Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп’ютерних наук

Допущено до захисту
Завідувач кафедри комп’ютерних наук
доктор технічних наук, професор
Андрій БОНДАРЧУК
« 01» червня 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня «бакалавр»

Спеціальність 122 Комп’ютерні науки

Освітня програма 122.00.01 Інформатика

Тема: ДЕСКТОПНИЙ ДОДАТОК ІНФОРМАЦІЙНОЇ ПІДТРИМКИ ОБЛІКУ ТА РЕАЛІЗАЦІЇ ЗООТОВАРІВ

Виконав
студент групи ІНБ-1-22-4.0д
Сидоренко Д.А
_____

Науковий керівник
Доктор філософії
Турукало А.В
_____

Київ – 2026

Київський столичний університет імені Бориса Грінченка
Факультет інформаційних технологій та математики
Кафедра комп'ютерних наук

«Затверджую»
Завідувач кафедри комп'ютерних наук
доктор технічних наук, професор
Андрій БОНДАРЧУК
«31»жовтня 2025 р.

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА
«Десктопний додаток інформаційної підтримки обліку та реалізації зоотоварів»

Виконавець – студент групи ІНБ-1-22-4.0д.
спеціальності 122 Комп'ютерні науки
освітньої програми 122.00.01 Інформатика
Сидоренко Дмитро Андрійович

1. Вихідні дані: інформація про бізнес-процеси та специфіку роботи адміністраторів ветеринарних клінік і зоомагазинів , офіційна документація мови програмування Python та фреймворку PyQt5 , технічна специфікація СКБД SQLite , принципи роботи з апаратним забезпеченням через віртуальні COM-порти (бібліотека pySerial), теоретичні основи проєктування реляційних баз даних.
2. Основні завдання: аналіз предметної області та недоліків ручного обліку; обґрунтування вибору інструментальних засобів розробки; проєктування багатошарової архітектури системи та реляційної структури бази даних; програмна реалізація модулів АРМ адміністратора (управління складом, оформлення продажів, контроль заборгованостей клієнтів); налаштування та інтеграція апаратного сканера штрих-кодів; тестування функціоналу додатка.
3. Пояснювальна записка: Обсяг близько 40 стор. формату А4 комп'ютерного набору з дотриманням вимог стандарту і методичних рекомендацій кафедри.
4. Графічні матеріали: діаграма прецедентів використання АРМ адміністратора , діаграма послідовностей обробки даних зі сканера штрих-кодів , ER-діаграма структури бази даних , скріншоти графічного інтерфейсу.
5. Додатки: Unittest проти pytest (Додаток А), Скрипт ініціалізації БД (Додаток Б), інфраструктура та код тестів (Додаток В, Г, Ґ та Д), тестування GUI (Додаток Е)
6. Строк подання роботи на кафедру «01» червня 2026р.

Науковий керівник
Доктор філософії
Турукало Андрій Валерійович
«___»_____2025 р.

Виконавець:
Сидоренко Дмитро Андрійович
«___»_____2025 р.

Календарний план

№	Етапи виконання роботи	Термін виконання	Примітка
1	Затвердження теми кваліфікаційної роботи бакалавра	31.10.25	Виконано
2	Аналіз проблеми, огляд існуючих аналогів та формування вимог до додатка	01.11.25 - 11.01.26	Виконано
3	Обґрунтування та вибір інструментальних засобів розробки	26.01.26 – 15.03.26	Виконано
4	Проєктування архітектури системи та структури реляційної бази даних	16.03.26- 31.03.26	Виконано
5	Програмна реалізація графічного інтерфейсу та основного функціоналу ПЗ для АРМ	01.04.26 – 15.04.26	Виконано
6	Інтеграція та налаштування апаратного сканера штрих-кодів, через віртуальний COM-порт	16.04.26 – 30.04.26	Виконано
7	Тестування розробленого програмного забезпечення та усунення виявлених недоліків	01.05.26 – 15.05.26	Виконано
8	Підготовка пояснювальної записки, графічних матеріалів, додатків.	25.05.25 – 12.06.26	Виконано
9	Захист кваліфікаційної роботи	18.06.26	

«Затверджую»
Науковий керівник
Доктор філософії,
Турукало А.В

_____

Індивідуальний план склав
Сидоренко Д.А
_____

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 62 с., 7 рис., 6 табл., 36 джерела, 7 додатків.

Актуальність. Сучасні заклади ветеринарної медицини поєднують надання медичних послуг із роздрібною реалізацією зоотоварів. Більшість існуючих систем є надлишковими або залежними від інтернету, через що облік часто ведеться вручну. Розробка спеціалізованого десктопного додатка для інформаційної підтримки та обліку є актуальним завданням для підвищення ефективності роботи адміністраторів.

Об'єкт дослідження: інформаційні процеси обліку та реалізації зоотоварів у закладах ветеринарної медицини

Предмет дослідження: програмні засоби та інформаційні технології розробки десктопного додатка для інформаційного забезпечення управління запасами та процесами реалізації зоотоварів.

Мета роботи: розробка десктопного додатка для інформаційної підтримки, обліку та реалізації зоотоварів, який дозволить оптимізувати рутинні процеси роботи адміністраторів та підвищити ефективність управління ресурсами ветеринарної клініки.

Основні завдання:

1. Аналіз існуючих аналогів та методів ручного обліку.
2. Обґрунтування вибору інструментальних засобів розробки.
3. Проєктування архітектури та структури реляційної бази даних.
4. Розробка графічного інтерфейсу та основного функціоналу АРМ з елементами CRM.
5. налаштування апаратного сканера штрих-кодів.

Основні результати. Розроблено автономний десктопний додаток на базі Python та PyQt5, що автоматизує оформлення роздрібних продажів, контроль складських залишків та ведення профілів клієнтів-боржників.

Спроєктовано локальну базу даних на базі SQLite із забезпеченням транзакційності операцій. Реалізовано асинхронну інтеграцію зі сканером штрих-кодів через віртуальний COM-порт.

Проведено порівняння з аналогами (ENOTE, JetVet, Торгсофт, Odoo, Checkbox), яке підтвердило переваги рішення у вигляді повної автономності та відсутності надлишкового медичного функціоналу.

Практичне значення дослідження полягає у можливості впровадження розробленого додатка в роботу ветеринарних клінік для автоматизації рутинних операцій, мінімізації ризику людських помилок, забезпечення оперативного контролю за товарами та зменшення фінансових втрат.

Ключові слова: Автоматизація, облік, зоотовари, ветклініка, Python, PyQt5, SQLite, АРМ, CRM, штрих-код.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

CRM	Customer Relationship Management
SQL	Structured Query Language
АРМ	Автоматизоване робоче місце
ПРРО	Програмний реєстратор розрахункових операцій
ERP	Enterprise Resource Planning
СКБД	Система керування базами даних
HID	Human Interface Device
CRUD	Create, Read, Update, Delete
UI	User Interface
COM	Communication port

ЗМІСТ

ВСТУП 4
РОЗДІЛ 1 6
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО ДОДАТКА ДЛЯ ОБЛІКУ ЗООТОВАРІВ 6
1.1 Дослідження предметної області та проблематики обліку у ветклініках 6
1.2 Огляд існуючих аналогів 7
1.3 Огляд технологій за інструментальних засобів для розробки десктопних додатків. 9
Висновки до розділу 1 13
РОЗДІЛ 2 15
ОБҐРУНТУВАННЯ ТА ВИБІР ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ 15
2.1 Обґрунтування та вибір мови програмування 15
2.2 Вибір фреймворку для створення графічного інтерфейсу 16
2.3 Обґрунтування вибору системи управління базами даних 17
2.4 Обґрунтування вибору засобів інтеграції з периферійним обладнанням ... 18
2.5 Обґрунтування вибору інструментальних засобів для автоматизованого тестування 20
2.6 Вибір інструменту для пакування та розгортання програмного забезпечення 20
Висновки до розділу 2 22
РОЗДІЛ 3 24
ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ОСНОВНОГО ФУНКЦІОНАЛУ ДОДАТКА 24
3.1 Архітектура системи та логіка взаємодії 24
3.2 Проєктування структури бази даних 27
3.3 Розробка графічного інтерфейсу 30

3.4 Програмна реалізація роботи сканера штрих-кодів у системі 32
3.5 Перспективи подальшого розвитку 33
Висновки до розділу 3 33
ОЦІНКА НАДІЙНОСТІ, ТЕСТУВАННЯ ТА БЕЗПЕКА СИСТЕМИ 35
4.1 Методологія тестування 35
4.2 Модульне тестування основних елементів бізнес-логіки 35
4.3 Функціональне тестування графічного інтерфейсу користувача 37
Висновки до розділу 4 38
ВИСНОВКИ 39
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 40
ДОДАТКИ 44

ВСТУП

Нині автоматизація бізнес-процесів є важливою складовою для забезпечення успішного функціонування підприємства, зокрема це стосується і реалізації зоотоварів. Незважаючи на існування великої кількості систем для автоматизації торгівлі та універсальних CRM-рішень, багато закладів досі продовжують використовувати застарілі методи на кшталт паперового обліку. У результаті цього знижується ефективність роботи персоналу та ускладнюється процес аналітики й управління запасами.

Актуальність роботи зумовлена специфікою роботи сучасних ветеринарних закладів, які поєднують надання медичних послуг із роздрібною реалізацією ветеринарних препаратів та зоотоварів. У досліджуваному закладі спостерігається часткова автоматизація: лікувальний процес уже ведеться за допомогою програмних засобів, у той час як облік та продаж товарів у зоомагазині при клініці досі здійснюється вручну в паперових журналах.

Такий підхід має чимало недоліків, зокрема значні витрати часу на ручну фіксацію кожної позиції. Це, у свою чергу, підвищує ймовірність помилок через людський фактор, особливо при інтенсивному потоці клієнтів. Окрім цього, паперовий облік не забезпечує необхідної інформативності для швидкого моніторингу залишків на складі, аналізу популярності окремих товарів та своєчасного виявлення продукції, термін придатності якої добігає кінця. Відсутність такого контролю призводить до втрати потенційного прибутку.

Також важливим аспектом варто виділити хаотичний облік заборгованостей клієнтів, втрата даних про які напряму шкодить фінансовому стану всього підприємства. Тому розробка спеціалізованого програмного забезпечення, здатного автоматизувати облік та реалізацію товарів, є цілком актуальним завданням.

Метою роботи є розробка десктопного додатка для інформаційної підтримки, обліку та реалізації зоотоварів, який дозволить оптимізувати рутинні процеси роботи адміністраторів та підвищити ефективність управління

ресурсами ветеринарної клініки.

Щоб досягти заданої мети, потрібно розв’язати наступні завдання:

1. Провести аналіз предметної області, виявити основні недоліки методів ручного обліку та сформулювати технічні вимоги до програмного продукту.

2. Обґрунтувати вибір методів та інструментальних засобів розробки.

3. Спроєктувати структуру бази даних для надійного збереження інформації про наявні товари, здійснені продажі та облік заборгованостей клієнтів.

4. Розробити та протестувати програмний продукт, що реалізує функції автоматизованого робочого місця (АРМ) адміністратора з елементами CRM для ведення складу, продажу товарів та фінансового обліку.

Об’єкт дослідження – інформаційні процеси обліку та реалізації зоотоварів у закладах ветеринарної медицини

Предмет дослідження – програмні засоби та інформаційні технології розробки десктопного додатка для інформаційного забезпечення управління запасами та процесами реалізації зоотоварів.

Методи дослідження. Для аналізу предметної області та виявлення недоліків існуючих підходів застосовано методи системного аналізу; для проєктування структури бази даних – положення теорії реляційних баз даних; для безпосередньої програмної реалізації додатка – методи об’єктно-орієнтованого програмування та емпіричного тестування.

Практичне значення одержаних результатів полягає у можливості впровадження розробленого додатка в роботу ветеринарних клінік та зоомагазинів. Його використання дозволить автоматизувати рутинні операції адміністраторів, знизити ризик виникнення помилок через людський фактор, забезпечити оперативний контроль за наявними товарами та мінімізувати фінансові втрати, спричинені недоліками ручного обліку.

РОЗДІЛ 1

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО ДОДАТКА ДЛЯ ОБЛІКУ ЗООТОВАРІВ

1.1 Дослідження предметної області та проблематики обліку у ветклініках

При аналізі ринку спеціалізованого програмного забезпечення для ветеринарної сфери з’являється розуміння, що сучасні клініки являють собою екосистему, що складається з різних відділів, таких як медична лікарня, лабораторія та магазин зоотоварів, до яких входять також і аптечні товари [1].

Процеси надання медичних послуг, такі як огляди, хірургічні втручання та вакцинації тварин, мають сувору специфіку і вимагають ведення детальної історії хвороб, планування щеплень та інтеграції з медичним обладнанням. Зважаючи на складність цих процесів, вони зазвичай уже автоматизовані за допомогою спеціалізованих медичних систем. У той же час управління продажами та складський облік є відокремленими бізнес-процесами. На практиці цей напрямок може бути винесений за межі основної медичної системи. Насамперед це зумовлено тим, що для виконання швидких завдань адміністратора використання перевантаженого медичного інтерфейсу часто є недоцільним і може негативно позначатися на швидкості обслуговування. Для ефективної роботи важливо мати під рукою саме релевантні інструменти.

Як наслідок, деякі клініки навіть сьогодні продовжують вести облік зоотоварів у ручному режимі, використовуючи паперові журнали. У такому форматі адміністратори змушені фіксувати всі продані позиції та їх вартість вручну, а підрахунок виручки здійснювати наприкінці зміни за допомогою калькулятора. Усе це вимагає що значних витрат часу, підвищує ймовірність помилок, сильно ускладнює відстеження товарних залишків та майже унеможливлює ведення статистики проданих товарів, що може призвести до закупівлі неліквідного товару або ж дефіциту ходових позицій.

