

Детектор Плагіату v. 2961 - Звіт оригінальності: 15.06.2026 9:14:13

Перевірений документ: Слинько.docx Ліцензія: ВОЛОДИМИР МАТІЄВСЬКИЙ

? Тип пошуку: Пошук Перепису ? Знайдено мову: Uk

? Тип перевірки: Інтернет-перевірка

ТЕЕ і кодування: DocX n/a

Детальний аналіз документа документа:

? Співвідношення:

Плагіат 0.95% Оригінал 95.59%
Цитати 3.46%

? Графік розподілу:



? Джерела плагіату: 8

- | | | |
|------|-----|--|
| 0.7% | 660 | 1. https://lingua.lnu.edu.ua/wp-content/uploads/2020/04/metodychni-rekomendatsii-z-napy... |
| 0.5% | 486 | 2. https://duikt.edu.ua/repositorii/Кафедра Інженерії програмного забезпечення/2022/Д... |
| 0.3% | 306 | 3. https://mif.cnu.edu.ua/wp-content/uploads/sites/23/2018/02/Tytulka_mag_rob-1.doc |

? Дані оброблених ресурсів: 197 - ОК / 12 - Помилка

? Важливі примітки:

Wikipedia:	Google Books:	Сервіси платних робіт:	Античіт:
			
[не знайдено]	[не знайдено]	[не знайдено]	Знайдена протидія!

? Звіт проти обману UACE:

- Статус: Аналізатор **Увімкнений** Нормалізатор **Увімкнений** схожість символів встановлено **100%**
- Виявлений відсоток забруднення UniCode: **13,2%** з обмеженням: 4%
- Відсоток невизнаних символів після нормалізації: **7,6%**
- Усі підозрілі символи будуть позначені фіолетовим кольором: **Abcd...**
- Знайдено невидимі символи: 0

Рекомендація з оцінки:

Аналізу цього звіту слід приділити особливу увагу! Підозрюється, що цей документ містить значну кількість символів, чужих мові документа. Це є прямим вказівкою на те, що автор документа використав спеціальне програмне забезпечення/онлайн -веб -сервіс, щоб ефективно заплутати текст, намагаючись уникнути потенційного виявлення плагіату. Наполегливо рекомендується ескалювати цю справу! У разі сумнівів зверніться до служби підтримки детектора плагіату!

Статистика алфавіту та аналіз символів:

 Активні посилання (URL-адреси, витягнуті з документа):

URL не знайдені

 Виключено:

URL не знайдені

 Включено:

URL не знайдені

 Детальний аналіз документу:

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ ДЗ

 Знайдено посилань: **0,01%** id: 1

«ЛУГАНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА»

 Знайдено плагіату: **0,04%** id: 2

<https://lingua.lnu.edu.ua/wp-content/uploads/2020/04/metod...> + 3 resyncis!
Факультет технологій та інформаційних систем (назва факультету, інституту)
Інформаційних технологій та систем (назва кафедри) Пояснювальна записка до
дипломного проекту (роботи) БАКАЛАВРА (освітньо-кваліфікаційний рівень) на тему:
СТВОРЕННЯ **WINDOWS FORMS** СИСТЕМИ ДЛЯ АВТОМАТИЗАЦІЇ ОБЛІКУ ТОВАРІВ НА СКЛАДІ
Виконав: студент 4 курсу напряму підготовки (спеціальності) 121

 Знайдено посилань: **0%** id: 3

«Інженерія програмного забезпечення»_

 Знайдено плагіату: **0,03%** id: 4

<https://lingua.lnu.edu.ua/wp-content/uploads/2020/04/metod...> + 3 resyncis!
(шифр і назва напряму підготовки, спеціальності) Слинко Д. В.
(прізвище та ініціали) Керівник __ Семенов М. А. (прізвище та ініціали) Рецензент
(прізвище та ініціали)

) Лубни 2026 ЗМІСТ ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ2 ВСТУП3 РОЗДІЛ 1 ТЕОРЕТИЧНІ ОСНОВИ
ТА АНАЛІЗ ДОСТУПНИХ РІШЕНЬ5 1.1. Сутність складського обліку та його роль в
підприємстві5 1.2. Огляд сучасних систем складського обліку8 1.3. Порівняльний аналіз
існуючих систем складського обліку12 1.4. Обґрунтування вибору технологій для
розробки14 1.5. Постановка задачі та формування вимог до системи17 1.6. Вимоги до
програмної системи автоматизації складського обліку20 1.7. Моделі даних та їх роль у
інформаційних системах22 Висновки до розділу24 РОЗДІЛ 2 ПРОЄКТУВАННЯ І РОЗРОБКА
СИСТЕМИ СКЛАДСЬКОГО ОБЛІКУ26 2.1. Загальна архітектура програмного забезпечення26
2.2. Розробка структури бази даних30 2.3. Реалізація основних модулів системи32 2.4.
Реалізація інтерфейсу користувача35 2.5. Реалізація обліку руху товарів38 2.6. Тестування
програмного забезпечення41 2.7. Порівняння з аналогами та переваги розробленої
системи44 2.8. Аналіз продуктивності та ефективності роботи системи46 2.9. Перспективи
розвитку та масштабування системи49 Висновки до розділу52 ВИСНОВКИ54 СПИСОК
ВИКОРИСТАНИХ ДЖЕРЕЛ56 Додаток А. Блок-схема роботи програми.59 Додаток Б. **UML**
Діаграма класів.60 Додаток В. Лістинг класу **StockMovementService.cs**61 Додаток Г. Лістинг
класу **ProductService.cs**64 Додаток Д. Лістинг класу **AuthService.cs**67 Додаток Е. Лістинг
класу **Database.cs**69 Додаток Є. Лістинг класу **MainForm.cs**70 Додаток Ж. Лістинг класу
IncomingForm.cs72 ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ **ERP** – **Enterprise Resource Planning** **CRM** –
Customer Relationship Management **IT** – **Information Technology** **UI** – **User Interface** **БД** – База
Даних **UID** – **User Identifier** **ПЗ** – Програмне Забезпечення **BAS** – **Business Automation Software**
РНБО – Рада національної безпеки і оборони України ВСТУП Актуальність роботи. У сучасних
умовах розвитку інформаційних технологій автоматизація бізнес-процесів є невід'ємною
частиною ефективного функціонування підприємств. Одним з ключових напрямків роботи,
необхідних для впровадження інформаційної системи, є складський облік. Від точності та
оперативності обліку товарів залежить ефективність управління запасами, терміни
поставки продукції та фінансові результати підприємства. Традиційні методи складського
обліку, такі як використання паперових документів або електронних таблиць, мають низку
суттєвих недоліків. До них належать більший ризик помилок, складність обробки великих
обсягів даних, низька швидкість доступу до інформації та відсутність централізованого
управління. У зв'язку з цим виникає потреба в розробці спеціалізованих програмних
рішень, які дозволять автоматизувати процеси обліку та управління товарами.
Актуальність роботи зумовлена зростаючими вимогами до ефективності управління
ресурсами підприємства, а також необхідністю забезпечення швидкого доступу до
достовірної інформації про стан складських запасів. Впровадження інформаційних систем
не тільки дозволяє зменшити кількість помилок, але й покращує бізнес-процеси, підвищує
продуктивність праці та покращує якість прийняття управлінських рішень. Метою
дипломної роботи є розробка системи **Windows Forms** для автоматизації обліку запасів на
складі, що забезпечує зручне управління товарами, контроль їх кількості та облік усіх

операцій переміщення. Об'єкт дослідження – є процес обліку товарів на складі підприємства. Тема дослідження – методи та інструменти розробки програмного забезпечення для автоматизації обліку запасів. Завдання – зібрати довідковий матеріал по темі, розробити схему класів та таблиць в базі даних, створити інтерфейс програми, написати логіку роботи системи, оцінити швидкодію створеної системи, провести тестування та виправити помилки. Під час роботи використовувалися засоби розробки програмного забезпечення, такі як мова програмування [C#](#), платформа [.NET](#) з технологією [Windows Forms](#) для створення користувацьких інтерфейсів та система управління базами даних [SQLite](#) для зберігання інформації в локальній базі даних. Практичне значення отриманих результатів полягає в можливості використання розробленої системи для автоматизації обліку запасів на малих та середніх підприємствах. Програмний продукт дозволяє ефективніше керувати запасами, зменшуючи кількість помилок та спрощуючи доступ до інформації. Структурно бакалаврська робота складається з двох розділів. У першому розділі розглядаються теоретичні основи обліку запасів та аналізуються існуючі рішення. У другій частині описується проектування та впровадження програмної системи, а також результати її тестування. РОЗДІЛ 1

ТЕОРЕТИЧНІ ОСНОВИ ТА АНАЛІЗ ДОСТУПНИХ РІШЕНЬ 1.1. Сутність складського обліку та його роль в підприємстві Складський облік є важливою частиною системи управління підприємством, оскільки він забезпечує контроль за рухом матеріальних ресурсів, їх зберіганням та ефективним використанням. У сучасних умовах ведення бізнесу, коли обсяг товарообігу постійно зростає та конкуренція загострюється, правильна організація обліку товарів на складі має першорядне значення. Складський облік являє собою систему безперервного кількісно-сортowego контролю за рухом запасів, що реалізується безпосередньо на місцях зберігання через заповнення відповідних карток обліку матеріалів [1]. Під складським обліком розуміється розроблена програма як система збору, реєстрації, обробки та зберігання інформації про надходження, рух та вибуття товарів. Головною метою цього виду обліку є надання достовірної та своєчасної інформації про наявність та стан запасів, що дозволяє керівництву приймати обґрунтовані рішення. Складський облік виконує багато важливих функцій. По-перше, він забезпечує контроль за рухом товарів, дозволяючи відстежувати всі операції, пов'язані з їх надходженням та використанням. Крім того, облік допомагає вести облік матеріальних цінностей, оскільки дозволяє виявляти нестачі, надлишки або інші відхилення. Ключову роль також відіграє інформаційне забезпечення, що надається керівництвом підприємства, яке дозволяє оцінювати ефективність використання ресурсів та планувати альтернативні варіанти.

Рис.1.1. Схема роботи доставки

” Знайдено посилань: 0%

id: 5

«НОВА ПОШТА».

Однією з ключових характеристик обліку на складі є його точність. Будь-яка бухгалтерська помилка може мати катастрофічні наслідки, такі як збій в процесі постачання, перевитрата коштів або втрачений прибуток. Саме тому сучасні підприємства все частіше переходять на автоматизовані системи, які мінімізують людський фактор. Функціонування складу базується на базових операціях обліку, що регламентують процеси переміщення та складування продукції. Належна виконавська дисципліна під час їх реалізації є гарантом точності інвентаризаційних показників, збереження майна та оптимізації операційної діяльності підприємства [2]. Організація складського обліку передбачає ведення інформації про кожну одиницю товару або групу товарів. Для цього використовуються різні характеристики, такі як найменування, коди, штрих-коди, одиниці вимірювання, ціни, кількості, а також дані про постачальників та категорії. Такий тип деталізації дозволяє отримати повну картину стану складу в будь-який момент часу. Рис.1.2. Зразок картки складського обліку. Картка складського обліку використовується як універсальний інструмент контролю за наявністю та рухом активів (матеріалів, готової продукції, продуктів харчування), крім специфічних груп товарів, таких як медикаменти, перев'язувальні матеріали та дорогоцінні метали для протезування [3]. В обліку руху товарів є особлива функція. Основні операції включають отримання товарів на склад, їх переміщення в межах підприємства та утилізацію при продажу або списанні. Кожна з цих операцій повинна бути задокументована, що дозволяє забезпечити чіткість та контроль над усіма процесами. У практичній діяльності підприємств складський облік пов'язаний з іншими формами обліку, особливо з бухгалтерським та управлінським. Дані про залишки товарів використовуються для створення фінансової звітності, аналізу витрат та

планування закупок. Таким чином, складський облік є важливою складовою загальної інформаційної системи підприємства. З розвитком інформаційних технологій відбулися значні зміни у способах ведення складського обліку. Якщо раніше акцент робився на паперових документах, то сьогодні програмні системи набувають все більшого поширення. Вони дозволяють автоматизувати багато операцій, значно підвищують швидкість обробки даних та забезпечують надійне зберігання. Системи автоматизації трансформують управління складськими силами, дозволяючи ефективно контролювати складні логістичні цикли та рух великої кількості товарних категорій [4]. Такі системи складського обліку забезпечують централізоване зберігання інформації, роблячи її доступною та ефективною для співробітників. Крім того, такі системи дозволяють реалізовувати додаткові функції, зокрема, формування звітів, аналіз руху товарів та моніторинг залишків у режимі реального часу. Також важливою є простота використання системи. Інтерфейс має бути інтуїтивно зрозумілим, щоб користувачі могли швидко виконувати необхідні операції без тривалого навчання. Це особливо важливо для малих та середніх підприємств, де співробітники можуть не отримувати спеціалізованої підготовки з інформаційних технологій. Таким чином, складський облік є важливою частиною діяльності будь-якого підприємства, що займається матеріальними ресурсами. Його тонка організація дозволяє забезпечити контроль руху товарів, підвищити точність даних та оптимізувати бізнес-процеси. Використання сучасних інформаційних систем дозволить створити високоякісну для даної бакалаврської роботи.

