Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Алгоритмічні та програмні засоби оптимізації розміщення посилок на складі в інтелектуальних логістичних системах»

Виконав: студент 4 курсу
групи 2ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Танань А.В.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Чехместрук Р.Ю.
(прізвище та ініціали)
Рецензент:. к.т.н., доц. каф. ОТ Дудник О.В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ О.Н. Романюк
«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О.Н. Романюк
«21» березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Тананю Артему Володимировичу

1. Тема роботи – алгоритмічні та програмні засоби оптимізації розміщення посилок на складі в інтелектуальних логістичних системах.
Керівник роботи: Чехместрук Роман Юрійович, к.т.н., доц. каф. ПЗ, затверджені наказом вищого навчального закладу від
«20» березня 2025 р. №97.
2. Строк подання студентом роботи 2 червня 2025.
3. Вихідні дані до роботи: характеристики посилок – унікальний ідентифікатор, вага, габарити, опис; дані про сектора складу – максимальна допустима вага, об'єм, поточна завантаженість; вхідні дії користувача – додавання, пошук, видалення, редагування записів; алгоритмічні параметри – межі допустимих характеристик, критерії оптимальності, рівень заповненості; облікові дані користувачів – логін, пароль, роль; вихідні дані – візуальне представлення заповненості складу, таблиці посилок, рекомендовані сектори.
4. Зміст текстової частини: вступ; аналіз розвитку систем складської логістики та постановка задачі бакалаврської кваліфікаційної роботи; розробка моделі та алгоритмів програмного продукту; розробка програмних модулів системи оптимізації розміщення посилок; тестування програми; висновки; список використаних джерел; додатки.
5. Перелік ілюстративного матеріалу: титульний слайд; актуальність роботи; мета, предмет та об'єкт дослідження; порівняльний аналіз аналогів; постановка задачі; новизна; практична цінність; принцип роботи алгоритму визначення сектору; блок-схема роботи програми; головне вікно програми; вікно «Посилки»; вікна «Інформація про посилку» та «Додати нову посилку»;

апробація матеріалів бакалаврської кваліфікаційної роботи; висновки.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдан прийн
1-4	Чехместрук Р. Ю., к.т.н., доцент кафедри ПЗ	24.03.2025	01.06.2

1. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Приміт
1	Аналіз та постановка задачі бакалаврської кваліфікаційної роботи	30.03.2025 – 05.04.2025	вик.
2	Аналіз розвитку систем складської логістики	06.04.2025 – 08.04.2025	вик.
3	Розробка інтерфейсу програмного додатку	09.04.2025 – 15.04.2025	вик.
4	Розробка моделі та алгоритмів роботи системи	16.04.2025 – 21.04.2025	вик.
5	Реалізація модуля керування посилками	22.04.2025 – 30.04.2025	вик.
6	Реалізація алгоритму визначення оптимального сектора	01.05.2025 – 08.05.2025	вик.
7	Тестування системи	09.05.2025 – 13.05.2025	вик.
8	Створення пояснювальної записки	14.05.2025 – 18.05.2025	вик.

Студент _____ **Танань А**
(підпис) (прізвище та ініці

Керівник бакалаврської кваліфікаційної роботи _____ **Чехместрук Р.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.42

Танань А.В. Алгоритмічні та програмні засоби оптимізації розміщення посилок на складі в інтелектуальних логістичних системах : бакалаврська кваліфікаційна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця : ВНТУ, 2025. 90 с.

На укр. Мові. Бібліогр. : 25 назв.; рис. : 29; табл. : 2.

У межах бакалаврської кваліфікаційної роботи розроблено алгоритмічні та програмні засоби для оптимізації процесу розміщення посилок на складі в інтелектуальних логістичних системах. Запропоноване рішення забезпечує автоматизований вибір оптимального сектору зберігання з урахуванням фізичних параметрів посилок (вага, об’єм) та поточного рівня завантаженості складу, що сприяє підвищенню ефективності просторового використання та мінімізації людського впливу при прийнятті рішень.

Система включає реалізацію механізмів авторизації користувачів з розмежуванням прав доступу, ведення журналу подій для фіксації активності, а також функціональний графічний інтерфейс, що забезпечує інтуїтивно зрозумілу взаємодію з користувачем.

Інтерфейс створено з використанням бібліотеки PyQt6, зберігання даних реалізовано за допомогою бази даних SQLite, що забезпечує автономну роботу програми без потреби у зовнішньому серверному середовищі.

Розроблені засоби можуть бути успішно впроваджені в логістичних підприємствах малого та середнього масштабу з метою підвищення точності, швидкості та надійності процесів управління складом.

ABSTRACT

Tanan A.V. Algorithmic and software tools for optimizing the placement of parcels in the warehouse in intelligent logistics systems. Vinnytsia : VNTU, 2025. 90 p.

In ukrainian language. Bibliobrapher: 25 titles; fig. : 29; tabl. : 2.

This bachelor's qualification work presents the development of algorithmic and software tools for optimizing parcel placement in a warehouse within intelligent logistics systems. The proposed solution provides automated selection of the optimal storage sector based on the physical parameters of parcels (weight, volume) and the current load level of the warehouse, contributing to efficient spatial utilization and minimizing human involvement in decision-making.

The system includes user authentication mechanisms with role-based access control, activity logging for event tracking, and a functional graphical interface designed for intuitive user interaction.

The interface is built using the PyQt6 library, while data storage is implemented with the SQLite database, enabling the application to operate autonomously without reliance on an external server environment.

The developed tools can be effectively implemented in small and medium-sized logistics enterprises to improve the accuracy, speed, and reliability of warehouse management processes.

ЗМІСТ

ВСТУП 3
1 АНАЛІЗ РОЗВИТКУ СИСТЕМ СКЛАДСЬКОЇ ЛОГІСТИКИ ТА ПОСТАНОВКА ЗАДАЧІ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ ... 7
1.1 Аналіз сучасного стану автоматизованих систем управління складом 7
1.2 Порівняльний аналіз аналогів 9
1.3 Аналіз методів розв’язання задачі 14
1.4 Постановка задач бакалаврської кваліфікаційної роботи 16
1.5 Висновки 18
2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ 20
2.1 Реалізація програмного забезпечення системи 20
2.2 Розробка інтерфейсу програмного додатку 22
2.3 Розробка моделі системи 26
2.4 Розробка алгоритмів роботи системи 29
2.5 Висновки 31
3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ ОПТИМІЗАЦІЇ РОЗМІЩЕННЯ ПОСИЛОК 32
3.1 Обґрунтування вибору засобів для реалізації 32
3.2 Реалізація модуля керування посилками та алгоритму визначення оптимального сектора 33
3.3 Висновки 38
4 ТЕСТУВАННЯ ПРОГРАМИ 39
4.1 Тестування системи 39
4.2 Розробка інструкції користувача 48
4.3 Висновки 50
ВИСНОВКИ 51
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 53
ДОДАТКИ 56
Додаток А. Технічне завдання ..... **Ошибка! Закладка не определена.**
Додаток Б. Протокол перевірки БКР на плагіат ..... **Ошибка! Закладка не определена.**
Додаток В. Лістинг програми 61
Додаток Г. Реалізація модулів та алгоритмів 79
Додаток Д. Ілюстративна частина 83

ВСТУП

Обґрунтування вибору теми дослідження. У сучасних умовах цифровізації, активного розвитку електронної комерції та постійного зростання обсягів товарообігу, перед логістичними компаніями та складськими підприємствами стоїть завдання ефективної організації внутрішніх процесів. Одним із ключових напрямів є оптимізація просторового розміщення вантажів, що забезпечує раціональне використання складських площ, мінімізацію часу обробки замовлень і зниження навантаження на персонал [1]. Актуальність теми посилюється потребою у впровадженні автоматизованих та інтелектуальних рішень, які здатні швидко приймати рішення на основі множини факторів.

Традиційне розміщення посилок зазвичай базується на простих правилах або інтуїції працівників, що не дозволяє досягнути оптимального використання ресурсів. У таких умовах виникають ситуації, коли частина складу перевантажена, а інша – недозавантажена, що знижує загальну ефективність [2]. Саме тому актуальною є розробка алгоритмічних засобів, які дозволяють враховувати фізичні параметри вантажів (вага, розміри, об’єм), а також поточну завантаженість секторів і інші логістичні фактори при розміщенні.

Окрему складність становить розробка системи, яка б була не лише функціонально ефективною, а й зручною у використанні для працівників, що не мають спеціальної технічної підготовки. Це вимагає створення інтуїтивно зрозумілого інтерфейсу з прозорою логікою, що приховує складні обчислення та алгоритми від кінцевого користувача [3].

У межах цієї бакалаврської кваліфікаційної роботи розроблено програмну систему, яка реалізує інтелектуальну оптимізацію розміщення посилок у складі. Основу рішення становить алгоритм, що враховує обмеження за вагою та об’ємом для кожного сектору, оцінює поточне завантаження й автоматично визначає оптимальне місце для зберігання. Реалізовані також модулі авторизації користувачів, графічного відображення завантаженості складу, пошуку й перегляду посилок. Система працює автономно, не потребує зовнішнього

серверного середовища і орієнтована на малі та середні логістичні підприємства.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення ефективності управління складським простором шляхом розробки інтелектуального програмного забезпечення, що забезпечує оптимальне розміщення посилок на основі фізичних параметрів вантажів та поточної завантаженості секторів складу.

Основними задачами дослідження є:

- провести аналіз сучасних програмних рішень для управління складом з точки зору їх функціональних можливостей та логістичної ефективності;
- дослідити існуючі алгоритми логістичної оптимізації та визначити придатні для задачі розміщення вантажів;
- запропонувати алгоритмічну модель, яка поєднує обмеження фізичних характеристик вантажів із критеріями рівномірного завантаження секторів;
- розробити програмні модулі для обробки даних про вантажі, розрахунку оптимального сектору, візуалізації стану складу;
- реалізувати інтерфейс користувача, адаптований для роботи непідготовленого персоналу;
- здійснити тестування програмного забезпечення, оцінити його функціональність, точність та продуктивність у модельних умовах.

Об'єкт дослідження – процес автоматизованого управління просторовим розміщенням посилок у складських умовах.

Предмет дослідження – алгоритми логістичної оптимізації, методи розподілу вантажів за секторами, архітектура та реалізація програмного забезпечення системи управління складом.

Методи дослідження. У процесі дослідження використовувалися такі методи: аналіз предметної області; порівняльний аналіз існуючих систем;
методи логістичної оптимізації, зокрема жадібні алгоритми, rule-based підходи,

constraint-based моделі; структурне та об’єктно-орієнтоване моделювання; комп’ютерне моделювання; експериментальне тестування.

Наукова новизна отриманих результатів.

1. Удосконалено комбінований алгоритм оптимізації розміщення посилок на складі, який, на відміну від відомих, враховує обмеження за вагою, об’ємом та рівнем заповненості секторів, що дозволяє уникати перевантаження окремих зон та покращити просторову рівномірність розподілу вантажів.

2. Подальшого розвитку отримав модуль динамічного оновлення стану секторів, що реагує на дії користувача в режимі реального часу, завдяки чому забезпечується підвищення оперативності управління складськими ресурсами.

3. Розроблено новий графічний інтерфейс користувача, який відрізняється від відомих тим, що забезпечує інтуїтивну взаємодію з основним функціоналом системи без потреби у спеціальній технічній підготовці, що значно розширює коло потенційних користувачів.

4. Створено архітектурно незалежне автономне програмне рішення, яке не вимагає інтеграції зі складними ERP-системами, що робить його придатним для впровадження в умовах малих і середніх логістичних підприємств, де важлива швидкість та простота розгортання.

Практична цінність отриманих результатів. Практична цінність результатів полягає в створенні функціонального і доступного програмного рішення, яке може бути впроваджене на логістичних об’єктах з метою підвищення точності розміщення вантажів; зниження навантаження на персонал завдяки автоматизації процесів; скорочення часу обробки запитів; забезпечення автономної роботи без зовнішніх серверів; покращення обслуговування завдяки візуалізації стану складу та простому інтерфейсу.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. У наукових працях, опублікованих у співавторстві, автору належать такі результати: оптимізація розміщення посилок на складі в інтелектуальних логістичних системах [4].

Апробація матеріалів бакалаврської кваліфікаційної роботи. Основні положення бакалаврської кваліфікаційної роботи доповідалися на 3 Міжнародній науково-практичній конференції «Modern Trends in the Development of Economy, Technology and Industry».

Публікації. Основні результати досліджень опубліковано у збірнику матеріалів конференції.

1 АНАЛІЗ РОЗВИТКУ СИСТЕМ СКЛАДСЬКОЇ ЛОГІСТИКИ ТА ПОСТАНОВКА ЗАДАЧІ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Аналіз сучасного стану автоматизованих систем управління складом

У сучасному світі стрімкої цифрової трансформації особливого значення набуває ефективне управління логістичними процесами, серед яких центральне місце посідає організація складського господарства. Зростання обсягів вантажоперевезень, швидкий розвиток електронної комерції, підвищення вимог до швидкості виконання замовлень і точності доставки формують нові стандарти функціонування складів, вимагаючи переходу від ручних операцій до повноцінної автоматизації та інтелектуального управління.

Традиційні підходи до зберігання вантажів, що базуються на досвіді працівників або жорстко фіксованих правилах, дедалі більше втрачають актуальність через низьку адаптивність до зміни умов, перевантаження окремих секторів та нераціональне використання складських площ. Такі недоліки спричиняють збільшення часу на пошук вантажів, ризик помилок при розміщенні, а також зниження загальної ефективності логістичної системи [5].

У відповідь на ці виклики активно впроваджуються автоматизовані системи управління складом — Warehouse Management Systems (WMS). Основне їхнє призначення — забезпечити комплексне управління потоками вантажів від моменту надходження до моменту відвантаження. WMS-рішення дозволяють здійснювати облік вантажів, контролювати їх місце розташування, планувати розміщення з урахуванням фізичних характеристик вантажів (вага, об’єм, габарити) та технічних параметрів складських зон (вантажопідйомність, залишковий простір, пріоритет обслуговування тощо) [6].

Сучасні WMS дедалі частіше інтегрують інтелектуальні алгоритми, що використовують дані аналітики та прогнозування для прийняття рішень про оптимальне розміщення вантажу. Це дозволяє системі не лише враховувати

поточний стан складу, а й прогнозувати можливі перевантаження, групувати вантажі за типами, маршрутами або термінами зберігання, що суттєво покращує логістичну продуктивність [7].

Окрему роль у розвитку таких систем відіграє візуалізація складських процесів. Сучасне програмне забезпечення дозволяє в реальному часі бачити рівень заповненості секторів, використовує графічні індикатори, теплові карти завантаженості, динамічну звітність та інтерактивні екрани управління. Це забезпечує ефективний моніторинг стану складу, дозволяє приймати обґрунтовані рішення щодо розміщення і переміщення вантажів.

Ще однією важливою характеристикою сучасних систем є дружній та адаптивний інтерфейс, який дозволяє непідготовленому персоналу швидко навчатися роботі в системі. Це досягається завдяки інтерактивним формам, зручним елементам керування, підказкам та мінімізації необхідності ручного введення даних [8].

Однак попри значні досягнення в галузі, більшість комерційно доступних WMS-рішень мають низку обмежень: складність налаштування і впровадження; висока вартість ліцензій та обслуговування; орієнтація переважно на великі підприємства; відсутність адаптації під специфіку невеликих складів.

Ці фактори створюють попит на легкі, доступні, автономні системи, які не поступаються у функціональності, але можуть бути швидко впроваджені без потреби в складній інфраструктурі чи ERP-інтеграції. Особливо актуально це для невеликих логістичних центрів, служб доставки, муніципальних складів і малих бізнесів.

Отже, аналіз сучасного стану автоматизованих систем управління складом дозволяє зробити висновок про необхідність подальшого розвитку спеціалізованих рішень, які поєднують в собі:

- інтелектуальні алгоритми оптимізації розміщення;
- автономну архітектуру без залежності від серверів;
- простоту інтерфейсу та доступність;
- гнучкість і масштабованість, що дозволяє адаптувати систему під

конкретні умови.

Саме на таких принципах ґрунтується розробка програмного забезпечення у межах цієї бакалаврської кваліфікаційної роботи.

