с.

13. Emily Jones. Mathematical Models in Software Engineering Applications. Springer, 2023 . – 360 с.

14. David Peng. Computer Graphics Techniques for the Modern Developer. O'Рейлі, 2022 . – 320 с.

15. Amanda Harris. Data Visualization for Desktop Applications. CRC Press, 2022 . – 280 с.

16. Must-Have Inventory Management System Features // NetSuite. – [Електронний ресурс]. – Режим доступу: https://www.netsuite.com/portal/resource/articles/inventory-management/features.shtml (дата звернення: 17.03.2025).

17. Microsoft Dynamics 365 Commerce. – [Електронний ресурс]. – Режим доступу: https://dynamics.microsoft.com/en-us/dynamics365/commerce/ (дата звернення: 17.03.2025).

18. SAP Business One. – [Електронний ресурс]. – Режим доступу: https://www.sap.com/products/erp/business-one.html (дата звернення: 17.03.2025).

19. Odoo. – [Електронний ресурс]. – Режим доступу: https://www.odoo.com/ (дата звернення: 17.03.2025).

20. QuickBooks Commerce. – [Електронний ресурс]. – Режим доступу: https://quickbooks.intuit.com/commerce/ (дата звернення: 17.03.2025).

21. Fishbowl Inventory. – [Електронний ресурс]. – Режим доступу: https://www.fishbowlinventory.com/ (дата звернення: 17.03.2025).

ДОДАТКИ

Додаток А. Реалізація UI головної сторінки системи

```
<Application.Resources>
    <ResourceDictionary>
        <DataTemplate DataType="{x:Type viewModels:LoginViewModel}">
            <views:Login />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:CreateOrEditItemViewModel}">
            <views:CreateOrEditItem  />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ManageItemsViewModel}">
            <views:ManageItems />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:HomeScreenViewModel}">
            <views:HomeScreen />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ScanItemsViewModel}">
            <views:ScanItems />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ScanAndPurchaseViewModel}">
            <views:ScanAndPurchase />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:FinalizePurchaseViewModel}">
            <views:FinalizePurchase />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:GenerateBarcodesViewModel}">
            <views:GenerateBarcodes />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:AdjustQuantityViewModel}">
            <views:AdjustQuantity />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ViewReportsViewModel}">
            <views:ViewReports />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ViewItemTypesViewModel}">
            <views:ViewItemTypes />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:CreateOrEditItemTypeViewModel}">
            <views:CreateOrEditItemType  />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ViewCurrenciesViewModel}">
            <views:ViewCurrencies />
```

```
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:CreateOrEditCurrencyViewModel}">
            <views:CreateOrEditCurrency />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ViewQuantityAdjustmentsViewModel}">
            <views:ViewQuantityAdjustments  />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ViewItemSoldInfoViewModel}">
            <views:ViewItemSoldInfo />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ManageUsersViewModel}">
            <views:ManageUsers />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:CreateOrEditUserViewModel}">
            <views:CreateOrEditUser />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ChangePasswordViewModel}">
            <views:ChangePassword />
        </DataTemplate>
        <DataTemplate DataType="{x:Type viewModels:ManageAppSettingsViewModel}">
            <views:ManageAppSettings />
        </DataTemplate>
        <Style TargetType="{x:Type mahapps:WindowButtonCommands}"
               BasedOn="{StaticResource MahApps.Styles.WindowButtonCommands.Win10}" />
        <Style TargetType="{x:Type Button}"  BasedOn="{StaticResource MahApps.Styles.Button}">
            <Setter Property="FontSize"Value="14" />
            <Setter   Property="mahapps:ControlsHelper.ContentCharacterCasing"Value="Normal"  />
        </Style>
        <Style x:Key="NoCapsColumnHeader"BasedOn="{StaticResource MahApps.Styles.DataGridColumnHeader}"
               TargetType="{x:Type DataGridColumnHeader}">
            <Setter   Property="mahapps:ControlsHelper.ContentCharacterCasing"Value="Normal"  />
        </Style>
        <ResourceDictionary.MergedDictionaries>
            <ResourceDictionary   Source="pack://application:,,,/MahApps.Metro;component/Styles/Controls.xaml"  />
            <ResourceDictionary   Source="pack://application:,,,/MahApps.Metro;component/Styles/Fonts.xaml"  />
            <ResourceDictionary   Source="pack://application:,,,/MahApps.Metro;component/Styles/Themes/Light.Blue.xaml"  />
        </ResourceDictionary.MergedDictionaries>
    </ResourceDictionary>
</Application.Resources>
```