1.2 Огляд сучасних систем складського обліку

У сучасних умовах розвитку інформаційних технологій існує безліч програмних рішень, призначених для автоматизації складського обліку. Вони відрізняються функціональністю, складністю впровадження, вартістю та орієнтацією на різні категорії підприємств. Аналіз таких систем є важливим етапом перед розробкою власних програмних продуктів, оскільки дозволяє виявити їх сильні та слабкі сторони, а також сформулювати вимоги до майбутніх систем.

Enterprise Resource Planning (ERP) виступає інструментом оптимізації внутрішніх бізнес-процесів компанії: від складського обліку та постачання до управління проектами й фінансами. На відміну від орієнтованих на клієнтів **CRM**, **ERP**-системи фокусуються на внутрішній ефективності та автоматизації регламентованих операцій підприємства [5]. У сучасних системах складського обліку можна виділити як складні корпоративні рішення, так і прості програмні продукти, орієнтовані на малий бізнес. До першої групи належать системи, що входять до складу більших **ERP**-рішень. Вони забезпечують інтеграцію складського обліку з іншими бізнес-процесами на підприємстві, такими як бухгалтерський облік, фінанси, управління персоналом та логістика. Такі системи мають широкий спектр функцій, включаючи управління запасами, прогнозування попиту, автоматизацію закупок та формування аналітичних звітів. Рис.1.3. Схема **ERP**. Але складні системи мають і деякі недоліки. Їх часто важко впроваджувати та налаштовувати, що вимагає значних витрат та спеціального навчання операторів. Крім того, їх функціональність стає надлишковою для малого бізнесу, що ускладнює їх використання та робить їх ефективнішими. Ще одна категорія - це спеціалізовані програми складського обліку, які не входять до складу більших інформаційних систем. Такі рішення, як правило, мають вузьку спеціалізацію та орієнтовані на виконання певних завдань, більшість із них ненааявні в публічному доступі. Вони забезпечують облік товарів, контроль залишків, реєстрацію переміщень продукції та формування базових звітів. Перевагою таких систем є їхня простота у використанні та економічна ефективність. Вони не потребують складного налаштування та можуть бути швидко впроваджені в робочий процес підприємства. Однак вони часто неефективні, створюючи проблеми в розширенні бізнесу або облікових процесах. Під хмарними сервісами розуміють інфраструктуру інтернет-серверів, що надають віртуальні потужності для обробки та зберігання інформації. Ця технологія дозволяє масштабувати доступ до ІТ-ресурсів (додатків, баз даних, мережевих сервісів) за запитом, забезпечуючи високу мобільність і доступність даних [6]. Окрему групу складають хмарні сервіси складського обліку. Вони працюють через веб-інтерфейс і не потребують встановлення на комп'ютер користувача. Дані зберігаються на віддалених серверах, що дозволяє отримати доступ до системи з будь-якого місця, де є підключення до Інтернету. Такі рішення більш доступні підприємствам, які мають кілька складів або широко розподілену структуру. Рис.1.4. Зразок хмарного сервісу складського обліку. Водночас використання хмарних сервісів пов'язане з певними ризиками. Зокрема, це зумовлено залежністю від стабільності підключення до Інтернету, проблемами безпеки даних та необхідністю регулярної оплати за користування послугою. Для деяких підприємств це може бути ключовим фактором при виборі системи. Перелік ризиків використання хмарних сервісів доповнюють потенційна обмеженість

автономії через залежність від провайдера та значні операційні труднощі під час перенесення систем на інші платформи [7]. Аналіз сучасних систем складського обліку показує, що кожна з них має свої переваги та обмеження. Великі корпоративні рішення пропонують багато можливостей, але є складними та дорогими. Спеціальні програми прості та доступні, але не завжди відповідають потребам підприємств, що розвиваються. Хмарні сервіси пропонують гнучкість та доступність, але мають свої технічні та організаційні обмеження. Тому вибір системи складського обліку залежить від конкретних потреб підприємства, його масштабу, фінансових можливостей та рівня підготовки персоналу. Проведений огляд дозволив нам зробити висновок, що бажано розробляти власний програмний продукт, який може поєднати в собі простоту використання, достатню функціональність та відсутність надлишкових можливостей.

1.3. Порівняльний аналіз існуючих систем складського обліку

Для кращого розуміння особливостей сучасних систем складського обліку корисно провести їх порівняльний аналіз. Це дозволяє не лише оцінити їхні функціональні можливості, але й визначити ключові критерії, що впливають на вибір програмного забезпечення для конкретного підприємства. Як об'єкти аналізу було обрано кілька типових представників різних класів систем: корпоративні [ERP](#)-рішення, специфічні локальні програми та хмарні сервіси. Такий підхід дозволяє отримати загальне уявлення про існуючі рішення та виявити їх якісні особливості. Основними критеріями порівняння є функціональність, складність впровадження, вартість використання, зручність інтерфейсу та масштабованість. Саме ці характеристики впливають на ефективність системи в реальних умовах. Для наведення загальних порівнянь різних типів систем складського обліку було створено таблицю. Таблиця 1.3.1. Порівняльний аналіз систем складського обліку.

Критерій	ERP -системи	Локальні програми	Хмарні сервіси
Функціональність	Висока	Середня	Середня
Складність впровадження	Висока	Низька	Низька
Вартість	Висока	Низька	Середня
Гнучкість використання	Середня	Висока	Висока
Масштабованість	Висока	Обмежена	Висока

Як видно з наведеної вище таблиці, [ERP](#)-системи забезпечують найширший функціонал та високий рівень масштабованості, що робить їх придатними для великих підприємств. Проте їх впровадження потребує значних ресурсів, як фінансових, так і часових. Крім того, складність інтерфейсу може ускладнювати роботу користувачів без спеціальної підготовки. Локальні програми, навпаки, відзначаються простотою використання та швидкістю впровадження. Вони не потребують значних витрат і можуть ефективно використовуватись на малих підприємствах. Однак їх можливості масштабування є обмеженими, що може стати проблемою при розширенні діяльності. Крім того, централізація даних у межах однієї платформи зумовлює високу вразливість компанії до системних збоїв. Техногенні фактори або порушення в роботі серверного обладнання, що призводять до недоступності [ERP](#)-системи, мають критичний вплив, спричиняючи припинення функціонування всіх підрозділів [8]. Хмарні сервіси займають проміжне положення. Вони поєднують відносну простоту використання з можливістю роботи в розподіленому середовищі. Завдяки доступу через Інтернет такі системи дозволяють організувати облік незалежно від місця знаходження користувача. Проте їх використання пов'язане з постійними витратами та залежністю від зовнішньої інфраструктури. Експлуатація хмарних сервісів базується на моделі регулярних орендних платежів. Недотримання термінів оплати за користування програмним інструментарієм призводить до тимчасового обмеження доступу до ресурсів, при цьому порядок і умови призупинення надання послуг чітко регламентуються договором постачання [9]. Проведений аналіз дозволяє зробити висновок, що універсального рішення, яке б однаково добре підходило для всіх підприємств, не існує. Кожен тип систем має свої переваги та обмеження, які необхідно враховувати при виборі. З урахуванням результатів порівняння можна сформулювати основні вимоги до розроблюваної системи. Вона повинна поєднувати простоту локальних програм із достатнім рівнем функціональності, необхідним для ефективного ведення складського обліку. Водночас важливо забезпечити можливість подальшого розширення системи без суттєвого ускладнення її структури. Проведений порівняльний аналіз став основою для визначення напрямку розробки та дозволив сформулювати підхід до створення програмного продукту, орієнтованого на практичні потреби користувачів.

1.4. Обґрунтування вибору технологій для розробки

Вибір технологій є одним з ключових етапів розробки програмного забезпечення, оскільки він впливає на функціональність, можливості, зручність використання та можливість подальшого розвитку системи. В рамках цієї бакалаврської роботи для створення графічного інтерфейсу користувача було обрано платформу [.NET](#), мову програмування [C#](#), технологію [Windows Forms](#) та систему керування

базами даних [SQLite](#). [C#](#) являє собою високорівневу об'єктно-орієнтовану мову програмування з акцентом на безпеку типів, адаптовану для екосистеми [.NET](#). Проєкт був реалізований під егідою [Microsoft](#) групою розробників у складі А. Гейлсберга, С. Вілтамута та П. Гольде, що заклало фундамент сучасної розробки на базі технологій [Microsoft](#) [10]. Вона поєднує в собі простоту синтаксису, високу продуктивність та універсальність у різних програмних продуктах. Використання об'єктно-орієнтованого підходу дозволяє структуровано організувати код, спрощуючи його підтримку та розширення. Крім того, [C#](#) має безліч бібліотек та вбудованих інструментів, які значно пришвидшують процес розробки. [.NET](#) являє собою середовище розробки з відкритим вихідним кодом, що забезпечує підтримку широкого спектра мов програмування та спеціалізованих бібліотек. Дана екосистема надає розробникам універсальний інструментарій для створення програмних продуктів, адаптованих під різні операційні системи [11]. Платформа [.NET](#) забезпечує середовище виконання програм та надає доступ до широкого спектру готових компонентів. Вона підтримує різноманітні програми, включаючи настільні, веб- та мобільні. Однією з найважливіших переваг платформи є автоматичне управління пам'яттю, що знижує ризик помилок, пов'язаних з неналежним використанням ресурсів. Платформа також забезпечує високий рівень безпеки та стабільності для програм. Рис.1.5. Логотипи [.NET](#) та [C#](#). Для реалізації графічного інтерфейсу користувача було обрано технологію [Windows Forms](#). У структурі [Microsoft .NET Framework Windows Forms](#) виступає ключовим інструментарієм для створення візуальних оболонок застосунків, регламентуючи принципи побудови та взаємодії користувача з графічним інтерфейсом [12]. Це дозволяє створювати настільні програми за допомогою візуального конструктора, що значно спрощує розробку інтерфейсу. Дана технологія надає зручний механізм обробки подій, який дозволяє швидко покращити взаємодію користувача з програмою. Ця технологія ідеально підходить для створення бізнес-додатків, особливо бухгалтерських систем, де важливі простота та зрозумілість інтерфейсу. Одна з причин вибору [Windows Forms](#) - це простота використання та освоєння. Це дозволяє швидко реалізувати необхідний функціонал без використання складних технологій. Крім того, програми, створені за допомогою цієї технології, потребують менше комп'ютерних ресурсів, що робить їх придатними для використання на різних комп'ютерах. В якості



Знайдено плагіату: 0,04%

id: 6

системи керування базами даних було обрано [SQLite](#). [SQLite](#) являє собою вбудовану реляційну бібліотеку для керування базами даних, що відповідає стандарту [SQL-92](#). Її ключовою особливістю є відсутність серверної частини, що класифікує її як полегшене рішення для роботи зі

структурованими даними [13]. Крім того це легка вбудована база даних, яка не потребує встановлення окремого сервера. Всі дані зберігаються в одному файлі, що спрощує встановлення та використання програми. Система підтримує стандартну мову запитів [SQL](#), що дозволяє ефективно працювати з даними. Рис.1.6. Логотип [SQLite](#). Перевагою [SQLite](#) є висока швидкість роботи під час виконання операцій читання та запису, а також легкість інтеграції з додатками [C#](#). Він ідеально підходить для малих та середніх програм, де немає потреби в складній клієнт-серверній архітектурі. Крім того, використання локальної бази даних забезпечує незалежність від мережевого підключення, що є важливим для стабільної роботи системи. Поєднання [C#](#), [.NET](#), [Windows Forms](#) та [SQLite](#) забезпечує баланс між функціональністю та простотою впровадження. Такий підхід особливо підходить для розробки програмного продукту, орієнтованого на малий та середній бізнес. Тому вибір цих технологій є виправданим та відповідає цілям бакалаврської роботи. Вони надають можливість створити повноцінно функціонуючу систему складського обліку, яка враховує сучасні вимоги до програмного забезпечення. 1.5. Постановка задачі та формування вимог до системи На основі аналізу предметної області та існуючих програмних рішень можна сформулювати основне завдання бакалаврської роботи. Воно полягає в розробці програмного комплексу, який забезпечує автоматизацію обліку товарів на складі та дозволяє ефективно керувати інформацією про їх рух, кількість та характеристики. Розроблена система повинна забезпечувати виконання ключових операцій складського обліку, зокрема, реєстрацію надходження товарів, їх облік, ведення поточної інформації про залишки, а також ведення історії всіх операцій. Водночас важливо забезпечити простоту використання системи, щоб користувачі могли ефективно використовувати її без спеціальної технічної підготовки. Однією з головних вимог до системи є забезпечення достовірності даних. Це означає, що всі операції повинні виконуватися точно, без втрати

інформації та створення помилок. Для цього необхідно передбачити механізми перевірки введених даних, а також забезпечення цілісності бази даних під час операцій. Системні вимоги включають можливість роботи з ключовими сутностями в предметній області. Система повинна забезпечувати додавання, редагування та видалення інформації про товари, категорії та постачальників. Користувач повинен мати можливість переглядати список товарів, виконувати пошук та отримувати інформацію про їх кількість та характеристики. Особлива увага впроваджена обліку руху товарів. Система повинна підтримувати операції надходження та списання, а також автоматично змінювати кількість товарів на складі. Кожна операція повинна мати журнальний запис, що містить інформацію про тип операції, кількість, дату операції та користувача. В ієрархії інформаційних технологій автентифікація визначається як регламентована процедура підтвердження автентичності суб'єкта або об'єкта системи [14]. В даній розробці, ця система є дуже важливою. Вона повинна забезпечувати можливість входу в систему за допомогою логіна та пароля, а також обмеження прав доступу. Це дозволяє обмежити доступ до певних функцій системи та підвищити рівень безпеки. Рис.1.7. Приклад схеми автентифікації.