Ведення списків щодо заборгованостей клієнтів також виходить в окремий процес, що створює хаос у роботі адміністраторів і стає причиною прямих фінансових збитків підприємства.

Для вирішення цих проблем компанії переходять до автоматизованих рішень, важливою частиною яких є управління базою клієнтів. Відповідно до термінології: «CRM (Customer Relationship Management, Управління взаємовідносинами з клієнтами) – це програмне забезпечення, що дозволяє автоматизувати управління взаємодіями та даними клієнтів протягом усього життєвого циклу. Основна мета CRM-системи – оптимізувати процеси та покращити відносини між компанією та її замовниками, що в результаті сприяє зростанню доходів та розвитку бізнесу»[5].

Хоча розроблений у рамках даної роботи додаток не є повноцінною CRM-системою, він використовує її головний принцип, а саме централізоване управління даними взаємодії з клієнтами, у даному випадку – це ведення інформації щодо заборгованостей клієнтів.

Враховуючи вищесказане, розробка програмного рішення фокусується саме на потребах зоомагазину при ветеринарній клініці. Цільовий додаток має функціонувати як прикладне програмне забезпечення АРМ адміністратора з елементами CRM. Він буде виконувати вузький спектр задач: швидке оформлення роздрібних продажів, автоматичний контроль складських залишків та ведення профілів клієнтів-боржників, не перевантажуючи інтерфейс медичним функціоналом.

1.2 Огляд існуючих аналогів

Щоб обґрунтувати доцільність розробки власного програмного забезпечення було проведено порівняльний аналіз існуючих на ринку програмних рішень.

На розгляд було взято системи різних класів: спеціалізовані ветеринарні CRM, універсальні десктопні системи обліку, глобальні ERP-системи та програмні реєстратори розрахункових операцій, або ж скорочено - ПРРО.

Для порівняння було обрано наступні програмні продукти:

1. ENOTE – являє собою комплексну хмарну систему управління ветеринарними клініками, яка забезпечує повний цикл автоматизації: від запису на прийом до лабораторних досліджень та складського обліку[6].

2. JetVet – спеціалізована CRM-система для ветеринарного бізнесу. Орієнтована на оптимізацію роботи лікарів та адміністраторів, має вбудовані інструменти фінансового контролю, ведення історії пацієнтів та контролю складу[7].

3. Торгсофт – це програмне забезпечення для автоматизації ведення фінансів, складу та управління на підприємстві. Вона допомагає фіксувати всі операції, стежити за рухом товарів і грошових коштів, формувати звітність і аналізувати бізнес. Має орієнтацію на ритейл[8].

4. Odoo – являє собою глобальну модульну ERP-систему, яка дозволяє автоматизувати будь-які бізнес-процеси підприємства, включно з POS, складом, бухгалтерією та управлінням персоналом[9].

5. Checkbox – це сервіс ПРРО, котрий надає бізнесу можливість фіскалізувати розрахункові операції в електронному вигляді. Він представляє з себе хмарний сервіс, що є цифровою альтернативою традиційним апаратним касовим апаратам[10].

Для оцінювання систем було обрано перелік критеріїв, які базуються на виявлених раніше недоліку ручного обліку та специфічних вимогах до автоматизації роботи адміністраторів. А саме: архітектура системи, залежність від інтернет-з’єднання, складність освоєння інтерфейсу, наявність надлишкового для касира функціоналу та можливість обліку заборгованостей. Результати порівняння наведено у таблиці 1.1

Виходячи з даних порівняльної таблиці, можна побачити, що жодне з готових рішень не задовольняє повною мірою потреби зоомагазину при клініці. Перш за все, більшість із них є залежними від інтернету, що в умовах його відсутності блокує можливість повноцінної роботи адміністратора. Також такі програми, як ENOTE, Odoo, JetVet та Торгсофт, мають багато надлишкового

функціоналу на кшталт медичних карток чи складних налаштувань, які уповільнюють роботу та потребують додаткового часу на навчання. Checkbox хоч і не має такого недоліку та чудово справляється зі своїми прямими задачами, втім, він є ПРРО і не має функціоналу для ведення інформації щодо заборгованостей клієнтів.

Таблиця 1.1

Порівняльний аналіз існуючих рішень та розроблюваного ПЗ

Програмний продукт	Архітектура системи	Залежність від інтернет-з'єднання	Складність освоєння інтерфейсу	Функціонал обліку заборгованостей
ENOTE	Хмарна (SaaS)	Повна	Висока	Наявний
JetVet	Хмарна (SaaS)	Повна	Середня	Наявний
Торгсофт	Десктопна	Відсутня	Висока	Наявний
Odoo	Хмарна / Web	Часткова (офлайн для POS)	Висока	Наявний
Checkbox	Хмарна / Web	Часткова (до 36 год офлайн) [10]	Низька	Відсутній
Розроблюване ПЗ	Десктопна	Відсутня	Низька	Наявний

Таким чином, розробка власного програмного рішення є технічно обґрунтованою. Створюване ПЗ поєднує в собі автономність роботи, низьку складність навчання завдяки відсутності зайвих модулів та наявність цільового функціоналу для продажів, контролю складських залишків і фінансових зобов’язань клієнтів.

1.3 Огляд технологій за інструментальних засобів для розробки десктопних додатків.

Щоб реалізувати ПЗ для автономного АРМ необхідно обрати три базові компоненти, які стануть скелетом додатка, а саме – мову програмування, фреймворк для створення графічного інтерфейсу та систему управління базами даних. Далі буде наведено огляд популярних рішень на ринку.

Мови програмування:

C#(.NET) – Вважається промисловим стандартом для розробки корпоративного ПЗ для операційної системи Windows. Славиться своєю високою продуктивністю та строгою типізацією. Має потужний інструментарій для створення різноманітних десктопних програм. Втім, для використання цієї екосистеми потрібні мати певні специфічна знання, на кшталт вміння працювати з мовою розмітки XAML, що, у свою чергу, підвищує поріг входження та може уповільнити розробку невеликих проєктів.

Python – Мова програмування, що має динамічну типізацію, орієнтована на гнучкість та швидкість написання коду. Має лаконічний синтаксис та величезну кількість різноманітних бібліотек, які дають змогу прискорити цикл розробки. Завдяки гнучкості динамічною типізації проєкти легше адаптувати під нові завдання. Із недоліків можна відзначити її інтерпретованість, через що вона поступається в швидкодії компільованим мовам, та потребує більше системних ресурсів.

Java – Являє собою мову програмування, головною особливістю якої є її кросплатформність та розвинена модель безпеки. За допомогою віртуальної машини Java мова здатна виконувати ідентичний байт-код на різних операційних системах без необхідності у повторній компіляції. Безпека ж базується на суворій типізації, вбудованих механізмах ізоляції та відсутності прямого доступу до адрес комірок пам'яті. До недоліків можна віднести відносно високе споживання оперативної пам’яті через роботу віртуальної машини та збирачів сміття, які є процесами автоматичного управління пам'яттю, що сприяють ефективній роботі програм на Java [15]. Порівняння мов програмування представлено у таблиці 1.2

Таблиця 1.2

Порівняння мов програмування

Критерій	**C# (.NET)**	**Python**	**Java**
Типізація	Статична, сувора	Динамічна	Статична, сувора
Середовище виконання	CLR (.NET Runtime)	Інтерпретатор CPython	JVM (Java Virtual Machine)
Швидкість виконання	Висока (CLR + JIT)	Нижча (інтерпретована)	Висока (JVM + JIT)
Швидкість розробки	Середня	Висока	Середня
Поріг входження	Середній	Низький	Середній
Екосистема бібліотек	Розвинена	Розвинена	Розвинена
Споживання ресурсів	Середнє (CLR + GC)	Середнє	Середнє (JVM + GC)

Фреймворки для створення GUI: WPF (C#) – є графічною підсистемою у складі .NET, що надає змогу створювати складні інтерфейси використовуючи апаратний прискорювач DirectX для рендерингу. Потребує глибокого розуміння мови розмітки XAML, та для ефективного використання важливим є використання архітектурного шаблону MVVM, який спрямований на чітке розділення графічного інтерфейсу та прикладної логіки[16].

JavaFX (Java) – є сучасним фреймворком для створення графічних інтерфейсів в екосистемі Java, який наразі є стандартом для цієї платформи. Підтримує стилізацію візуальних компонентів за допомогою CSS, що дозволяє гнучко налаштовувати інтерфейс. Має доволі значні вимоги до оперативної пам’яті та необхідність включення оточення JavaFX Runtime до інсталяційного пакету.

PyQt(Python) – являє собою програмну оболонку для кросплатформної бібліотеки Qt, написаної мовою програмування C++. Фреймворк надає великий набір готових і оптимізованих віджетів, а однією з головних переваг є механізм сигналів і слотів, який використовується для обробки подій. До недоліків можна віднести специфічні умови ліцензування та значне збільшення розміру програми як наслідок наявності великої кількості залежностей.

Tkinter(Python) – є стандартною бібліотекою мови програмування Python для створення графічних інтерфейсів. Головними перевагами є низька вага та відсутність потреби у встановленні додаткових залежностей. Втім, фреймворк

має і вагомі недоліки, а саме: обмежені засоби стилізації компонентів, що призводить до застарілого вигляду інтерфейсу, а також недостатню кількість інструментів для відображення великих масивів даних чи будування складних табличних структур.

Порівняння фреймворків для створення графічного інтерфейсу наведено у таблиці 1.3

Таблиця 1.3

Порівняння фреймворків для створення GUI

Критерій	**WPF (C#)**	**JavaFX (Java)**	**PyQt5 (Python)**	**Tkinter (Python)**
Базова мова	C# / XAML	Java / FXML	Python (обгортка Qt/C++)	Python (Tcl/Tk)
Набір віджетів	Широкий	Широкий	Широкий	Обмежений
Підтримка табличних структур	Повна (DataGrid)	Повна (TableView)	Повна (QTableWidget)	Відсутня (лише Treeview)
Стилізація	XAML-стилі	CSS	QSS	Мінімальна
Обробка подій	MVVM, прив'язка даних	Обробники подій	Сигнали і слоти	Прямі колбеки
Кросплатформність	Тільки Windows	Повна	Повна	Повна
Поріг входження	Вищий (XAML, MVVM)	Середній	Середній	Низький
Розмір залежностей	Середній	Великий (JavaFX Runtime)	Великий (Qt-бібліотеки)	Мінімальний
Ліцензія	MIT (.NET)	GPLv2 з Classpath Exception	GPL / Комерційна	PSF (вільна)

Системи управління базами даних:

Клієнт-серверні СКБД (PostgreSQL, MySQL) – це потужні системи, які є оптимальними для мережевих рішень з багатьма користувачами. Вони забезпечують високий рівень безпеки та підтримку паралельних запитів, що є надважливим, коли зміни до бази вносяться більш ніж одною людиною одночасно.

До недоліків відноситься потреба у розгортанні та підтримці окремого серверного процесу, і необхідність регулярного технічного адміністрування.

Вбудовані СКБД (SQLite) – належать до категорії безсерверних рішень, і не потребують додаткових налаштувань. Уся база зберігається локально у вигляді єдиного файлу, що у свою чергу забезпечує максимальну автономність та легкість у розгортанні додатка на кінцевому пристрої. Ключовим обмеженнями цього рішення є відсутність повноцінної підтримки одночасних запитів від кількох користувачів, через що, цей варіант не є оптимальним для колективних систем.

Порівняння систем керування базами даних наведено у таблиці 1.4

Таблиця 1.4

Порівняння СКБД

Критерій	**Клієнт-серверні СКБД (PostgreSQL, MySQL)**	**Вбудована СКБД (SQLite)**
Архітектура	Клієнт-серверна, окремий серверний процес	Безсерверна, бібліотека вбудована в застосунок
Розгортання	Потребує встановлення і налаштування сервера	Не потребує інсталяції, база, бо є єдиним файлом
Адміністрування	Регулярне технічне обслуговування	Не потребує
Паралельні запити	Повна підтримка	Обмежена (блокування на запис)
Багатокористувацький режим	Підтримується	Не підтримується
Безпека даних	Висока (автентифікація та ролі)	Базова (захист на рівні ОС)
Автономність	Залежить від серверного процесу	Повна

Висновки до розділу 1

У ході робіт над першим розділом було проведено системний аналіз предметної області та визначено основні вимоги до розробки програмного забезпечення для АРМ адміністратора, за результатами роботи можна зробити наступні висновки:

1. Аналіз бізнес-процесів: Дослідження існуючих методів роботи показало, що ведення касового обліку, контролю складських залишків та реєстру заборгованостей клієнтів у ручному форматі або за допомогою перевантажених медичних систем суттєво знижує ефективність роботи персоналу. Обґрунтовано необхідність розробки відокремленого програмного рішення для зоомагазину при клініці з елементами CRM.

2. Вимоги до додатка**:** Порівняльний аналіз існуючих рішень показав, що жодне з готових програмних забезпечень не задовольняє повною мірою потреби цільового робочого місця. Визначено, що створюваний додаток повинен бути незалежним від інтернет-з’єднання, не мати надлишкового функціоналу та відрізнятися низькою складністю освоєння для більшості працівників.

3. Огляд технологій та інструментальних засобів: У ході огляду популярних мов програмування, фреймворків для створення графічних інтерфейсів та систем управління базами даних можна побачити, що сучасні рішення пропонують широку варіативність у питаннях швидкодії, апаратних вимог та складності реалізації. Виходячи з цього підрозділу можна сформувати оптимальний технологічний стек для розробки додатка.