1.2 Порівняльний аналіз аналогів

Розвиток логістики та зростання обсягів товарообігу стимулювали активне впровадження спеціалізованих програмних засобів для автоматизації управління складськими процесами. У сучасних умовах це стало не лише бажаним, а й необхідним компонентом ефективної діяльності логістичних підприємств. На ринку представлено низку готових рішень, орієнтованих як на великі корпорації з розгалуженою інфраструктурою, так і на підприємства середнього та малого бізнесу. Такі системи покликані оптимізувати ключові етапи логістичного циклу: приймання, зберігання, облік, пошук та видачу товарів. Їх основним завданням є мінімізація впливу людського фактору, зменшення кількості помилок та забезпечення повного контролю над обігом вантажів. У цьому підрозділі розглянуто найбільш відомі та поширені аналоги, що використовуються на практиці.

Найбільш відомі та поширені аналоги:

- Warehouse Expert;
- SAP Extended Warehouse Management (SAP EWM);
- WMS від IT-Enterprise;
- SmartStock WMS;
- Odoo Inventory.

Warehouse Expert — система для автоматизованого обліку на складах, яка використовується здебільшого в малому та середньому бізнесі (рис. 1.1). Програма забезпечує облік товарів, друк супровідних документів, ведення звітності, інвентаризацію та інтеграцію з популярними бухгалтерськими системами. Проте логіка розміщення товарів у Warehouse Expert базується на фіксованих зонах і не підтримує адаптивних або інтелектуальних алгоритмів оптимізації.

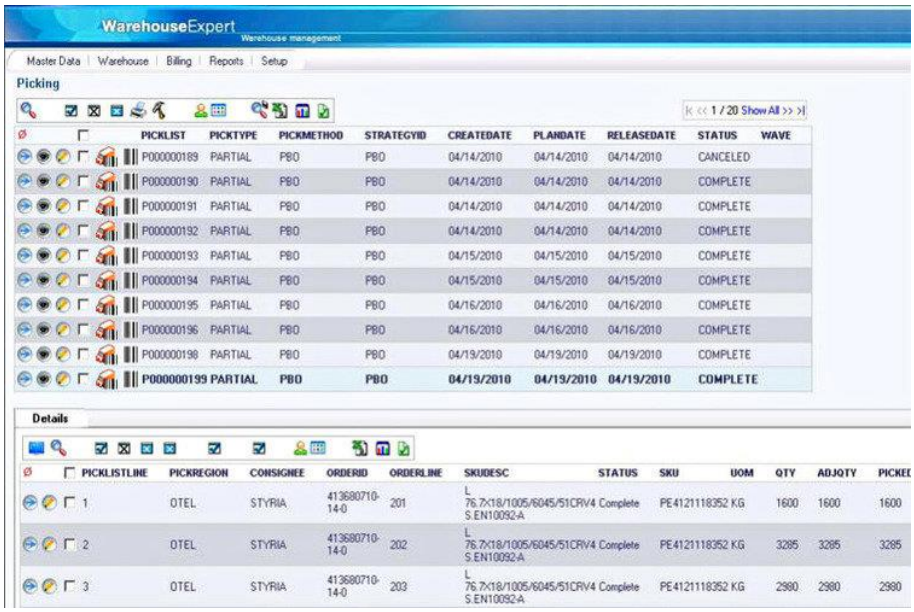

Рисунок 1.1 – Вікно застосунку Warehouse Expert

SAP EWM (рис. 1.2) — один із найпотужніших інструментів на ринку систем управління складом. Продукт підтримує моделювання складських процесів, планування розміщення товарів, оптимізацію маршрутів та інтеграцію з системами постачання [9].

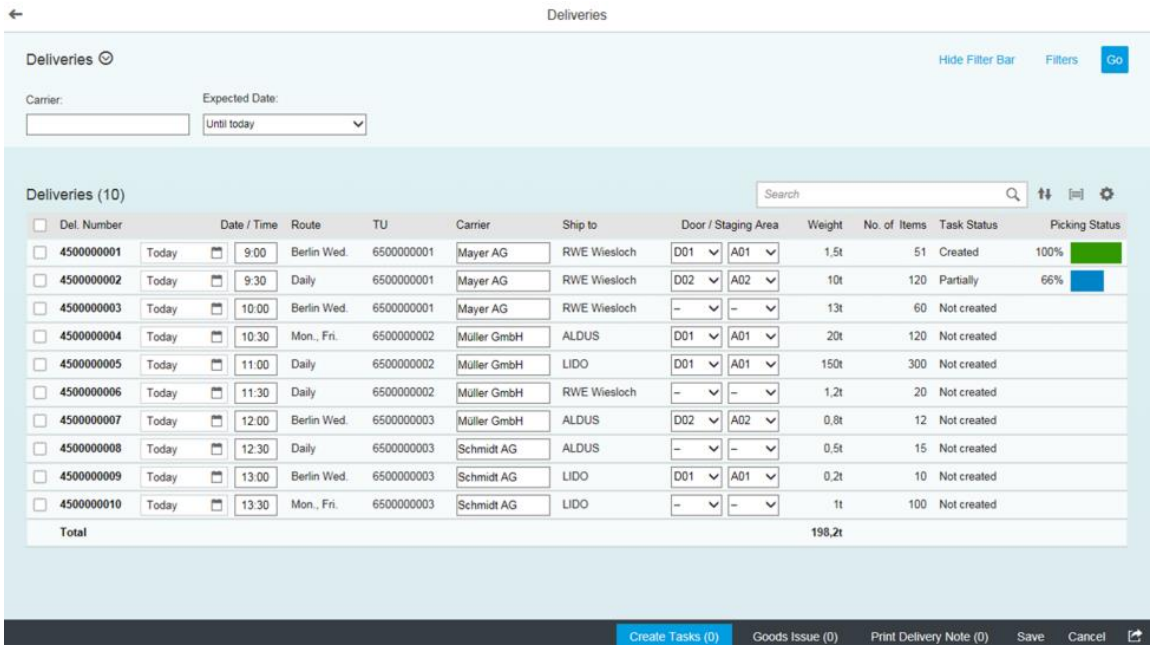

Рисунок 1.2 – Вікно застосунку SAP EWM

Основним недоліком є висока вартість ліцензії та складність впровадження. Система потребує глибокої адаптації під конкретне підприємство, значного часу на налаштування та команду фахівців.

Вітчизняна система WMS від IT-Enterprise (рис. 1.3) розроблена з урахуванням потреб українських підприємств. Вона підтримує функціонал повноцінної WMS-системи, включаючи контроль складських залишків, облік руху товарів, обробку замовлень і управління персоналом [10]. Однак подібно до SAP, вона орієнтована на великі компанії та має складну архітектуру, що ускладнює впровадження на складах малого масштабу.

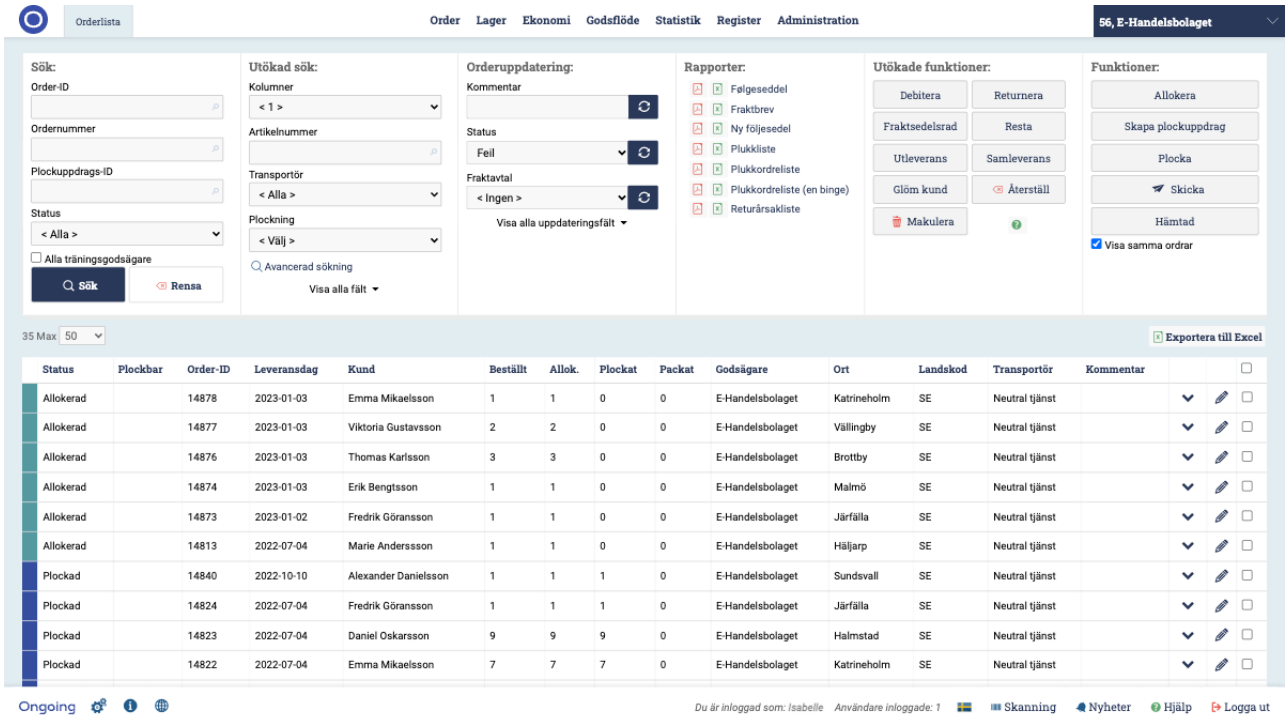

Рисунок 1.3 – Вікно застосунку IT-Enterprise WMS

SmartStock WMS (рис. 1.4) — легке та доступне рішення для автоматизації складських процесів, орієнтоване переважно на малий і середній бізнес. Система пропонує базовий функціонал, зокрема інтерфейс управління залишками, формування звітів, генерацію штрихкодів, а також облік переміщень товарів між зонами складу. Вона вирізняється зрозумілим інтерфейсом, невибагливістю до

ресурсів і швидким налаштуванням, що робить її зручною для впровадження в обмежених умовах. Попри це, система не реалізує інтелектуальних алгоритмів автоматичного розміщення вантажів. Вибір місця зберігання здійснюється вручну або відповідно до заздалегідь визначених шаблонів, що знижує гнучкість і ефективність при великій динаміці обігу.

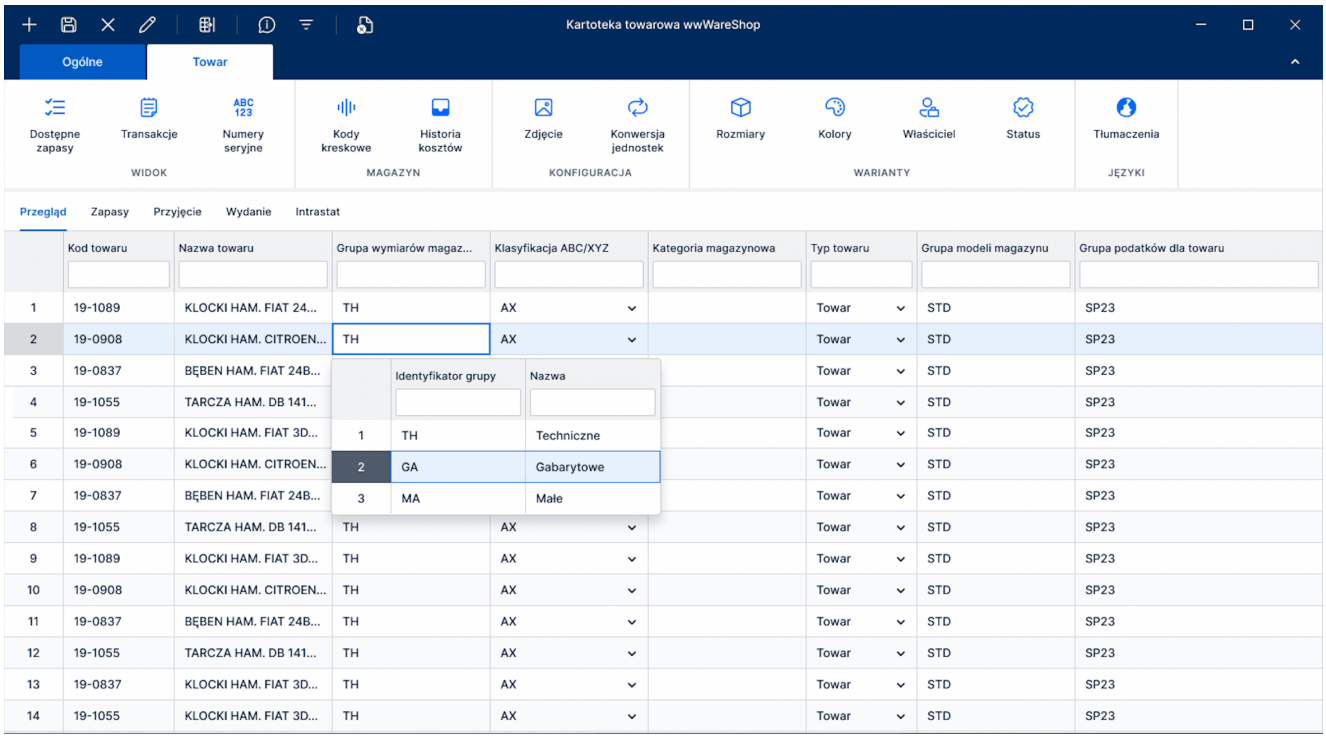

Рисунок 1.4 – Вікно застосунку SmartStock WMS

Odoo Inventory (рис. 1.5) — модульна ERP-система з відкритим кодом, яка дозволяє реалізувати базовий функціонал обліку товарів, контролю складу та обробки замовлень. Має можливості інтеграції з іншими модулями, такими як продажі, закупівлі чи CRM. Основним плюсом є гнучкість і безкоштовна базова версія [11]. Проте, як і SmartStock, Odoo не має вбудованої логіки для оптимального розміщення товарів у складських зонах.

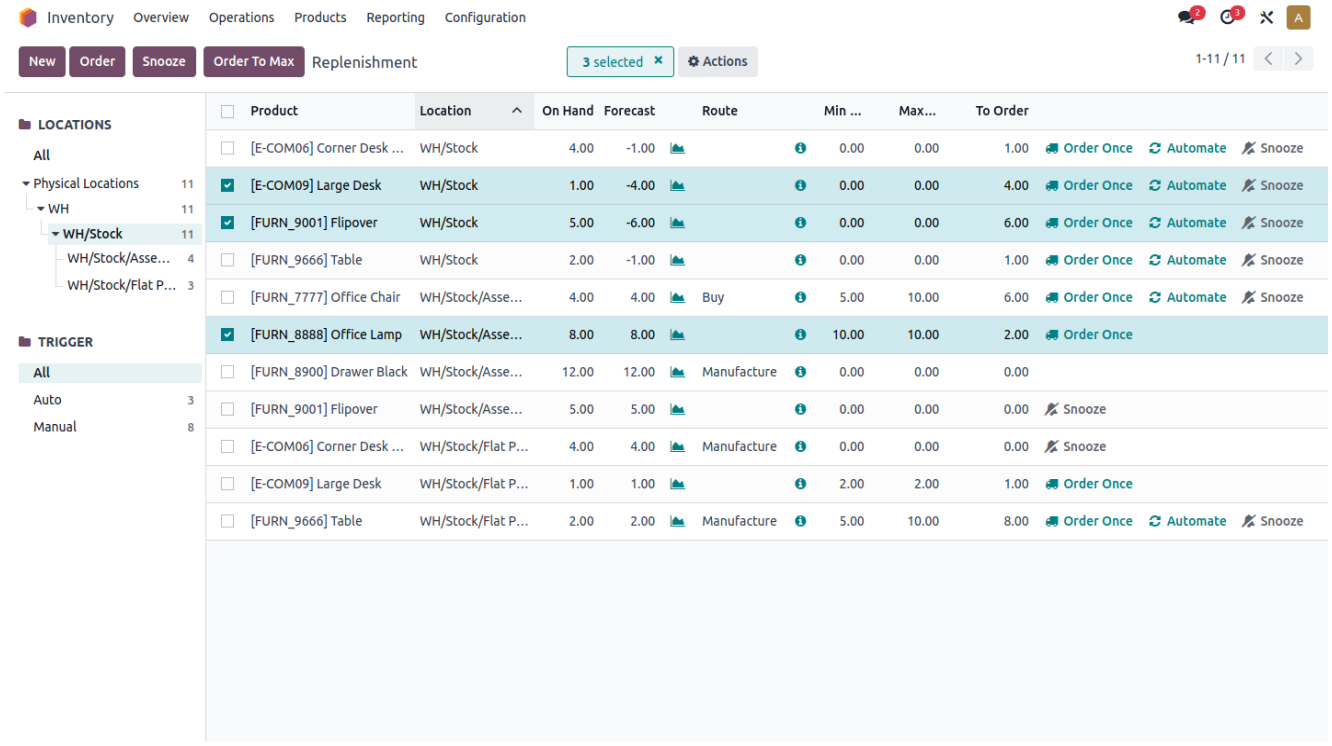

Рисунок 1.5 – Вікно застосунку Odoo Inventory

Для оцінки ефективності та відповідності потребам роботи виконано порівняльний аналіз зазначених аналогів за критеріями, які безпосередньо впливають на процес оптимізації розміщення посилок (див. табл. 1.1).