Окрім вимог до системи, необхідно враховувати характеристики відмов системи. До них належать зручність інтерфейсу, швидкість роботи, надійність та можливість подальшого розвитку. Інтерфейс має бути інтуїтивно зрозумілим, щоб користувач міг швидко опанувати завдання з програмою. Швидкість операцій повинна забезпечувати оптимальну продуктивність навіть при обробці великих обсягів даних. Система також повинна бути безпомилковою. У разі помилки, такої як неправильне введення даних або відсутність необхідної інформації, програма повинна реагувати правильно, запобігаючи аварійному вимкненню. Крім того, корисно знати вимоги до зберігання даних. Використання локальної бази даних дозволяє забезпечити автономність системи, але вимагає впровадження механізмів захисту інформації. Зокрема, важливо забезпечити зберігання паролів у зашифрованому вигляді та безпеку даних. Витрати на передачу даних у мережевому середовищі є значно вищими порівняно з локальними операціями доступу. Сукупна вартість транзакції охоплює не лише обсяг переданого трафіку, а й обчислювальні ресурси, необхідні для забезпечення роботи багаторівневих протоколів, що відповідають за інкапсуляцію, маршрутизацію та дефрагментацію пакетів на вузлі-отримувачі [15]. Таким чином, постановка завдання включає створення програмної системи, яка забезпечує автоматизацію ключових процесів складського обліку, відповідає вимогам зручності та надійності, а також може бути використана в реальних умовах на підприємстві.

1.6. Вимоги до програмної системи автоматизації складського обліку

Визначення вимог до програмної системи являє собою дуже важливу та ключову частину етапів розробки, від цього залежить не тільки функціональність, а ще й подальший розвиток програмного продукту, з особливою увагою було виділено відповідність реальним потребам користувачів даної системи. Функціональні вимоги визначають завдання для розробки програми, які система повинна виконувати. Основною функцією являється облік товарів на складі, що передбачає збереження інформації про назву товару, його кількість, ціну, категорії та постачальників та іншу не менш важливу інформацію про товар. Розроблювана система повинна мати можливість додавати нові товари з урахуванням її характеристик, змінювати наявні записи та видаляти непотрібні. Важливим є те, що пошук товарів та записів переміщення чи списання товарів має бути швидким і гнучким, для максимально ефективної роботи підприємства. Рис.1.8. Приклад пошуку товару та його дані. На даному зображенні наглядно показано процес пошуку товару, використовуючи який, не обов'язково вводити ім'я товару повністю, а достатньо тільки частини назви. Як згадувалось, однією з дуже важливих вимог є запис обліку руху товарів. Система повинна реєструвати надходження та списання продукції, автоматично оновлювати кількість на складі і вести журнал операцій, за допомогою якого користувач може відстежувати історію змін та аналізувати роботу складу. Також система має забезпечувати роботу з довідковими даними, такими як категорії товарів та інформація про постачальників з можливістю їх додавання в базу даних. Це дозволяє підтримувати дані актуальними і полегшує управління складом. Нефункціональні вимоги визначають якість роботи системи. Інтерфейс має бути інтуїтивно зрозумілим, щоб навіть користувачі з обмеженим технічним досвідом могли швидко розібратися з роботою програми. Надійність системи передбачає збереження даних у разі помилок або збоїв, для чого використовується механізм транзакцій у базі даних. Продуктивність системи передбачає швидке оброблення запитів та виконання операцій навіть при збільшенні обсягів інформації. Використання легких систем без перевантаження надлишковим функціоналом, полегшує управління базами даних та допомагає забезпечити

стабільну та ефективну роботу програмного забезпечення, також інтерфейс був розроблений без використання візуальних ефектів та анімацій, для забезпечення максимальної продуктивності та мінімального навантаження на систему, що дозволяє встановити її навіть на дуже слабкі пристрої. Таким чином, сформовані вимоги до розробки відображають основні характеристики розроблюваної системи та визначають її здатність ефективно виконувати завдання автоматизації складського обліку, створюючи зручний та надійний інструмент для роботи користувачів.

1.7. Моделі даних та їх роль у інформаційних системах

Моделі даних є фундаментальною складовою будь-якої інформаційної системи, оскільки визначають, як організована, зберігається та обробляється інформація. Точність побудови моделі впливає на ефективність роботи системи, її надійність та зручність подальшого розвитку.

Рис.1.9. Зображення прикладу реляційної моделі.

Модель даних можна розглядати як формальний опис об'єктів в предметній області, їхніх властивостей і взаємозв'язків. Ще вона дозволяє перетворювати реальні процеси у структуровану форму, яку можна зберігати у базі даних та обробляти за допомогою програмного забезпечення. Реляційна модель даних базується на математичному понятті відношення та поданні цих відношень у формі таблиць. Вона була запропонована на початку 1970-х років американським ученим Е. Коддом. У будь-якій реляційній СУБД передбачається, що користувач сприймає базу даних як сукупність таблиць. Це стосується лише логічної структури бази даних і належить до концептуального та зовнішнього рівнів представлення [16]. У розробленій системі складського обліку реляційна модель використовується для таких сутностей, як товари, категорії, постачальники, користувачі та рух товарів. Кожна сутність у програмі відповідає певному класу, а у базі даних - таблиці. Сутність

Знайдено посилань: 0%

id: 7

«товар»

містить основну інформацію: назву, унікальний ідентифікатор, штрих-код, одиницю вимірювання, ціну та кількість, а також ідентифікатор категорії та постачальника, для зв'язку з іншими таблицями. Ці зв'язки забезпечують структуроване представлення даних. Категорії дозволяють групувати товари для зручності пошуку та аналізу. Інформація про постачальників допомагає планувати закупки та вести бухгалтерський облік. Сутність

Знайдено посилань: 0%

id: 8

«рух товарів»

фіксує всі операції над продукцією: надходження, списання, дату та дії користувача. Це забезпечує прозорість обліку та можливість аналізу історії змін. Сутність

Знайдено посилань: 0%

id: 9

«користувач»

визначає доступ до системи, зберігає дані для ідентифікації користувача, такі як логін та хеш паролю, для входу. При побудові моделі даних було забезпечено узгодженість інформації. Наприклад, кожен товар обов'язково пов'язаний із категорією та постачальником, що запобігає появі аномальних записів та мінімізує помилки в базі даних. Моделі було розроблено гнучкими і з можливістю масштабування: при необхідності можна додавати нові таблиці чи поля, не змінюючи існуючу структуру. Отже, розроблена та спроектована мною модель даних забезпечує надійну основу для роботи системи складського обліку, підтримує логічні зв'язки між об'єктами, спрощує обслуговування та дозволяє розвивати систему у майбутньому. Для користувача це означає швидку, стабільну та зручну роботу з усіма аспектами обліку товарів.

Висновки до розділу У першій частині бакалаврської роботи розглянуто теоретичні основи складського обліку, проаналізовано сучасні програмні рішення та обґрунтовано вибір технологій для розробки персональної інформаційної системи. Дослідження показує, що складський облік є важливим інструментом управління підприємством, що забезпечує контроль за рухом матеріальних ресурсів, їх зберіганням та ефективним використанням. Від якості організації бухгалтерського обліку залежить терміни прийняття управлінських рішень, оптимізація витрат та підвищення загальної ефективності підприємства. Аналіз сучасних систем складського обліку виявляє широкий спектр програмних рішень, які відрізняються функціональністю, складністю та вартістю. Було визначено три основні групи систем: корпоративні ERP-рішення, спеціалізовані локальні програми та хмарні сервіси. Кожна з цих груп має свої переваги та недоліки, що впливають на їх правильне використання в різних умовах. Порівняльний аналіз дозволяє визначити ключові критерії оцінки систем

складського обліку, включаючи функціональність, зручність використання, вартість та масштабованість. Встановлено, що не існує універсального рішення, яке може повністю задовольнити всі потреби підприємств, що підкреслює актуальність розробки власного програмного продукту. В результаті обґрунтування вибору технологій було визначено доцільність використання мови програмування [C#](#), платформи [.NET](#), технології [Windows Forms](#) та системи керування базами даних [SQLite](#). Також було сформульовано постановку задачі та визначено основні вимоги до розробленої системи. Вони охоплюють як практичні, так і неефективні характеристики, такі як надійність, зручність використання та безпека даних. Більше уваги приділено забезпеченню точності обліку товарів та контролю їх руху. Таким чином, результати першого розділу формують теоретико-методологічну основу для подальшого розвитку програмної системи складського обліку. Отримані висновки вказують на напрямки проєктування та впровадження системи, які розглядаються у другому розділі бакалаврської роботи. РОЗДІЛ 2

ПРОЄКТУВАННЯ І РОЗРОБКА СИСТЕМИ СКЛАДСЬКОГО ОБЛІКУ 2.1. Загальна архітектура програмного забезпечення Архітектура програмного забезпечення є основою будь-якої інформаційної системи, оскільки вона визначає структуру програми, принципи взаємодії її компонентів та загальну логіку роботи. В рамках цієї бакалаврської роботи було розроблено додаток для автоматизації складського обліку, побудований з використанням багатоварової архітектури. Багатоварова архітектура, або [n](#)-рівневий архітектурний стиль, є однією з найбільш затребуваних парадигм у проєктуванні систем. Даний підхід став галузевим стандартом для більшості прикладних рішень завдяки високій ергономічності розробки, широкій поширеності в інженерному середовищі та економічній ефективності впровадження [17]. Її вибір передбачає логічний поділ системи на окремі компоненти, кожен з яких відповідає за певні функції. Це дозволяє зменшити складність програмного коду, підвищити його зрозумілість та забезпечити можливість подальшого розвитку системи без суттєвих змін у вже реалізованих модулях. Структура проєкту організована у вигляді окремих каталогів, що відповідають різним рівням архітектури. Такий підхід дозволяє чітко розмежувати функціональність та уникнути змішування логіки для різних об'єктів. Рис.2.1. Папки проєкту. Однією з ключових особливостей є рівень доступу до даних, який реалізовано в каталозі [Data](#). Він відповідає за встановлення зв'язу з базою даних та її ініціалізацію. На цьому рівні можна створювати необхідні таблиці без бази даних, що дозволяє системі автоматично підготувати середовище до роботи при першому запуску. Наступним важливим рівнем є рівень моделі, який представлений у каталозі [Models](#). Він включає класи, що охоплюють ключові аспекти предметної області, такі як продукт, категорія, постачальник, споживач та рух продукту. Кожен з цих класів має властивості, що відповідають структурі відповідних таблиць у базі даних. Моделі користувачів дозволяють уніфікувати роботу з даними та полегшити передачу між різними частинами системи. Під бізнес-логікою розуміють програмну імплементацію предметної області, що охоплює систему залежностей між даними та механізми їхнього опрацювання. Даний компонент формується на основі унікальних бізнес-правил замовника з метою забезпечення коректного функціонування автоматизованих систем керування [18]. Вона була реалізована в каталозі [Service](#). Тут розташовані класи, які виконують обробку даних, перевіряють правильність операцій та взаємодіють з базою даних. Наприклад, окремі сервіси відповідають за роботу з продуктами, категоріями, постачальниками та рухом продукту. Такий підхід дозволяє централізувати логіку обробки даних та уникнути її дублювання в різних частинах програми. Окремий модуль безпеки розташований у каталозі [Security](#). Він відповідає за автентифікацію користувачів та захист їхніх даних. Цей модуль реалізує механізм хешування паролів, який гарантує їх зберігання в безпечній формі. Рис.2.2. Список написаних класів. Інтерфейс користувача є функціональним середовищем для забезпечення ефективної взаємодії оператора з інформаційною системою. Ключове призначення [UI](#) полягає в оптимізації користувацького досвіду, що дозволяє повноцінно експлуатувати програмний комплекс без необхідності володіння спеціалізованими технічними компетенціями [19]. Його було реалізовано в каталозі [Forms](#). Він складається з набору форм, кожна з яких відповідає за певні завдання. Наприклад, є форми для входу в систему, управління товарами, категоріями, постачальниками, а також виконання операцій надходження та списання товару. Кожна форма взаємодіє з відповідними сервісами, передаючи їм дані та отримуючи результати для відображення. Рис.2.3. Список всіх форм. Взаємодія між компонентами системи організована таким чином, щоб мінімізувати залежності між ними. Форми не працюють безпосередньо з базою даних, а використовують для цього сервіси. У свою чергу, сервіси використовують класи доступу

до даних. Такий підхід дозволяє змінювати реалізацію одного з рівнів без необхідності вносити зміни до інших частин системи. Важливим елементом архітектури є використання методів транзакцій, які змінюють стан даних. Це особливо важливо для операцій надходження та списання, де одночасно змінюється кількість товару та робиться запис у журналі руху. Використання транзакцій забезпечує цілісність даних та запобігає помилкам у разі збою. Ще одним важливим аспектом є організація запуску програми. Відправною точкою для входу є файл програми, де проходить ініціалізація бази даних та відкривається форма авторизації. Після успішного входу користувач отримує доступ до основного функціоналу системи. Таким чином, архітектура розробленої системи забезпечує чіткий розподіл обов'язків між компонентами, зручність підтримки та можливість розширення функціональності. Обраний вами метод дозволяє створити надійну та ефективну програмну систему, яка відповідає вимогам автоматизації складського обліку.