РОЗДІЛ 2

ОБҐРУНТУВАННЯ ТА ВИБІР ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ РОЗРОБКИ

2.1 Обґрунтування та вибір мови програмування

Розробка ПЗ для АРМ адміністратора потребує використання мови програмування, яка зможе забезпечити швидку ітеративну розробку, стабільну та ефективну роботу з базами даних та підтримку актуальних графічних фреймворків. Врахувавши огляд, проведений у першому розділі, вибір упав саме на Python. Підґрунтям для цього рішення став порівняльний аналіз з мовами C++ та Java. Щодо першої, то хоча C++ забезпечує максимальну швидкодію за рахунок прямого управління пам’яттю, для виконання поставлених завдань такий рівень низькорівневого контролю є надлишковим. Через необхідність ручного керування ресурсами значно зростає час розробки бізнес-логіки та підвищується ризик появи критичних помилок. З іншого ж боку, ми маємо Java, яка пропонує безпечне середовище з автоматичним керуванням пам’яттю, що покращує роботу додатка, але ціною за це є необхідність у написанні значної кількості шаблонного коду, який забезпечуватиме працездатність ПЗ, та побудова складних ієрархій класів навіть для простих операцій. У свою чергу, Python пропонує високорівневі структури даних та динамічну типізацію, що дозволяє сфокусуватися саме на реалізації бізнес-процесів. Простий і лаконічний синтаксис забезпечує високу швидкість прототипування та помітно зменшує загальну кодову базу проєкту[21]. Це, у свою чергу, спрощує подальше масштабування та підтримку системи. Звісно, важливим фактором також є екосистема мови. Завдяки наявності великої кількості оптимізованих бібліотек, зокрема вбудованих, на кшталт sqlite3, яка створена для роботи з реляційними базами даних, можна реалізувати ключовий функціонал ПЗ без залучення сторонніх сервісів.

Окрім об’єктивних технічних переваг, вирішальним фактором у виборі саме цієї мови став наявний досвід роботи з її екосистемою, що сприятиме кращій якості кінцевого продукту.

2.2 Вибір фреймворку для створення графічного інтерфейсу

Графічний інтерфейс є особливо важливою частиною проєкту, оскільки від його вигляду, швидкодії та стабільності безпосередньо залежить ефективність обслуговування клієнтів адміністратором. Тому як основний інструмент розробки GUI було обрано фреймворк PyQt5. Цей вибір було зумовлено порівнянням із базовою бібліотекою Tkinter та специфікою завдань. Головним обмеженням останньої є суттєво нижчий рівень функціональності при роботі з великими масивами даних та побудові складних табличних структур: наявний у Tkinter віджет ttk.Treeview сильно поступається за можливостями та продуктивністю аналогічним рішенням PyQt5. У середовищі, коли адміністратор має постійно взаємодіяти з різними частинами системи, це обмеження є дуже відчутним, і у випадку використання Tkinter для таких завдань це призвело б до падіння швидкодії додатка. Натомість PyQt5 вже з коробки надає набір потужних та оптимізованих віджетів, які забезпечують швидку обробку та плавне відображення інформації. Як приклад готового рішення можна навести QTableWidget, призначенням якого є відображення, редагування та керування табличними даними. Не менш вагомим аргументом на користь PyQt5 стали його широкі можливості зі стилізації компонентів, завдяки чому можна зручно та зрозуміло оформляти елементи інтерфейсу для різних задач. Через обмежені засоби стилізації у Tkinter буває набагато складніше виконувати ті ж задачі, на додачу до цього, кінцевий вигляд додатка без використання додаткових модулів, на кшталт Tkinter ttk, буде виглядати відверто застарілим. Механізм QSS, що інтегрований у PyQt5, дає можливість гнучко налаштовувати інтерфейс через зовнішні файли стилів, що дозволяє створити сучасний вигляд вікон та забезпечити високий рівень UX-дизайну без втручання в алгоритми обробки подій.

Якщо ж дивитися з точки зору архітектури ПЗ на подійно-орієнтований характер взаємодії у PyQt5, в основі якого лежить механізм сигналів і слотів, то ми отримаємо значно більшу гнучкість у порівнянні зі стандартною системою зв’язування подій у Tkinter. Взаємодія між елементами управління базується на технологічній логіці, де «сигнал генерується, коли відбувається певна подія; слот є будь-яким Python-викликом, який виконується у відповідь на отриманий сигнал»[17]. Враховуючи, що ПЗ передбачає інтеграцію з апаратним сканером штрих-кодів, архітектура PyQt5 дозволяє без особливих проблем організувати асинхронне перехоплення даних та їх передачу поміж шарами програми без потреби створювати жорсткі залежності між класами.

2.3 Обґрунтування вибору системи управління базами даних

Ефективне функціонування ПЗ для АРМ адміністратора напряму пов’язане з надійністю, швидкістю та стабільністю підсистеми збереження даних. Оскільки розроблюваний додаток на нинішньому етапі свого розвитку позиціонується як локальне десктопне рішення для автоматизації одного робочого місця, розгортання повноцінних клієнт-серверних систем для управління базами даних на кшталт PostgreSQL чи MySQL наразі є надлишковим. Вони потребують встановлення та постійного функціонування окремого сервера, котрий вимагає регулярного адміністрування і також підвищує вимоги до апаратних ресурсів ПК. У масштабах локального додатка, що працює на одному пристрої, архітектура клієнт-сервер створює додаткові точки відмови на кшталт блокування портів брандмауером або раптової зупинки сервісу СКБД та зумовлює зайві витрати часу на міжпроцесну взаємодію і мережевий стек, навіть при роботі через локальний інтерфейс.

Врахувавши ці обмеження, як оптимальне рішення для наявного рівня задач було обрано SQLite, яка є вбудованою реляційною СКБД. Її головною особливістю можна виділити те, що вона не має окремого серверного процесу, а її рушій інтегрується у кодову базу Python за допомогою вбудованої бібліотеки sqlite3. Також вона працює за принципом «zero-configuration», тобто не потребує

первинних налаштувань для початку роботи[25]. Це дозволяє забезпечити максимальну автономність ПЗ та легкість його розгортання на кінцевому пристрої. Уся структура бази даних зберігається в одному локальному файлі на диску, що суттєво спрощує обслуговування системи. У свою чергу, відсутність серверних затримок та мережевого обміну забезпечує максимальну швидкість виконання запитів в умовах однокористувацького режиму роботи, що повністю відповідає архітектурі розроблюваного ПЗ. Незважаючи на свою компактність, SQLite є повноцінною реляційною системою, яка підтримує стандартний синтаксис SQL і гарантує повну транзакційність, механізм якої у контексті баз даних являє собою послідовність однієї або декількох операцій з базою даних, які виконуються як єдиний робочий блок [26]. Завдяки використанню стандартної мови запитів, масштабування проєкту та перехід до хмарних СКБД буде суттєво спрощено, що знизить потенційні витрати часу у майбутньому.

2.4 Обґрунтування вибору засобів інтеграції з периферійним обладнанням

Надійність та швидкість функціонування АРМ адміністратора ветеринарної клініки суттєво залежить від ефективності інтеграції програмного забезпечення з апаратним комплексом, зокрема зі сканером штрих-кодів. При проєктуванні підсистеми взаємодії з сканером розглядалися наступні архітектурні рішення:

1. Використання сканера в режимі емуляції клавіатури(HID)
2. Режим емуляції COM порту із програмним перехопленням даних.

Перший підхід є найпростішим у реалізації, бо операційна система сприймає сканер як звичайну клавіатуру, яка автоматично вставляє зчитані символи у поточне активне текстове поле графічного інтерфейсу додатка. Схема працююча, але має суттєві недоліки, головних з яких є необхідність постійно контролювати, щоб активним було саме поле вводу штрих-коду, у противному випадку дані можуть бути внесені некоректно, що призведе до збоїв чи інших проблем при оформленні чека.

Альтернативним варіантом у межах HID-режиму є використання глобального перехоплювача подій клавіатури у поєднанні з унікальним префіксом або у налаштуваннях сканера, що дає змогу програмно відрізнити введення сканера від введення користувача. Однак такий підхід ускладнює логіку обробки подій і є не сильно надійним коли у середовищі активні одразу декілька застосунків.

Врахувавши ці обмеження, основним для розробки ПЗ для АРМ було другий підхід, а саме – переведення сканера у режим роботи через віртуальний COM порт, для роботи з яким обрано бібліотеку pySerial, яка надає доступ до системних COM-портів у середовищі ОС Windows через високорівневий API мови Python. Вибір саме цієї бібліотеки зумовлений наступними перевагами:

1. Асинхронне автономне перехоплення даних: бібліотека дає змогу організувати безперервний моніторинг порту в окремому фоновому потоці виконання, завдяки чому програма перехоплює штрих-код безпосередньо з буфера драйвера послідовного порту, незалежно від того, яке вікно додатка зараз активне і де знаходиться курсор. Це усуває необхідність ручного фокусування на полях введення та підвищує загальний рівень зручності використання системи.

2. Надійність та швидкість передачі. Взаємодія через pySerial забезпечує читання даних безпосередньо з буфера драйвера послідовного порту з передбачуваними таймаутами, що дозволяє детерміновано обробляти потік байтів на рівні застосунку. Також бібліотека містить гнучкі інструменти для конфігурування параметрів з'єднання та тонкого налаштування таймаутів зчитування.

3. Ізоляція бізнес-логіки: використання послідовного порту дозволяє чітко розмежувати потік даних від сканера та потік введення з клавіатури користувача. Це прибирає ризик випадкового перемішування символів штрих-коду з текстом, який адміністратор може паралельно вводити в іншій програмі.

Хоча використання pySerial потребує додаткових зусиль для написання для написання багатопотокового коду порівняно з методами на основі HID-емуляції, цей цілком компенсується високою стабільністю.

2.5 Обґрунтування вибору інструментальних засобів для автоматизованого тестування

Для реалізації автоматизованих модульних та інтеграційних тестів було обрано фреймворк pytest. Його вибір обґрунтовано порівнянням зі стандартною бібліотекою unittest, а також тим, що на сьогодні він є індустріальним стандартом у сфері тестування Python-застосунків. І хоча unittest надає достатній базовий функціонал, він потребує написання значної кількості шаблонного коду, тоді як pytest пропонує сучасніший та ефективніший підхід до організації тестової бази. Зокрема:

1. Лаконічність синтаксису: тести оголошуються як звичайні функції з використанням стандартного оператора assert, що зменшує обсяг кодової бази та підвищує її читабельність.
2. Гнучкий механізм фікстур: дозволяє керувати станом тестового середовища з можливістю налаштування області видимості та повторного використання між модулями.
3. Автоматичне виявлення тестів: фреймворк самостійно сканує директорії проєкту та ідентифікує всі файли і функції з префіксом test_.

Детальне порівняння pytest та unittest наведено у додатку А

2.6 Вибір інструменту для пакування та розгортання програмного забезпечення

Щоб забезпечити зручність для кінцевого користувача та спростити процес розгортання додатку на цільовому пристрої, постає завдання конвертувати вихідний код програми у виконуваний модуль. Враховуючи, що інтерпретатор Python та потрібні залежності зазвичай відсутні на звичайних ПК, програмний продукт повинен містити всі необхідні для його роботи в операційній системі компоненти.

На практиці для реалізації цього завдання існують два основних підходи:

1. Заморожування – являє собою архівацію вихідного байт-коду разом із необхідним середовищем виконання.

2. AOT-компіляція – є процесом перекладу вихідного коду високого рівня у низькорівневі мови програмування, на кшталт C, з подальшим збиранням бінарного файлу системними засобами.

Було проведено порівняння трьох популярних інструментів, а саме: PyInstaller, Nuitka та cx_Freeze. Результати порівняння наведено у таблиці 2.1.

Проведене порівняння дає змогу зробити висновок, що всі засоби мають власну область ефективного застосування. cx_Freeze є найменш гнучким під час роботи з розгалуженими залежностями графічних фреймворків. Компілятор Nuitka надає максимальний рівень безпеки у контексті захисту від недозволеної декомпіляції коду, проте ціною за це є суттєво більший час компіляції та потреба у налаштованому середовищі компіляторів MSVC чи GCC.

Враховуючи специфіку розроблюваного додатка, важливою є безпомилкова обробка складних динамічних імпортів, у чому позитивно виділяється PyInstaller, який має найбільш зрілу екосистему автоматичного аналізу модулів, що допомагає мінімізувати ризики втрати динамічних бібліотек під час інсталяції.

Таблиця 2.1

Порівняння PyInstaller, Nuitka та cx_Freeze

Критерій порівняння	PyInstaller	Nuitka	cx_Freeze
Архітектурний підхід	Заморожування байт-коду, створення ізольованого середовища виконання	AOT-трансляція в код на мові C та подальша компіляція	Заморожування байт-коду у вигляді відокремленої структури
Продуктивність виконання	Базова швидкість інтерпретатора Python без оптимізації	Оптимізована, з можливим прискорення обчислень на 10-30%	Базова швидкість інтерпретатора Python без оптимізації
Захист інтелектуальної власності	Низький, бо можлива швидка декомпіляція архіву до вихідних .py файлів.	Високий, завдяки генерації машинного коду стійкого до реверс-інжинірингу	Низький, через те що код доступний у напіввідкритому вигляді як .pyc файли
Час збирання проєкту	Висока швидкість (процес триває від кількох секунд до кількох хвилин)	Низька швидкість (вимагає тривалого часу та значних ресурсів CPU)	Висока швидкість (лінійне пакування без глибокого аналізу)
Автоматичне визначення залежностей	Відмінне, завдяки розвиненій системі плагінів та хуків для складних бібліотек	Добре, із мінусом у вигляді потреби періодичного ручного налаштування параметрів компілятора	Задовільне, бо часто вимагає ручного опису динамічних залежностей у конфігурації

Тому, врахувавши такі критерії, як стабільність та надійність інтеграції сторонніх пакетів, для реалізації збірки програмного модуля в інсталяційний файл було обрано PyInstaller.