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерії	Warehouse Expert	SAP EWM	IT-Enterprise WMS	SmartStock WMS	Odoo Inventory	Власна програма
Автоматичне визначення сектору	-	+	+	-	-	+
Адаптованість для малого бізнесу	+	-	-	+	+	+

Продовження таблиці 1.1

Критерії	Warehouse Expert	SAP EWM	IT-Enterprise WMS	SmartStock WMS	Odoo Inventory	Власна програма
Вартість та доступність	+	-	-	+	+	+
Інтуїтивність інтерфейсу	+	-	-	+	+	+
Візуалізація заповненості складу	-	+	+	+	-	+
Загальна оцінка	60%	40%	40%	80%	60%	100%

Як свідчить порівняльна характеристика, жодна з наявних систем не поєднує одночасно автоматизоване визначення сектору, доступність, гнучкість і візуалізацію стану складу. Саме ці особливості реалізовані у власній розробці, що робить її ефективною альтернативою для малих та середніх логістичних об’єктів.

Таким чином, створення власного програмного рішення з інтелектуальними алгоритмами розміщення, простим інтерфейсом та можливістю автономного використання є доцільним і актуальним. Це дозволяє заповнити прогалини, притаманні наявним аналогам, і надати користувачам саме ті інструменти, які потрібні у щоденній роботі зі складом.

1.3 Аналіз методів розв’язання задачі

Розробка програмних модулів для системи оптимізації розміщення посилок на складі передбачає вибір ефективних методів і підходів, здатних враховувати параметри вантажів, поточний стан секторів і забезпечувати

максимально рівномірне заповнення складських площ. У контексті поставленої задачі доцільно проаналізувати основні методи логістичної оптимізації, що можуть бути адаптовані до реалізації у програмному забезпеченні.

Одним із найпоширеніших підходів є метод жорстко заданих правил (rule-based). Він полягає у використанні фіксованих умов для прийняття рішень: наприклад, якщо вага менше 10 кг – помістити у сектор А, якщо об’єм більше певного значення – у сектор В тощо. Такий підхід є простим у реалізації, не потребує складних обчислень і легко змінюється в коді. Проте основним його недоліком є відсутність гнучкості та неможливість адаптації до змін у завантаженості складу – навіть за мінімального перевищення ліміту система не зможе запропонувати альтернативне рішення [12].

Іншим методом, що часто застосовується у логістиці, є жадібний алгоритм (greedy algorithm). Він приймає рішення на основі найближчої локальної оптимізації – наприклад, посилка розміщується у сектор, де наразі найбільше вільного місця. Перевага цього підходу – швидкість прийняття рішень та простота реалізації. Однак він не враховує довгострокових наслідків: заповнюючи найбільш вільні сектори без урахування типу або групування вантажів, можна швидко створити ситуацію, коли залишаться лише невідповідні за параметрами місця [13].

Більш універсальним є алгоритм «на основі обмежень» (constraint-based placement). У цьому випадку кожен сектор розглядається як об’єкт, що має обмеження (максимальна вага, об’єм, кількість місць), а кожна посилка – як об’єкт з конкретними вимогами. Система підбирає сектор, який задовольняє всі обмеження. Такий метод дозволяє динамічно враховувати зміну параметрів, а також обирати з кількох допустимих варіантів найкращий – наприклад, за мінімальним відхиленням від середнього навантаження. Він також добре масштабується, але потребує налаштування логіки перевірки обмежень та додаткових перевірок на конфлікти [14].

У складніших системах можуть використовуватись евристичні або комбіновані методи, які поєднують декілька підходів. Наприклад, жадібний

алгоритм можна обмежити рамками допустимих значень об’єму або ваги, що дає змогу враховувати як оптимальність, так і допустимість. Такий підхід може дати хороші результати при малих ресурсах, але його якість залежить від налаштування логіки, яка визначає, що вважати допустимим або пріоритетним [15].

Також розглядається застосування елементів кластеризації — коли сектори групуються за типом посилок або завантаженістю, а розміщення здійснюється у межах найбільш релевантної групи. Цей метод особливо корисний для великих складів, однак потребує додаткових обчислень та логіки підтримки стану кожного кластера [16].

У межах бакалаврської кваліфікаційної роботи було вирішено застосувати метод обмежень у поєднанні з логікою жадібного вибору. Зокрема, кожна посилка аналізується на відповідність доступним секторам за об’ємом і вагою, після чого з допустимих варіантів обирається той, який має найменший рівень завантаження. Це дозволяє уникнути як перевантаження окремих зон, так і втрати простору в менш заповнених секторах. Реалізація алгоритму виконана у вигляді окремої функції, що автоматично викликається під час додавання нової посилки та передає результат у форму збереження.

Такий підхід забезпечує баланс між простотою реалізації, швидкістю обробки і логічною обґрунтованістю вибору. Крім того, у подальшому його можна масштабувати або вдосконалити шляхом введення додаткових критеріїв — наприклад, категорій вантажів, групування за маршрутами тощо.

Отже, аналіз існуючих методів оптимізації розміщення показує, що комбінування методів обмежень і локальної оптимізації є найбільш прийнятним для проєктованої системи. Обрана стратегія дозволяє забезпечити ефективне управління простором складу та покращити логістику зберігання без ускладнення структури програмного забезпечення.

1.4 Постановка задач бакалаврської кваліфікаційної роботи

На основі аналізу сучасного стану автоматизованих систем управління

складом, порівняння аналогічних програмних рішень і розгляду можливих методів реалізації алгоритмів логістичної оптимізації було сформульовано комплекс задач, вирішення яких є необхідним для досягнення мети бакалаврської роботи.

Основним завданням є розробка програмних модулів, що забезпечують автоматизоване розміщення посилок на складі з урахуванням їхніх фізичних характеристик та завантаженості секторів. Для цього необхідно реалізувати низку функціональних і технічних рішень, які забезпечать ефективну взаємодію між користувачем та системою, обробку даних, а також наочне представлення результатів.

У рамках бакалаврської кваліфікаційної роботи передбачається виконання таких задач:

- розробити структуру локальної бази даних для зберігання інформації про посилки;

- створити модуль додавання нових посилок, який дозволяє вводити основні параметри (ідентифікатор, вага, габарити, опис) і автоматично визначати рекомендований сектор зберігання;

- реалізувати алгоритм оптимізації розміщення, який враховує поточну завантаженість секторів, обмеження за вагою та об’ємом і виконує вибір найоптимальнішого сектору;

- забезпечити функціональність перегляду та пошуку посилок;

- розробити модуль з можливістю перегляду характеристик секторів та рівня заповненості;

- створити систему авторизації користувачів з розмежуванням доступу (адміністратор, оператор);

- реалізувати графічний інтерфейс користувача, що забезпечує доступ до всіх основних функцій та інтуїтивну взаємодію з системою;

- здійснити тестування функціоналу системи, перевірку правильності обчислень, коректності розміщення посилок і надійності збереження даних.

Усі перелічені задачі мають бути реалізовані у вигляді єдиної програмної

системи, що функціонує автономно, без потреби у зовнішньому серверному середовищі. Основними критеріями ефективності реалізації є зручність користування, логічність побудови, коректність обчислень і стабільність роботи застосунку в умовах реального навантаження.

1.5 Висновки

У першому розділі було проведено ґрунтовний аналіз сучасного стану автоматизованих систем управління складом, а також розглянуто існуючі підходи до вирішення задач логістичної оптимізації з урахуванням фізичних параметрів вантажів, обмежень складських зон і наявного вільного простору. Окрему увагу приділено оцінці ефективності інтеграції таких систем у логістичні процеси з позиції гнучкості, масштабованості та адаптивності до потреб підприємств різного масштабу.

Було виявлено, що більшість сучасних рішень або занадто складні у впровадженні та потребують значних витрат на налаштування і навчання персоналу, або не мають вбудованої логіки для автоматичного визначення оптимального місця зберігання вантажів, обмежуючись шаблонними механізмами розміщення.

Серед найбільш відомих аналогів було проаналізовано системи Warehouse Expert, SAP EWM, WMS від IT-Enterprise, SmartStock WMS та Odoo Inventory. Кожна з них має свої переваги та недоліки, які було детально розглянуто у відповідних підрозділах.

Проведений порівняльний аналіз дозволив об’єктивно оцінити сильні та слабкі сторони кожного з інструментів і водночас підкреслити відсутність комплексного рішення, яке б одночасно поєднувало інтелектуальні алгоритми розміщення, високу зручність використання та доступність для підприємств малого й середнього бізнесу без потреби в складній інфраструктурі.

Також було розглянуто методи, які можна використати для автоматизації розміщення посилок на складі. Зокрема, проаналізовано методи жорстких правил, жадібної логіки, підходів на основі обмежень і евристичних стратегій.

На основі порівняння переваг і недоліків було обрано комбінований підхід, який дозволяє забезпечити гнучкість, точність і швидкість прийняття рішень.

У результаті аналізу було сформульовано перелік завдань, які необхідно вирішити у межах бакалаврської кваліфікаційної роботи.

Таким чином, було обґрунтовано доцільність розробки власного програмного рішення, яке забезпечить автоматизоване та оптимальне управління розміщенням посилок на складі з урахуванням актуальних логістичних вимог і можливостей сучасних інформаційних технологій.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Реалізація програмного забезпечення системи

Для реалізації програмного забезпечення системи оптимізації розміщення посилок на складі було обрано мову програмування Python, яка на сьогодні є однією з найпопулярніших платформ для розробки інформаційних систем. Python забезпечує високу швидкість розробки завдяки своїй лаконічній та зручній синтаксичній структурі, широкому спектру вбудованих можливостей, а також активній спільноті, що постійно підтримує й оновлює численні бібліотеки [17]. Особливо корисними для цієї бакалаврської роботи стали бібліотеки для роботи з базами даних, обробки структурованої інформації та реалізації алгоритмів логістичної оптимізації.

Для створення графічного інтерфейсу користувача було обрано бібліотеку PyQt6, яка дозволяє реалізувати сучасний, адаптивний, багатофункціональний GUI. Вона забезпечує інтеграцію вікон, форм введення, панелей керування, повідомлень, таблиць та інструментів візуалізації, що значно підвищує зручність використання системи навіть непідготовленими користувачами.

У якості сховища даних обрано вбудовану базу даних SQLite. Цей вибір зумовлено її простотою у використанні, відсутністю потреби у встановленні зовнішнього серверного ПЗ, можливістю роботи в автономному режимі, а також достатньою функціональністю для зберігання й обробки даних в умовах невеликого або середнього логістичного підприємства [18].

Система функціонує на основі обробки таких вхідних даних:

1. Дані про посилки — унікальний трекінг-номер, ПІБ відправника й одержувача, вага, габарити (довжина, ширина, висота), дата прибуття, опис вмісту.

2. Інформація про сектори складу — максимально допустима вага, об’єм, поточний рівень заповненості, унікальний ідентифікатор сектору.

3. Облікові дані користувачів — логін, пароль, роль у системі (адміністратор або оператор).

4. Параметри пошуку — ключові атрибути для ідентифікації або фільтрації посилок у базі даних.

Для обробки вхідних даних реалізовано модуль введення посилок, який надає користувачеві форму для введення всіх обов’язкових параметрів з валідацією полів. Після підтвердження введення активується алгоритм оптимального розміщення, який виконує пошук найкращого сектору відповідно до допустимих обмежень.

Модуль авторизації забезпечує автентифікацію користувачів шляхом перевірки логіна та пароля. Усі користувачі після входу мають доступ до основного функціоналу системи, включно з додаванням нових посилок, переглядом і пошуком записів. На поточному етапі розробки система не передбачає диференціації прав доступу за ролями (адміністратор, оператор), однак архітектура проєкту дозволяє реалізувати цей механізм у подальших версіях, що забезпечить гнучке управління доступом до функціональних модулів.

Модуль пошуку реалізовано з можливістю фільтрації та миттєвого оновлення даних без перезавантаження інтерфейсу. Користувач може здійснювати пошук за унікальним номером посилки.

Вся система взаємодіє з внутрішньою структурою бази даних, що була ретельно спроєктована з урахуванням принципів нормалізації та забезпечення логічного поділу даних між таблицями.

Запити до бази даних реалізовані за допомогою бібліотеки sqlite3, що дозволяє виконувати операції швидко й без зайвого навантаження на систему. Усі дії з базою, включно з додаванням, оновленням і пошуком, реалізовані через безпечні SQL-запити.

Таким чином, система оперує багатим набором вхідних даних, що дозволяє формувати повну картину складських операцій і забезпечувати точну логістичну оптимізацію розміщення посилок. Поєднання технологій Python, PyQt6 і SQLite створює надійну основу для побудови гнучкої, автономної та зручної у використанні системи для логістичних задач.

2.2 Розробка інтерфейсу програмного додатку

Під час розробки графічного інтерфейсу програмного забезпечення особливу увагу було приділено його логічній структурі, зручності у використанні та візуальній зрозумілості для кінцевого користувача. Інтерфейс створено з використанням бібліотеки PyQt6, яка надає широкі можливості для побудови інтерактивних вікон, форм, таблиць, панелей та інших елементів GUI.

Інтерфейс системи побудований таким чином, щоб мінімізувати кількість кроків для виконання основних дій: додавання посилки, перегляд секторів складу, пошук за параметрами, авторизація користувача тощо. Основним кольором оформлення було обрано світло-сірий фон із синіми акцентами для кнопок та активних елементів, що забезпечує візуальний комфорт при щоденному використанні [19].

Після запуску програми, користувача зустрічає екран авторизації (рис. 2.1), де необхідно ввести логін і пароль.

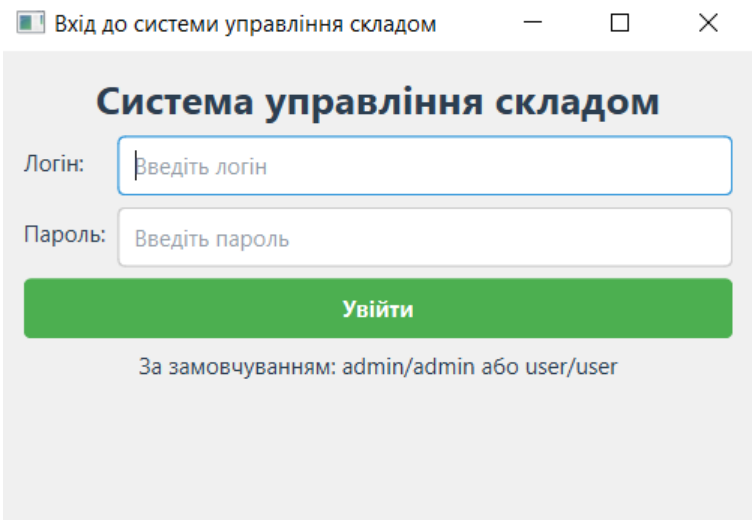

Рисунок 2.1 – Вікно авторизації

Після входу у систему відкривається головне вікно, яке містить у собі дані про користувача та кнопку «Вийти» (рис. 2.2).