Додаток Б. Програмний код для створення бази даних SQLite

```
private void CreateDatabase(){
  if (!Directory.Exists(_directory)) {Directory.CreateDirectory(_directory);}
    SQLiteConnection.CreateFile(GetFilePath());
    using (var conn = GetDatabaseConnectionWithoutMigrating()) {
        using (var command = GetSQLiteCommand(conn)){
            string createUsersTable = "CREATE TABLE Users (" +
                "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +
"Name TEXT," +"Username TEXT," +  "PasswordHash TEXT)";
            command.CommandText = createUsersTable;
            command.ExecuteNonQuery();
            string createCurrenciesTable = "CREATE TABLE Currencies (" +
                "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +
                "Name TEXT," +"Abbreviation TEXT," +"Symbol TEXT," +
                "ConversionRateToUSD TEXT," +"IsDefaultCurrency INTEGER DEFAULT 0)";
            command.CommandText = createCurrenciesTable;
            command.ExecuteNonQuery();
            string createInventoryItemTable = "CREATE TABLE InventoryItems (" +
                "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +  "Name TEXT," +"Description TEXT," +
                "PicturePath TEXT," +  "Cost TEXT," +  "CostCurrencyID INTEGER REFERENCES Currencies(ID)," +
                "ProfitPerItem TEXT," +  "ProfitPerItemCurrencyID INTEGER REFERENCES Currencies(ID)," +
                "Quantity INTEGER," +"BarcodeNumber TEXT," +  "WasDeleted INTEGER DEFAULT 0," +
                "CreatedByUserID INTEGER REFERENCES Users(ID))";
            command.CommandText = createInventoryItemTable;
            command.ExecuteNonQuery();
            string createItemSoldInfoTable = "CREATE TABLE ItemsSoldInfo (" +
                "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +  "DateTimeSold TEXT," +
                "QuantitySold INTEGER DEFAULT 1," +"Cost TEXT," +"CostCurrencyID INTEGER REFERENCES Currencies(ID)," +
                "Paid TEXT," +"PaidCurrencyID INTEGER REFERENCES Currencies(ID)," +"Change TEXT," +
                "ChangeCurrencyID INTEGER REFERENCES Currencies(ID)," +  "ProfitPerItem TEXT," +
                "ProfitPerItemCurrencyID INTEGER REFERENCES Currencies(ID)," +
"InventoryItemID INTEGER REFERENCES InventoryItems(ID)," +"SoldByUserID INTEGER REFERENCES Users(ID) )";
            command.CommandText = createItemSoldInfoTable;
            command.ExecuteNonQuery();
            string createGeneratedBarcodesTable = "CREATE TABLE GeneratedBarcodes (" +
                "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +  "Number INTEGER," +
                "DateTimeGenerated TEXT," +"GeneratedByUserID INTEGER REFERENCES Users(ID) )";
            command.CommandText = createGeneratedBarcodesTable;
            command.ExecuteNonQuery();
            string addInitialUser = "" +
                "INSERT INTO Users (Name, Username, PasswordHash) VALUES (@name, @username, @passwordHash)";
            command.CommandText = addInitialUser;
            command.Parameters.Clear();
            command.Parameters.AddWithValue("@name", "Administrator");
```

```
command.Parameters.AddWithValue("@username", "admin");
command.Parameters.AddWithValue("@passwordHash", User.HashPassword("changeme"));
command.ExecuteNonQuery();
string addCurrency = "" +
    "INSERT INTO Currencies (Name, Abbreviation, Symbol, ConversionRateToUSD, IsDefaultCurrency) " +
    "VALUES (@name, @abbreviation, @symbol, @conversion, @isDefault)";
command.CommandText = addCurrency;
command.Parameters.Clear();
command.Parameters.AddWithValue("@name", "US Dollar");
command.Parameters.AddWithValue("@abbreviation", "USD");
command.Parameters.AddWithValue("@symbol", "$");
command.Parameters.AddWithValue("@conversion", "1.0");
command.Parameters.AddWithValue("@isDefault", false);
command.ExecuteNonQuery();
command.Parameters.Clear();
command.Parameters.AddWithValue("@name", "Cambodian Riel");
command.Parameters.AddWithValue("@isDefault", true);
command.ExecuteNonQuery();
command.CommandText = "PRAGMA user_version = 0";
command.Parameters.Clear();
command.ExecuteNonQuery();
conn.Close();  }}}
```