Висновки до розділу 2

Другий розділ обґрунтував вибір технологічного стека для розробки додатка для АРМ. Результатом порівняльного аналізу став вибір мови програмування Python як основної, оскільки вона забезпечує високу швидкість розробки, лаконічний синтаксис та має розвинену екосистему бібліотек. Для створення графічного інтерфейсу обрано фреймворк PyQt5, що надає потужний набір готових та оптимізованих віджетів, гнучкий механізм стилізації QSS та архітектуру сигналів і слотів, яка спрощує організацію асинхронної взаємодії між компонентами системи. Як СКБД обрано SQLite, що являє собою вбудоване

рішення без серверного процесу, яке відповідає наявним вимогам до застосунку та надає максимальну автономність і простоту розгортання. Щоб інтегрувати сканер штрих-кодів у обраному режимі, а саме віртуального COM-порту, було обрано бібліотеку pySerial, що дозволить здійснювати незалежне від фокусу активного вікна перехоплення даних у фоновому потоці та розмежувати потоки введення від сканера і клавіатури користувача. Для забезпечення якості програмного продукту було обрано фреймворк pytest як інструмент автоматизованого тестування. Порівняльний аналіз з бібліотекою unittest підтвердив його переваги з точки зору лаконічності синтаксису, гнучкості механізму фікстур та зручності параметризації тестових сценаріїв. Для пакування програмного продукту у самодостатній виконуваний модуль обрано інструмент PyInstaller. Порівняльний аналіз з альтернативами Nuitka та cx_Freeze засвідчив, що PyInstaller має найбільш зрілу екосистему автоматичного аналізу залежностей та розвинену систему плагінів і хуків, що мінімізує ризики втрати динамічних бібліотек під час розгортання на цільовому пристрої. Як результат, отримано технологічний стек, що цілком відповідає функціональним вимогам до системи.

РОЗДІЛ 3

ПРОЄКТУВАННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ОСНОВНОГО ФУНКЦІОНАЛУ ДОДАТКА

3.1 Архітектура системи та логіка взаємодії

Додаток побудований на принципі багатошарової архітектури, метою якої є поділ складних процесів на менші керовані рівні.[18] Це дає змогу чітко розділити зони відповідальності компонентів. Використання фреймворку PyQt5 зумовлює подійно-орієнтований характер взаємодії об’єктів, реалізований через механізм сигналів та слотів.

За своєю структурою додаток поділяється на три автономні рівні, а саме:

- Рівень представлення - графічний інтерфейс, що включає класи MainWindow, CheckoutWidget та WarehouseDialog. Відповідає за візуалізацію та перехоплення дій користувача.
- Сервісний рівень - інкапсулює бізнес-логіку. Сервіси CheckoutService та ComScannerService координують операції, що охоплюють декілька таблиць БД, забезпечуючи транзакційність.
- Рівень доступу до даних - репозиторії inventory_repository та sales_repository, що ізолюють SQL-запити до SQLite.

Візуалізація функціональних можливостей системи на високому рівні абстракції представлена у вигляді діаграми на (рис 3.1). Актором діаграми є «Адміністратор». Визначено такі базові прецеденти: «Оформити продаж», що через зв’язки «include» включає «Керувати чеком» і «Провести оплату»; «Переглянути склад» із розширеннями «extend» до «Оприбуткувати товар» та «Керувати товарами (CRUD)», де «CRUD – це абревіатура, яка стосується чотирьох основних функцій, необхідних для реалізації програми постійного зберігання: створення (create), читання (read), оновлення (update) та видалення (delete) [19].»; а також «Переглянути журнал продажів», який розширюється («extend») прецедентами «Керувати боргами» та «Оформити повернення».

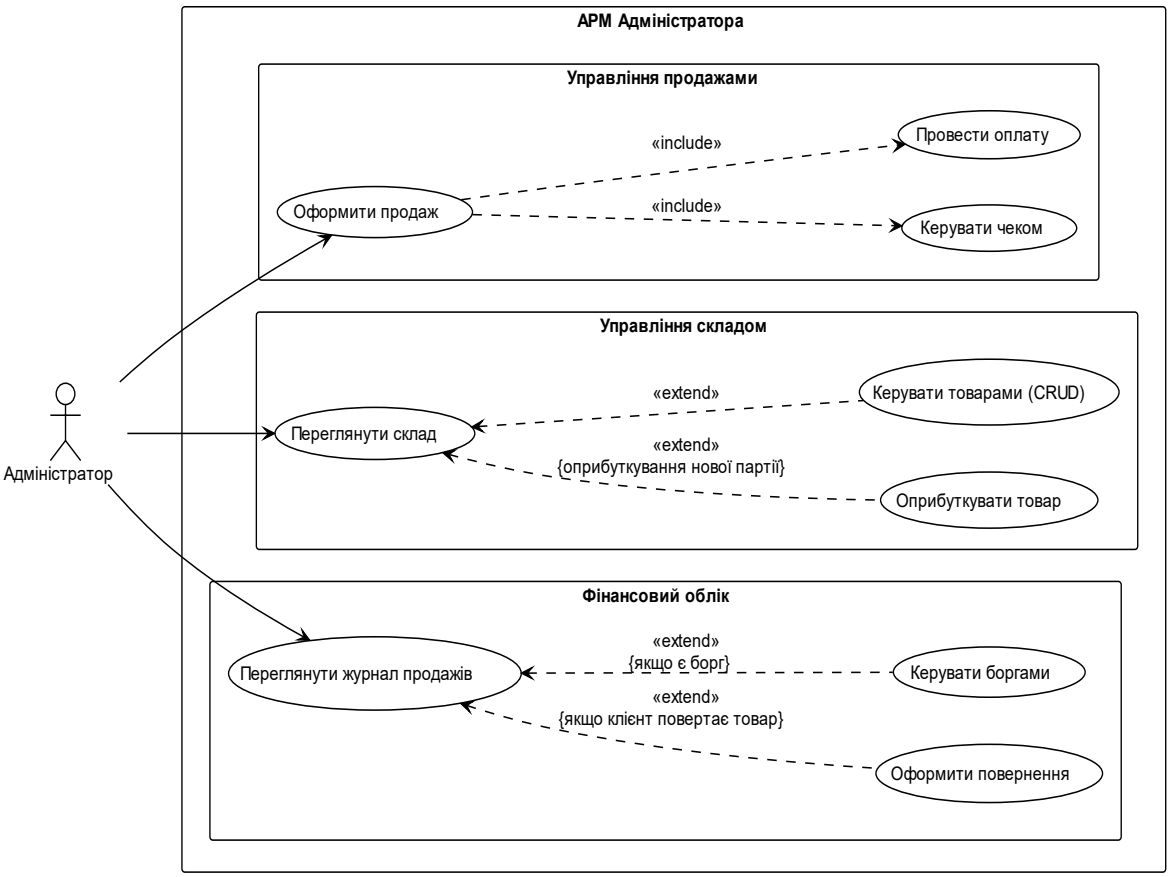

Рисунок 3.1. Кейси використання

Логіка обробки подій і шлях проходження даних у системі грунтуються на асинхронному зчитуванні інформації. Одним з найбільш важливих процесів є обробка даних зі сканера штрих-кодів у режимі роботи через COM-порту. Сам COM-порт – це фізичний інтерфейс на комп’ютері, що служить для обміну даними між комп’ютером і пристроями підключеними до нього. Він може бути реалізований як за допомогою спеціальних роз’ємів, так і можуть бути створені програмними методами [20]. У даному випадку, порт – віртуальний:

1. Апаратна ініціалізація: пристрій асинхронно передає байти на послідовний порт. Сервіс ComScannerService використовує окремий потік QThread для читання, що відокремлює низькорівневий ввід та запобігає блокуванню UI.

2. Обробка потоку: воркер _ReaderWorker накопичує дані в буфері до ідентифікації символу-термінатора (\r або \n), після чого емітує сигнал barcode_scanned.

3. Пошук та мапінг: метод _handle_scanned_barcode інтерфейсу MainWindow перехоплює сигнал та ініціює запит до InventoryRepository. Репозиторій виконує вибірку з бази даних і трансформує сирі рядки в об'єкт товару (Product).

4. Умовне розгалуження: залежно від результату пошуку в БД, система або викликає метод add_item для додавання позиції в чек, або ініціює сценарій обробки виняткової ситуації..

Принцип взаємодії об’єктів під час цього процесу відображено на (рис 3.2).

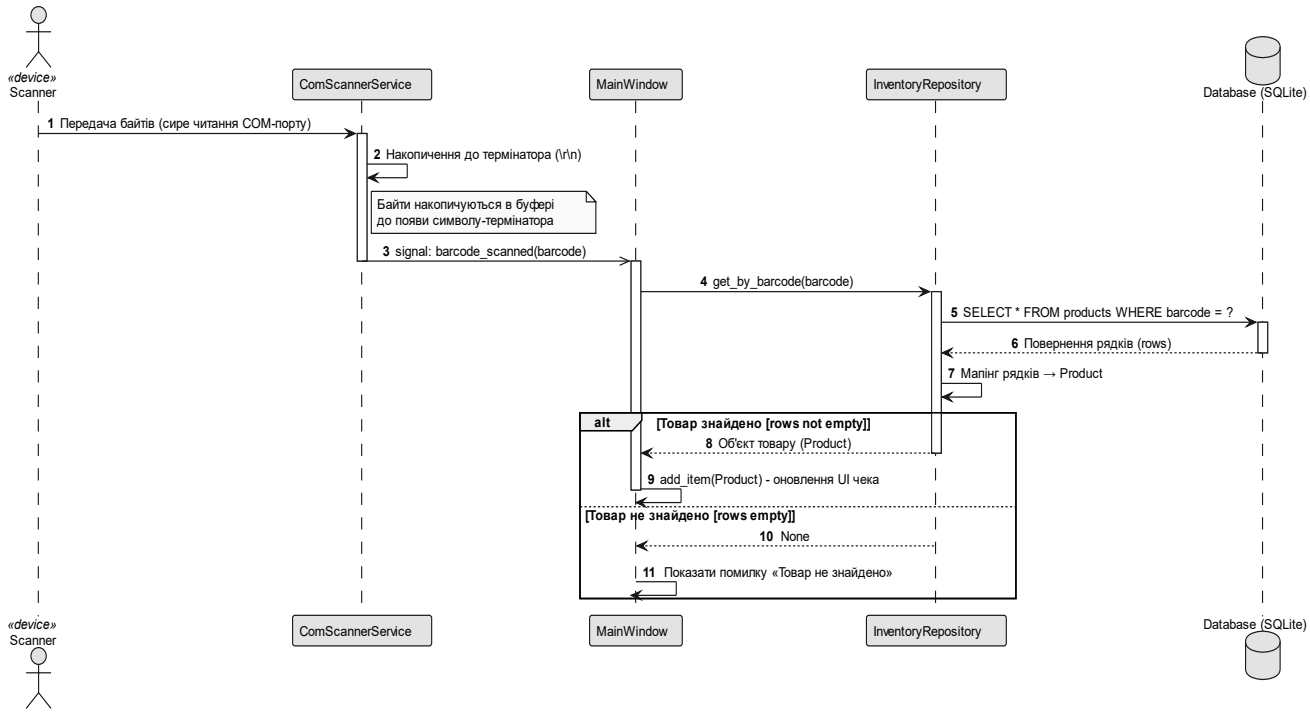

Рисунок 3.2. Діаграма послідовності процесу сканування та обробки штрихкоду товару

На діаграмі зображено лінії життя таких компонентів: Scanner (що є зовнішнім пристроєм зі стереотипом <<device>>), ComScannerService, MainWindow, InventoryRepository та Database (SQLite). Потік повідомлень починається від Scanner до ComScannerService (де проходить сире читання порту), далі асинхронний сигнал barcode_scanned передається до MainWindow.

Потім MainWindow викликає метод get_by_barcode() у InventoryRepository, який виконує SELECT-запит до Database та здійснює мапінг отриманих рядків у сутність програми.

Завершується цикл у межах умовного оператора alt: за умови успішного знаходження запису виконується виклик add_item() у віджеті чека для оновлення UI, а у разі порожньої відповіді від бази даних – інтерфейс виводить повідомлення «Товар не знайдено».

Використання такої архітектури допомагає забезпечити високу простежуваність і надійність при обробці вводу-виводу, через те що кожен етап має суворо визначену зону відповідальності відповідно до принципу єдиної відповідальності (SRP).

3.2 Проєктування структури бази даних

Проєктування бази даних є фундаментальним етапом розробки, адже від її архітектури залежить швидкодія програми та надійність збереження інформації зоомагазину. Структура БД була спроєктована так, щоб уникати дублювання інформації та забезпечити її цілісність під час роботи з касою та складом. SQL-скрипт ініціалізації бази винесений у додаток Б.

Візуалізація зв'язків між таблицями показана на ER-діаграмі на (рис 3.3).

На діаграмі зображено п'ять базових таблиць: product_list (каталог товарів), receipts (загальна інформація про чеки), receipt_items (вміст чека), sales_journal (журнал продажів) та debt_payments (відстеження виплат боргів). Таблиця receipts має зв'язок «один до багатьох» (1:N) із таблицями receipt_items та debt_payments. Таблиця product_list також пов'язана відношенням 1:N із вмістом чека та журналом, що було реалізовано через механізм зовнішніх ключів (FOREIGN KEY).