2.2. Розробка структури бази даних

Проектування бази даних є одним з ключових етапів розробки інформаційної системи, оскільки воно забезпечує зберігання, структурування та обробку даних. Від правильної побудови структури бази даних залежить ефективність усієї системи, швидкість виконання запитів та цілісність інформації. Реляційна БД базується на табличному представленні даних, що дозволяє реалізовувати складні запити та працювати з інформацією багатьма способами. Це забезпечує високу цілісність даних і виключає необхідність структурної перебудови бази при зміні вимог до вибірки чи аналізу активів [20]. У цьому проєкті використовується реляційна база даних [SQLite](#), яка зберігає дані у вигляді таблиць, пов'язаних логічними зв'язками. Такий підхід виключає дублювання та забезпечує узгодженість. На етапі проектування були визначені основні сутності предметної області, які визначають структуру складського обліку. До них належать товари, категорії, постачальники, використання та облік руху товарів. Кожна з цих сутностей представлена у вигляді окремої таблиці. Таблиця товарів є основним компонентом системи, оскільки вона містить інформацію про всі товари, що зберігаються на складі. Для кожного товару зберігається його унікальний ідентифікатор, код, назва, штрих-код, одиниця вимірювання, ціна, кількість, а також посилання на категорію та постачальника. Таблиця категорій використовується для групування товарів за певними характеристиками. Це полегшує пошук та аналіз даних. Таблиця постачальників містить інформацію про організації або осіб, які постачають продукцію. Таблиця користувачів використовується для зберігання даних про окремих користувачів системи. Вона містить логін, хеш пароля та роль користувача. Ролі користувачів дозволяють реалізувати розмежування прав доступу до функціональності системи. Особливо важлива таблиця руху товарів. Вона містить інформацію про всі операції, пов'язані з отриманням та списанням продукції. Кожен запис містить ідентифікатор продукції, її назву, кількість, режим роботи, дату виготовлення, використання та інші коментарі. Наявність такої таблиці дозволяє відстежувати історію змін та забезпечувати прозорість обліку.

2.2.2. База даних системи.

[Users](#) [Products](#) [Categories](#) [Suppliers](#) [StockMovements](#) [Id](#) [Id](#) [Id](#) [Id](#) [Id](#) [Login](#) [Code](#) [Name](#) [Name](#) [ProductId](#) [PasswordHash](#) [Name](#) [Phone](#) [ProductName](#) [Role](#) [Barcode](#) [Email](#) [Quantity](#) [Unit](#) [MovementType](#) [Price](#) [UserLogin](#) [Quantity](#) [MovementDate](#) [CategoryId](#) [Comment](#) [SupplierId](#)

Зв'язки між таблицями реалізовані за допомогою зовнішніх ключів. Зокрема, таблиця товарів містить посилання на категорію та постачальника, що дозволяє отримувати додаткову інформацію без дублювання даних. Таблиця руху товарів пов'язана з таблицею товарів через ідентифікатор товару. Під час проектування бази даних враховувалися принципи нормалізації. Це дозволило зменшити надлишковість даних та підвищити їх цілісність. Кожна таблиця відповідає окремій сутності, а всі залежності між даними реалізуються через зв'язки. Під нормалізацією розуміють методологію поетапного розбиття таблиць відповідно до нормальних форм. Цей процес базується на аналізі зв'язків між атрибутами й дозволяє трансформувати вихідне відношення у групу логічно обґрунтованих структур, що відповідають вимогам реляційної моделі [21]. Особливу увагу було приділено типам даних. Для ідентифікаторів використовуються типи цілих чисел, для текстових полів - рядкові типи, а для зберігання дат і часу - відповідні формати, що дозволяють здійснювати точне порівняння та сортування. Важливим питанням є забезпечення цілісності даних. Для цього використовуються функції, які перешкоджають додаванню відповідних записів. Наприклад, неможливо створити товар без вказівки його назви або категорії. Вони також перевіряються під час операцій, що змінюють кількість товарів. Таким чином, розроблена структура бази даних забезпечує належне зберігання інформації, підтримує всі необхідні складські операції та відповідає вимогам до надійності та масштабованості. Це основа для реалізації функціональності системи, яка обговорюється в наступних розділах.

2.3.

Реалізація основних модулів системи Реалізація основних модулів системи є центральним етапом розробки програмного забезпечення, оскільки саме на цьому рівні виконуються основні функції складського обліку. В рамках цього проєкту функціональність системи організована за модульним принципом, що дозволяє розділити логіку програми на різні компоненти та спростити її для подальшої підтримки. Основна частина логіки реалізована у вигляді сервісів, кожен з яких має працювати з певною сутністю в предметній області. Такий підхід дозволяє ізолювати бізнес-логіку від інтерфейсу користувача та забезпечити повторне використання коду. Рис.2.4. Список сервісів. Модуль роботи з товарами реалізовано у вигляді відповідного сервісу, який забезпечує виконання операцій з додавання, редагування, видалення та отримання інформації про товари. При додаванні товару система зберігає всі його характеристики, такі як коди, назви, штрих-коди, одиниці вимірювання, ціни, кількості, а також посилання на категорію та постачальника. При оновленні даних введена інформація перевіряється на точність, що запобігає помилкам. Окремий модуль створено для роботи з категоріями товарів. Його функція полягає у створенні, редагуванні та видаленні нових категорій. Використання категорій дозволяє структурувати інформацію про товар та спрощує пошук у системі. Так само впроваджується модуль для роботи з постачальниками, який забезпечує збереження інформації про джерела товарів. Це дозволяє відстежувати походження товарів та аналізувати взаємодію з постачальниками. Одним з найважливіших є модуль обліку руху товарів. Забезпечує операції з приймання та списання товарів. При надходженні товару система автоматично збільшує його кількість на складі та створює відповідний запис у журналі руху. Під час списання товар перевіряється наявність оптом, після чого його кількість зменшується, а інформація про операцію записується в базу даних. Особливістю реалізації цього модуля є використання транзакцій, які забезпечують узгодженість даних. Це означає, що зміна кількості товару та створення запису руху можуть бути виконані як одна операція. У разі виникнення помилки всі зміни скасовуються, що усуває недійсні дані. Модуль автентифікації користувачів забезпечує контроль доступу до системи. Він перевіряє введені облікові дані та визначає роль користувача. Паролі зберігаються у вигляді хешів, що підвищує рівень безпеки. Після успішного входу користувач отримує доступ до функцій відповідно до своїх прав. Рис.2.5. Модуль автентифікації користувачів. Фільтрування даних являє собою високоефективний інструмент виокремлення цільових підмножин інформації в межах визначеного діапазону або табличного масиву. Ця процедура забезпечує візуалізацію виключно тих записів, що відповідають заданим критеріям, шляхом тимчасового приховування неактуальних рядків [22]. Для підвищення ефективності системи реалізовано пошук та фільтрацію даних. Це дозволяє швидко знаходити потрібну інформацію навіть за великої кількості даних. Зокрема, користувач може шукати товари за назвою або іншими характеристиками. Взаємодія між модулями відбувається через чітко визначені інтерфейси. Користувач отримує доступ до сервісів через форми для виконання операцій, а сервіси, у свою чергу, взаємодіють з базою даних. Такий підхід дозволяє уникнути дублювання коду та забезпечує централізовану обробку даних. Під час впровадження модулів враховувалася необхідність обробки помилок. У разі неточності, такої як нестача товару або недостатня кількість, система повідомляє користувача та не виконує операцію. Це запобігає порушенню логіки обліку. Таким чином, впровадження основних модулів системи забезпечує виконання всіх необхідних функцій складського обліку. Використання модульного підходу дозволяє створити гнучку та зручну систему, яку можна легко розширювати або модифікувати в майбутньому.

2.4. Реалізація інтерфейсу користувача

Інтерфейс користувача є важливою частиною програмної системи, оскільки саме через нього користувач взаємодіє з функціональністю програми. Від якості реалізації інтерфейсу залежить зручність використання системи, швидкість операцій та загальна ефективність роботи користувача. В рамках цього проєкту інтерфейс користувача реалізовано з використанням технології [Windows Forms](#). Такий підхід дозволяє створити додаток з інтуїтивно зрозумілим графічним інтерфейсом, доступним для користувачів. Використання візуального конструктора форм значно спрощує процес розробки та дозволяє швидко створювати необхідні елементи інтерфейсу. Дизайн інтерфейсу користувача ([UI/UID](#)) охоплює сукупність методологій та етапів розробки візуально-функціонального середовища для апаратних і програмних комплексів. Ключовим пріоритетом цього процесу є створення максимально ергономічної та інтуїтивно зрозумілої моделі взаємодії оператора з електронними пристроями різного призначення [23]. Структура інтерфейсу побудована на основі різних форм, кожна з яких відповідає за виконання певного набору завдань. Центральною формою системи є головне вікно, яке

відкривається після успішної авторизації користувача. Воно містить основні елементи навігації, що забезпечують доступ до всіх функціональних можливостей системи. Рис.2.6. Вікно авторизації. Першим етапом роботи з програмою є форма авторизації. Вона призначена для введення логіна та пароля користувача. Якщо облікові дані були успішно перевірені, користувач отримує доступ до основної функції. В іншому випадку система повідомляє про помилку та каже, що спробує ще раз. Такий підхід дозволяє забезпечити базовий рівень безпеки доступу до системи. Головна форма виконує роль центру керування програмою. Вона має елементи керування, що дозволяють отримати доступ до різних розділів системи, зокрема, до управління товарами, категоріями, постачальниками та операціями руху товарів. Інтерфейс організовано таким чином, щоб користувач міг швидко знайти потрібну функцію та виконати потрібну операцію. Рис.2.7. Головне вікно програми. Для роботи з товарами передбачена окрема форма, що відображає список усіх доступних позицій. У ній реалізовано можливість перегляду інформації, додавання нових товарів, редагування існуючих та їх видалення. Для зручності користувача передбачені функції пошуку, які дозволяють йому швидко знаходити потрібний товар за назвою або іншими характеристиками. Аналогічно реалізовані форми для роботи з категоріями та постачальниками. Вони забезпечують базові операції з управління даними та мають просту та зрозумілу структуру. Такий підхід уніфікує інтерфейс та спрощує робочий процес користувача. Особлива увага приділяється формам, що стосуються обліку руху товарів. Розроблено окремі інтерфейси для операцій надходження та списання, що дозволяють користувачеві вибрати товар, вказати кількість та додати коментарі. Водночас система автоматично виконує всі необхідні обчислення та зберігає результати в базі даних. Рис.2.8. Вікно з товарами та вікно додавання товару. У процесі розробки інтерфейсу враховувалися принципи зручності та інтуїтивності. Масив елементів керування, таких як кнопки, поля введення та списки, розроблений таким чином, щоб користувач міг легко орієнтуватися в інтерфейсі. Використання стандартних компонентів [Windows Forms](#) забезпечує звичний вигляд програми, знижуючи поріг входу для нових користувачів. Під подією розуміють програмно розпізнаваний інцидент, що потребує виконання специфічного набору інструкцій. Джерелом виникнення подій можуть виступати дії суб'єкта, внутрішні операції операційного середовища або сигнали від сторонніх програмних компонентів [24]. Важливим аспектом є обробка подій, що відбуваються під час взаємодії користувача з інтерфейсом. Натискання кнопок, вибір елементів зі списку або введення даних у поля може викликати відповідні методи, що реалізують необхідну логіку. Це забезпечує безперебійну роботу програми та дозволяє швидко реагувати на дії користувача. Також передбачено відображення повідомлень про результати операцій. Наприклад, коли продукт успішно додано або виникає помилка, система повідомляє користувача за допомогою діалогових вікон. Це спрощує роботу та дозволяє швидше виявляти та виправляти помилки. Таким чином, реалізований інтерфейс користувача забезпечує зручну та ефективну взаємодію з системою. Він дозволяє користувачам виконувати всі необхідні операції складського обліку без зайвих труднощів.