Додаток В. Реалізація програмного коду для міграції бази даних SQLite

```
private void PerformMigrationsAsNecessary(SQLiteCommand command){
    command.CommandText = "PRAGMA user_version";
    using (var reader = command.ExecuteReader()){
        if (reader.Read()){
            var userVersion = reader.GetInt32(0);
            reader.Close();commands
            switch (userVersion + 1)  {
                case 1:
                    string createQuantityAdjustmentsTable = "CREATE TABLE QuantityAdjustments (" +
                        "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +"AmountChanged TEXT," +
                        "DateTimeChanged TEXT," +  "InventoryItemID INTEGER REFERENCES InventoryItems(ID)," +
                        "AdjustedByUserID INTEGER REFERENCES Users(ID))";
                    command.CommandText = createQuantityAdjustmentsTable;
                    command.ExecuteNonQuery();
                    command.CommandText = "PRAGMA user_version = 1;";
                    command.ExecuteNonQuery();
                    command.Parameters.Clear();
                    goto case 2;
                case 2:
                    string addIsDrinkColumn = "" +"ALTER TABLE InventoryItems " +
                       "ADD COLUMN IsDrink INTEGER DEFAULT 0;";
                    command.CommandText = addIsDrinkColumn;
                    command.ExecuteNonQuery();
                    command.CommandText = "PRAGMA user_version = 2;";
                    command.ExecuteNonQuery();
                    command.Parameters.Clear();
                    goto case 3;
                case 3:
                    command.CommandText = "PRAGMA foreign_keys = 0";
                    command.ExecuteNonQuery();
                    command.CommandText = "BEGIN TRANSACTION;";
                    command.ExecuteNonQuery();
                    string addItemTypesTable = "" +"CREATE TABLE ItemTypes (" +
                        "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +"Name TEXT," +"Description TEXT)";
                    command.CommandText = addItemTypesTable;
                    command.ExecuteNonQuery();
                    string addInitialItemTypes = "" +
                        "INSERT INTO ItemTypes (Name, Description) VALUES (\"School supplies\", \"Pencils, pens, etc.\")";
                    command.CommandText = addInitialItemTypes;
                    command.ExecuteNonQuery();
                    addInitialItemTypes = "" +
                        "INSERT INTO ItemTypes (Name, Description) VALUES (\"Drinks\", \"Water, milk, etc.\")";
                    command.CommandText = addInitialItemTypes;
```

```
    command.ExecuteNonQuery();
    addInitialItemTypes = "" +
       "INSERT INTO ItemTypes (Name, Description) VALUES (\"Meal tickets\", \"Tickets for student meals\")";
    command.CommandText = addInitialItemTypes;
    command.ExecuteNonQuery();
    string recreateInventoryItemTable = "CREATE TABLE New_InventoryItems (" +
       "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +"Name TEXT," +    "Description TEXT," +
       "PicturePath TEXT," +"Cost TEXT," +"CostCurrencyID INTEGER REFERENCES Currencies(ID)," +
       "ProfitPerItem TEXT," +"ProfitPerItemCurrencyID INTEGER REFERENCES Currencies(ID)," +
       "Quantity INTEGER," +"BarcodeNumber TEXT," +"WasDeleted INTEGER DEFAULT 0," +
       "CreatedByUserID INTEGER REFERENCES Users(ID)," +
      "ItemTypeID INTEGER REFERENCES ItemTypes(ID))";
    command.CommandText = recreateInventoryItemTable;
    command.ExecuteNonQuery();
    string moveInventoryData = "" +
       "INSERT INTO New_InventoryItems (Name, Description, PicturePath, " +
       "Cost, CostCurrencyID, ProfitPerItem, ProfitPerItemCurrencyID," +
       "Quantity, BarcodeNumber, WasDeleted, CreatedByUserID, ItemTypeID) " +
       "SELECT Name, Description, PicturePath, " +
       "Cost, CostCurrencyID, ProfitPerItem, ProfitPerItemCurrencyID," +
       "Quantity, BarcodeNumber, WasDeleted, CreatedByUserID, 1 " +
       "FROM InventoryItems " +"ORDER BY ID;";
    command.CommandText = moveInventoryData;
    command.ExecuteNonQuery();