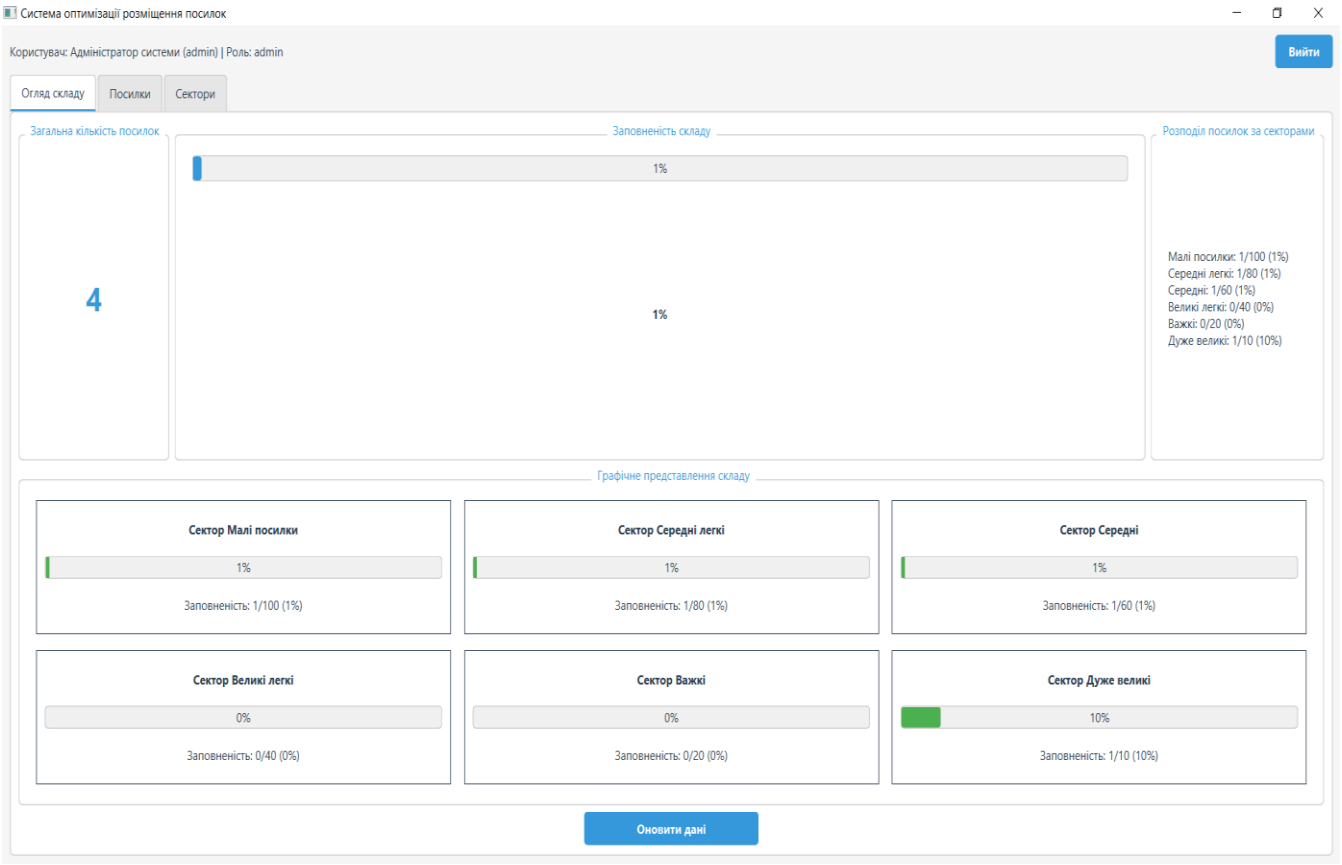

Рисунок 2.2 – Головне вікно програми

Головне вікно містить у собі кілька вкладок. Першою є вкладка «Огляд складу», яка реалізує наочну візуалізацію поточного стану заповненості всіх секторів. Ця вкладка дозволяє користувачеві миттєво оцінити, наскільки завантажений кожен сектор, і зрозуміти загальний стан складських площ.

У вікні «Огляд складу» кожен сектор відображається у вигляді окремого блоку, що містить:

- назву сектору;
- допустиму максимальну кількість посилок;
- поточні показники завантаженості;
- індикатор заповненості у вигляді кольорових прогрес-барів.

Другою вкладкою є вкладка «Посилки» (рис. 2.3), яка відображає таблицю з усіма доданими посилками, їх ідентифікаторами, відправниками/отримувачами, вагою, сектором та статусом. Також ця вкладка

містить у собі функцію пошуку посилки за номером та функцію «Додати посилку».

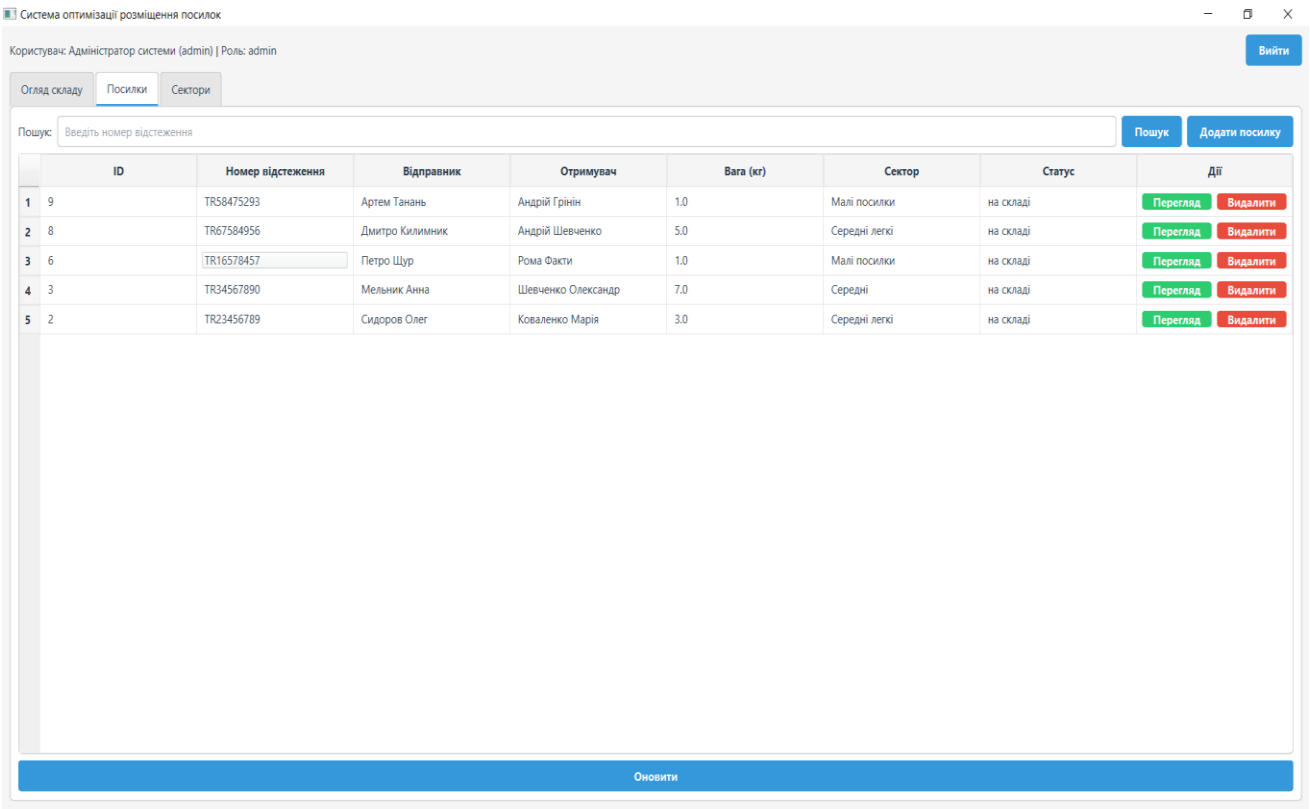

Рисунок 2.3 – Вкладка «Посилки»

При натисканні кнопки «Додати посилку» відкривається вікно «Додати нову посилку» (рис. 2.4), у якому потрібно вказати дані про номер посилки, відправника та отримувача, вагу та габарити. Також за бажанням можна додати опис посилки.

Після того, як користувач увів всі необхідні дані, він може натиснути кнопку «Розрахувати рекомендований сектор» та програма автоматично визначить місце, яке слід зайняти відправленню.

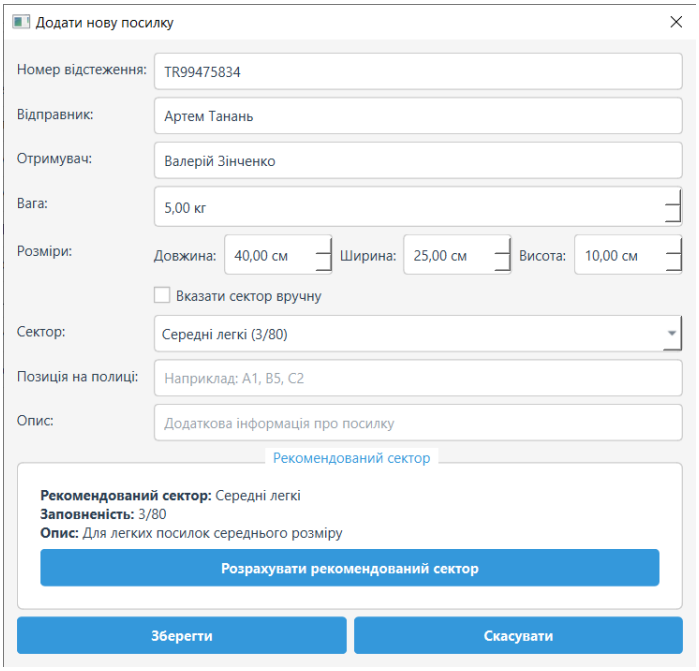

Рисунок 2.4 – Вікно «Додати нову посилку»

Останнім вікном є вікно «Сектори», яке дає змогу переглядати існуючі сектори, їх обмеження (діапазон ваги, діапазон об’єму), опис, а також відсоток заповненості, який візуалізується через прогрес-бари (рис. 2.5).

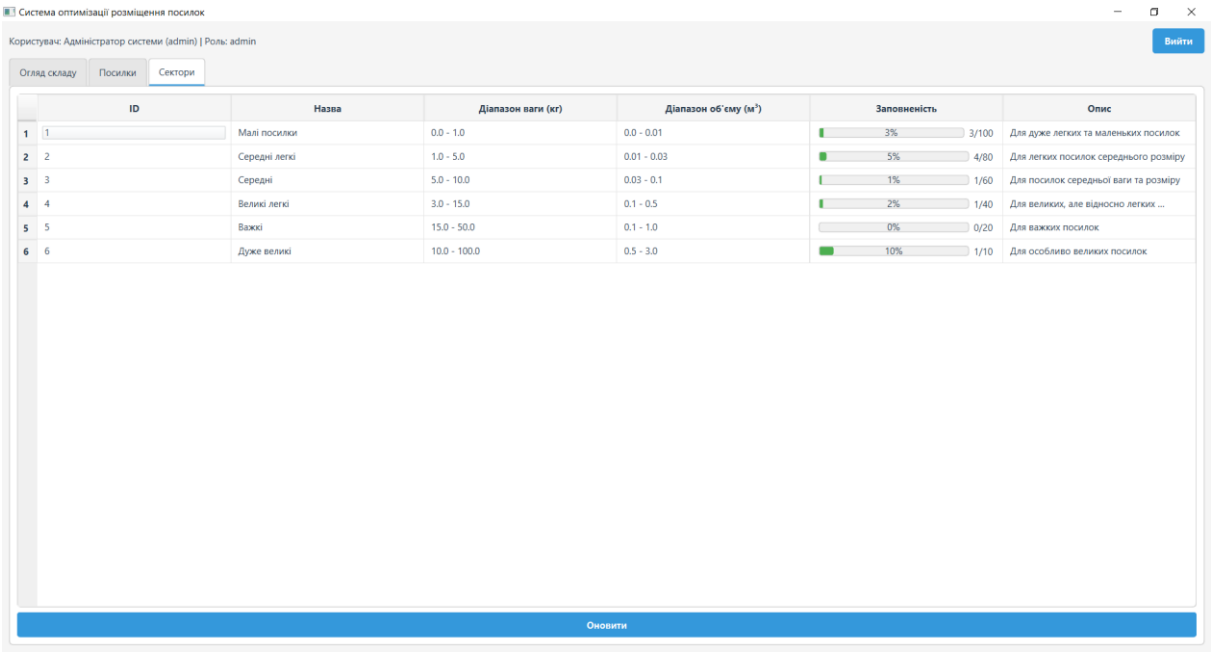

Рисунок 2.5 – Вкладка «Сектори»

Також передбачено контекстне меню та кнопки швидкого доступу для редагування або видалення записів, оновлення таблиць, очищення полів форми, виходу із системи. Навігація між вкладками виконується через бічне меню, яке розташоване зліва і дозволяє швидко перемикатися між основними розділами.

Інтерфейс розроблено з урахуванням принципів мінімалізму — кожна дія доступна у межах одного чи двох кліків, всі елементи мають підписи, а системні повідомлення інформують користувача про результати його дій (рис. 2.6).

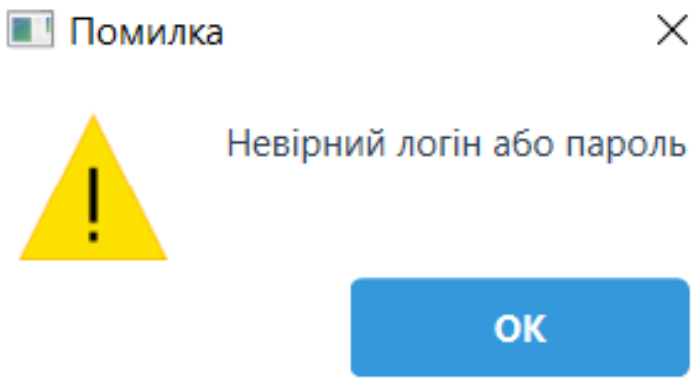

Рисунок 2.6 – Приклад системного повідомлення

Інтерфейс протестовано на різних роздільностях екранів і забезпечує стабільне відображення як у віконному, так і повноекранному режимі. Усі текстові поля перевіряються на коректність введення, що запобігає збереженню некоректних або неповних даних.

Таким чином, розроблений графічний інтерфейс забезпечує повний функціонал системи в зручному, інтуїтивно зрозумілому вигляді, що дає змогу ефективно взаємодіяти з системою навіть користувачам без спеціальної технічної підготовки.

2.3 Розробка моделі системи

Для формалізації логіки роботи розробленої системи оптимізації розміщення посилок на складі було створено модель її функціонування у вигляді

UML-діаграми діяльності, яка відображає послідовність дій користувача та відповідні реакції системи (рис. 2.7). Модель дозволяє краще зрозуміти структуру взаємодії між модулями інтерфейсу, алгоритмами та базою даних [20].

Робота з програмним забезпеченням починається з вікна авторизації, де користувач вводить логін і пароль. У разі успішної перевірки даних система відкриває головне вікно, що містить бокове меню для навігації між основними вкладками: «Огляд складу», «Посилки», «Сектори».

На вкладці «Огляд складу» користувач бачить блоки, що відповідають кожному сектору складу. Кожен блок містить інформацію про назву сектору, допустиму кількість посилок, поточну кількість і графічний індикатор заповненості. Це дозволяє миттєво оцінити ситуацію на складі та за необхідності перейти до редагування секторів або додавання нових посилок.

При переході на вкладку «Посилки» користувач переглядає таблицю з усіма відправленнями, здійснює пошук за номером, а також має можливість додати нову посилку. Натискання кнопки «Додати посилку» відкриває окреме вікно, де користувач вводить необхідні дані: номер, відправника, отримувача, вагу, габарити та, за бажанням, опис. Всі поля перевіряються на коректність заповнення. Після цього користувач натискає кнопку «Розрахувати рекомендований сектор».

У цей момент запускається алгоритм визначення оптимального сектору, який аналізує поточну заповненість секторів, допустимі вагові та об’ємні обмеження, і повертає назву найбільш придатного сектору. Якщо підходящого сектору не знайдено, система повідомляє про це. У разі успіху рекомендований сектор додається до інформації про посилку, після чого всі дані зберігаються до бази даних.

На вкладці «Сектори» адміністратор може переглядати наявні сектори, їх характеристики, а також рівень заповнення кожного. Крім цього, реалізовано можливість редагування секторів, додавання нових або видалення неактуальних.