2.5. Реалізація обліку руху товарів Облік руху товарів є однією з основних функцій розробленої системи, оскільки забезпечує контроль за змінами кількості продукції на складі. Успішне впровадження цього механізму дозволяє отримувати актуальну інформацію про залишки товарів, відстежувати всі операції та забезпечувати прозорість обліку. Рух товару являє собою сукупність логістичних процесів, спрямованих на фізичне транспортування продукції від моменту виходу з виробництва до її передачі кінцевому споживачеві. Цей цикл охоплює всі етапи дистрибуції та операційного супроводу вантажів [25]. В рамках цієї системи під рухом товарів розуміються всі операції, пов'язані з надходженням продукції на склад та її відвантаженням. Для реалізації цього функціоналу використовується спеціальний модуль, який взаємодіє з таблицями товарів та журналом руху. Основними операціями, що підтримуються системою, є надходження товарів та їх документальне оформлення. Під час операції надходження користувач вибирає відповідний товар, вказує кількість та, за необхідності, додає коментар. Після підтвердження операції система автоматично збільшує кількість товарів на складі та робить запис у журналі руху. Рис.2.9. Вікно історії руху товарів. Операція списання реалізується аналогічно, але перед виконанням система перевіряє кількість наявних товарів. Якщо кількість, що підлягає списанню, перевищує наявний залишок, операція не виконується, і користувач отримує невірне повідомлення. Це запобігає конфіденційності даних та забезпечує надійність обліку. Важливою особливістю реалізації є використання транзакцій під час операцій переміщення товарів. Кожна операція включає дві взаємопов'язані дії: зміну кількості товару в таблиці товарів та

додавання запису до таблиці переміщення. Використання транзакції гарантує, що обидві дії будуть виконані як єдине ціле. У разі виникнення помилки жодна зі змін не буде збережена, що забезпечує цілісність даних. Журнал переміщення товарів відіграє важливу роль у системі, оскільки містить історію всіх проведених операцій. Кожен запис у журналі містить ідентифікатор товару, його назву, кількість, тип операції, дату, ім'я співробітника та коментар. Це дозволяє відстежувати всі зміни та аналізувати діяльність складу за певний період часу. Для зручності використання реалізовано можливість перегляду історії переміщення товарів. Дані можна фільтрувати за датою, що дозволяє отримати доступ до оперативної інформації за певний період часу. Такий підхід корисний для аналізу та моніторингу складських операцій. Рис.2.10. Вікно звітів з руху товарів. Особлива увага приділяється збереженню дати та часу операції. Це дозволяє точно визначити час виконання кожної дії та детально проаналізувати події. Використання стандартних форматів дати та часу гарантує точність обробки та відображення. Під час впровадження також враховувалася необхідність забезпечення зручності введення даних. Інтерфейс форм надходження та списання товарів розроблений таким чином, щоб користувач міг швидко виконати операцію, не витрачаючи час на зайве введення інформації. Товар вибирається зі списку, що мінімізує помилки. Система також забезпечує обробку помилок, що виникають під час операцій. Якщо дані введені неправильно або виникла технічна проблема, користувач отримує повідомлення, яке дозволяє швидко виявити та усунути причину помилки. Таким чином, впровадження механізмів обліку руху товарів забезпечує точність, надійність та зручність проведення операцій. Це дозволяє ефективно контролювати стан складських запасів та є важливою особливістю загальної системи автоматизації обліку.

2.6. Тестування програмного забезпечення

Тестування є важливою частиною процесу розробки програмного забезпечення, оскільки воно дозволяє перевірити точність роботи системи, виявити помилки та оцінити її надійність. В рамках цього проєкту було проведено тестування для перевірки точності реалізації функціональності складського обліку та забезпечення стабільної роботи програми. Під тестуванням ПЗ розуміють процедуру системного дослідження об'єкта розробки для верифікації показників його якості. Ключовою метою є формування інформаційної бази про стан продукту відносно контекстуальних умов та сценаріїв його використання кінцевими користувачами [26].

Процес тестування в даній розробці включає перевірку базової функціональності системи, зокрема авторизації користувачів, роботи з товарами, категоріями та постачальниками, а також виконання операцій з надходження та списання товарів. Більше уваги було приділено перевірці точності обробки даних та збереженню інформації в базі даних. Тестування авторизації включало перевірку точності введення логіна та пароля, а також поведінки системи у разі некоректності даних. Було встановлено, що система правильно обробляє як неправильні, так і некоректні спроби входу, забезпечуючи необхідний рівень безпеки. Під час тестування модуля продукту було перевірено операції додавання, редагування та видалення записів. Вони підтвердили, що всі зміни правильно відображаються в базі даних та одразу стають видимими в інтерфейсі користувача.

Рис.2.11. Вікно списання товарів та вікно помилки. На зображенні приклад помилки списання товару. В базі даних кількість вказаного товару тільки 9 і система не дозволяє списати більше чим це можливо, що є захистом від випадкового введення неправильних даних та для більшого контролю за кількістю товарів. Особливу увагу було приділено тестуванню обліку руху товарів, оскільки цей модуль є критично важливим для функціонування системи і має працювати без помилок. Було перевірено точність збільшення та зменшення кількості товарів, а також створення відповідних записів у журналі переміщення. Результати тестування показують, що система виконує всі операції правильно та забезпечує цілісність даних завдяки використанню транзакцій. Також було проведено тестування введення даних на обробку помилок. Зокрема, було перевірено ситуації введення текстових замість числових даних, відсутності товарів або недостатньої кількості на складі. Також було перевірено систему пошуку товарів в відповідній вкладці товарів. У всіх випадках система реагувала відповідно, перешкоджаючи виконанню правильних операцій.

Очікуваний результат	Фактичний результат	
Вхід із правильними даними	Успішна авторизація	
Виконано успішно	Вхід із неправильним паролем	Відмова у доступі
Виконано успішно	Додавання нового товару	Запис додається у базу даних
Виконано успішно	Редагування товару	Дані оновлюються
Виконано успішно	Видалення товару	Запис видаляється
Виконано успішно	Надходження товару	Кількість збільшується, запис додається
Виконано успішно	Списання товару (достатня кількість)	Кількість зменшується
Виконано успішно		

Списання товару (недостатня кількість) Операція не виконується Виконано успішно
Перегляд історії руху Відображення списку операцій Виконано успішно У процесі
тестування також оцінювалася стабільність системи під час кількох операцій. Було
виявлено, що програма працює правильно, коли вона не працює, і що всі дані зберігаються
в базі даних без втрат. Окрім функціонального тестування, було проведено оцінку
зручності використання системи. Інтерфейс було визначено як інтуїтивно зрозумілий, а
основні операції можна виконувати швидко та без проблем. Це підтверджує, що система
відповідає вимогам користувацького досвіду. Таким чином, результати тестування
підтверджують, що розроблена система працює правильно, забезпечує всі необхідні
функції та відповідає вимогам надійності. Незначні проблеми, виявлені під час процесу
доопрацювання, були усунені, що дозволило покращити якість програмного продукту. 2.7.
Порівняння з аналогами та переваги розробленої системи Оцінка продуктивності
розробленого програмного забезпечення передбачає його порівняння з існуючими
аналогами. Такий тип аналізу дозволяє виявити конкурентні переваги системи, а також
оцінити доцільність її використання в практичній діяльності. На сучасному ринку
програмного забезпечення доступна значна кількість систем для автоматизації
складського обліку. Серед них можна виділити масштабні корпоративні рішення,
спеціалізовані локальні програми та хмарні сервіси. Популярними представниками є такі
системи, як [BAS](#), [1C](#), а також різні [CRM](#) та [ERP](#) рішення. Корпоративні системи, такі як [BAS](#)
або [1C](#), є універсальними та дозволяють автоматизувати не тільки складський облік, але й
інші бізнес-процеси на підприємстві. Однак їх використання часто пов'язане з високими
витратами на впровадження, складністю налаштування та необхідністю навчання
персоналу. Крім того, такі системи можуть бути надлишковими для малого бізнесу, який не
потребує складного функціоналу. 1C є російським розробником у сфері системної інтеграції
та дистрибуції ПЗ. Спеціалізація компанії включає проектування баз даних та програмних
комплексів ділового призначення, а також забезпечення їхньої технічної підтримки на
ринку [27]. З січня 2026 року всі продукти 1C були заборонені на території України
відповідно до рішень РНБО та Держспецв'язку і внесені до офіційного переліку
забороненого ПЗ через загрозу національній безпеці. Виходячи з даної інформації, для
безпечного та більш сучасного ведення бізнесу без ризиків зі сторони закону на території
України, рекомендовано використовувати систему [BAS](#). Хмарні сервіси спрощують
використання та не вимагають встановлення програмного забезпечення на локальний
комп'ютер. Однак вони залежать від підключення до Інтернету, що в деяких ситуаціях
може бути жакхливим. Також виникають питання щодо безпеки даних та їх зберігання на
сторонніх серверах. Система, розроблена в рамках цієї роботи, належить до класу
локальних настільних додатків і орієнтована на використання в малих та середніх
підприємствах. Однією з головних переваг є простота використання. Інтерфейс системи
інтуїтивно зрозумілий і не потребує тривалого навчання користувачів. Це дозволяє швидко
впровадити програму в робочий процес і мінімізує час, витрачений на адаптацію
персоналу. Ще однією важливою перевагою є відсутність залежності від Інтернету.
Оскільки система працює локально, її можна використовувати в умовах обмеженого або
нестабільного доступу до мережі та локально в офісних мережах для обміну даними
виключно між працівниками. Наприклад звіти з руху товару, історія списання чи
надходження, та інші дані. Це забезпечує надійну роботу та дозволяє отримувати доступ
до даних у будь-який час. Використання легкої системи управління базами даних [SQLite](#)
дозволяє зберігати всі дані в одному файлі, що значно спрощує процес розгортання та
обслуговування системи. Користувачеві не потрібно встановлювати додаткове програмне
забезпечення або налаштовувати складні серверні компоненти. Система також забезпечує
достатній рівень функціональності для виконання основних завдань складського обліку.
Вона підтримує управління товарами, категоріями та постачальниками, а також облік
переміщення продукції з можливістю перегляду історії транзакцій. Це дозволяє ефективно
контролювати складські запаси та проводити аналіз активності. Важливою перевагою є
можливість подальшого розширення системи. Завдяки модульній архітектурі, програму
можна підтримувати новими функціями без значних змін у вже реалізованих компонентах.
Це відкриває можливості для розвитку системи відповідно до потреб користувача. Таблиця
2.7.1. Порівняння з іншими системами. Критерій Розроблена система [ERP](#)-системи (1C, [BAS](#))
Хмарні сервіси Вартість Низька Середня Простота використання Висока Середня Висока
Потреба в інтернеті Ні Ні Так Складність впровадження Низька Висока Низька
Масштабованість Середня Висока Висока Таким чином, розроблена система має
оптимальне співвідношення функціональності, простоти та вартості. Вона є ефективним

рішенням для автоматизації складського обліку в невеликих організаціях, де використання складних корпоративних систем неможливе. 2.8. Аналіз продуктивності та ефективності роботи системи

Продуктивність програмної системи є одним із ключових показників її якості, оскільки саме від швидкості обробки даних та стабільності роботи залежить зручність використання програмного забезпечення в реальних умовах. Для систем автоматизації складського обліку це має особливе значення, адже під час роботи користувачі постійно виконують операції додавання, редагування, пошуку та переміщення товарів. У випадку низької швидкодії навіть прості дії можуть займати значний час, що негативно впливає на ефективність роботи персоналу та загальну організацію облікових процесів. Розроблена система використовує локальну базу даних [SQLite](#), яка є компактною та достатньо швидкою системою управління базами даних для програм такого типу. Використання [SQLite](#) дозволяє уникнути необхідності налаштування окремого серверного середовища, що спрощує встановлення та подальше використання програмного забезпечення. Оскільки база даних зберігається безпосередньо на локальному комп'ютері, доступ до інформації виконується значно швидше, ніж у випадку роботи через мережу. Це позитивно впливає на швидкість виконання запитів та загальну продуктивність системи.