    string removeOldTable = "" +"DROP TABLE InventoryItems;";
    command.CommandText = removeOldTable;
    command.ExecuteNonQuery();
    string renameNewTable = "" +"ALTER TABLE New_InventoryItems RENAME TO InventoryItems;";
    command.CommandText = renameNewTable;
    command.ExecuteNonQuery();
    command.CommandText = "PRAGMA user_version = 3;";
    command.ExecuteNonQuery();
    command.Parameters.Clear();
    command.CommandText = "COMMIT TRANSACTION;";
    command.ExecuteNonQuery();
    command.CommandText = "PRAGMA foreign_keys = 1";
    command.ExecuteNonQuery();
    command.CommandText = "VACUUM;";
    command.ExecuteNonQuery();
    goto case 4;
case 4:
    string addIsDefaultColumn = "" +"ALTER TABLE ItemTypes " +
       "ADD COLUMN IsDefault INTEGER DEFAULT 0;";
    command.CommandText = addIsDefaultColumn;
    command.ExecuteNonQuery();
    command.CommandText = " UPDATE ItemTypes SET IsDefault = 1 WHERE ID = 1";
```

```
                    command.ExecuteNonQuery();
                    command.CommandText = "PRAGMA user_version = 4;";
                    command.ExecuteNonQuery();
                    command.Parameters.Clear();
                    goto case 5;
                case 5:
                    command.CommandText = "PRAGMA foreign_keys = 0";
                    command.ExecuteNonQuery();
                    string recreateQuantityAdjustmentsTable = "CREATE TABLE New_QuantityAdjustments (" +
                        "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +  "AmountChanged INTEGER," +
                        "DateTimeChanged TEXT," +"InventoryItemID INTEGER REFERENCES InventoryItems(ID)," +
                        "AdjustedByUserID INTEGER REFERENCES Users(ID))";
                    command.CommandText = recreateQuantityAdjustmentsTable;
                    command.ExecuteNonQuery();
                    string moveQuantityAdjustmentData = "" +
                        "INSERT INTO New_QuantityAdjustments (AmountChanged, DateTimeChanged, InventoryItemID,
AdjustedByUserID) " +
                        "SELECT AmountChanged, DateTimeChanged, InventoryItemID, AdjustedByUserID " +
                        "FROM QuantityAdjustments " +"ORDER BY ID;";
                    command.CommandText = moveQuantityAdjustmentData;
                    command.ExecuteNonQuery();
                    string removeOldQuantityAdjustmentTable = "" +
                        "DROP TABLE QuantityAdjustments;";
                    command.CommandText = removeOldQuantityAdjustmentTable;
                    command.ExecuteNonQuery();
                    string renameNewQuantityAdjustmentsTable = "" +
                        "ALTER TABLE New_QuantityAdjustments RENAME TO QuantityAdjustments;";
                    command.CommandText = renameNewQuantityAdjustmentsTable;
                    command.ExecuteNonQuery();
                    command.CommandText = "PRAGMA user_version = 5;";
                    command.ExecuteNonQuery();
                    command.Parameters.Clear();
                    command.CommandText = "PRAGMA foreign_keys = 1";
                    command.ExecuteNonQuery();
                    goto case 6;
                case 6:
                    string addUserPermissionsTable = "" +"CREATE TABLE UserPermissions (" +
                            "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +"CanAddEditItems INTEGER," +
                        "CanAdjustItemQuantity INTEGER," +"CanViewDetailedItemQuantityAdjustments INTEGER," +
                            "CanScanItems INTEGER," +  "CanGenerateBarcodes INTEGER," +"CanViewReports INTEGER," +
                            "CanViewDetailedItemSoldInfo INTEGER," +"CanSaveReportsToPDF INTEGER," +
                            "CanDeleteItemsFromInventory INTEGER," +  "CanManageItemCategories INTEGER," +
                            "UserID INTEGER REFERENCES Users(ID))";
                    command.CommandText = addUserPermissionsTable;
                    command.ExecuteNonQuery();
                    string addDefaultPermissions = "" +
```

```
                    "INSERT INTO UserPermissions (CanAddEditItems, CanAdjustItemQuantity, " +