У будь-який момент користувач може оновити таблиці або вийти із системи за допомогою відповідних кнопок. Усі дії супроводжуються

повідомленнями про успіх чи помилки, а перемикання між вкладками виконується через ліву верхню панель навігації.

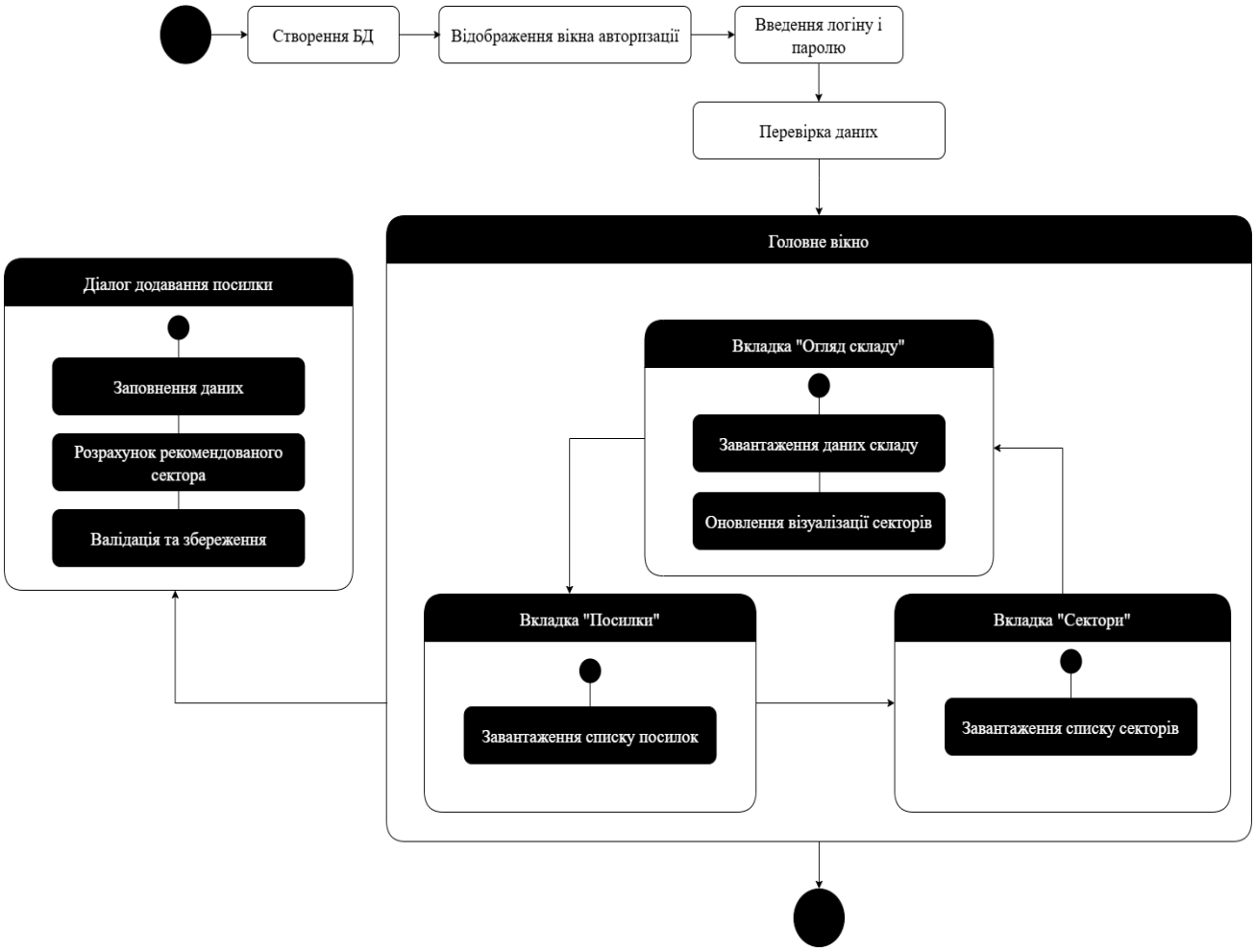

Рисунок 2.7 – Модель роботи програми у вигляді діаграми діяльності програмних засобів

Таким чином, розроблена UML-діаграма діяльності успішно формалізує логіку роботи системи, відображаючи послідовність взаємодій між користувачем та програмними модулями. Вона наочно демонструє основні етапи роботи з інтерфейсом, запуск алгоритмів оптимізації та обробку даних у базі, що забезпечує цілісне розуміння структури та функціонування системи.

2.4 Розробка алгоритмів роботи системи

Для забезпечення коректної послідовності взаємодії користувача з програмним забезпеченням було розроблено алгоритм роботи системи у вигляді блок-схеми (рис. 2.8), що відображає загальну логіку функціонування додатку — від моменту запуску до виходу з програми. Алгоритм охоплює процес авторизації, перемикання між функціональними вкладками, роботу з даними складу, посилками та секторами.

Робота програми починається з ініціалізації, під час якої виконується створення або перевірка існування бази даних. Якщо база відсутня, вона створюється автоматично. Наступним етапом є відображення вікна авторизації, де користувач має ввести логін і пароль для доступу до системи.

Після введення облікових даних система виконує перевірку правильності введення. Якщо логін або пароль некоректні, користувач отримує повідомлення про помилку, і система повертає його до повторного введення. У разі успішної авторизації відкривається головне вікно програми.

Далі запускається основний цикл взаємодії користувача з програмою, що передбачає роботу з функціональними вкладками. Користувач здійснює вибір однієї з доступних функцій:

- «Огляд складу» – перегляд поточного стану заповненості секторів у вигляді блоків з індикаторами;
- «Посилки» – перегляд, пошук, додавання нових записів;
- «Сектори» – перегляд списку секторів, їх обмежень та рівня заповненості.

Вибір вкладки ініціює запуск відповідного функціонального модуля. Наприклад, при переході на вкладку «Посилки» стає доступним додавання нової посилки, запуск алгоритму визначення оптимального сектору тощо. Усі дії супроводжуються перевіркою коректності введених даних і збереженням інформації в локальну базу даних SQLite.

Після завершення роботи з вкладками користувач може вийти з програми, натиснувши кнопку «Вихід». Це призводить до завершення основного циклу і закриття програми.

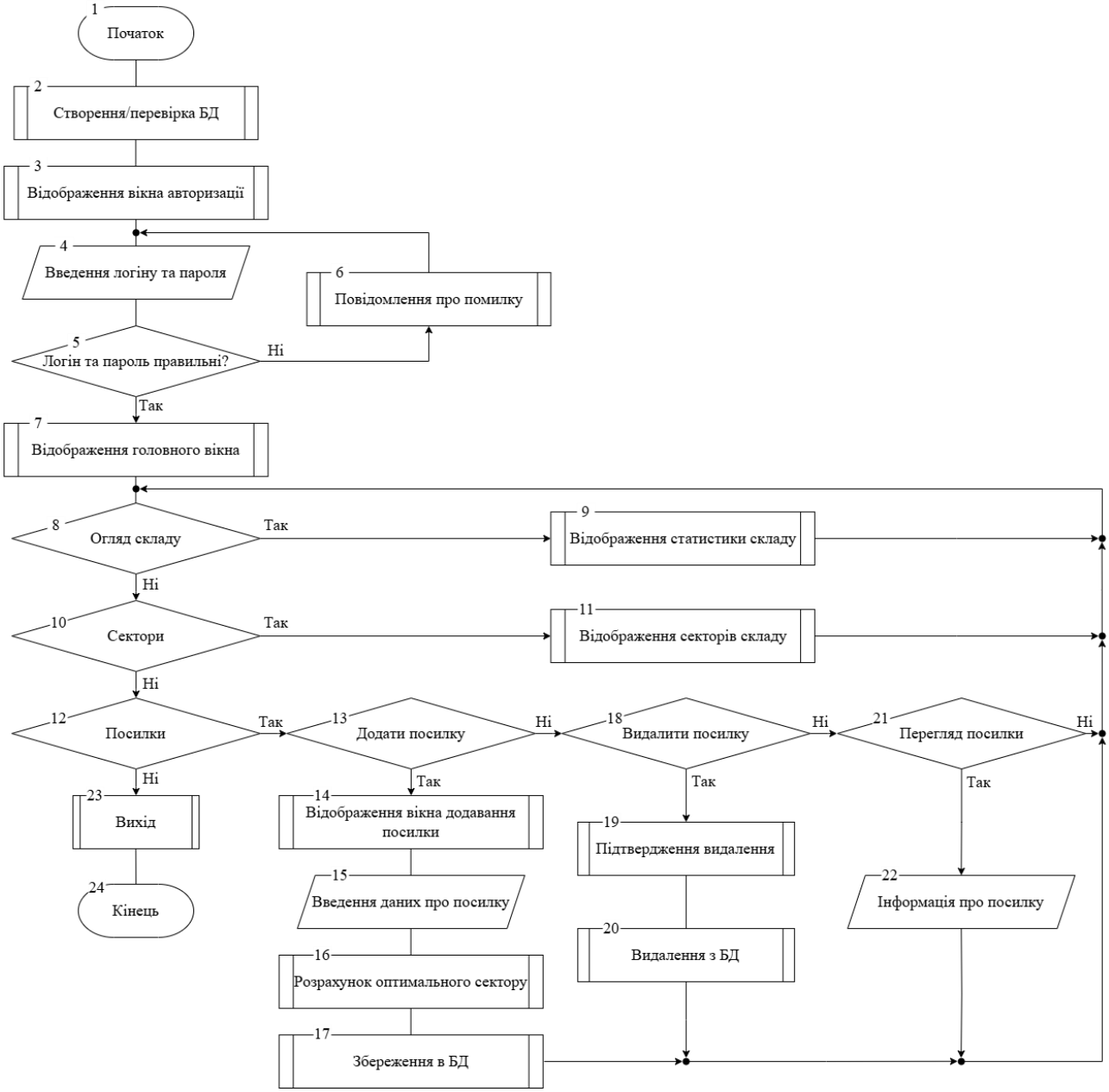

Рисунок 2.8 – Алгоритм роботи системи

Таким чином, запропонований алгоритм забезпечує логічну, послідовну та інтуїтивно зрозумілу структуру роботи з програмним забезпеченням. Він дозволяє легко орієнтуватися у функціоналі системи, що є особливо важливим для кінцевих користувачів без спеціальної технічної підготовки. Побудована схема також стала основою для реалізації програмного коду, тестування функціональних модулів і подальшого розширення можливостей системи.

2.5 Висновки

У другому розділі було проаналізовано вхідні дані, які обробляються програмною системою під час взаємодії з користувачем, а також визначено основні типи інформації, що вводяться, зберігаються та використовуються для прийняття рішень.

Окрему увагу приділено створенню графічного інтерфейсу користувача з використанням бібліотеки PyQt6. Інтерфейс було реалізовано відповідно до принципів мінімалізму та функціональної зручності, що дозволяє ефективно виконувати всі необхідні дії з управління складом, посилками та секторами.

Було розроблено модель функціонування системи у вигляді UML-діаграми діяльності, яка демонструє загальну послідовність дій користувача та логіку переходів між етапами роботи програми. Також описано ключові алгоритми системи, зокрема алгоритм роботи системи.

Усі ці елементи лягли в основу логічної структури програмного забезпечення, що дозволяє забезпечити його стабільну, передбачувану та зручну експлуатацію у реальних умовах.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ СИСТЕМИ ОПТИМІЗАЦІЇ РОЗМІЩЕННЯ ПОСИЛОК

3.1 Обґрунтування вибору засобів для реалізації

У процесі розробки програмного забезпечення для оптимізації розміщення посилок на складі ключовим етапом стало обґрунтоване обрання технологій, інструментів та мов програмування, які б забезпечували ефективну реалізацію всіх функціональних модулів системи — від інтерфейсу користувача до логіки обробки даних та зберігання інформації.

Основною мовою програмування було обрано Python — високорівневу мову з простим синтаксисом, яка підтримує швидку розробку, інтеграцію з базами даних, обробку логіки програми та створення зручних GUI-додатків. Python має багату екосистему бібліотек, що дозволяє ефективно реалізовувати навіть складні алгоритми оптимізації та працювати з локальними базами даних.

Для створення графічного інтерфейсу користувача використано PyQt6 — популярну бібліотеку, яка дозволяє створювати сучасні, адаптивні та багатофункціональні графічні інтерфейси. Вона підтримує роботу з віджетами, таблицями, діалоговими вікнами, панелями управління та іншими елементами GUI, що забезпечило реалізацію усіх необхідних вікон: авторизації, перегляду посилок, секцій складу, додавання нових записів тощо. Перевагою PyQt6 є також сумісність із Windows і Linux, а також можливість масштабування на повноекранні режими.

Для зберігання даних обрана SQLite — вбудована реляційна база даних, яка не потребує окремого сервера та дозволяє працювати з файлами бази локально. Її використання дає змогу реалізувати повністю автономну програму без залежності від мережевого середовища. Через бібліотеку sqlite3, яка входить до стандартної бібліотеки Python, забезпечено повноцінну взаємодію з базою даних — зчитування, запис, редагування та пошук даних.

Для візуалізації заповненості секторів складу використано стандартні графічні елементи PyQt6, зокрема прогрес-бари, які динамічно оновлюються

залежно від рівня заповнення. Це дало змогу надати користувачу візуальне представлення поточного стану складських приміщень без потреби в зовнішніх візуалізаційних фреймворках.

Альтернативними варіантами для реалізації GUI були Tkinter, Kivy та wxPython, однак ці бібліотеки не забезпечували достатнього рівня кастомізації або мали обмеження в роботі з таблицями та стилізацією, що є критичним для системи складського обліку. Саме тому PyQt6 було обрано як найбільш гнучке та потужне рішення.

Для середовища розробки використовувалося Visual Studio Code, що забезпечує підтримку інтеграції з Python, зручне автодоповнення, перевірку синтаксису, керування проєктами, а також зручний перегляд структури коду.

Таким чином, поєднання Python, PyQt6, SQLite та Visual Studio Code дозволило реалізувати всі необхідні компоненти системи в єдиному середовищі без потреби у складних зовнішніх інструментах. Вибрані технології забезпечують простоту, стабільність, розширюваність та кросплатформеність програмного продукту.

3.2 Реалізація модуля керування посилками та алгоритму визначення оптимального сектора

Розробка програмного модуля спрямована на забезпечення повноцінного управління посилками у системі оптимізації розміщення посилок на складі. Для взаємодії з користувачем застосовується графічний інтерфейс, побудований з використанням PyQt6, а зберігання даних реалізовано на базі SQLite. Програма включає комплексну реалізацію функціональних можливостей: від введення та збереження нових посилок до визначення оптимального сектора, в якому має бути розміщена посилка, а також подальшого перегляду та видалення інформації.

Для введення даних про нову посилку створено клас AddPackageDialog. Цей клас формується на основі діалогового вікна, яке містить низку елементів:

- поля для введення номера відстеження, відправника, отримувача, ваги та

розмірів (довжина, ширина, висота);

- можливість ручного або автоматичного вибору сектора за допомогою прапорця (QCheckBox) та випадаючого списку (QComboBox);

- елемент для введення додаткової інформації (позиція на полиці, опис посилки).

При натисканні кнопки «Зберегти» відбувається перевірка обов'язкових полів, а також даних щодо наявності вільного місця у вибраному секторі. У разі відсутності помилок викликається метод accept, який підтверджує введення даних. Блок-схему методу зображено на рисунку 3.1.

Рисунок 3.1 – Блок-схема методу «accept»

Фрагмент коду з методу «accept» наведено на рисунку Г.1.

Цей код гарантує, що користувач не зможе зберегти посилку з неповними даними або з вибраним сектором, що вже заповнений.

Для відображення докладної інформації про посилку реалізовано клас PackageInfoDialog. Він здійснює завантаження даних із бази даних за допомогою SQL-запиту із використанням з'єднання з SQLite, після чого інформація красиво формується у вигляді таблиці з використанням віджетів QLabel, QGridLayoutQLabel, QGridLayout. Це дозволяє користувачу швидко переглянути деталі посилки, такі як вага, розміри, сектор розміщення, позиція на полиці, статус та інші параметри.