Оптимізація, - це процес удосконалення певного об'єкта, системи чи процесу з метою досягнення найефективніших характеристик, показників і співвідношень. Вона передбачає пошук найвигідніших рішень для підвищення продуктивності, економії ресурсів та покращення загальної ефективності. Наприклад, оптимізація може застосовуватися у виробничих процесах, організації виробництва, використанні енергії та багатьох інших сферах діяльності [28]. Під час розробки, особлива увага приділялася оптимізації основних операцій, які найчастіше використовуються в роботі користувача. До таких операцій належать додавання нових товарів, оновлення інформації про наявні записи, видалення даних та пошук необхідної інформації у базі даних. Аналіз роботи програми показав, що зазначені операції виконуються за короткий проміжок часу навіть при поступовому збільшенні кількості записів у системі. Це досягається завдяки використанню відносно простих [SQL](#)-запитів та раціональній структурі таблиць. Пошук товарів за назвою або окремими характеристиками реалізований таким чином, щоб користувач міг швидко отримати необхідну інформацію без тривалого очікування. У процесі тестування було встановлено, що навіть при роботі з великою кількістю товарних позицій результати пошуку відображаються практично миттєво. Це значно покращує зручність роботи з програмою та скорочує час виконання повсякденних операцій. Окрему увагу в системі приділено реалізації механізму обліку руху товарів. Такі операції часто пов'язані зі зміною кількох записів одночасно, тому для забезпечення цілісності даних використовуються транзакції. Використання механізму транзакцій дозволяє виконувати комплекс взаємопов'язаних дій як єдину операцію. У випадку виникнення помилки всі зміни можуть бути автоматично скасовані, що запобігає появі некоректних або неповних даних у базі. Крім підвищення надійності роботи системи, це також дозволяє уникнути повторного виконання окремих запитів і тим самим зменшує навантаження на програму. Важливим фактором, який впливає на ефективність роботи системи, є структура бази даних. У розробленому програмному забезпеченні використовується реляційна модель даних із чітко визначеними зв'язками між таблицями. Такий підхід забезпечує логічну організацію інформації та дозволяє уникнути дублювання даних. Для отримання пов'язаної інформації використовуються [SQL](#)-конструкції типу [JOIN](#), які забезпечують швидке об'єднання даних із кількох таблиць. Це дозволяє формувати необхідні результати без створення надлишкових копій інформації. Продуктивність системи також значною мірою залежить від обсягу даних, які зберігаються у базі. Для малих та середніх підприємств, на які орієнтований розроблений програмний продукт, кількість записів зазвичай не є надмірно великою. Саме по цій причині мною було вибрано використання [SQLite](#), що є цілком виправданим та ефективним рішенням, оскільки така база даних здатна забезпечити стабільну та швидку роботу навіть при поступовому збільшенні інформації. Під час тестування система демонструвала стабільну роботу без помітних затримок при виконанні типових операцій. Під час тестування часу запуску системи на комп'ютері [FIREBAT T8 Plus](#) зі встановленою ОС [Windows 10 LTSC](#), процесором [Intel N100](#), Оперативною пам'яттю 16Гб [DDR5](#) та [SSD](#) накопичувачем на 512Гб, час запуску був від 1 до 3 секунд, в залежності від фонових навантажень комп'ютера, що говорить про чудовий результат та хорошу оптимізацію. Не менш важливим аспектом є швидкість взаємодії користувача з інтерфейсом програми. Під час проектування системи було враховано необхідність забезпечення простого та зрозумілого інтерфейсу, який не перевантажує користувача зайвими елементами. Завдяки

цьому основні функції програми виконуються швидко та не викликають труднощів у процесі роботи. Форми введення даних, таблиці та елементи керування реагують без суттєвих затримок, що створює відчуття стабільності та надійності програмного забезпечення. Загальні результати проведеного аналізу свідчать про те, що розроблена система забезпечує достатній рівень продуктивності для виконання поставлених завдань. Використання локальної бази даних, оптимізованих [SQL](#)-запитів та простої архітектури дозволило досягти швидкої роботи програми без значних витрат системних ресурсів. Система успішно виконує базові операції складського обліку, забезпечує стабільну обробку даних та може ефективно використовуватися в практичній діяльності підприємств малого та середнього масштабу. Таким чином, проведений аналіз підтверджує, що програмна система відповідає основним вимогам щодо швидкодії, надійності та ефективності роботи. Навіть за умови поступового збільшення обсягів інформації система зберігає стабільність та забезпечує комфортну роботу користувачів, що є важливим показником якості розробленого програмного продукту.

2.9. Перспективи розвитку та масштабування системи

Розроблена програмна система автоматизованого обліку товарів в достатній мірі забезпечує виконання основних функцій, необхідних для організації ефективної роботи складу на малих та середніх підприємствах. Вона дозволяє здійснювати облік товарів, контролювати їх переміщення, оновлювати інформацію про залишки та забезпечувати швидкий доступ до необхідних даних. Однак з розвитком сучасних інформаційних систем, має сенс в покращенні та нововведеннях системи, а потреби користувачів з часом змінюються та стають важчими в реалізації. Саме тому важливим етапом оцінки програмного продукту є визначення можливостей його подальшого вдосконалення та масштабування для максимально довгого та ефективного існування розробленої системи в підприємствах. Дуже важливим із напрямків розвитку системи є розширення її функціональних можливостей. На даному етапі розробки система забезпечує виконання базових операцій складського обліку, проте в майбутньому доцільно реалізувати додаткові інструменти для аналізу та контролю діяльності підприємства. Зокрема, корисним буде впровадження модуля формування звітності. Автоматичне створення звітів, наприклад в форматі [excel](#), дозволить користувачам отримувати інформацію про залишки товарів, рух продукції, найбільш активні операції та інші показники діяльності складу. Наявність такої функціональності значно спростить процес аналізу інформації та допоможе керівництву підприємства швидше приймати управлінські рішення. Не варто виключати те, що системою будуть користуватись багато різних працівників, де не всі повинні мати доступ до деяких функцій системи. Саме тому, напрямком удосконалення є також розширення можливостей управління користувачами системи. На практиці різні працівники підприємства можуть мати різний рівень доступу до інформації та функцій програми. Наприклад, адміністратор повинен мати можливість змінювати структуру бази даних та налаштування системи, тоді як звичайний працівник складу повинен мати доступ лише до виконання основних операцій обліку. У зв'язку з цим доцільно впровадити більш гнучку систему ролей і прав доступу. Це дозволить підвищити рівень безпеки, забезпечити контроль дій користувачів та зменшити ризик випадкового пошкодження або видалення важливої інформації. Ще одним важливим напрямком розвитку є інтеграція системи із зовнішніми програмними продуктами. У сучасних умовах підприємства часто використовують декілька інформаційних систем одночасно, наприклад бухгалтерські програми, [CRM](#)-системи або сервіси для управління підприємством. Інтеграція між такими системами дозволяє автоматизувати обмін даними та уникнути дублювання інформації. Наприклад, дані про надходження або списання товарів можуть автоматично передаватися до бухгалтерської системи без необхідності повторного введення інформації вручну. Це не лише підвищить ефективність роботи персоналу, а й зменшить ймовірність виникнення помилок. З точки зору масштабування одним із найбільш важливих напрямків є перехід від локальної бази даних до клієнт-серверної архітектури. На даному етапі використання [SQLite](#) є доцільним рішенням для невеликих підприємств, однак зі збільшенням кількості користувачів та обсягу інформації можуть виникати обмеження щодо продуктивності та одночасного доступу до даних з різних місць. Використання серверних систем управління базами даних, таких як [MySQL](#) або [PostgreSQL](#), дозволить забезпечити підтримку багатокористувацького режиму роботи, підвищити рівень безпеки та покращити можливості резервного копіювання інформації. Це особливо актуально для підприємств, які мають декілька складів або працюють у різних містах чи країнах. Крім цього, перспективним напрямком є створення веб-версії системи або адаптація програмного забезпечення для роботи в мережевому середовищі. Підхід орієнтований на

веб системи, дозволить користувачам отримувати доступ до системи з різних пристроїв незалежно від їхнього місцезнаходження. Це може бути особливо корисним для керівників підприємств, які потребують швидкого доступу до актуальної інформації про стан складу в будь-який час. Також використання веб-технологій дозволить спростити процес оновлення програмного забезпечення та забезпечити централізоване зберігання даних оновлюючи систему лише в одному місці, а не окремо для кожного комп'ютера. Окрему увагу в подальшому розвитку системи доцільно приділити вдосконаленню інтерфейсу користувача. Зручність та простота використання програмного забезпечення безпосередньо впливають на ефективність роботи працівників та навантаження на пристрій. Подальше покращення інтерфейсу може включати оптимізацію навігації, удосконалення форм введення даних, адаптацію дизайну до сучасних стандартів та використання більш інтуїтивно зрозумілих елементів керування. Це дозволить зменшити час, необхідний для навчання користувачів, та підвищити комфорт роботи з програмою. Також потрібно звернути увагу на напрямок розвитку в реалізації підвищенні рівня надійності та захисту даних. Для цього доцільно реалізувати механізми автоматичного резервного копіювання та відновлення інформації. У випадку технічних несправностей або помилок користувача це дозволить уникнути втрати важливих даних та забезпечити безперервність роботи системи. Також можна впровадити використання сучасних технологій автоматизації складських процесів. Наприклад, систему можна доповнити підтримкою штрих-кодів або QR-кодів для швидкого обліку товарів. Це дозволить значно скоротити час на введення інформації та зменшити кількість помилок під час роботи персоналу. Розроблена система має значний потенціал для подальшого розвитку та вдосконалення. Основні напрямки модернізації пов'язані з розширенням функціональності, покращенням безпеки, підтримкою багатокористувацького режиму роботи та інтеграцією із зовнішніми сервісами. Реалізація таких можливостей дозволить адаптувати програмний продукт до складніших умов використання та забезпечить його ефективне застосування в діяльності підприємств різного масштабу. Таким чином, перспективи розвитку системи свідчать про можливість її подальшого вдосконалення відповідно до сучасних вимог інформаційних технологій та потреб бізнесу. Гнучкість архітектури та можливість масштабування роблять розроблену систему перспективною основою для створення більш складних та функціональних рішень у сфері автоматизації складського обліку. Висновки до розділу У другому розділі бакалаврської роботи розглянуто процес розробки програмної системи для автоматизації обліку запасів, починаючи від проектування архітектури та бази даних і закінчуючи реалізацією основних функціональних можливостей та їх тестуванням. Під час виконання розділу було визначено загальну архітектуру програмного забезпечення, яка базується на багаторівневому підході з поділом на рівні доступу до даних, бізнес-логіці та інтерфейсі користувача.

 **Знайдено плагіату: 0,01%**

id: 10

https://www.scs.kpi.ua/storage/2025/09/bazy_danyh_ta_zas
Було розроблено структуру бази даних, реалізовано механізм автоматичного створення бази даних та

таблиць, що призвело до спрощення процесу розгортання системи. Під час впровадження програмного забезпечення було розроблено ключові модулі, що забезпечують важливу функціональність складського обліку. Зокрема, реалізовано роботу з товарами, категоріями, постачальниками, а також механізм автентифікації користувачів. Графічний інтерфейс користувача було розроблено з використанням технології **Windows Forms**. Побудова інтерфейсу системи враховує принципи зручності та інтуїтивності. У рамках розділу також було проведено тестування програмного забезпечення, яке підтвердило правильність функціональності та відповідність системи. Крім того, було проведено порівняльний аналіз розробленої системи з існуючими аналогами. Було виявлено, що запропоноване рішення має багато переваг, зокрема, простоту використання, низькі витрати на впровадження та незалежність від Інтернету. Тому на другому розділі було повністю розроблено програмну систему для складського обліку, яка відповідає вимогам та може бути використана на практиці. Отримані результати підтверджують ефективність обраних методів і технологій і складають основу для формування загальних висновків бакалаврської роботи. ВИСНОВКИ У процесі виконання бакалаврської роботи на тему

 **Знайдено посилань: 0,01%**

id: 11

«Створення **Windows Form**-системи для автоматизації обліку товарів на складі»

було досягнуто поставленої мети та вирішено основні задачі, пов'язані з проектуванням і

реалізацією програмного забезпечення для автоматизації складського обліку. У першому розділі проаналізовано предметну область, розглянуто теоретичні основи складського обліку та досліджено сучасні програмні рішення, які можна використовувати для автоматизації цих процесів. Аналіз показує, що існуючі системи мають широкий спектр функціональних можливостей, але часто є складними у використанні або економічно недоцільними для малого бізнесу. Це підтвердило актуальність розробки власної системи, яка зосереджена на простоті, доступності та ефективності. На другому етапі було розроблено програмний комплекс, що реалізує основні функції складського обліку. Було розроблено архітектуру програмного забезпечення, яка базується на розподілі функціональності між окремими модулями. Це дозволило створити структуровану та зрозумілу систему, яку легко модифікувати та розширювати. Було створено базу даних, яка забезпечує зберігання всієї необхідної інформації про товари, категорії, постачальників, використання та рух товарів. Було реалізовано автоматизований метод створення бази даних, що спрощує процес встановлення та початку роботи з системою. У процесі впровадження програмного забезпечення було розроблено модулі для управління товарами, категоріями та постачальниками, а також механізм автентифікації користувачів. Особлива увага була приділена впровадженню обліку руху товарів, що забезпечує точність та надійність даних завдяки використанню транзакцій. Розроблено привабливий графічний інтерфейс користувача на основі технології [Windows Forms](#), що забезпечує ефективну взаємодію з системою. Інтерфейс дозволяє користувачам швидко та без додаткових клопотів виконувати всі необхідні операції. Тестування підтверджує точність системи та її відповідність вимогам. Було виявлено, що система стабільно виконує всі заплановані операції, забезпечує цілісність даних та правильно обробляє помилки. Порівняно з існуючими аналогами, розроблена система має багато переваг, таких як простота використання, низькі витрати на впровадження, незалежність від Інтернету та можливість подальшого розвитку. Це робить її ефективним рішенням для автоматизованого складського обліку на малих підприємствах. Таким чином, розроблений програмний проєкт повністю відповідає поставленим цілям та може бути використаний у практичній діяльності. Перспективою подальшого розвитку є покращення функціональності, зокрема, додавання аналітичних інструментів, інтеграція з іншими системами та покращення інтерфейсу користувача. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ Термін

” Знайдено посилань: 0%

id: 12

«Складський облік».