                    "CanViewDetailedItemQuantityAdjustments, CanScanItems, CanGenerateBarcodes, CanViewReports," +
                    "CanViewDetailedItemSoldInfo, CanSaveReportsToPDF, CanDeleteItemsFromInventory,
CanManageItemCategories," +"UserID) " +  "VALUES (1, 1, 1, 1, 1, 1, 1, 1, 1, 1)";
                command.CommandText = addDefaultPermissions;
                command.ExecuteNonQuery();
                command.CommandText = "PRAGMA user_version = 6;";
                command.ExecuteNonQuery();
                command.Parameters.Clear();
                goto case 7;
            case 7:
                string addCanManageUsersColumn = "" +
                    "ALTER TABLE UserPermissions " +"ADD COLUMN CanManageUsers INTEGER DEFAULT 1;";
                command.CommandText = addCanManageUsersColumn;
                command.ExecuteNonQuery();
                command.CommandText = "PRAGMA user_version = 7;";
                command.ExecuteNonQuery();
                command.Parameters.Clear();
                goto case 8;
            case 8:
                string addWasDeletedUsersColumn = "" +
                    "ALTER TABLE Users " +  "ADD COLUMN WasDeleted INTEGER DEFAULT 0;";
                command.CommandText = addWasDeletedUsersColumn;
                command.ExecuteNonQuery();
                command.CommandText = "PRAGMA user_version = 8;";
                command.ExecuteNonQuery();
                command.Parameters.Clear();
                goto case 9;
            case 9:
                string addCanDeleteItemsSold = "" +
                    "ALTER TABLE UserPermissions " +  "ADD COLUMN CanDeleteItemsSold INTEGER DEFAULT 1;";
                command.CommandText = addCanDeleteItemsSold;
                command.ExecuteNonQuery();
                command.CommandText = "PRAGMA user_version = 9;";
                command.ExecuteNonQuery();
                command.Parameters.Clear();
                goto case 10;
            case 10:
                string addExplanationColumn = "" +
                    "ALTER TABLE QuantityAdjustments " +   "ADD COLUMN Explanation TEXT DEFAULT ";";
                command.CommandText = addExplanationColumn;
                command.ExecuteNonQuery();
                command.CommandText = "PRAGMA user_version = 10;";
                command.ExecuteNonQuery();
                command.Parameters.Clear();
                goto case 11;
```

```
                case 11:
                    string addCanViewManageInventoryQuantityColumn = "" +
                        "ALTER TABLE UserPermissions " +  "ADD COLUMN CanViewManageInventoryQuantity INTEGER
DEFAULT 1;";
                    command.CommandText = addCanViewManageInventoryQuantityColumn;
                    command.ExecuteNonQuery();
                    command.CommandText = "PRAGMA user_version = 11;";
                    command.ExecuteNonQuery();
                    command.Parameters.Clear();
                    goto case 12;
                case 12:
                    string addWasAdjustedForStockPurchase = "" +
                        "ALTER TABLE QuantityAdjustments " +"ADD COLUMN WasAdjustedForStockPurchase INTEGER DEFAULT
0;";
                    command.CommandText = addWasAdjustedForStockPurchase;
                    command.ExecuteNonQuery();
                    command.CommandText = "PRAGMA user_version = 12;";
                    command.ExecuteNonQuery();
                    command.Parameters.Clear();
                    goto case 13;
                case 13:
                    string addPurchaseCostAndItemsPerPurchase = "" +  "ALTER TABLE InventoryItems " +
                        "ADD COLUMN ItemPurchaseCost TEXT DEFAULT '0'; " +  "ALTER TABLE InventoryItems " +
                        "ADD COLUMN ItemPurchaseCostCurrencyID INTEGER DEFAULT 0;" +"ALTER TABLE InventoryItems " +
                        "ADD COLUMN ItemsPerPurchase INTEGER DEFAULT 0";
                    command.CommandText = addPurchaseCostAndItemsPerPurchase;
                    command.ExecuteNonQuery();
                    command.CommandText = "PRAGMA user_version = 13;";