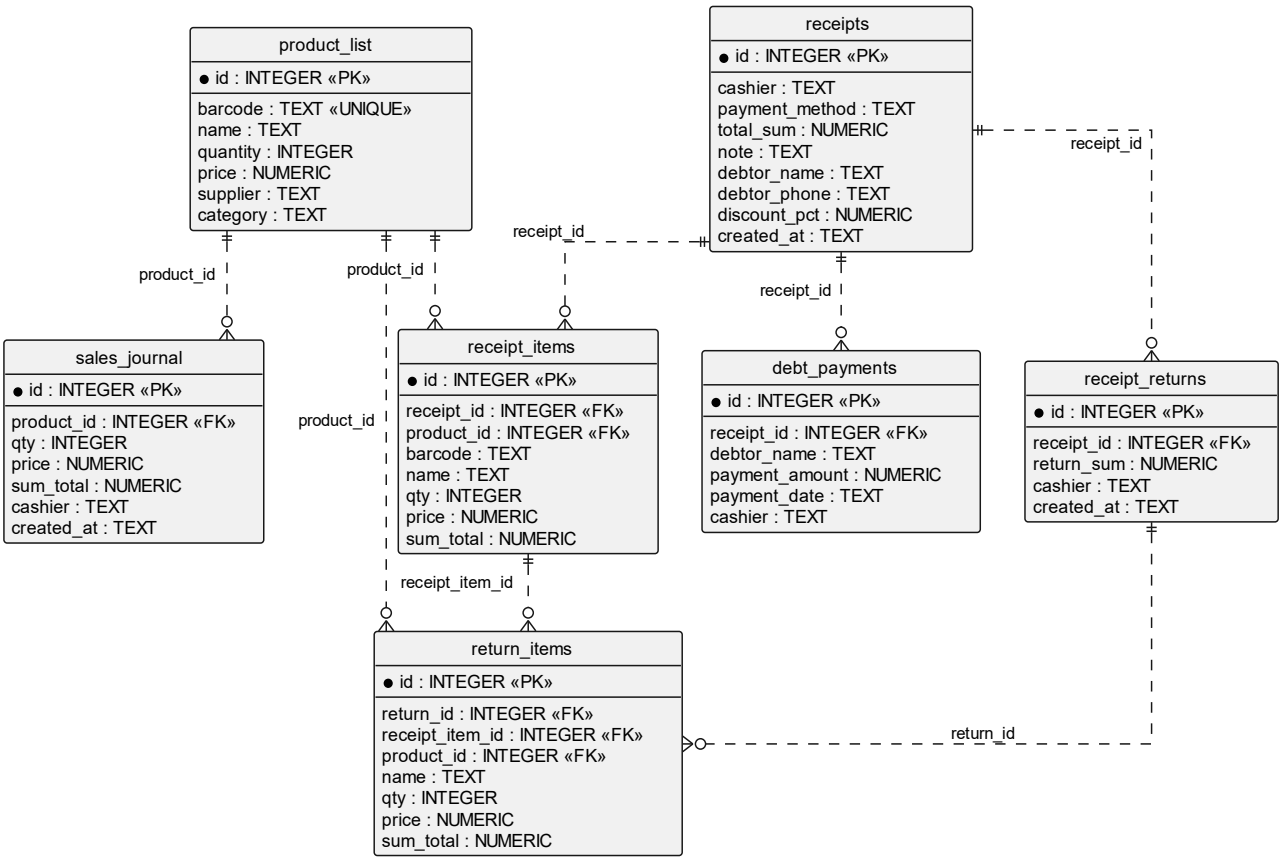

Рисунок. 3.3. Схема фізичної моделі бази даних

Схема бази даних складається з таких ключових таблиць:

- Таблиця product_list (Склад товарів): Зберігає актуальний перелік номенклатури та її залишки. Містить унікальний ідентифікатор id, штрих-код, найменування, кількість, ціну, а також дані про постачальника та категорію. Для кількості та ціни на рівні БД встановлено обмеження CHECK, щоб унеможливити випадкове збереження від'ємних значень.
- Таблиця receipts (Заголовки чеків): Фіксує базову інформацію про транзакцію. Містить дані про касира, спосіб оплати, підсумкову суму та час створення. Додаткові атрибути note, debtor_name, debtor_phone та discount_pct забезпечують обробку знижок і оформлення боргових зобов'язань конкретних клієнтів.
- Таблиця receipt_items (Позиції чека): Деталізує склад кожного чека, зв'язуючи його з товаром через зовнішні ключі receipt_id та product_id. Таблиця реалізує архітектурний патерн свідомої денормалізації: вона зберігає історичний

зліпок даних на момент продажу (barcode, name, qty, price, sum_total). Це гарантує, що сума в архівних чеках ніколи не зміниться, навіть якщо в майбутньому ціна чи назва товару в головному каталозі буде відредагована. Суть полягає в тому, щоб сума в старих чеках не змінювалася, якщо в майбутньому ціна товару в каталозі зросте або впаде.

- Таблиця sales_journal (Журнал продажів): Веде наскрізний транзакційний облік кожної проданої одиниці для спрощення аналітики та швидкого формування звітів без необхідності складних об'єднань (JOIN) з чеками.
- Таблиця debt_payments (Реєстр виплат): Відповідає за облік погашення боргів. Зв'язується з оригінальним борговим чеком через receipt_id та фіксує ім'я боржника, суму внеску (payment_amount), дату платежу та касира, який його прийняв.
- Таблиці receipt_returns та return_items (Повернення): Структурно дублюють логіку чеків для коректного фінансового обліку операцій повернення товару.

Використання SQLite дозволяє групувати SQL-запити в атомарні транзакції, що є дуже важливим для надійності системи.

Наприклад, під час продажу створення запису чека в receipts, додавання позицій у receipt_items та списання товару зі складу йдуть як єдина неподільна операція. За умови успішного виконання зміни фіксуються командою COMMIT. У разі незапланованого закриття програми чи системного збою операція автоматично скасовується командою ROLLBACK.

Це гарантує захист бази від збереження часткових, пошкоджених або неточних даних.

3.3 Розробка графічного інтерфейсу

Архітектура GUI орієнтована на високу швидкість обробки транзакцій та зручність повсякденної роботи адміністратора. При розробці UX-дизайну основний акцент було зроблено на тому, щоб користувач міг виконувати рутинні касові та складські операції з мінімальними витратами часу. Зокрема, завдяки налаштованому перехопленню подій від апаратного сканера штрих-кодів, адміністратор має змогу швидко додавати товари до чека без необхідності ручного пошуку або введення даних.

- Логіка навігації та динамічна зміна контенту. Головне вікно застосунку є базовим контейнером інтерфейсу. Перемикання між головним екраном з пошуком і екраном оформлення чека відбувається всередині одного вікна за рахунок стеку віджетів (QStackedWidget):

перед користувачем не відкривається окремого вікна для оформлення чека, а замість цього змінюється лише активна сторінка вмісту. Після успішної авторизації відкривається головне вікно на початковій сторінці: логотип системи та центральне поле для пошуку товару або вводу штрих-коду. Коли за допомогою сканера, або підтвердження вводу у полі пошуку, програма перемикається на екран кошика й додає позицію до кошика, починаючи формування нового чеку. Ці дії відбуваються у межах одного вікна. Вигляд головного та екрану оформлення чеку представлені у (рис 3.4) та (рис 3.5).

- Багатовіконний режим і паралельні операції: Модуль «Склад» (WarehouseDialog) реалізовано у вигляді незалежного вікна. Після натискання відповідної кнопки на головному екрані відкриваються нове вікна поверх головного, що дає змогу адміністратору переглядати потрібну інформацію на складі. Вигляд вікна складу представлено у (рис 2.6)

- Бічна панель: Реалізоване меню забезпечує доступ до списку категорій товарів, та відповідно товарів обраної категорії. Подвійне натискання на обраний товар переносить користувача на екран оформлення чека й автоматично додає цю позицію до чека.

Рисунок 3.4. Вигляд головного екрану додатку

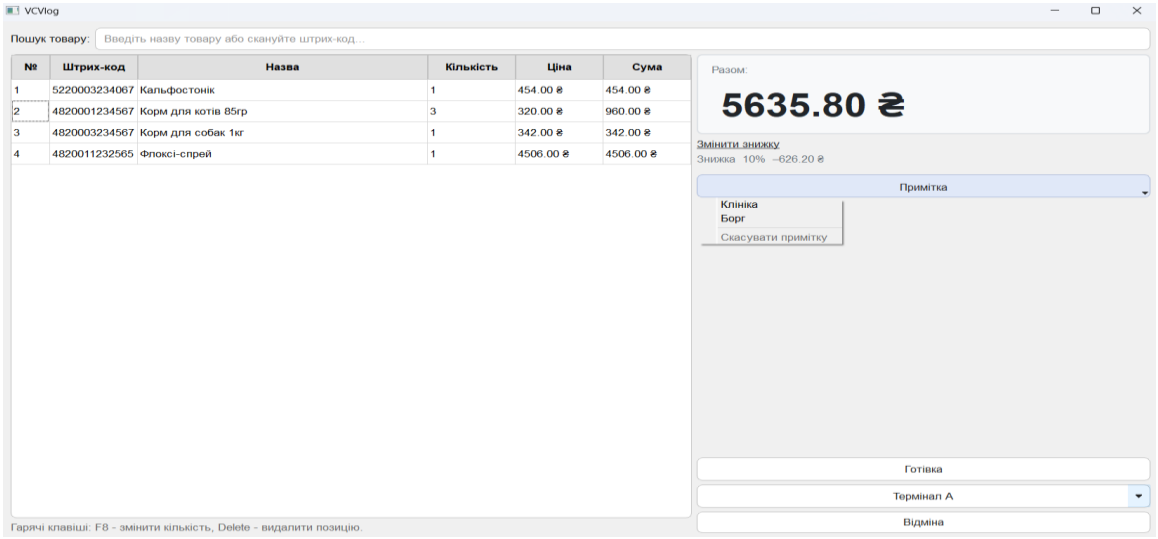

Рисунок 3.5. Вигляд екрану формування чека

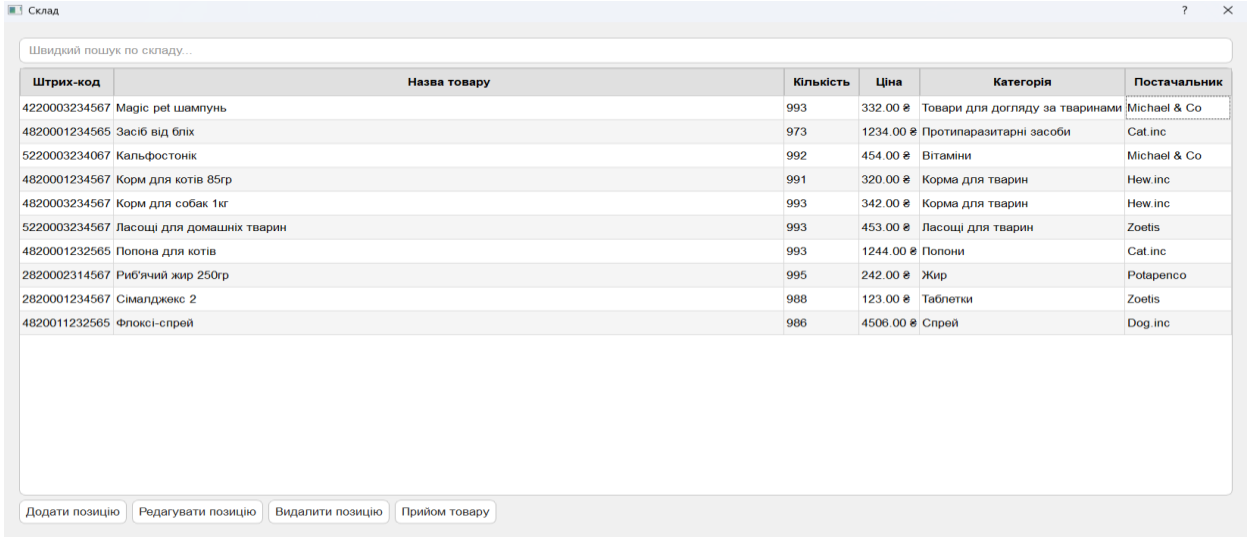

Рисунок 3.6. Вигляд інтерфейса складу

3.4 Програмна реалізація роботи сканера штрих-кодів у системі

Взаємодія зі сканером штрих-кодів була винесена в окремий сервісний рівень, для того, щоб низькорівнева робота з каналом даних не перемішувалась з відображенням форм. Обраний режим сканера та параметрів з’єднання зберігається у зовнішньому JSON файлі конфігурації, що дозволяє змінювати параметри роботи сканера без перезапуску застосунку.

Архітектура сканування. Додатком підтримуються два різні режими роботи зі сканерами штрих-кодів:

- Послідовний порт (COM). Дані зчитуються у вигляді потоку байтів за допомогою бібліотеки pySerial, яка призначена для роботи з послідовним портом. Розбір кадру завершується за символами-термінаторами, а саме повернення каретки або нового рядка. Блокуюче читання виконується в окремому потоці виконання, тож інтерфейс користувача не зупиняється на очікуванні порту. Головною перевагою цього підходу є незалежність від фокусу вводу в системі: сканеру не потрібно друкувати у активний віджет як клавіатура. Готовий рядок зі штрих-кодом передається в основну логіку застосунку для пошуку товару.
- Режим емуляції клавіатури (HID). Сканер подає символи так, ніби їх набирають на клавіатурі. Для збору цього типу вводу використовується фільтр подій на рівні застосунку, який за потреби перехоплює натискання клавіш до стандартної роздачі подій віджетам, але з урахуванням винятків для окремих полів, де ручний ввід залишається звичайним. Для того, щоб відрізняти ввід сканером від повільного вводу користувача, використовується інтервал між вводом символів та таймер паузи після останнього символу. Коли таймер добігає кінця аналізується вміст активного поля, і виконується евристична перевірка того, чи зміст рядка нагадує штрих-код за формою. Після цього виконується пошук товару за кодом у даних інвентаризації, і у разі успішного знаходження позиції відбувається додавання товару до чеку, або ж до відповідного поля, наприклад при заповненні карточки товару на складі.