Однією з ключових задач системи є автоматизований вибір оптимального сектора для розміщення посилки. Цей алгоритм імплементовано у методі calculate_recommended_sector класу AddPackageDialog. Основні етапи алгоритму можна описати наступним чином:

1. Перетворення розмірів. Дані користувача вводяться у сантиметрах, тому необхідно перетворити їх у метри для розрахунку об'єму (рис. Г.2).

2. Пошук посилки за критеріями. Алгоритм порівнює вагу та об'єм посилки із межами допустимих значень для кожного сектора. Якщо значення входять у встановлені межі між minweight і maxweight, minvolume і maxvolume, між min_weight і max_weight, min_volume і max_volume, проводиться подальша оцінка. Розраховується коефіцієнт заповненості сектора fill_factor, а також визначається наскільки параметри посилки (вага та об'єм) наближені до середнього значення даного сектора (рис. Г.3).

Таким чином, остаточна оцінка match_score по кожному сектору дозволяє вибрати найбільш відповідний варіант. Якщо повна відповідність не знайдена, виконується пошук за мінімальною "відстанню" від параметрів посилки до допустимих значень даного сектора.

3. Автоматичне відображення рекомендації. Отриманий результат використовується для автоматичного вибору рекомендованого сектора у списку, після чого користувачу виводиться інформація про обраний сектор: його назва,

заповненість та опис. Нижче наведено фрагмент коду з методу calculate_recommended_sector (рис. Г.4), який відповідає за автоматичне відображення рекомендації. Як тільки обрано оптимальний сектор, його дані використовуються для автоматичного вибору в комбобоксі.

Таким чином, алгоритм забезпечує гнучку та адаптивну логіку розподілу посилок за секторами, що дозволяє врахувати як фізичні характеристики посилки, так і поточний стан завантаженості секторів.

Головне вікно системи, реалізоване у класі MainWindow, інтегрує функціонал керування посилками та відображення статистичних даних:

- на вкладці «Посилки» користувач може здійснювати пошук, перегляд, додавання й видалення посилок. Для цього використовується таблиця, яка заповнюється даними із бази даних;

- метод add_new_package викликає діалог додавання посилки, після чого, за допомогою методу save_new_package, здійснюється перевірка унікальності номера відстеження, запис даних у таблицю packages та оновлення поточного заповнення сектора.

- також реалізоване журналювання дій користувача через вставку записів у таблицю activity_logs.

Фрагмент коду з методу збереження посилки наведено на рисунку Г.5.

Цей підхід гарантує узгоджене оновлення даних із бази та забезпечує цілісність інформації при виконанні операцій над посилками.

Реалізація даного модуля демонструє інтегроване використання графічного інтерфейсу для роботи із базою даних, що робить систему гнучкою та масштабованою. Завдяки чітко структурованим класам і методам забезпечується як ефективне введення та обробка даних, так і зручний алгоритм розподілу посилок по секторах, що враховує як характеристику посилки, так і поточну завантаженість сектора. Це дозволяє оптимізувати логістичні процеси та підвищити продуктивність роботи складу.

Блок-схему методу save_new_package зображено на рисунку 3.2.

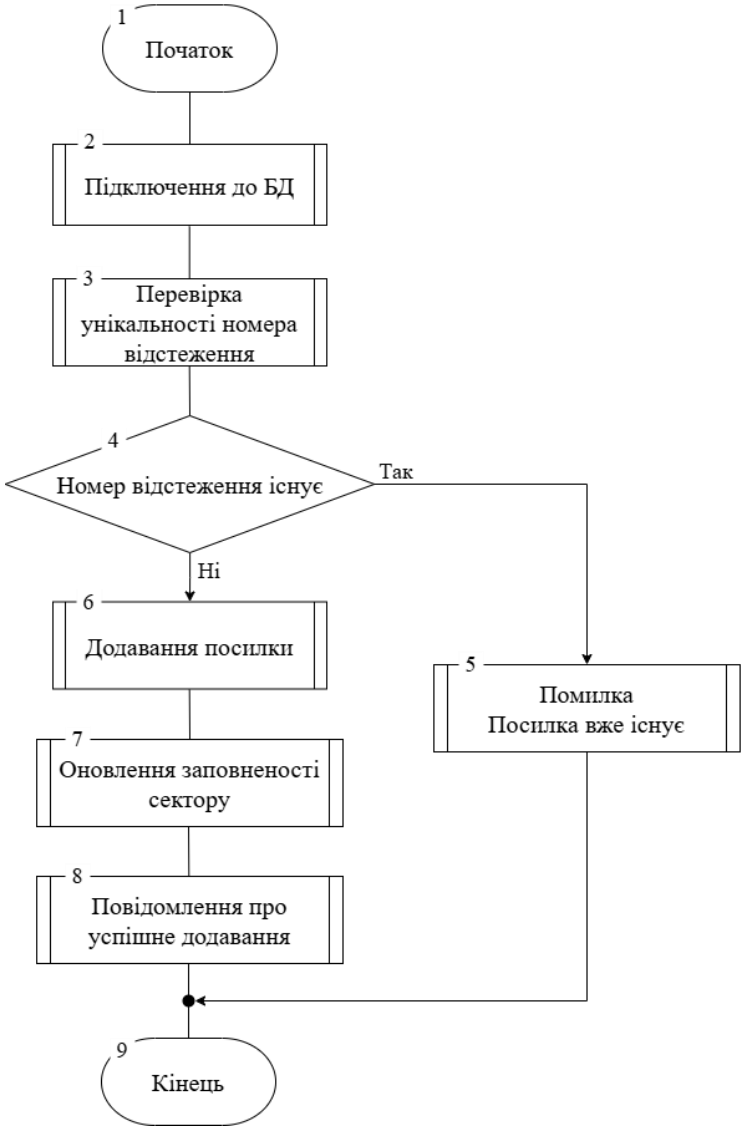

Рисунок 3.2 – Блок-схема методу збереження посилки

Розглянемо метод filter_packages, який реалізовує фільтрацію посилок за пошуковим запитом. Цей метод знаходиться в класі MainWindow та виконує наступні завдання:

1. Зчитування запиту. Метод отримує текст із поля введення пошукового запиту. Якщо поле порожнє, відображаються всі посилки за допомогою виклику методу load_packages_data.

2. Формування SQL-запиту. Якщо запит не пустий, виконується підключення до бази даних SQLite та виконання SQL-запиту, який шукає посилки за відповідністю номеру відстеження, імені відправника або

отримувача. Пошук здійснюється у нижньому регістрі для забезпечення нечутливості до регістру.

3. Оновлення таблиці. Отримані результати заповнюються у таблицю посилок. Для кожного рядка створюється набір кнопок (перегляд та видалення), які дозволяють користувачу виконати відповідні дії над вибраною посилкою.

Фрагмент коду методу filter_packages наведено на рисунку Г.6.

Цей метод дозволяє динамічно оновлювати відображення посилок в залежності від введеного запиту, що сприяє зручнішій роботі користувача із системою оптимізації розміщення посилок. Таким чином, фільтрація даних забезпечує швидкий доступ до потрібної інформації та інтегрується з іншими модулями додатку.

3.3 Висновки

У третьому розділі було виконано аналіз та обґрунтування вибору технологій і інструментів, що застосовувались для реалізації програмних модулів системи оптимізації розміщення посилок на складі. Зокрема, було обрано мову програмування Python як основну платформу для розробки логіки програми, бібліотеку PyQt6 — для створення зручного графічного інтерфейсу, а також SQLite — як вбудовану систему керування базами даних.

У розділі детально описано реалізацію модуля керування посилками та секторами, який забезпечує можливість додавання, перегляду, редагування та видалення записів у таблицях. Було реалізовано функціональність автоматичного оновлення заповненості секторів при кожній зміні. Користувацький інтерфейс реалізовано у вигляді діалогових вікон і таблиць із кнопками керування, що забезпечують інтерактивну взаємодію з даними.

Також у цьому розділі розглянуто структуру модуля додавання нової посилки: описано компонування інтерфейсу, логіку перевірки введених даних, механізм вибору сектору, а також методи отримання інформації з бази та її збереження. Фрагменти коду, наведені у підрозділах, демонструють взаємодію між користувачем, інтерфейсом та внутрішньою логікою системи.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування програмного забезпечення для системи оптимізації розміщення посилок на складі є критично важливим етапом, що дозволяє переконатися в надійності, точності та відповідності реалізованого функціоналу поставленим вимогам. Основною метою тестування є виявлення можливих помилок, перевірка логіки розміщення відправлень, коректності взаємодії з базою даних, а також зручності користування графічним інтерфейсом [21].

Під час підготовки до тестування система була встановлена на комп’ютери з операційною системою Windows 10 із встановленим Python 3.11, а також необхідними бібліотеками, зокрема PyQt6 для GUI та sqlite3 для обробки запитів до локальної бази даних. Було підготовлено тестову базу з секторами та змодельовано кілька сценаріїв введення користувачем різних типів посилок.

Тестування проводилось згідно з підготовленим планом та умовами, які охоплювали кілька основних етапів:

1. Функціональне тестування – охоплювало перевірку ключових операцій, таких як додавання нової посилки, вибір сектору (вручну або автоматично), збереження даних у базу, виведення таблиць посилок та секторів, фільтрація, очищення полів форми тощо. Окрему увагу приділено перевірці коректності розрахунку рекомендованого сектору згідно із заданими габаритами та вагою посилки [22].

2. Тестування відмов – перевіряло, як система поводиться при введенні некоректних даних (наприклад, відсутній номер відстеження, вага = 0, некоректна позиція на полиці тощо), при повному заповненні сектору, відсутності з'єднання з базою тощо [23]. Система має механізми інформування користувача про помилки через повідомлення типу QMessageBox.warning.

У ході тестування програмного забезпечення було здійснено моніторинг навантаження на ресурси комп’ютерної системи, зокрема – на центральний процесор, оперативну пам’ять та графічний процесор. Моніторинг проводився за

допомогою бібліотек psutil та GPUtil, що дозволило у реальному часі фіксувати зміну показників системних ресурсів під час активної роботи програми.

Згідно з отриманими результатами (рис. 4.1), середнє навантаження на CPU коливалося в межах 1.6%–30.8%, що свідчить про загальну невисоку обчислювальну складність виконуваних операцій. Споживання оперативної пам’яті варіювалося від 2.5 до 3.5 ГБ, залежно від кількості оброблюваних записів та стану інтерфейсу. Навантаження на GPU залишалося на рівні 0%–14%, що підтверджує відсутність інтенсивної графічної обробки, характерної для даного класу програм.

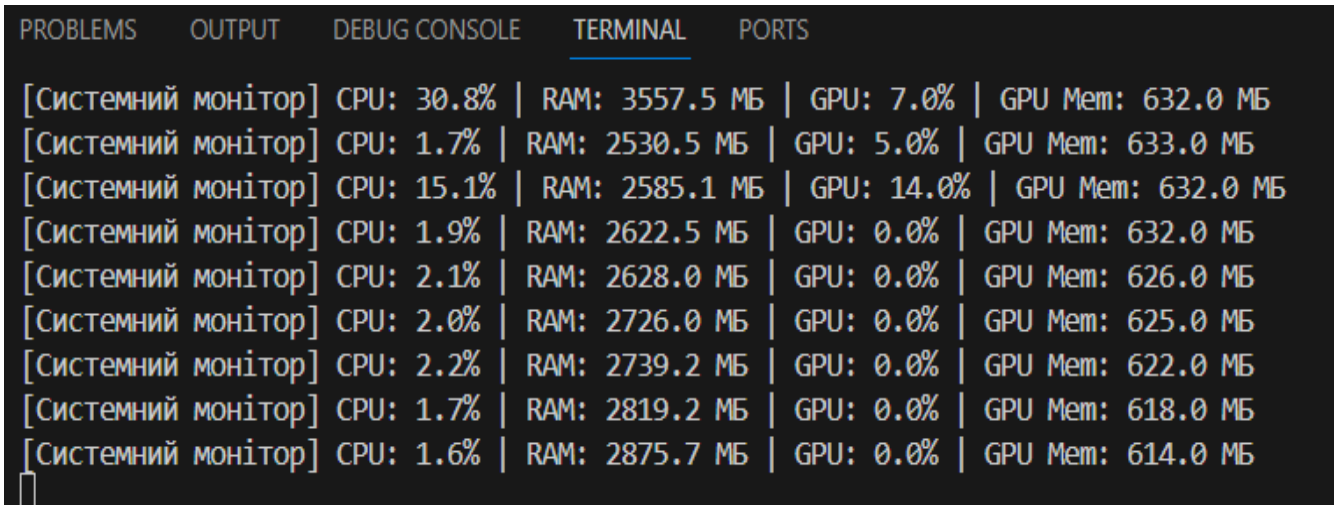

Рисунок 4.1 – Навантаження на ресурси комп’ютера

Отримані дані підтверджують, що система працює стабільно, не створює критичного навантаження на апаратне забезпечення та може бути ефективно використана в умовах обмежених ресурсів.

Функціональне тестування було почато з запуску програми. Після запуску застосунку з’являється вікно авторизації користувача (рис. 4.2).

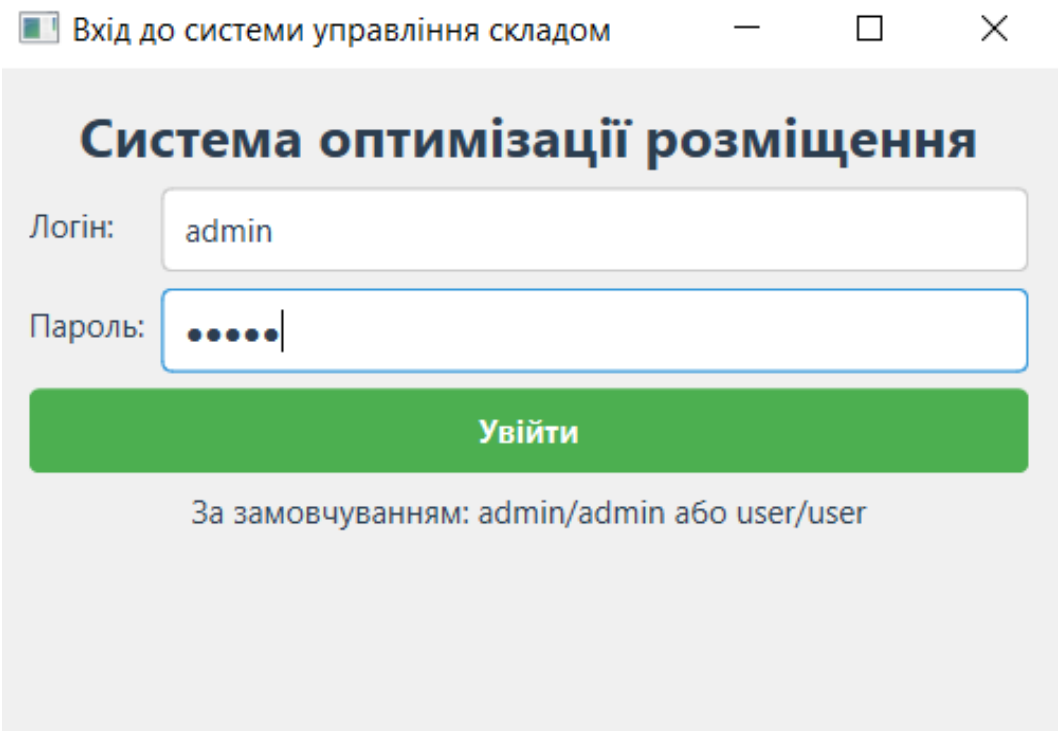

Рисунок 4.2 – Вікно «Вхід до системи управління складом»

Спочатку у поля «Логін» та «Пароль» було введено коректні дані та натиснуто кнопку «Увійти». Після цього вхід був виконаний успішно.

Для тестування відмов у ту ж саму форму було введено некоректні дані. Після натискання кнопки «Увійти» з’явилось системне вікно з помилкою «Невірний логін або пароль» (рис. 4.3).