                    command.ExecuteNonQuery();
                    command.Parameters.Clear();
                    goto case 14;
                case 14:
                    string addCanEditAppSettings = "" +
                        "ALTER TABLE UserPermissions " +"ADD COLUMN CanEditAppSettings INTEGER DEFAULT 0;";
                    command.CommandText = addCanEditAppSettings;
                    command.ExecuteNonQuery();
                    command.CommandText = "PRAGMA user_version = 14;";
                    command.ExecuteNonQuery();
                    command.Parameters.Clear();
                    goto case 15;
                case 15:
                    string addCanEditCurrencies = "" +
                        "ALTER TABLE UserPermissions " +  "ADD COLUMN CanManageCurrencies INTEGER DEFAULT 0;";
                    command.CommandText = addCanEditCurrencies;
                    command.ExecuteNonQuery();
                    command.CommandText = "PRAGMA user_version = 15;";
```

```
                command.ExecuteNonQuery();
                command.Parameters.Clear();
                goto case 16;
            case 16:
                string addPurchases = "" +  "CREATE TABLE Purchases ("
+"ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +"DateTimePurchased TEXT," +"TotalCost TEXT," +
                  "Name TEXT," +"Phone TEXT," +  "Email TEXT," +"UserID INTEGER REFERENCES Users(ID))";
                command.CommandText = addPurchases;
                command.ExecuteNonQuery();
                string addPurchasedItems = "" +"CREATE TABLE PurchasedItems (" +
                    "ID INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT," +"Quantity INTEGER," +
                    "Name TEXT,"+"Type TEXT," +  "Cost TEXT," +"CostCurrencySymbol TEXT," +
                    "CostCurrencyConversionRate TEXT," +"Profit TEXT," +   "ProfitCurrencySymbol TEXT," +
                    "ProfitCurrencyConversionRate TEXT," +"PurchaseID INTEGER REFERENCES Purchases(ID))";
                command.CommandText = addPurchasedItems;
                command.ExecuteNonQuery();
                command.CommandText = "PRAGMA user_version = 16;";
                command.ExecuteNonQuery();
                command.Parameters.Clear();
                goto case 17;
            case 17:
                string addPurchaseColumns = "" +"ALTER TABLE Purchases " +
                  "ADD COLUMN CostCurrencySymbol TEXT DEFAULT '$'; " +"ALTER TABLE Purchases " +
                  "ADD COLUMN CostCurrencyConversionRate TEXT DEFAULT '1';";
                command.CommandText = addPurchaseColumns;
                command.ExecuteNonQuery();
                command.CommandText = "PRAGMA user_version = 17;";
                command.ExecuteNonQuery();
                command.Parameters.Clear();
                break;
            case 18:
                string addMorePurchaseColumns = "" +"ALTER TABLE PurchasedItems " +
                  "ADD COLUMN InventoryItemID INTEGER REFERENCES InventoryItems(ID); " +
                  "ALTER TABLE Purchases " +  "ADD COLUMN ChangeCurrencySymbol TEXT DEFAULT '$';" +
                  "ALTER TABLE Purchases " + "ADD COLUMN ChangeCurrencyConversionRate TEXT DEFAULT '1';";
                command.CommandText = addMorePurchaseColumns;
                command.ExecuteNonQuery();
                command.CommandText = "PRAGMA user_version = 18;";
                command.ExecuteNonQuery();
                command.Parameters.Clear();
                break;
            case 19:
                string addPurchaseMethodColumn = "" +  "ALTER TABLE Purchases " +
                  "ADD COLUMN PurchaseMethod INTEGER DEFAULT 1; " +  "ALTER TABLE ItemsSoldInfo " +
                  "ADD COLUMN PurchaseMethod INTEGER DEFAULT 1; ";
                command.CommandText = addPurchaseMethodColumn;
```

```
            command.ExecuteNonQuery();
            command.CommandText = "PRAGMA user_version = 19;";
            command.ExecuteNonQuery();
            command.Parameters.Clear();
            break;  }   }
else{   reader.Close();}  }  }
```