Верховна Рада України. 20.08.1999. URL: <https://zakon.rada.gov.ua/laws/?term=27597:40013> (дата звернення: 19.03.2026). Роман К. Основні операції складського обліку. [Abmcloud](#). Складський облік: методи, документація та автоматизація. 05.01.2026. URL: <https://ink.ua/tAdLWGLR3> (дата звернення: 19.03.2026). Картка (книга) складського обліку запасів [держсектор]. [Document](#). 22.12.2022. URL: <https://document.vobu.ua/doc/16849> (дата звернення: 19.04.2026). Автоматизована система складського обліку. [Sklad service](#). 01.05.2018. URL: <https://ssk.ua/ua/blog/avtomatizirovannaya-sistema-skladskogo-ucheta-364> (дата звернення: 20.03.2026). Анастасія С. [ERP vs CRM](#): ключові відмінності та особливості систем автоматизації бізнесу. [Gudhub](#). 14.03.2025. URL: <https://ink.ua/FmlIG3Dti6> (дата звернення: 20.03.2026). Хмарні сервіси для бізнесу: особливості використання. [Roapp](#). 08.01.2026. URL: <https://roapp.com.ua/blog/business-data-in-cloud/> (дата звернення: 20.03.2026). [De Novo Cloud Expert](#). Переваги та недоліки хмарних сервісів — чесно та без прикрас. [Denovo](#). 29.07.2025. URL: <https://ink.ua/iRV5NVOSQ> (дата звернення: 20.03.2026). 1. [ERP](#)-система. Що це і для чого вона потрібна?. [Softinform](#). ПУБЛІКАЦІЇ. URL: <https://ink.ua/aHZlfwOoS> (дата звернення: 20.03.2026). Герасименко Ю. Облік бізнесу у хмарі: варто чи ні. [Cityhost](#). Недоліки хмарних сервісів для обліку бізнесу. 27.07.2022. URL: <https://cityhost.ua/uk/blog/uchet-biznesa-v-oblake-stoit-ili-net.html> (дата звернення: 20.03.2026). [C Sharp](#). [Wikipedia](#). 23.09.2025. URL: https://uk.wikipedia.org/wiki/C_Sharp (дата звернення: 20.03.2026). [.NET](#). [Wikipedia](#). 29.01.2026. URL: <https://uk.wikipedia.org/wiki/.NET> (дата звернення: 20.03.2026). [Windows Forms](#). [Wikipedia](#). 16.05.2022. URL: https://uk.wikipedia.org/wiki/Windows_Forms (дата звернення: 20.03.2026). [SQLite](#). [Wikipedia](#). 28.11.2024. URL: <https://uk.wikipedia.org/wiki/SQLite> (дата звернення: 20.03.2026). Автентифікація (інформаційні технології) // Велика українська енциклопедія. URL: [https://vue.gov.ua/Автентифікація_\(інформаційні_технології\)](https://vue.gov.ua/Автентифікація_(інформаційні_технології)) (дата звернення: 20.03.2026). Петух А., Романюк О., Романюк О. БАЗИ ДАНИХ. МОВИ ЗАПИТІВ, УПРАВЛІННЯ ТРАНЗАКЦІЯМИ, РОЗПОДІЛЕНА ОБРОБКА ДАНИХ. Навчальні посібники Вінницького

національного технічного університету МЕРЕЖНІ ЕЛЕКТРОННІ ВИДАННЯ. URL: https://web.posibnyky.vntu.edu.ua/fitki/11petuh_bazdanyh_movy_zalitiv/31.htm#314 (дата звернення: 20.03.2026). Полотай О. ЛЕКЦІЯ. РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ. Віртуальний університет: навчальне середовище Львівського державного університету безпеки життєдіяльності. URL: <https://lnk.ua/djAuUXqVf> (дата звернення: 01.05.2026). Thomas S. Багатошаровий архітектурний стиль. LinkedIn. 28.09.2022. URL: <https://ua.linkedin.com/pulse/layered-architecture-style-thomas-saied?tl=uk> (дата звернення: 01.04.2026). Бізнес-логіка. Wikipedia. 29.11.2025. URL: <https://lnk.ua/S3rCCzxvx> (дата звернення: 01.04.2026). Інтерфейс користувача. Wikipedia. 21.07.2025. URL: <https://lnk.ua/lJfSwongM> (дата звернення: 01.04.2026). Реляційна база даних. Wikipedia. 27.07.2024. URL: <https://lnk.ua/f9fwBA8I3> (дата звернення: 01.04.2026). Нормалізація баз даних. Wikipedia. 02.01.2024. URL: <https://lnk.ua/4FC8tUcqD> (дата звернення: 01.04.2026). Джей Д. Що таке фільтрація даних?. Dovidka.biz. URL: <https://dovidka.biz.ua/shho-take-filtratsiya-danih> (дата звернення: 01.04.2026). Дизайн інтерфейсу користувача. Wikipedia. 14.02.2026. URL: <https://lnk.ua/UzEonPmRi> (дата звернення: 01.04.2026). Подія (інформатика). Wikipedia. 18.08.2022. URL: <https://lnk.ua/wnmTPIsvk> (дата звернення: 01.04.2026). Дистрибуція (логістика). Wikipedia. Процес руху товару. 17.10.2025. URL: <https://lnk.ua/Hwhlybqct> (дата звернення: 01.04.2026). Тестування програмного забезпечення. Wikipedia. 29.09.2025. URL: <https://lnk.ua/essyHQQ9R> (дата звернення: 01.04.2026). 1C. Wikipedia. 31.01.2026. URL: <https://uk.wikipedia.org/wiki/1%D0%A1> (дата звернення: 03.04.2026). Оптимізація. Wikipedia. 07.04.2024. URL: <https://lnk.ua/mNuSuQCFN> (дата звернення: 01.05.2026). Додаток А. Блок-схема роботи програми. Рис.А.1. Блок-схема роботи. Додаток Б. UML Діаграма класів. Рис. Б.1. UML Діаграма класів програми. Додаток В. Лістинг класу `StockMovementService.cs` using `System; using System.Collections.Generic; using Microsoft.Data.Sqlite; using WarehouseAccounting.Data; using WarehouseAccounting.Models; namespace WarehouseAccounting.Services { public class StockMovementService { public void Incoming(int productId, int quantity, string userLogin, string comment) =`

” Знайдено посилань: 0% id: 13

```

) { SqliteConnection connection = Database.GetConnection(); connection.Open();
SqliteTransaction transaction = connection.BeginTransaction(); SqliteCommand updateCmd =
connection.CreateCommand(); updateCmd.Transaction = transaction; updateCmd.CommandText =

```

” Знайдено посилань: 0,01% id: 14

```

"UPDATE Products SET Quantity = Quantity + @q WHERE Id = @d"

```

```

; updateCmd.Parameters.AddWithValue(

```

” Знайдено посилань: 0% id: 15

```

"@d",

```

```

quantity); updateCmd.Parameters.AddWithValue(

```

” Знайдено посилань: 0% id: 16

```

"@d",

```

```

productId); updateCmd.ExecuteNonQuery(); SqliteCommand insertCmd =
connection.CreateCommand(); insertCmd.Transaction = transaction; insertCmd.CommandText =
@

```

” Знайдено посилань: 0,02% id: 17

```

"INSERT INTO StockMovements (ProductId, ProductName, Quantity, MovementType, UserLogin,
MovementDate, Comment) VALUES (@p, @name, @q, 'IN', @u, @d, @c)"

```

```

; insertCmd.Parameters.AddWithValue(

```

” Знайдено посилань: 0% id: 18

```

"@d",

```

```

productId); ProductService productService = new ProductService(); Product product =
productService.GetById(productId); insertCmd.Parameters.AddWithValue(

```

” Знайдено посилань: 0% id: 19

```
"@name",
product.Name); insertCmd.Parameters.AddWithValue(
"Знайдено посилань: 0% id: 20
"@1",
quantity); insertCmd.Parameters.AddWithValue(
"Знайдено посилань: 0% id: 21
"@1",
userLogin); insertCmd.Parameters.AddWithValue(
"Знайдено посилань: 0% id: 22
"@1",
DateTime.Now.ToString(
"Знайдено посилань: 0% id: 23
"yyyy-MM-dd HH:mm:ss"
)); insertCmd.Parameters.AddWithValue(
"Знайдено посилань: 0% id: 24
"@1",
comment); insertCmd.ExecuteNonQuery(); transaction.Commit(); connection.Close(); } public
void Outgoing(int productId, int quantity, string userLogin, string comment =
"Знайдено посилань: 0% id: 25
""
) { SqlConnection connection = Database.GetConnection(); connection.Open();
SQLiteTransaction transaction = connection.BeginTransaction(); SQLiteCommand checkCmd =
connection.CreateCommand(); checkCmd.Transaction = transaction; checkCmd.CommandText =
"Знайдено посилань: 0,01% id: 26
"SELECT Quantity, Name FROM Products WHERE Id = @id"
; checkCmd.Parameters.AddWithValue(
"Знайдено посилань: 0% id: 27
"@id",
productId); SqlDataReader reader = checkCmd.ExecuteReader(); if (!reader.Read()) {
reader.Close(); transaction.Rollback(); connection.Close(); throw new Exception(
"Знайдено посилань: 0% id: 28
"Товар не знайдено"
); } int currentQty = reader.GetInt32(reader.GetOrdinal(
"Знайдено посилань: 0% id: 29
"Quantity"
)); string productName = reader.GetString(reader.GetOrdinal(
"Знайдено посилань: 0% id: 30
"Name"
)); reader.Close(); if (currentQty < quantity) { transaction.Rollback(); connection.Close(); throw new
Exception(
"Знайдено посилань: 0% id: 31
"Недостатньо товару на складі"
); } SQLiteCommand updateCmd = connection.CreateCommand(); updateCmd.Transaction =
transaction; updateCmd.CommandText =
"Знайдено посилань: 0,01% id: 32
"UPDATE Products SET Quantity = Quantity - @q WHERE Id = @id"
; updateCmd.Parameters.AddWithValue(
```