3.5 Перспективи подальшого розвитку

Розроблений додаток закладає надійний технологічний фундамент для подальшого масштабування та функціонального розширення системи за такими напрямами:

- Модернізація архітектури даних**,** шляхом переходу від локальної до клієнт-серверної СКБД але з максимальним збереженням можливості роботи офлайн. Це дозволить підтримувати синхронну роботу кількох робочих місць, забезпечить віддалений доступ до бази та збереже повну офлайн-автономність додатка в умовах нестабільного мережевого з’єднання.
- Розширення сервісної та апаратної інтеграції. Впровадження модулів взаємодії з API існуючих сервісів ПРРО для автоматизації процесу фіскалізації чеків та прямого відправлення розрахункових даних до податкових органів. Додатково в планах пряма інтеграція з банківськими платіжними терміналами.
- Розвиток аналітичного та звітного функціоналу. Розробка модуля бізнес-аналітики для збору, обробки та графічної візуалізації статистики продажів, що дозволить відстежувати динаміку попиту й визначати рентабельність позицій. Поряд із цим планується реалізація інструментів для експорту розширеної фінансово-складської звітності в уніфіковані формати.
- Автоматизація адміністрування та безпеки. Інтеграція підсистеми автоматичних сповіщень керівництва щодо фінансових підсумків зміни, а також автоматизація створення резервних копій локальної бази даних із їх завантаженням у хмарне сховище.

Висновки до розділу 3

Третій розділ було присвячено проєктуванню та програмній реалізації програмного забезпечення для АРМ. Було зроблено наступне:

1. Розроблено багатошарову архітектуру додатка, яка чітко розмежовує рівень представлення, сервісну бізнес-логіку та шар доступу до даних.

2. Спроєктовано локальну реляційну базу даних на основі SQLite, що включає п'ять взаємопов'язаних таблиць. Застосовано механізм транзакцій для гарантування цілісності фінансової інформації та складських залишків під час комплексних операцій.

3. Створено оптимізований графічний інтерфейс за допомогою фреймворку PyQt5. Використання стеку віджетів дозволило реалізувати навігацію в межах єдиного головного вікна, що суттєво пришвидшує рутинні дії користувача.

4. Інтегровано апаратний сканер штрих-кодів через віртуальний COM-порт. Асинхронне зчитування потоку байтів в окремому потоці виконання забезпечило безперебійне перехоплення даних без блокування інтерфейсу та залежності від активного фокусу вводу. В результаті, отримано програмний продукт для автоматизації дій адміністратора у зоомагазині.

5. Сформовано напрями подальшого розвитку додатка, які передбачають перехід на клієнт-серверне рішення, інтеграцію з сервісами ПРРО й банківськими терміналами, а також розширення аналітичного функціоналу системи.

РОЗДІЛ 4

ОЦІНКА НАДІЙНОСТІ, ТЕСТУВАННЯ ТА БЕЗПЕКА СИСТЕМИ

4.1 Методологія тестування

Щоб забезпечити безперебійну роботу АРМ необхідно комплексно підійти до тестування, оскільки програмні помилки у підсистемах розрахунку або збереження даних можуть призвести до прямих фінансових збитків підприємтва.

У даній роботі було застовано декілька рівнів тестування системи, а саме:

1. Модульне тестуваненя – Щоб перевірити ізольовані компоненти бізнес-логіки
2. Функціональне тестування графічного інтерфейсу – Щоб верифікувати сценарії користувача, як от коректність навігації чи обробка хибного вводу,

Архітектура тестів у проєкті базується на патерні AAA, який є індустріальним стандартом, і розшифровується як (Arrange, Act, Assert), де:

1. Arrange: ініціалізація тестових даних
2. Act: виклик цільового методу сервісу або репозиторію з підготовленими параметрами.
3. Assert: порівняння фактичного результату виконання з очікуваним еталонним значенням.

Використовуючи таку методологію, можна швидко локалізувати архітектурні дефекти, та забезпечити стабільну роботу додатка.

4.2 Модульне тестування основних елементів бізнес-логіки

Модульне тестування є першим і найважливішим рівнем верифікації програмного забезпечення: воно дозволяє перевірити правильність бізнес-правил кожного компонента системи в повній ізоляції від решти коду та від реальної бази даних. Для написання тестів використано фреймворк pytest. Щоб жоден тест не торкався робочої бази даних, усі звернення до SQLite автоматично

перенаправляються на тимчасову базу, що існує лише в оперативній пам'яті та знищується після завершення тесту. Ця інфраструктура описана у спільному конфігураційному файлі conftest.py (код наведено у додатку В).

Тестуванню підлягали чотири ключові компоненти бізнес-логіки системи. Для кожного компонента відібрано найбільш показові сценарії, що охоплюють як штатну роботу, так і обробку нештатних ситуацій. Перелік тестових сценаріїв наведено у таблиці 4.1, а ілюстрація виконання показана на (рис 4.1)

Таблиця 4.1

Перелік модульних тестів

Компонент системи	**Що перевіряється**	**Кількість тестів**
Сервіс проведення чеків	Списання залишків після продажу, застосування знижки, блокування при дефіциті товару, запис до журналу продажів	4
Управління складом	Збереження картки товару, унікальність штрих-коду, регістронезалежний пошук по кирилиці, прийом партії товару	4
Журнал продажів та повернення	Відновлення залишків при поверненні, розрахунок залишку боргу, збереження виплати	3
Автентифікація	Вхід із правильним паролем, відхилення хибного пароля, чутливість до регістру	3

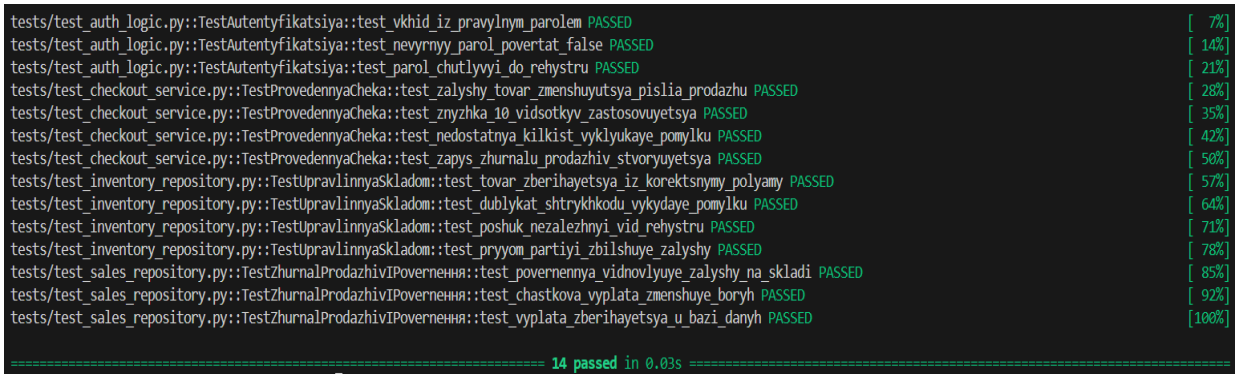

Рисунок 4.1. Демонстрація успішного проходження тестів

Нижче наведено приклад тесту, який перевіряє одне з бізнес-правил, а саме контроль залишків товарів на складі при проведенні чека. Система блокує додавання товару до чека, якщо додана кількість перевищує наявний залишок

```
    def test_insufficient_quantity_raises(self, mock_db):
        _add_product(mock_db, 2, "Вітаміни", 1)
        with pytest.raises(ValueError, match="Недостатньо на складі для Вітаміни"):
            CheckoutService().finalize_receipt(
                [{"product_id": 2, "name": "Вітаміни", "price": 50.0, "qty": 5}],
                payment_method="cash",
            )
```

У цьому тесті ми додаємо товар, якого на складі залишилася всього 1 одиниця, і пробуємо додати до чека одразу 5 одиниць. Очікувано має вискочити помилка ValueError. Це підтвердить, що система заблокувала некоректну операцію і дані в базі не змінилися. Код основних тестів було винесено у додатки Г, Ґ та Д.

4.3 Функціональне тестування графічного інтерфейсу користувача

Функціональне тестування GUI спрямовано на те, щоб перевірити коректність візуального відображення елементів керування, реакцію інтерфейсу на дії користувача та відповідність поведінки вікон і діалогів логіці бізнес-процесів застосунку. Оскільки інтерфейс користувача розроблено на PyQt5, який використовує власний цикл обробки подій та графічний рушій для рендерингу віджетів, автоматизація GUI-тестів не є виправданою для системи такого масштабу. Тому графічну оболонку буде перевірено вручну за допомогою методу чорної скриньки. «Це метод тестування програмного забезпечення, який перевіряє функціональність системи без доступу до її внутрішньої структури коду. У цьому підході тестувальник працює як кінцевий користувач, оцінюючи, чи правильно система реагує на вхідні дані» [33]. Результати проведення ручного функціонального тестування інтерфейсу наведено у додатку Е

Протестовані елементи інтерфейсу продемонстрували стабільну роботу, правильну реакцію на некоректні дії користувача та відсутність графічних артефактів чи зависань. Окрему увагу було приділено модальним вікнам попереджень, які слугують для того, щоб користувач не зміг випадково здійснити помилкову транзакцію чи внести некоректні дані до бази даних.

Висновки до розділу 4

Модульне тестування: за допомогою фреймворку pytest реалізовано 14 автоматизованих тестів на базі патерну AAA. Вони підтвердили коректність роботи основних бізнес-правил: розрахунку сум, списання та відновлення товарних залишків під час повернень, а також логіки авторизації. Усі сценарії виконано успішно. Далі проведено функціональне тестування GUI: шляхом ручної перевірки за допомогою методу чорної скриньки верифіковано 10 сценаріїв взаємодії користувача з інтерфейсом PyQt5.

Результати показали стабільність графічної оболонки, відсутність великих затримок рендерингу та надійність вбудованої валідації. Загалом комплексне оцінювання головних модулів бізнес-логіки та графічного інтерфейсу підтвердило стійкість системи до хибних дій користувача та програмних збоїв. Застосовані методи дали можливість верифікувати механізм захисту даних, показавши, що система коректно обробляє виняткові ситуації та забезпечує цілісність чутливої інформації.

ВИСНОВКИ

Результатом роботи над кваліфікаційною роботою стало розв’язання актуального завдання шляхом розробки десктопного додатка для автоматизації обліку та реалізації зоотоварів. Провівши системний аналіз предметної області, було доведено вкрай низьку ефективність паперового обліку та надмірність порівнюваних систем для виконання завдань адміністратора зоомагазину, після чого було сформовано технічні вимоги до програмного продукту, головними з яких стали автономність, легкість в освоєнні та наявність CRM-функціоналу для обліку дебіторської заборгованості. Наступним кроком стало обґрунтування інструментальних засобів розробки, а саме вибір мови програмування Python та фреймворку PyQt5 як бази, що дало змогу створити гнучкий та сучасний інтерфейс із високою швидкістю обробки даних. Зі свого боку, інтеграція СКБД SQLite забезпечила додатку незалежність від мережевої інфраструктури та стабільну роботу. Було спроєктовано та програмно реалізовано багаторівневу архітектуру системи, що включає реляційну базу даних із п’яти таблиць для ведення товарної номенклатури, фіксації чеків та виплат. Додатковим способом підвищення відмовостійкості, зручності та стабільності стала реалізація роботи сканера штрих-кодів через віртуальний COM-порт, а для збереження цілісності фінансових даних впроваджено транзакційну модель запитів. Фінальним етапом розробки стало проведення комплексного тестування програмного забезпечення, а саме автоматизованих тестів, що перевірили коректність роботи основних бізнес-правил та ізольованих компонентів, і функціонального тестування, спрямованого на перевірку графічного інтерфейсу, яке було проведене за методом чорної скриньки, і результати якого підтвердили стабільність роботи додатка. У підсумку, розроблений програмний продукт уже є готовим до практичного використання, і його впровадження поза сумнівом сприятиме підвищенню ефективності та комфорту роботи працівників, а також зниженню фінансових ризиків для підприємства. Завдяки надійній технічній базі додаток володіє значним потенціалом для подальшого масштабування та вдосконалення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. ENOTE - автоматизація процесів та контроль для ветеринарних клінік і аптек. URL: https://enote.vet/ (дата звернення: 11.02.2026).

2. Richards M., Ford N. Fundamentals of software architecture: an engineering approach. O'Reilly Media, 2020. 432 p.

3. Parker D. Learn Python : : A Crash Course on Python Programming and How to Start Coding with It. Learn the Basics of Machine Learning and Data Analysis. Independently Published, 2019.

4. Fatouretchi M. The Art of CRM: Proven strategies for modern customer relationship management. Packt Publishing, 2019. 360 p.

5. Що таке CRM: визначення, переваги, ключові можливості. Creatio. URL: https://www.creatio.com/ua/crm/what-is-crm (дата звернення: 10.02.2026).

6. SQL in a Nutshell: A Desktop Quick Reference. O'Reilly Media, Incorporated, 2022. 650 p.

7. Jet Vet додаток і програма для ветклініки і ветеринарів. URL: https://jet.vet/ (дата звернення: 11.02.2026).

8. Програма обліку товарів та автоматизація магазину. URL: https://torgsoft.ua/ (дата звернення: 11.02.2026).