Додаток Г. Реалізація програмного коду для створення звіту у форматі Excel з детальною інформацією про запаси товарів за певний період часу

```
class StockInfoExcelGenerator{
    public void ExportStockInfo(List<DetailedStockReportInfo> items, DateTime startDate, DateTime endDate, string path){
        items.Sort((a, b) => (a.Item.Name + a.Item.Description).ToLower().CompareTo((b.Item.Name + b.Item.Description).ToLower()));
        var startDateString = startDate.ToString(Utilities.DateTimeToFriendlyFullDateTimeStringFormat());
        var endDateString = endDate.ToString(Utilities.DateTimeToFriendlyFullDateTimeStringFormat());
        using (var workbook = new XLWorkbook()){
            var worksheet = workbook.Worksheets.Add("Stock Info");
            worksheet.Cell("A1").Value = "SimpleInventory -- Stock Info Report for Sold Items";
            worksheet.Cell("A1").Style.Font.Bold = true;
            worksheet.Cell("A2").Value = startDateString + " - " + endDateString;
            worksheet.Cell("A4").SetValue("Name").Style.Font.SetBold(true);
            worksheet.Cell("B4").SetValue("Description").Style.Font.SetBold(true);
            worksheet.Cell("C4").SetValue("Beginning Stock (Computer)").Style.Font.SetBold(true);
            worksheet.Cell("D4").SetValue("Ending Stock (Computer)").Style.Font.SetBold(true);
            worksheet.Cell("E4").SetValue("Ending Stock (Manual Entry)").Style.Font.SetBold(true);
            worksheet.Cell("F4").SetValue("Computer Difference").Style.Font.SetBold(true);
            worksheet.Cell("G4").SetValue("Manual Difference").Style.Font.SetBold(true);
            worksheet.Cell("H4").SetValue("Stock Difference").Style.Font.SetBold(true);
            worksheet.Cell("I4").SetValue("Item Cost").Style.Font.SetBold(true);
            worksheet.Cell("J4").SetValue("Cost Difference (Missing Items)").Style.Font.SetBold(true);
            worksheet.Cell("K4").SetValue("Cost Difference (Extra Items)").Style.Font.SetBold(true);
            var currentCell = worksheet.Cell("A5");
            var lastRow = currentCell.WorksheetRow();
            IXLCell firstCellWithData = null;
            foreach (DetailedStockReportInfo item in items)  {
                lastRow = currentCell.WorksheetRow();
                if (firstCellWithData == null){firstCellWithData = currentCell; }
                currentCell.Value = item.Item.Name;
                currentCell.CellRight(1).Value = item.Item.Description;
                currentCell.CellRight(2).Value = item.StartStockWithPurchaseStockIncrease;
                currentCell.CellRight(3).Value = item.EndStock; // computer
                currentCell.CellRight(4).Value = "";
                currentCell.CellRight(4).AddConditionalFormat().WhenEquals("\"\"")
                    .Fill.SetBackgroundColor(XLColor.Yellow);
                currentCell.CellRight(5).FormulaA1 = "=SUM(-" + currentCell.CellRight(2).Address.ToStringFixed() + ","
                    + currentCell.CellRight(3).Address.ToStringFixed() + ")";
                currentCell.CellRight(6).FormulaA1 = "=IF(" + currentCell.CellRight(3).Address.ToStringFixed() + "=\"\", \"-\", "
                        + "SUM(-" + currentCell.CellRight(2).Address.ToStringFixed() + ","
                            + currentCell.CellRight(4).Address.ToStringFixed() + "))";
                currentCell.CellRight(5).AddConditionalFormat()
                    .WhenNotEquals("=" + currentCell.CellRight(6).Address.ToStringFixed())
                    .Fill.SetBackgroundColor(XLColor.LightPink);
```

```
            currentCell.CellRight(6).AddConditionalFormat()
               .WhenNotEquals("=" + currentCell.CellRight(5).Address.ToStringFixed())
               .Fill.SetBackgroundColor(XLColor.LightPink);
            currentCell.CellRight(7).SetFormulaA1("=ABS(SUM(" + currentCell.CellRight(5).Address.ToStringFixed() + ", -"
               + currentCell.CellRight(6).Address.ToStringFixed() + "))").AddConditionalFormat().WhenNotEquals("0")
               .Fill.SetBackgroundColor(XLColor.LightPink);
            currentCell.CellRight(8).Value = item.Item.Cost;
            currentCell.CellRight(9).SetFormulaA1(formula);
            formula = string.Format("=IF({0}<>\"\", IF({1}<{2},{3}*{4},\"\"), \"\")",
               currentCell.CellRight(4).Address.ToStringFixed(),
               currentCell.CellRight(3).Address.ToStringFixed(),
               currentCell.CellRight(4).Address.ToStringFixed(),
               currentCell.CellRight(7).Address.ToStringFixed(),
               currentCell.CellRight(8).Address.ToStringFixed()  );
            currentCell.CellRight(10).SetFormulaA1(formula);
            if (currentCell.WorksheetRow().RowNumber() % 2 == 0){
               currentCell.WorksheetRow().Style.Fill.BackgroundColor = XLColor.LightGray;}
            currentCell = currentCell.CellBelow();}
         if (items.Count > 0){
            currentCell.CellRight(8).SetValue("Cost Discrepency").Style.Font.SetBold(true);
            currentCell.CellRight(9).SetFormulaA1("=SUM(" + firstCellWithData.CellRight(9).Address.ToStringFixed()
               + ":" + currentCell.CellAbove(1).CellRight(9).Address.ToStringFixed() + ")").Style.Font.SetBold(true);
            currentCell.CellRight(10).SetFormulaA1("=SUM(" + firstCellWithData.CellRight(10).Address.ToStringFixed()
               + ":" + currentCell.CellAbove(1).CellRight(10).Address.ToStringFixed() + ")").Style.Font.SetBold(true);   }
         worksheet.Columns().AdjustToContents(4, 4, 10, 25);
         worksheet.PageSetup.PrintAreas.Clear();
         var firstCellForPrinting = worksheet.Cell("A1");
         var lastCellForPrinting = items.Count > 0 ? currentCell.CellRight(10) : worksheet.Cell("J4");
         worksheet.PageSetup.PrintAreas.Add(firstCellForPrinting.Address.ToStringRelative()  +  ":"  +
lastCellForPrinting.Address.ToStringRelative());
         worksheet.PageSetup.SetRowsToRepeatAtTop("4:4");
         worksheet.PageSetup.PagesWide = 1;
         worksheet.PageSetup.PageOrientation = XLPageOrientation.Landscape;
         workbook.SaveAs(path);
         Process.Start(path);}}}
```