” Знайдено посилань: 0%	id: 33
"@_1", quantity); updateCmd.Parameters.AddWithValue(” Знайдено посилань: 0%	id: 34
"@_d", productId); updateCmd.ExecuteNonQuery(); SQLiteCommand insertCmd = connection.CreateCommand(); insertCmd.Transaction = transaction; insertCmd.CommandText = @ ” Знайдено посилань: 0,02%	id: 35
"INSERT INTO StockMovements (ProductId, ProductName, Quantity, MovementType, UserLogin, MovementDate, Comment) VALUES (@p, @name, @q, 'OUT', @_1, @_d, @_c)" ; insertCmd.Parameters.AddWithValue(” Знайдено посилань: 0%	id: 36
"@_q", productId); insertCmd.Parameters.AddWithValue(” Знайдено посилань: 0%	id: 37
"@_name", productName); insertCmd.Parameters.AddWithValue(” Знайдено посилань: 0%	id: 38
"@_1", quantity); insertCmd.Parameters.AddWithValue(” Знайдено посилань: 0%	id: 39
"@_1", userLogin); insertCmd.Parameters.AddWithValue(” Знайдено посилань: 0%	id: 40
"@_1", DateTime.Now.ToString(” Знайдено посилань: 0%	id: 41
"yyyy-MM-dd HH:mm:ss")); insertCmd.Parameters.AddWithValue(” Знайдено посилань: 0%	id: 42
"@_c", comment); insertCmd.ExecuteNonQuery(); transaction.Commit(); connection.Close(); } public List StockMovement GetMovements(DateTime fromDate, DateTime toDate, int? categoryId = null, int? supplierId = null) { List StockMovement movements = new List StockMovement (); SQLiteConnection connection = Database.GetConnection(); connection.Open(); string query = @"SELECT sm.Id, sm.ProductId, sm.MovementType, sm.Quantity, sm.UserLogin, sm.Comment, sm.MovementDate FROM StockMovements sm INNER JOIN Products p ON sm.ProductId = p.Id WHERE sm.MovementDate = @fromDate AND sm.MovementDate = @toDate"; if (categoryId != null) query += " AND p.CategoryId = @categoryId" ” Знайдено посилань: 0,01%	id: 43
"; if (supplierId != null) query += " AND p.SupplierId = @supplierId ” Знайдено посилань: 0,01%	id: 44
"; using (SQLiteCommand command = new SQLiteCommand(query, connection)) { command.Parameters.AddWithValue(" @fromDate ” Знайдено посилань: 0%	id: 45

```

", fromDate); command.Parameters.AddWithValue("
@toDate
” Знайдено посилань: 0,01% id: 46
", toDate); if (categoryId != null) command.Parameters.AddWithValue("
@categoryId
” Знайдено посилань: 0,01% id: 47
", categoryId); if (supplierId != null) command.Parameters.AddWithValue("
@supplierId", supplierId); using (SqliteDataReader reader = command.ExecuteReader()) { while
(reader.Read()) { StockMovement sm = new StockMovement { Id = reader.GetInt32(0), ProductId
= reader.GetInt32(1), MovementType = reader.GetString(2), Quantity = reader.GetInt32(3),
UserLogin = reader.GetString(4), Comment = reader.GetString(5), MovementDate =
DateTime.Parse(reader.GetString(6)) }; movements.Add(sm); } } } return movements; } } }
Додаток Г. Лістинг класу ProductService.cs using Microsoft.Data.Sqlite; using System; using
System.Collections.Generic; using WarehouseAccounting.Data; using
WarehouseAccounting.Models; namespace WarehouseAccounting.Services { public class
ProductService { public ProductService() { } public void Add(Product product) { using
(SqliteConnection connection = Database.GetConnection()) { connection.Open(); string query =
@"INSERT INTO Products (Code, Name, Barcode, Unit, Price, Quantity, CategoryId, SupplierId)
VALUES (@code, @name, @barcode, @unit, @price, @quantity, @categoryId, @supplierId)
” Знайдено посилань: 0,01% id: 48
"; using (SqliteCommand command = new SqliteCommand(query, connection)) {
command.Parameters.AddWithValue("
@code
” Знайдено посилань: 0,01% id: 49
", product.Code); command.Parameters.AddWithValue("
@name
” Знайдено посилань: 0,01% id: 50
", product.Name); command.Parameters.AddWithValue("
@barcode
” Знайдено посилань: 0,01% id: 51
", product.Barcode); command.Parameters.AddWithValue("
@unit
” Знайдено посилань: 0,01% id: 52
", product.Unit); command.Parameters.AddWithValue("
@price
” Знайдено посилань: 0,01% id: 53
", product.Price); command.Parameters.AddWithValue("
@quantity
” Знайдено посилань: 0,01% id: 54
", product.Quantity); command.Parameters.AddWithValue("
@categoryId
” Знайдено посилань: 0,01% id: 55
", product.CategoryId); command.Parameters.AddWithValue("
@supplierId
” Знайдено посилань: 0,03% id: 56
", product.SupplierId); command.ExecuteNonQuery(); } } } public void Update(Product product) {
using (SqliteConnection connection = Database.GetConnection()) { connection.Open(); string
query = @"
UPDATE Products SET Code=@code, Name=@name, Barcode=@barcode, Unit=@unit,

```

Price=@price, Quantity=@quantity, CategoryId=@categoryId, SupplierId=@supplierId WHERE Id=@id

” Найдено посилань: **0,01%** id: 57

```
using (SqlCommand command = new SqlCommand(query, connection)) {
    command.Parameters.AddWithValue("

```

@code

” Найдено посилань: **0,01%** id: 58

```
product.Code); command.Parameters.AddWithValue("

```

@name

” Найдено посилань: **0,01%** id: 59

```
product.Name); command.Parameters.AddWithValue("

```

@barcode

” Найдено посилань: **0,01%** id: 60

```
product.Barcode); command.Parameters.AddWithValue("

```

@unit

” Найдено посилань: **0,01%** id: 61

```
product.Unit); command.Parameters.AddWithValue("

```

@price

” Найдено посилань: **0,01%** id: 62

```
product.Price); command.Parameters.AddWithValue("

```

@quantity

” Найдено посилань: **0,01%** id: 63

```
product.Quantity); command.Parameters.AddWithValue("

```

@categoryId

” Найдено посилань: **0,01%** id: 64

```
product.CategoryId); command.Parameters.AddWithValue("

```

@supplierId

” Найдено посилань: **0,01%** id: 65

```
product.SupplierId); command.Parameters.AddWithValue("

```

@id

” Найдено посилань: **0,03%** id: 66

```
product.Id); command.ExecuteNonQuery(); } } } public void Delete(int id) { using
(SqliteConnection connection = Database.GetConnection()) { connection.Open(); string query =
"

```

DELETE FROM Products WHERE Id=@id

” Найдено посилань: **0,01%** id: 67

```
using (SqlCommand command = new SqlCommand(query, connection)) {
    command.Parameters.AddWithValue("

```

@id

” Найдено посилань: **0,03%** id: 68

```
id); command.ExecuteNonQuery(); } } } public List Product GetAll() { List Product products =
new List Product (); using (SqliteConnection connection = Database.GetConnection()) {
    connection.Open(); string query = "

```

```
SELECT Id, Code, Name, Barcode, Unit, Price, Quantity, CategoryId, SupplierId FROM Products";
using (SqlCommand command = new SqlCommand(query, connection)) { using
(SqliteDataReader reader = command.ExecuteReader()) { while (reader.Read()) { Product product
= new Product() { Id = reader.GetInt32(0), Code = reader.GetString(1), Name =
reader.GetString(2), Barcode = reader.GetString(3), Unit = reader.GetString(4), Price =
reader.GetDecimal(5), Quantity = reader.GetInt32(6), CategoryId = reader.GetInt32(7), SupplierId

```

```
= reader.GetInt32(8) }; products.Add(product); } } } return products; } public List Product
SearchByName(string name) { List Product allProducts = GetAll(); List Product result = new List
Product (); for (int i = 0; i allProducts.Count; i++) { if
(allProducts[i].Name.ToLower().Contains(name.ToLower())) { result.Add(allProducts[i]); } } return
result; } public Product GetById(int id) { Product? product = null; using (SqlConnection
connection = Database.GetConnection()) { connection.Open(); string query = @"SELECT Id, Code,
Name, Barcode, Unit, Price, Quantity, CategoryId, SupplierId FROM Products WHERE Id=@id
```

” Знайдено посилань: **0,01%**

id: **69**

```
"; using (SqlCommand command = new SqlCommand(query, connection)) {
command.Parameters.AddWithValue("
@id", id); using (SqlDataReader reader = command.ExecuteReader()) { if (reader.Read()) {
product = new Product() { Id = reader.GetInt32(0), Code = reader.GetString(1), Name =
reader.GetString(2), Barcode = reader.GetString(3), Unit = reader.GetString(4), Price =
reader.GetDecimal(5), Quantity = reader.GetInt32(6), CategoryId = reader.GetInt32(7), SupplierId
= reader.GetInt32(8) }; } } } return product; } } } Додаток Д. Лістинг класу AuthService.cs
using Microsoft.Data.Sqlite; using System.IO; using WarehouseAccounting.Data; using
WarehouseAccounting.Enums; using WarehouseAccounting.Models; namespace
WarehouseAccounting.Security { public class AuthService { public AuthService() { } public User?
Login(string login, string password) { using (SqlConnection connection =
Database.GetConnection()) { connection.Open(); string query = "SELECT Id, Login, PasswordHash,
Role FROM Users WHERE Login = @login
```

” Знайдено посилань: **0,01%**

id: **70**

```
"; using (SqlCommand command = new SqlCommand(query, connection)) {
command.Parameters.AddWithValue("
@login", login); using (SqlDataReader reader = command.ExecuteReader()) { if (reader.Read())
{ int id = reader.GetInt32(0); string dbLogin = reader.GetString(1); string dbPasswordHash =
reader.GetString(2); string roleString = reader.GetString(3); if (PasswordHasher.Verify(password,
dbPasswordHash)) { UserRole role = (UserRole)Enum.Parse(typeof(UserRole), roleString); User
user = new User() { Id = id, Login = dbLogin, Role = role }; return user; } } } } return null; }
public void Register(User user, string password) { using (SqlConnection connection =
Database.GetConnection()) { connection.Open(); string query = "INSERT INTO Users (Login,
PasswordHash, Role) VALUES (@login, @hash, @role)
```

” Знайдено посилань: **0,01%**

id: **71**

```
"; using (SqlCommand command = new SqlCommand(query, connection)) {
command.Parameters.AddWithValue("
@login", user.Login); command.Parameters.AddWithValue("@hash",
PasswordHasher.Hash(password)); command.Parameters.AddWithValue("@role",
user.Role.ToString()); command.ExecuteNonQuery(); } } } public List User GetAllUsers() { List
User users = new List User (); using (SqlConnection connection = Database.GetConnection()) {
connection.Open(); string query = "SELECT Id, Login, Role FROM Users"; using (SqlCommand
command = new SqlCommand(query, connection)) { using (SqlDataReader reader =
command.ExecuteReader()) { while (reader.Read()) { User user = new User() { Id =
reader.GetInt32(0), Login = reader.GetString(1), Role = (UserRole)Enum.Parse(typeof(UserRole),
reader.GetString(2)) }; users.Add(user); } } } } return users; } } } Додаток Е. Лістинг класу
Database.cs using Microsoft.Data.Sqlite; namespace WarehouseAccounting.Data { public static
class Database { private static readonly string _connectionString = "Data
Source=WarehouseDB.db"; public static SqlConnection GetConnection() { return new
SqlConnection(_connectionString); } } } Додаток Є. Лістинг класу MainForm.cs using System;
using System.Windows.Forms; using WarehouseAccounting.Enums; using
WarehouseAccounting.Forms; using WarehouseAccounting.Models; namespace
WarehouseAccounting.Forms { public partial class MainForm : Form { private User LoggedUser;
public MainForm(User user) { InitializeComponent(); LoggedUser = user; lblUser.Text =
"Користувач: " + LoggedUser.Login + " (" + LoggedUser.Role +
```

” Знайдено посилань: **0%**

id: **72**

```
");
; ApplyRolePermissions(); } private void ApplyRolePermissions() { if (LoggedUser.Role !=
```

```
UserRole.Admin) { //btnUsers.Enabled = false; } } private void btnProducts_Click(object sender,
EventArgs e) { ProductListForm form = new ProductListForm(); form.ShowDialog(); } private void
btnIncoming_Click(object sender, EventArgs e) { IncomingForm form = new
IncomingForm(LoggedUser); form.ShowDialog(); } private void btnOutgoing_Click(object sender,
EventArgs e) { OutgoingForm form = new OutgoingForm(LoggedUser); form.ShowDialog(); }
private void btnStockHistory_Click(object sender, EventArgs e) { StockHistoryForm form = new
StockHistoryForm(); form.ShowDialog(); } private void btnMovementReport_Click(object sender,
EventArgs e) { MovementReportForm form = new MovementReportForm(); form.ShowDialog(); }
private void btnCategories_Click(object sender, EventArgs e) { CategoryForm form = new
CategoryForm(); form.ShowDialog(); } private void btnSuppliers_Click(object sender, EventArgs e)
{ SupplierForm form = new SupplierForm(); form.ShowDialog(); } } } Додаток Ж. Лістинг класу
IncomingForm.cs using System; using System.Windows.Forms; using System.Xml.Linq; using
WarehouseAccounting.Models; using WarehouseAccounting.Services; namespace
WarehouseAccounting.Forms { public partial class IncomingForm : Form { private User
currentUser; private StockMovementService stockService; public IncomingForm(User user) {
InitializeComponent(); currentUser = user; stockService = new StockMovementService();
LoadProducts(); } private void LoadProducts() { ProductService productService = new
ProductService(); cmbProduct.Items.Clear(); foreach (Product product in productService.GetAll()) {
cmbProduct.Items.Add(product); } cmbProduct.DisplayMember =
```

```
” Знайдено посилань: 0% id: 73
"Name"
; cmbProduct.ValueMember =
” Знайдено посилань: 0% id: 74
"Id"
; } private void btnAdd_Click(object sender, EventArgs e) { if (cmbProduct.SelectedItem == null)
{ MessageBox.Show(
” Знайдено посилань: 0% id: 75
"Оберіть товар."
); return; } Product selectedProduct = (Product)cmbProduct.SelectedItem; int qty; if
(!int.TryParse(txtQuantity.Text, out qty) || qty == 0) { MessageBox.Show(
” Знайдено посилань: 0% id: 76
"Введіть правильну кількість."
); return; } string comment = txtComment.Text; try { stockService.Incoming(selectedProduct.Id,
qty, currentUser.Login, comment); MessageBox.Show(
” Знайдено посилань: 0% id: 77
"Товар додано на склад."
); txtQuantity.Clear(); txtComment.Clear(); } catch (Exception ex) { MessageBox.Show(
” Знайдено посилань: 0% id: 78
"Помилка: "
+ ex.Message); } } } }
```

Відмова від відповідальності:

Цей звіт повинен бути правильно інтерпретований та проаналізований кваліфікованою особою, яка несе відповідальність за оцінку!

Будь -яка інформація, наведена у цьому звіті, не є остаточною та підлягає ручному огляду та аналізу. Дотримуйтесь вказівок: [Рекомендації з оцінки](#)

Детектор Плагіату - Право знати автора! ☐ SkyLine LLC