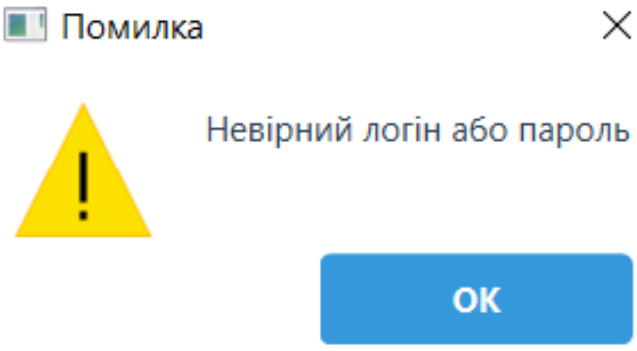

Рисунок 4.3 – Вікно «Помилка»

Після виконання успішного входу, вікно «Вхід до системи управління складом» зникло, а замість нього відкрилось головне вікно програми «Система оптимізації розміщення посилок» (рис. 4.4).

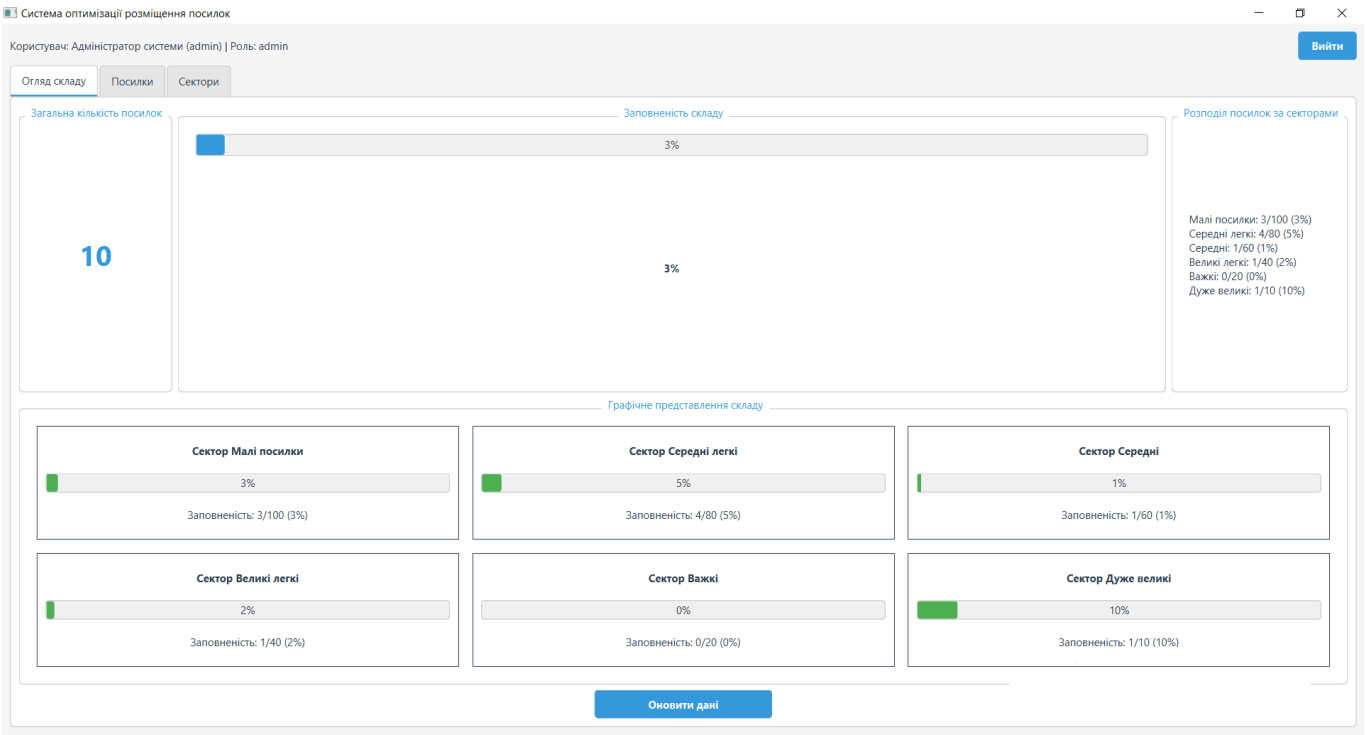

Рисунок 4.4 – Головне вікно «Система оптимізації розміщення посилок»

Головне вікно містить у собі три основні вкладки: «Огляд складу», «Посилки», «Сектори». За замовчування, при запуску програми, завжди відкривається вкладка «Огляд складу».

Вкладка «Огляд складу» поділена на секції:

1. «Загальна кількість посилок» – це секція, у якій зображена загальна кількість посилок, яка є на складі (рис. 4.5).

2. «Заповненість складу» – це секція, яка містить у собі загальний прогрес-бар, який відсотково показує, наскільки є заповненим склад.

Рисунок 4.5 – Секція «Загальна кількість посилок»

На рисунку 4.6 зображено секцію «Заповненість складу».

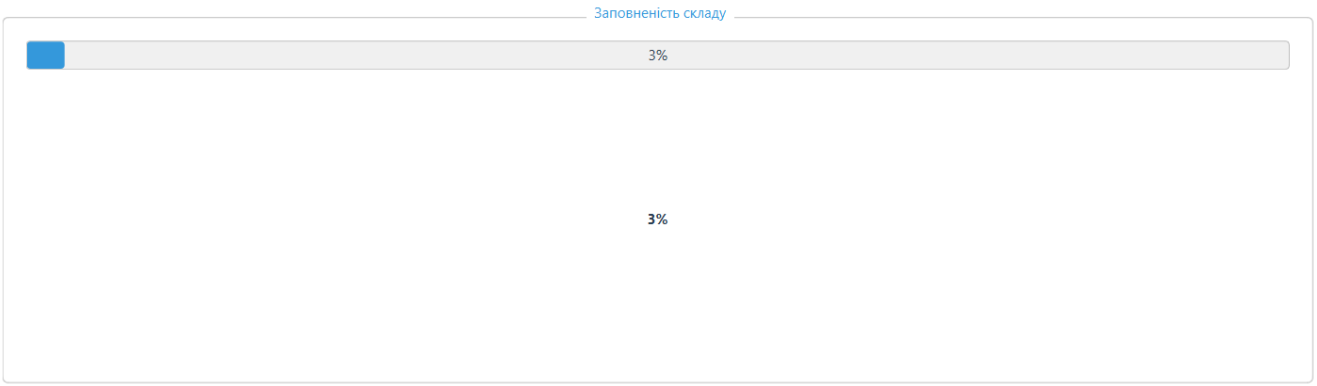

Рисунок 4.6 – Секція «Заповненість складу»

3. «Розподіл посилок за секторами» – це секція, що містить у собі інформацію про окрему числову та відсоткову кількість посилок кожної групи розмірів (рис. 4.7).

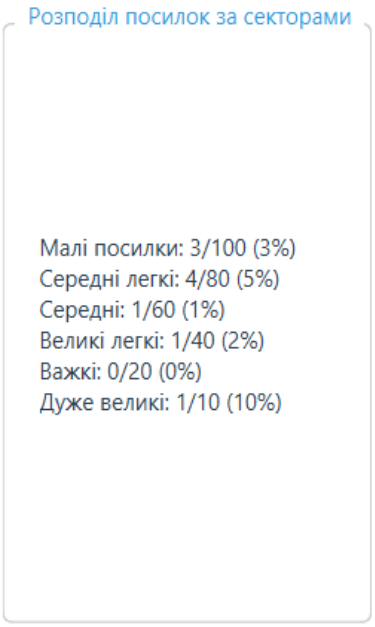

Рисунок 4.7 – Секція «Розподіл посилок за секторами»

4. «Графічне представлення складу» – це секція, яка складається з окремих боксів груп посилок. Кожен бокс містить у собі прогрес бар, який графічно демонструє заповненість певної категорії посилок (рис. 4.8). Також кожен бокс містить у собі цифрову інформацію про кількість посилок. Також секція містить кнопку «Оновити дані», яка оновлює статистику заповненості складу.

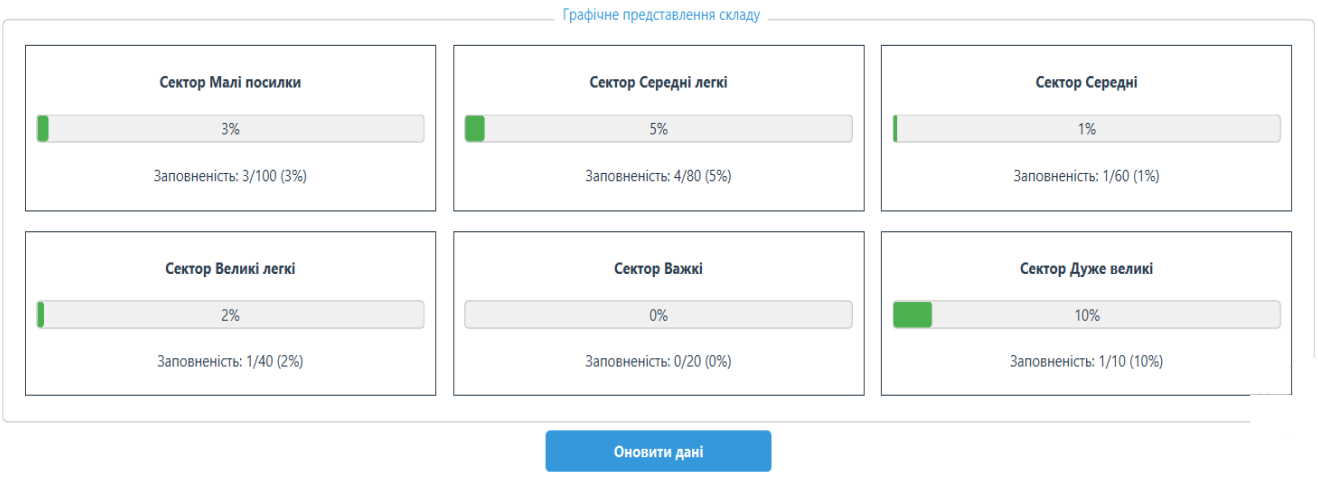

Рисунок 4.8 – Секція «Графічне представлення складу»

При збільшенні відсотка заповненості певної групи посилок, колір прогрес-бару міняється. Ледь заповненій групі відповідає зелений колів, середньо заповненій – помаранчевий та сильно заповненій – червоний (рис. 4.9).

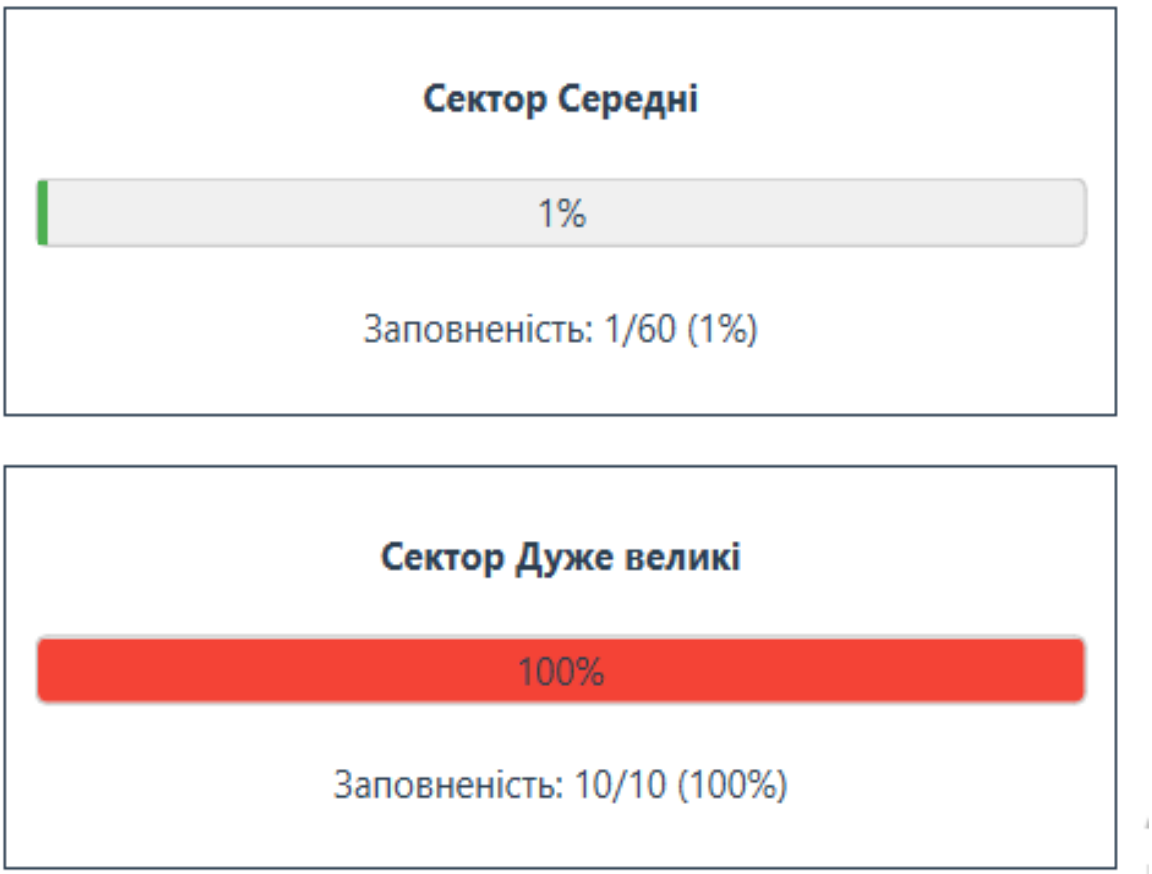

Рисунок 4.9 – Зміна кольорів прогрес-бару

Також головне вікно містить у собі кнопку «Вийти», при натисканні на яку вікно закривається, і з’являється вікно «Вхід до системи управління складом».

При натисканні на кнопку «Посилки» відкривається вкладка посилки, яка містить у собі таблицю посилок, що знаходяться на складі. Таблиця відображає інформацію про посилку, а також має функції перегляду та видалення посилок, за які відповідають однойменні кнопки (рис.4.10).

	ID	Номер відстеження	Відправник	Отримувач	Вага (кг)	Сектор	Статус	Дії
1	15	TR99475834	Артем Танань	Валерій Зінченко	5.0	Середні легкі	на складі	Перегляд Видалити
2	14	TR85947568	Ігор Мельник	Руслан Сорока	20.0	Дуже великі	на складі	Перегляд Видалити
3	13	TR75869485	Михайло Грушевський	Йосип Вареник	10.0	Великі легкі	на складі	Перегляд Видалити
4	11	TR16538695	Максим Шимко	Стецюк Анна	1.0	Малі посилки	на складі	Перегляд Видалити
5	10	TR68406912	Петро Моставчук	Станіслав Онуфрійчук	4.0	Середні легкі	на складі	Перегляд Видалити
6	9	TR58475293	Артем Танань	Андрій Грінін	1.0	Малі посилки	на складі	Перегляд Видалити
7	8	TR67584956	Дмитро Килимник	Андрій Шевченко	5.0	Середні легкі	на складі	Перегляд Видалити
8	6	TR16578457	Петро Щур	Рома Факти	1.0	Малі посилки	на складі	Перегляд Видалити
9	3	TR34567890	Мельник Анна	Шевченко Олександр	7.0	Середні	на складі	Перегляд Видалити
10	2	TR23456789	Сидоров Олег	Коваленко Марія	3.0	Середні легкі	на складі	Перегляд Видалити

Рисунок 4.10 – Таблиця посилок

При натисканні на кнопку «Перегляд», відкривається вікно «Інформація про посилку» з повною інформацією про посилку та кнопкою «Закрити», яка закриває це вікно (рис. 4.11).