Додаток Д. Реалізація генерації PDF файліз зі штрихкодами

```
class BarcodePDFGenerator{
    public BarcodePDFGenerator(){
        IsDryRun = false;
        BarcodeType = BarcodeLib.TYPE.CODE128;
        PageSize = PdfSharp.PageSize.A4;
        NumberOfPages = 1;}
    private BitmapSource ConvertImageToBitmapImage(Image img){
        using (var memory = new MemoryStream()){
            img.Save(memory, ImageFormat.Jpeg);
            memory.Position = 0;
            var bitmapImage = new BitmapImage();
            bitmapImage.BeginInit();
            bitmapImage.StreamSource = memory;
            bitmapImage.CacheOption = BitmapCacheOption.OnLoad;
            bitmapImage.EndInit();
            return bitmapImage;}}
    private Bitmap ResizeImage(Image image, int width, int height, int resolution = 96){
        var destRect = new Rectangle(0, 0, width, height);
        var destImage = new Bitmap(width, height);
        using (var graphics = Graphics.FromImage(destImage)){
            graphics.CompositingMode = System.Drawing.Drawing2D.CompositingMode.SourceCopy;
            graphics.CompositingQuality = System.Drawing.Drawing2D.CompositingQuality.HighQuality;
            graphics.InterpolationMode = System.Drawing.Drawing2D.InterpolationMode.HighQualityBicubic;
            graphics.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;
            graphics.PixelOffsetMode = System.Drawing.Drawing2D.PixelOffsetMode.HighQuality;
            using (var wrapMode = new ImageAttributes()){
                wrapMode.SetWrapMode(System.Drawing.Drawing2D.WrapMode.TileFlipXY);
                graphics.DrawImage(image, destRect, 0, 0, image.Width, image.Height, GraphicsUnit.Pixel, wrapMode);}}
        destImage.SetResolution(resolution, resolution);
        return destImage;}
    public bool IsDryRun { get; set; }
    public BarcodeLib.TYPE BarcodeType { get; set; }
    public PdfSharp.PageSize PageSize { get; set; }
    public int NumberOfPages { get; set; }
    public int GenerateBarcodes(string outputPath){
        if (NumberOfPages > 0){
            PdfDocument document = new PdfDocument();
            document.Info.Title = "Inventory Barcodes";
            long barcodeToUse = GeneratedBarcode.GetLatestBarcodeNumber() + 1;
            var barcodesGenerated = new List<long>();
            for (int i = 0; i < NumberOfPages; i++){
                PdfPage page = document.AddPage();
                page.Size = PageSize;
```

```
            XGraphics gfx = XGraphics.FromPdfPage(page);
            XFont font = new XFont("Verdana", 20, XFontStyle.Bold);
            XUnit yCoord = XUnit.FromInch(1);
            gfx.DrawString("Inventory Barcodes", font, XBrushes.Black,
               new XRect(0, yCoord, page.Width, page.Height), XStringFormats.TopCenter);
            yCoord += XUnit.FromInch(0.7);
            var barcodeCreator = new BarcodeLib.Barcode();
            barcodeCreator.ImageFormat = ImageFormat.Jpeg;
            barcodeCreator.IncludeLabel = false;
            barcodeCreator.Alignment = BarcodeLib.AlignmentPositions.CENTER;
            bool isPageFull = false;
            XUnit imageHeight = XUnit.FromPoint(60);
            while (!isPageFull){
               var isWidthFull = false;
               XUnit xCoord = XUnit.FromInch(1);
               while (!isWidthFull){
                  var image = barcodeCreator.Encode(BarcodeType, barcodeToUse.ToString());
                  if (image != null){
                     double ratioTo192 = (192 / image.VerticalResolution);
                     int resizeHeight = (int)(image.Height / ratioTo192);
                     int resizeWidth = (int)(image.Width / ratioTo192);
                     image = ResizeImage(image, resizeWidth, resizeHeight, (int)image.VerticalResolution);
                     XImage pdfImage = XImage.FromBitmapSource(ConvertImageToBitmapImage(image));
                     gfx.DrawImage(pdfImage, xCoord, yCoord);
                     XFont barcodeFont = new XFont("Verdana", 16, XFontStyle.Bold);
                     gfx.DrawString(barcodeToUse.ToString(), barcodeFont, XBrushes.Black,
                        new XRect(xCoord, yCoord + pdfImage.PointHeight + 2, pdfImage.PointWidth, 150), XStringFormats.TopCenter);
                     xCoord += XUnit.FromPoint(pdfImage.PointWidth);
                     imageHeight = XUnit.FromPoint(pdfImage.PointHeight);
                     XUnit spaceBetweenBarcodes = XUnit.FromInch(0.75);
                     if (xCoord + XUnit.FromPoint(pdfImage.PointWidth) + spaceBetweenBarcodes > page.Width - XUnit.FromInch(1)){
isWidthFull = true;  }
                     barcodesGenerated.Add(barcodeToUse);
                     barcodeToUse++;
                     xCoord += spaceBetweenBarcodes;}
                  else{  isWidthFull = true;  isPageFull = true;  break;  }}
               yCoord += imageHeight;
               yCoord += XUnit.FromInch(0.7);
               if (yCoord + imageHeight > page.Height - XUnit.FromInch(1)){  isPageFull = true;}}}
         if (!IsDryRun){
            GeneratedBarcode.AddGeneratedCodes(barcodesGenerated, DateTime.Now, 1);
            document.Save(outputPath);
            Process.Start(outputPath);}
         return barcodesGenerated.Count;}  return 0;}}
```