9. Відкритий код ERP та CRM *Odoo*. URL: https://www.odoo.com/uk_UA (дата звернення: 11.02.2026).

10. ПРРО checkbox | програмний РРО № 1 для бізнесу в україні. *Checkbox*. URL: https://checkbox.ua/ (дата звернення: 11.02.2026).

11. Моделювання програмного забезпечення : навч.-метод. посіб. [Електронний ресурс] / С. Ю. Манаков та ін. Одеса : Фенікс, 2023. URL: https://doi.org/10.32837/11300.25952 (дата звернення: 09.03.2026).

12. Чикунов П. Об’єктно-орієнтоване програмування : навч.-метод. посіб. для здобувачів першого (бакалаврського) рівня вищої освіти галузей знань F «Інформаційні технології», A «Освіта» та G «Електроніка, автоматизація та електронні комунікації». Нац. ун-т "Од. юрид. акад.", 2025. URL:

https://doi.org/10.32837/11300.30862 (дата звернення: 12.03.2026).

13. SQLite documentation. *SQLite Home Page*. URL: https://sqlite.org/docs.html (дата звернення: 13.03.2026).

14. Бабурнич Н. І. Автоматизація обліку бізнес-процесів у торгівлі: проблеми, етапи здійснення та огляд програмних рішень. *Трансформаційна економіка*. 2024. № 2 (07). С. 7–12. URL: https://doi.org/10.32782/2786-8141/2024-7-1 (дата звернення: 13.03.2026).

15. GeeksforGeeks. Garbage collection in java - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/java/garbage-collection-in-java/ (дата звернення: 14.03.2026).

16. Застосування шаблону MVVM у реалізації UI-логіки десктопних додатків Прикладні інформаційні технології. *Прикладні інформаційні технології*. URL: https://jait.donnu.edu.ua/article/view/19308 (дата звернення: 14.03.2026).

17. Support for signals and slots - pyqt documentation v5.15.7. *Riverbank Computing*.URL: https://www.riverbankcomputing.com/static/Docs/PyQt5/signals_slots.html (дата звернення: 14.03.2026).

18. GeeksforGeeks. Layered architecture in computer networks - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/computer-networks/layered-architecture-in-computer-networks/ (дата звернення: 01.04.2026).

19. CRUD-перевірка бази даних. *Онлайн-курси від компанії QATestLab | Головна сторінка*. URL: https://training.qatestlab.com/blog/technical-articles/crud-database-validation/ (дата звернення: 01.04.2026).

20. COM-порт: означення, використання та особливості. *GSMhub.com.ua*. URL: https://gsmhub.com.ua/glossary/com-port (дата звернення: 01.04.2026).

21. Трейтяк Д. Мова програмування python як інструмент у науково-педагогічній діяльності. *Information technologies and learning tools*. 2025. Т. 107, № 3. С. 168–181. URL: https://doi.org/10.33407/itlt.v107i3.6007 (дата звернення: 15.04.2026).

22. Welcome to pySerial’s documentation – pySerial 3.5 documentation.

Welcome to pySerial's documentation – pySerial 3.5 documentation. URL: https://pySerial.readthedocs.io/en/latest/ (дата звернення: 16.04.2026).

23. Buelta J. Python Architecture Patterns: Master API design, event-driven structures, and package management in Python. Packt Publishing, 2022. 594 с.

24. Sqlite3 DB-API 2.0 interface for SQLite databases. *Python documentation*. URL: https://docs.python.org/3/library/sqlite3.html (дата звернення: 19.04.2026).

25. What is sqlite? The database that runs inside your app. *MindStudio*. URL: https://www.mindstudio.ai/blog/what-is-sqlite (дата звернення: 20.04.2026).

26. Kuku G. Understanding transactions in SQL: ensuring data integrity and consistency. *LinkedIn: Log In or Sign Up*. URL: https://www.linkedin.com/pulse/understanding-transactions-sql-ensuring-data-integrity-ganiu-kuku/ (дата звернення: 21.04.2026).

27. Kuku G. Understanding transactions in SQL: ensuring data integrity and consistency. *LinkedIn: Log In or Sign Up*. URL: https://www.linkedin.com/pulse/understanding-transactions-sql-ensuring-data-integrity-ganiu-kuku/ (дата звернення: 21.04.2026).

28. Java проти Python: базовий Python для Java розробників: Стаття з блогу IT-школи Hillel. *Корисні матеріали: Статті та новини IT-індустрії | Комп'ютерна школа Hillel*. URL: https://blog.ithillel.ua/articles/java-vs-python-python-basics-for-java-developers (дата звернення: 26.04.2026).

29. GeeksforGeeks. Python GUI - PyQt VS TKinter - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/python/python-gui-pyqt-vs-tkinter/ (дата звернення: 28.04.2026).

30. Athreya aka Maneshwar. PostgreSQL vs sqlite: dive into two very different databases. *DEV Community*. URL: https://dev.to/lovestaco/postgresql-vs-sqlite-dive-into-two-very-different-databases-5a90 (дата звернення: 02.05.2026).

31. SQLite vs. mysql: a comprehensive guide to choosing the right database. *Blog*. URL: https://elementor.com/blog/sqlite-vs-mysql-a-guide/ (дата звернення: 02.05.2026).

32. Pytest documentation. *pytest documentation*. URL: https://docs.pytest.org/en/stable/ (дата звернення: 06.05.2026).

33. Black Box тестування: що це таке, методи, техніки, переваги та недоліки. *Sigma Software University*. URL: https://university.sigma.software/black-box-testing/ (дата звернення: 07.05.2026).

34. Cx_Freeze 8.6.4 documentation. *cx_Freeze 8.6.4 documentation*. URL: https://cx-freeze.readthedocs.io/en/stable/ (дата звернення: 08.05.2026).

35. PyInstaller Manual - PyInstaller 6.20.0 documentation. *PyInstaller Manual - PyInstaller 6.20.0 documentation*. URL: https://pyinstaller.org/en/stable/ (дата звернення: 08.05.2026).

36. User documentation - nuitka the python compiler. *Nuitka the Python Compiler – Nuitka the Python Compiler*. URL: https://nuitka.net/user-documentation/ (дата звернення: 08.05.2026).

37. Бондарчук, А. П., & Глушак, О. М. (2025). Предиктивне управління оновленнями програмного забезпечення в інтернеті речей. Зв'язок, (5), 13-17.

38. Ruzhinsky, V. G., Bondarchuk, A. P., Kuzminykh, V. O., Otrokh, S. I., & Taranenko, R. A. (2022). Event-oriented architecture system for collecting and processing bibliographic information. P 90-98. https://doi.org/10.31673/2412-4338.2022.049098

39. Abramov, V., Astafieva, M., Boiko, M., Bodnenko, D., Bushma, A., Vember, V., ... & Yaskevych, V. (2021). Theoretical and practical aspects of the use of mathematical methods and information technology in education and science.

40. Машкіна, І. В., Абрамов, В. О., Носенко, Т. І., Іваніченко, Є. В., & Яскевич, В. О. (2024). Навчально-методичний посібник до виконання і захисту кваліфікаційної роботи бакалавра спеціальностей 122 Комп’ютерні науки для здобувачів вищої освіти денної форми навчання.

ДОДАТКИ

ДОДАТОК А

Детальне порівняння unittest та pytest

Критерій порівняння	**Фреймворк unittest**	**Фреймворк pytest**
Синтаксис та декларація тестів	Передбачає обов'язкове оголошення класів з успадкуванням від unittest.TestCase. Тестові методи повинні відповідати встановленим угодам іменування.	Тести оголошуються як звичайні функції без необхідності створення класів. Це суттєво зменшує обсяг шаблонного коду та спрощує структуру тестового модуля.
Базовий інструмент перевірки	Надає власний набір спеціалізованих методів перевірки: assertEqual(), assertTrue(), assertRaises() та інші. Використання стандартного оператора assert не рекомендується.	Спирається на стандартний оператор assert мови Python. При невдалій перевірці фреймворк автоматично формує детальне повідомлення з різницею між очікуваним і фактичним значеннями.
Керування станом	Ініціалізація та завершення тестового оточення реалізовані через фіксовані методи життєвого циклу класу: setUp() і tearDown(). Область їхньої дії обмежена рівнем класу.	Використовує гнучкий механізм фікстур через декоратор @pytest.fixture з можливістю налаштування області видимості: функція, модуль або сесія. Фікстури можна перевикористовувати між різними тестовими модулями.
Параметризація тестових сценаріїв	Не має вбудованої підтримки параметризації. Для її реалізації необхідне підключення сторонніх бібліотек або ручна генерація тестових методів.	Підтримується безпосередньо через декоратор @pytest.mark.parametrize. Дозволяє запускати один тест з набором вхідних значень без дублювання коду.
Автоматичне виявлення тестів	Для виявлення тестів потрібен явний запуск через команду python -m unittest discover або ручне налаштування тестового завантажувача.	Автоматично сканує директорії проєкту та ідентифікує тестові файли і функції за масками test__ та __test без додаткового налаштування.
Сумісність із стороннім кодом	Не підтримує виконання тестів, розроблених під інші тестові фреймворки. Міграція на unittest вимагає переписування існуючих тестів.	Забезпечує повну зворотну сумісність з unittest: наявні тести можна запускати без жодних модифікацій, що значно спрощує поступову міграцію проєкту.

ДОДАТОК Б

DDL-скрипт ініціалізації та міграції бази даних

```
def init_db():
    with get_conn() as conn:
        # 1. Товари (склад)
        conn.execute("""
        CREATE TABLE IF NOT EXISTS product_list(
          id        INTEGER PRIMARY KEY,
          barcode   TEXT UNIQUE,
          name      TEXT NOT NULL CHECK(LENGTH(name) <= 255),
          quantity  INTEGER NOT NULL DEFAULT 0 CHECK(quantity >= 0),
          price     NUMERIC NOT NULL DEFAULT 0 CHECK(price >= 0),
          supplier  TEXT
        );
        """)
        conn.execute("CREATE INDEX IF NOT EXISTS idx_product_list_name ON
product_list(name);")

        # 2. Журнал продажів за позиціями
        conn.execute("""
        CREATE TABLE IF NOT EXISTS sales_journal(
          id         INTEGER PRIMARY KEY,
          product_id INTEGER NOT NULL,
          qty        INTEGER NOT NULL,
          price      NUMERIC NOT NULL,
          sum_total  NUMERIC NOT NULL,
          cashier    TEXT,
          created_at TEXT NOT NULL DEFAULT (datetime('now','localtime')),
          FOREIGN KEY(product_id) REFERENCES product_list(id)
        );
        """)

        # 3. Чеки (заголовки транзакцій)
```

```
        conn.execute("""
        CREATE TABLE IF NOT EXISTS receipts(
          id             INTEGER PRIMARY KEY,
          cashier        TEXT,
          payment_method TEXT,
          total_sum      NUMERIC NOT NULL DEFAULT 0,
          note           TEXT,
          debtor_name    TEXT,
          created_at     TEXT NOT NULL DEFAULT (datetime('now','localtime'))
        );
        """)

        # 4. Блок міграцій схеми (оновлення структури)
        try:
            conn.execute("ALTER TABLE receipts ADD COLUMN note TEXT;")
            conn.commit()
        except sqlite3.OperationalError:
            pass

        try:
            conn.execute("ALTER TABLE receipts ADD COLUMN debtor_name TEXT;")
            conn.commit()
        except sqlite3.OperationalError:
            pass

        try:
            conn.execute("ALTER TABLE receipts ADD COLUMN debtor_phone TEXT;")
            conn.commit()
        except sqlite3.OperationalError:
            pass

        try:
            conn.execute("ALTER TABLE receipts ADD COLUMN discount_pct NUMERIC
NOT NULL DEFAULT 0;")
```

```
    conn.commit()
except sqlite3.OperationalError:
    pass

try:
    conn.execute("ALTER TABLE product_list ADD COLUMN category TEXT;")
    conn.commit()
except sqlite3.OperationalError:
    pass

# 5. Виплати за боргами
conn.execute("""
CREATE TABLE IF NOT EXISTS debt_payments(
  id             INTEGER PRIMARY KEY,
  receipt_id     INTEGER NOT NULL,
  debtor_name    TEXT NOT NULL,
  payment_amount NUMERIC NOT NULL,
  payment_date   TEXT NOT NULL DEFAULT (datetime('now','localtime')),
  cashier        TEXT,
  FOREIGN KEY(receipt_id) REFERENCES receipts(id)
);
""")

# 6. Позиції чека (деталізація)
conn.execute("""
CREATE TABLE IF NOT EXISTS receipt_items(
  id          INTEGER PRIMARY KEY,
  receipt_id  INTEGER NOT NULL,
  product_id  INTEGER NOT NULL,
  barcode     TEXT,
  name        TEXT,
  qty         INTEGER NOT NULL,
  price       NUMERIC NOT NULL,
  sum_total   NUMERIC NOT NULL,
```

```
  FOREIGN KEY(receipt_id) REFERENCES receipts(id),
  FOREIGN KEY(product_id) REFERENCES product_list(id)
);
""")

# 7. Повернення чеків
conn.execute("""
CREATE TABLE IF NOT EXISTS receipt_returns(
  id             INTEGER PRIMARY KEY,
  receipt_id     INTEGER NOT NULL,
  return_sum     NUMERIC NOT NULL DEFAULT 0,
  cashier        TEXT,
  created_at     TEXT NOT NULL DEFAULT (datetime('now','localtime')),
  FOREIGN KEY(receipt_id) REFERENCES receipts(id)
);
""")

# 8. Позиції повернення
conn.execute("""
CREATE TABLE IF NOT EXISTS return_items(
  id             INTEGER PRIMARY KEY,
  return_id      INTEGER NOT NULL,
  receipt_item_id INTEGER NOT NULL,
  product_id     INTEGER NOT NULL,
  name           TEXT,
  qty            INTEGER NOT NULL,
  price          NUMERIC NOT NULL,
  sum_total      NUMERIC NOT NULL,
  FOREIGN KEY(return_id) REFERENCES receipt_returns(id),
  FOREIGN KEY(receipt_item_id) REFERENCES receipt_items(id),
  FOREIGN KEY(product_id) REFERENCES product_list(id)
);
""")
conn.commit()
```

ДОДАТОК В

Тестова інфраструктура conftest.py

```
import sys
import os
sys.path.insert(0,
os.path.dirname(os.path.dirname(os.path.abspath(__file__))))
import pytest
import sqlite3
from contextlib import contextmanager
_DDL = """
CREATE TABLE IF NOT EXISTS product_list (
    id        INTEGER PRIMARY KEY AUTOINCREMENT,
    barcode   TEXT UNIQUE,
    name      TEXT NOT NULL,
    quantity  INTEGER NOT NULL DEFAULT 0,
    price     NUMERIC NOT NULL DEFAULT 0,
    supplier  TEXT,
    category  TEXT
);
CREATE TABLE IF NOT EXISTS receipts (
    id             INTEGER PRIMARY KEY AUTOINCREMENT,
    cashier        TEXT,
    payment_method TEXT,
    total_sum      NUMERIC NOT NULL DEFAULT 0,
    note           TEXT,
    debtor_name    TEXT,
    debtor_phone   TEXT,
    discount_pct   NUMERIC NOT NULL DEFAULT 0,
    created_at     TEXT NOT NULL DEFAULT (datetime('now','localtime'))
);
CREATE TABLE IF NOT EXISTS receipt_items (
    id         INTEGER PRIMARY KEY AUTOINCREMENT,
    receipt_id INTEGER NOT NULL,
    product_id INTEGER NOT NULL,
    barcode    TEXT,
    name       TEXT,
    qty        INTEGER NOT NULL,
    price      NUMERIC NOT NULL,
    sum_total  NUMERIC NOT NULL
);
```

```
CREATE TABLE IF NOT EXISTS sales_journal (
    id         INTEGER PRIMARY KEY AUTOINCREMENT,
    product_id INTEGER NOT NULL,
    qty        INTEGER NOT NULL,
    price      NUMERIC NOT NULL,
    sum_total  NUMERIC NOT NULL,
    cashier    TEXT,
    created_at TEXT NOT NULL DEFAULT (datetime('now','localtime'))
);
CREATE TABLE IF NOT EXISTS debt_payments (
    id             INTEGER PRIMARY KEY AUTOINCREMENT,
    receipt_id     INTEGER,
    debtor_name    TEXT NOT NULL,
    payment_amount NUMERIC NOT NULL,
    payment_date   TEXT NOT NULL DEFAULT (datetime('now','localtime')),
    cashier        TEXT
);
CREATE TABLE IF NOT EXISTS receipt_returns (
    id          INTEGER PRIMARY KEY AUTOINCREMENT,
    receipt_id  INTEGER NOT NULL,
    returned_at TEXT NOT NULL DEFAULT (datetime('now','localtime')),
    cashier     TEXT
);
CREATE TABLE IF NOT EXISTS return_items (
    id         INTEGER PRIMARY KEY AUTOINCREMENT,
    return_id  INTEGER NOT NULL,
    item_id    INTEGER NOT NULL,
    product_id INTEGER NOT NULL,
    qty        INTEGER NOT NULL,
    price      NUMERIC NOT NULL
);
"""
@pytest.fixture
def mock_db(monkeypatch):
    conn = sqlite3.connect(":memory:")
    conn.row_factory = sqlite3.Row
    conn.executescript(_DDL)
    conn.commit()

    @contextmanager
    def fake_get_conn():
        yield conn
```

```
        monkeypatch.setattr("services.checkout_service.get_conn",
fake_get_conn)
        monkeypatch.setattr("data.inventory_repository.get_conn",
fake_get_conn)
        monkeypatch.setattr("data.sales_repository.get_conn", fake_get_conn)
        return conn
```

ДОДАТОК Г

Тести сервісу проведення чеків

```
import pytest
from services.checkout_service import CheckoutService
def _dodaty_tovar(conn, product_id, name, kilkist, tsina=100.0):
    conn.execute(
        "INSERT INTO product_list (id, name, quantity, price) VALUES (?,?,?,?);",
        (product_id, name, kilkist, tsina),
    )
    conn.commit()
class TestProvedennyaCheka:
    def test_zalyshy_tovar_zmenshuyutsya_pislia_prodazhu(self, mock_db):
        _dodaty_tovar(mock_db, 1, "Корм", 10)
        CheckoutService().finalize_receipt(
            [{"product_id": 1, "name": "Корм", "price": 100.0, "qty": 3}],
            payment_method="cash",
        )
        row = mock_db.execute(
            "SELECT quantity FROM product_list WHERE id=1;"
        ).fetchone()
        assert row["quantity"] == 7

    def test_znyzhka_10_vidsotkyv_zastosovuyetsya(self, mock_db):
        _dodaty_tovar(mock_db, 1, "Корм", 10, tsina=100.0)
        CheckoutService().finalize_receipt(
            [{"product_id": 1, "name": "Корм", "price": 100.0, "qty": 2}],
            payment_method="cash",
            discount_pct=10.0,
        )
        row = mock_db.execute("SELECT total_sum FROM receipts LIMIT 1;").fetchone()
        assert float(row["total_sum"]) == pytest.approx(180.0)

    def test_nedostatnya_kilkist_vyklyukaye_pomylku(self, mock_db):
        _dodaty_tovar(mock_db, 2, "Вітаміни", 1)
        with pytest.raises(ValueError, match="Недостатньо на складі для Вітаміни"):
            CheckoutService().finalize_receipt(
                [{"product_id": 2, "name": "Вітаміни", "price": 50.0,
```

```
"qty": 5}],
                    payment_method="cash",
                )
        def test_zapys_zhurnalu_prodazhiv_stvoryuyetsya(self, mock_db):
            _dodaty_tovar(mock_db, 1, "Корм", 10, tsina=100.0)
            CheckoutService(cashier="Марія").finalize_receipt(
                [{"product_id": 1, "name": "Корм", "price": 100.0, "qty": 2}],
                payment_method="cash",
            )
            rows = mock_db.execute(
                "SELECT qty, price FROM sales_journal WHERE product_id=1;"
            ).fetchall()
            assert len(rows) == 1
            assert rows[0]["qty"] == 2
            assert float(rows[0]["price"]) == pytest.approx(100.0)
```

ДОДАТОК Ґ

Тести сервісу управління складом

```
import pytest
import data.inventory_repository as inv
class TestUpravlinnyaSkladom:
    def test_tovar_zberihayetsya_iz_korektsnymy_polyamy(self, mock_db):
        pid = inv.create_product("BAR-001", "Вітаміни", 5, 50.0, None)
        tovar = inv.get_by_id(pid)
        assert tovar is not None
        assert tovar["name"] == "Вітаміни"
        assert tovar["quantity"] == 5
        assert float(tovar["price"]) == pytest.approx(50.0)
    def test_dublykat_shtrykhkodu_vykydaye_pomylku(self, mock_db):
        inv.create_product("BAR-DUP", "Корм", 10, 100.0, None)
        with pytest.raises(ValueError, match="Штрих-код 'BAR-DUP'"):
            inv.create_product("BAR-DUP", "Ласощі", 5, 30.0, None)
    def test_poshuk_nezalezhnyi_vid_rehystru(self, mock_db):
        inv.create_product(None, "Вітаміни для собак", 3, 30.0, None)
        results = inv.search_by_name_part("вітам")
        nazvy = [r["name"] for r in results]
        assert "Вітаміни для собак" in nazvy
    def test_pryyom_partiyi_zbilshuye_zalyshy(self, mock_db):
        pid = inv.create_product(None, "Корм", 5, 10.0, None)
        inv.receive_products([{"product_id": pid, "qty": 10, "price":
12.0}])
        assert inv.get_product_quantity(pid) == 15
```

ДОДАТОК Д

Тести журналу продажів та повернення

```
import pytest
import data.sales_repository as sales
def _dodaty_tovar(conn, nazva="Товар", kilkist=10, tsina=100.0):
    cur = conn.execute(
        "INSERT INTO product_list (name, quantity, price) VALUES
(?,?,?);",
        (nazva, kilkist, tsina),
    )
    conn.commit()
    return cur.lastrowid
def _dodaty_chek(conn, suma=200.0, nota=None, im_ya_borzhnyka=None):
    cur = conn.execute(
        "INSERT INTO receipts (payment_method, total_sum, note,
debtor_name) "
        "VALUES (?,?,?,?);",
        ("cash", suma, nota, im_ya_borzhnyka),
    )
    conn.commit()
    return cur.lastrowid
def _dodaty_pozytsiyu_cheka(conn, receipt_id, product_id, kilkist=2,
tsina=100.0):
    cur = conn.execute(
        "INSERT INTO receipt_items "
        "(receipt_id, product_id, name, qty, price, sum_total) "
        "VALUES (?,?,?,?,?,?);",
        (receipt_id, product_id, "Товар", kilkist, tsina, kilkist *
tsina),
    )
    conn.commit()
    return cur.lastrowid


class TestZhurnalProdazhivIPovernennya:
    def test_povernennya_vidnovlyuye_zalyshy_na_skladi(self, mock_db):
        pid = _dodaty_tovar(mock_db, kilkist=5)
        rid = _dodaty_chek(mock_db)
        item_id = _dodaty_pozytsiyu_cheka(mock_db, rid, pid, kilkist=2)
        sales.process_return(
```

```
                receipt_id=rid,
                items=[{
                    "item_id": item_id, "product_id": pid,
                    "name": "Товар", "qty": 2, "price": 100.0,
                }],
                cashier="Антон",
            )
            row = mock_db.execute(
                "SELECT quantity FROM product_list WHERE id=?;", (pid,)
            ).fetchone()
            assert row["quantity"] == 7

        def test_chastkova_vyplata_zmenshuye_boryh(self, mock_db):
            rid = _dodaty_chek(mock_db, suma=500.0, nota="Борг",
                               im_ya_borzhnyka="Коваленко")
            mock_db.execute(
                "INSERT INTO debt_payments (receipt_id, debtor_name,
payment_amount) "
                "VALUES (?,?,?);",
                (rid, "Коваленко", 200.0),
            )
            mock_db.commit()
            result = sales.get_debt_balance(rid)
            assert result["paid_total"] == pytest.approx(200.0)
            assert result["total_sum"] - result["paid_total"] ==
pytest.approx(300.0)

        def test_vyplata_zberihayetsya_u_bazi_danyh(self, mock_db):
            rid = _dodaty_chek(mock_db, suma=300.0, nota="Борг",
                               im_ya_borzhnyka="Сидоренко")
            sales.save_debt_payment(
                "Сидоренко", amount=150.0, cashier="Марія", receipt_id=rid
            )
            row = mock_db.execute(
                "SELECT payment_amount FROM debt_payments WHERE
receipt_id=?;",
                (rid,),
            ).fetchone()
            assert row is not None
            assert float(row["payment_amount"]) == pytest.approx(150.0)
```

ДОДАТОК Е

Результати функціонального тестування GUI

№ тесту	Опис сценарію дій користувача	Очікуваний результат системи	Результат тестування
1	Сканування валідного штрих–коду товару в активному вікні оформлення касового чека.	Товар автоматично додається до таблиці кошика, кількість встановлюється рівною 1, проміжний підсумок перераховується.	Успішно
2	Повторне сканування штрих–коду того самого товару, який уже присутній у поточному чеку.	Новий рядок у таблиці не створюється, значення поля «Кількість» збільшується на 1, загальна сума чека оновлюється.	Успішно
3	Введення неіснуючого або пошкодженого штрих–коду, якого немає в реляційній базі даних.	Система перехоплює подію, відображає модальне вікно з помилкою «Товар не знайдено», фокус вводу повертається на касове поле.	Успішно
4	Спроба проведення фінального чека з повністю порожнім списком товарних позицій.	Програма блокує генерацію транзакції, операція переривається з виведенням попереджувального діалогового вікна «Немає позицій у чеку».	Успішно
5	Введення значення відсоткової знижки в полі каси, що виходить за межі діапазону 0-99%.	Сервісний шар автоматично обмежує граничні значення або коригує підсумкову суму, унеможливлюючи від'ємний баланс.	Успішно
6	Перемикання між функціональними вкладками («Каса», «Склад», «Журнал продажів») через головне меню.	Візуальні віджети змінюються у QStackedWidget без затримок графіки, стан незавершеного чека на вкладці каси повністю зберігається.	Успішно
7	Зміна кількості товару безпосередньо у таблиці кошика користувачем	Загальна вартість відповідного рядка та підсумкова сума всього чека автоматично перераховуються.	Успішно
8	Видалення окремої товарної позиції з таблиці кошика натисканням кнопки F9	Товар вилучається з таблиці кошика, загальна сума чека миттєво оновлюється, решта товарів залишаються без змін.	Успішно
9	Спроба проведення чека, якщо замовлена кількість товару перевищує фактичний залишок на складі.	Система блокує проведення транзакції, виводить попереджувальне вікно з повідомленням про дефіцит та інформацією про доступну кількість.	Успішно
10	Пошук товару на вкладці «Склад» за назвою або штрих-кодом у полі пошуку.	Таблиця залишків динамічно фільтрується, відображаючи лише ті позиції, що містять пошуковий запит.	Успішно