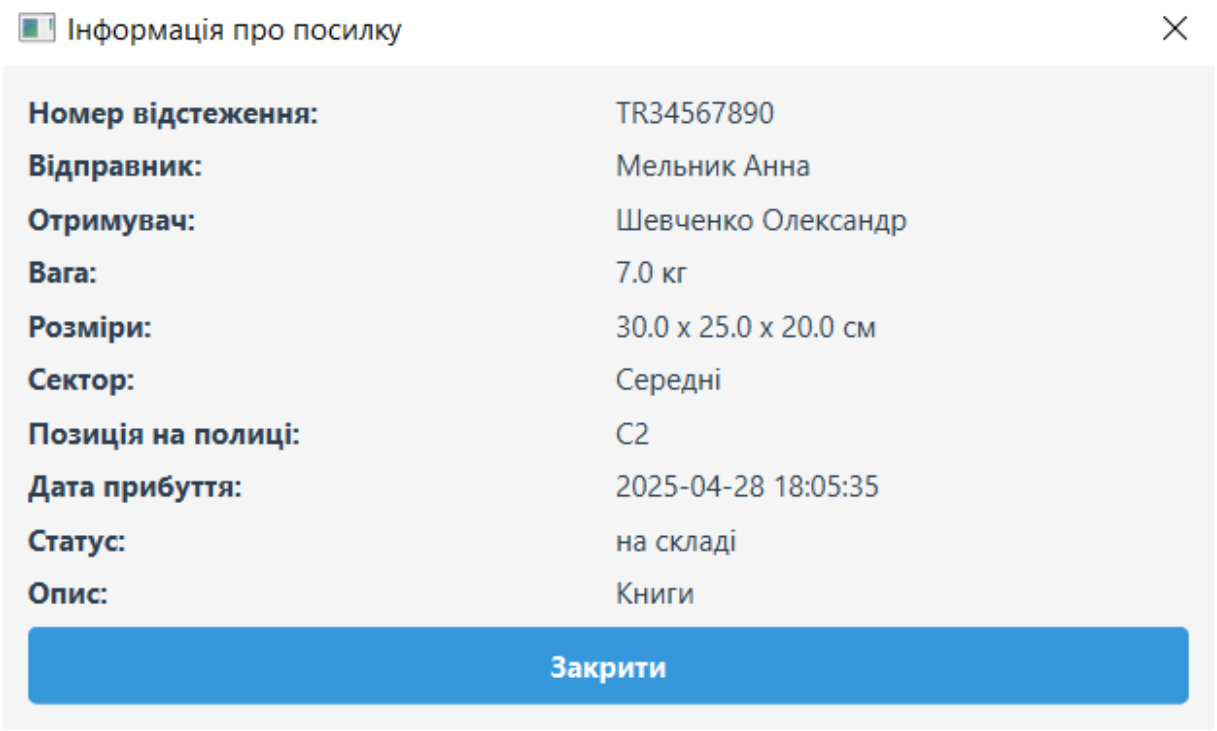

Рисунок 4.11 – Вікно «Інформація про посилку»

При натисканні на кнопку «Видалити», відкривається діалогове вікно «Підтвердження видалення», у якому потрібно підтвердити дію видалення посилки з бази даних. Функцію введено для запобігання випадкового видалення посилок з повною інформацією про посилку та кнопкою «Закрити», яка закриває це вікно (рис. 4.12).

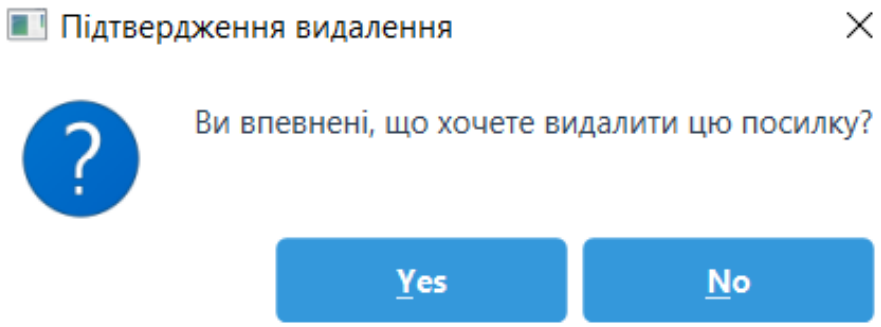

Рисунок 4.12 – Діалогове вікно «Підтвердження видалення»

Також вкладка містить у собі стрічку пошуку, що знаходиться над таблицею. Результат роботи пошуку програми зображено на рисунку 4.13.

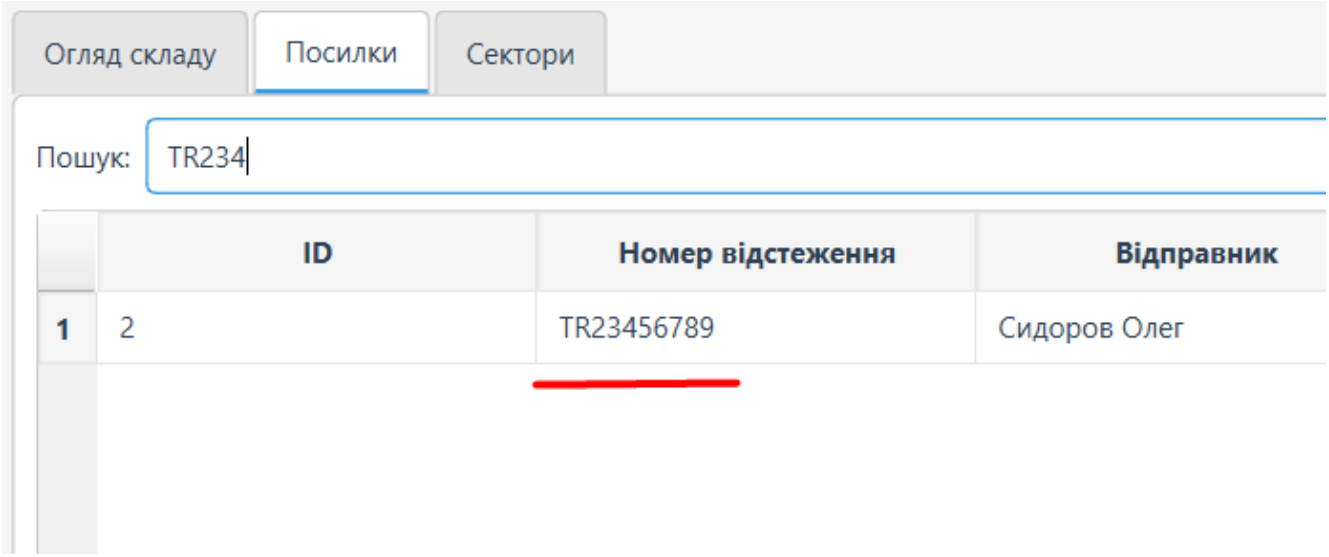

Рисунок 4.13 – Результат роботи пошуку

Вкладка також містить у собі функцію «Додати посилку». Після натиснення на однойменну кнопку, відкривається вікно, у якому, або додавання посилки пройшло успішно, потрібно заповнити усі обов’язкові поля. У іншому випадку при спробі зберегти посилку у базі даних, з’явиться вікно «Помилка», з повідомленням «Введіть номер відстеження» (рис. 4.14).

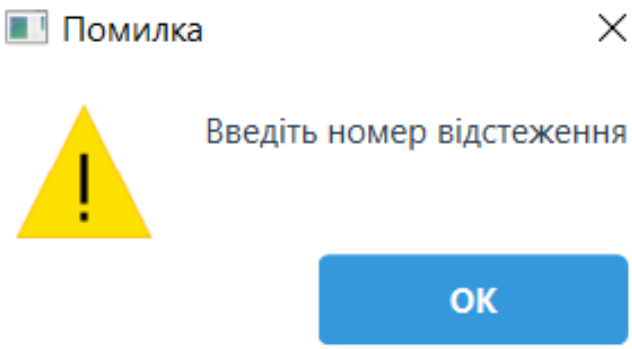

Рисунок 4.14 – Вікно «Помилка», Введіть номер відстеження

4.2 Розробка інструкції користувача

Інструкція користувача розроблена з метою забезпечення ефективної взаємодії з програмним забезпеченням, створеним для оптимізації розміщення посилок на складі. У ній описані вимоги до системи, порядок встановлення програми, запуску, а також поетапне використання її основного функціоналу.

Вимоги до апаратного забезпечення наведено в таблиці 4.1.

Таблиця 4.1 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i5	Intel Core i5
Оперативна пам'ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	2 ГБ	1 ГБ
Відеокарта	NVIDIA GeForce GTX 1050 Ti	NVIDIA GeForce GTX 1030
Операційна система	Windows 10	Windows 7

Після встановлення програми, користувач запускає виконуваний файл або відкриває головний скрипт у середовищі Python. Після запуску відкривається вікно авторизації, де необхідно ввести логін та пароль. У разі успішної автентифікації користувач потрапляє до головного вікна системи, що складається з декількох вкладок: «Огляд складу», «Посилки» та «Сектори».

Вкладка «Посилки». Щоб додати нову посилку, необхідно натиснути кнопку «Додати посилку». Відкриється діалогове вікно, в якому користувач заповнює дані:

- номер відстеження;
- ім’я відправника та отримувача;
- габарити та вага;
- опис (необов’язковий);
- позиція на полиці (наприклад, A1, B2);
- можливість ручного вибору сектору.

За бажанням, користувач може розрахувати рекомендований сектор, натиснувши відповідну кнопку. Система автоматично обере найбільш підходящий варіант на основі алгоритму, який враховує вагу, об’єм та заповненість секторів.

Після натискання кнопки «Зберегти» дані посилки передаються до бази, а інформація про завантаження сектору оновлюється автоматично.

Вкладка «Сектори». У цій вкладці відображається візуалізація поточного стану складу: сектори з підписами, заповненість у вигляді кольорових прогрес-барів, а також кількість розміщених посилок у кожному з них. Це дозволяє користувачу швидко оцінити навантаження та визначити переповнені зони.

Ця вкладка дає змогу переглядати:

- усі доступні сектори;
- граничні значення ваги та об’єму, які приймає сектор;
- поточну заповненість;
- опис сектору.

Система має вбудовану перевірку введених даних. У разі, якщо користувач

не заповнив обов’язкові поля або обрав переповнений сектор, виводяться інформативні повідомлення (наприклад: "Сектор повністю заповнений")

Після завершення роботи користувач може вийти з системи за допомогою відповідної кнопки, що закриває поточну сесію.

4.3 Висновки

У четвертому розділі було проведено всебічне тестування програмних модулів системи оптимізації розміщення посилок на складі. Основну увагу було приділено перевірці функціональності основних сценаріїв взаємодії користувача із системою, включаючи додавання нових посилок, автоматичний розрахунок сектору, а також реакцію програми на помилки введення.

Окрім того, було створено інструкцію користувача, яка містить детальні рекомендації щодо запуску, використання функціоналу програми та мінімальних технічних вимог до комп’ютера. Це забезпечує простоту впровадження системи навіть для користувачів без спеціальної підготовки.

ВИСНОВКИ

У бакалаврській кваліфікаційній роботі було реалізовано повноцінну систему для автоматизованого управління складським простором з використанням алгоритмів розміщення посилок. Робота охопила усі етапи життєвого циклу розробки програмного забезпечення — від аналізу предметної області до тестування і підготовки супровідної документації. Реалізація бакалаврської кваліфікаційної роботи відповідає методичним вказівкам [20, 21].

На основі проведеного аналізу існуючих рішень і методів було сформульовано технічне завдання, яке включало розробку програмних модулів з можливістю динамічного керування посилками та секторами складу. В результаті реалізації програмного продукту були досягнуті наступні цілі:

- розроблено програмні модулі для автоматизованого управління складом;
- забезпечено функціональність додавання, перегляду, пошуку та видалення посилок;
- реалізовано автоматичний розрахунок рекомендованого сектора для розміщення нової посилки на основі її ваги та об'єму;
- забезпечено візуалізацію заповненості складу з використанням графічних компонентів;
- реалізовано динамічне оновлення даних про стан складу;
- розроблено зручний, сучасний та адаптивний графічний інтерфейс користувача;
- здійснено тестування функціоналу системи відповідно до поставлених вимог.

Також у бакалаврській роботі було розроблено UML-діаграму діяльності, яка відображає логіку роботи системи, та реалізовано окремі модулі, зокрема модуль для взаємодії з базою даних SQLite. Було створено інструкцію користувача з описом функціоналу, вимог до апаратного забезпечення та основних сценаріїв використання програми.

Розроблений програмний продукт відповідає поставленим технічним

вимогам і може бути використаний як у навчальних цілях, так і як основа для подальшої розробки масштабованих систем управління складськими приміщеннями на основі логістичних алгоритмів оптимізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Сокур І. М., Сокур Л. М., Герасимчук В. В. Транспортна логістика. – Київ: Центр навчальної літератури, 2019. – 222 с.

2. Кислий В. М., Біловодська О. А., Олефіренко О. М., Соляник О. М. Логістика: теорія та практика. – Київ: Центр учбової літератури, 2019. – 360 с.

3. Марчук В. Є., Григорак М. Ю., Гармаш О. М. Складська логістика – Вінниця : ТОВ "ОЛДІ-ПЛЮС", 2020. – 256 с.

4. Modern Trends in the Development of Economy, Technology and Industry: Collection of Scientific Papers "International Scientific Unity" with Proceedings of the 3 rd International Scientific and Practical Conference. April 9-11, 2025. Toronto, Canada. 325 p.

5. Окландр М. А. Логістика : підручник. – Київ : Центр учбової літератури, 2021. – 345 с.

6. Інформаційні системи та технології в логістиці [Електронний ресурс] – Режим доступу до ресурсу: https://osvita.ua/vnz/reports/logika/25322/ (дата звернення: 03.04.2025)

7. Honeywell Intelligrated – What Is a Warehouse Management System (WMS)? [Електронний ресурс] – Режим доступу до ресурсу: https://www.intelligrated.com/en/resources/what-is-wms (дата звернення: 07.04.2025)

8. Infor – Why Modern WMS Systems Are Critical for Supply Chain Efficiency [Електронний ресурс] – Режим доступу до ресурсу: https://www.infor.com/resources/modern-wms-critical-for-supply-chain (дата звернення 09.04.2025)

9. SAP EWM [Електронний ресурс] – Режим доступу до ресурсу: https://www.sap.com/central-asia-caucasus/products/scm/extended-warehouse-management.html (дата звернення: 15.04.2025)

10. IT-Enterprise WMS [Електронний ресурс] – Режим доступу до ресурсу: https://www.it.ua/products/zakupki-i-logistika/upravlenie-skladom-materialov-wms (дата звернення: 17.04.2025)

11. Odoo Inventory [Електронний ресурс] – Режим доступу до ресурсу: https://www.odoo.com/uk_UA/app/inventory (дата звернення: 19.04.2025)

12. Посилкіна О. В., Деренська Я. М. Оптимізація логістичних рішень та управління логістичними ризиками. – Харків: НФаУ, 2018. – 61 с.

13. Чупріна М. О. Інформаційні технології в логістиці. – Київ: КПІ ім. Ігоря Сікорського, 2025. – 214 с.

14. Криворучко О. М., Кривенко Л. Ф. Основні напрями удосконалення логістичних процесів: реінжиніринг, оптимізація, автоматизація, 2021. – С. 45–50.

15. Головатенко І. А. Методи та засоби управління автономними логістичними кіберфізичними системами з використанням технологій штучного інтелекту. – Київ: КПІ ім. Ігоря Сікорського, 2024. – 148 с.

16. Кривещенко В. В., Москаленко І. А. Організація складських операцій. Сучасні технології комерційної діяльності і логістики. – Київ : КНЕУ, 2023. – С. 117–120.

17. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/ (дата звернення: 25.04.2025)

18. SQLite [Електронний ресурс] – Режим доступу до ресурсу: https://sqlite.org/ (дата звернення: 28.04.2025)

19. Mathis, L. Designed for Use: Create Usable Interfaces for Applications and the Web. – 2nd ed. – Raleigh: Pragmatic Bookshelf, 2016. – 338 p.

20. Fowler, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. – 3rd ed. – Boston: Addison-Wesley, 2018. – 208 p.

21. Тестування ПЗ [Електронний ресурс] – Режим доступу до ресурсу: https://university.sigma.software/what-is-software-testing/ (дата звернення: 30.04.2025)

22. Функціональне тестування інформаційних систем [Електронний ресурс] – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/vidy-funkcionalnogo-i-nefunkcionalnogo-testirovanija/ (дата звернення: 02.05.2025)

23. Тестування відмов у програмних системах [Електронний ресурс] – Режим доступу до ресурсу: https://qalearning.com.ua/theory/lectures/material/nonfunctional-testing/ (дата звернення: 12.05.2025)

24. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 51 с.

25. Методичні вказівки до виконання курсових проєктів з дисципліни "Проєктний практикум" для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. В.В. Войтко, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 44 с.

ДОДАТКИ

57

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"25" 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Алгоритмічні та програмні засоби оптимізації розміщення посилок на складі в інтелектуальних логістичних системах» за спеціальністю
121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. каф. ПЗ Р.Ю. Чехместрук
"24" 03 2025 р.

Виконав:
студент гр.2ПІ-21Б А.В. Танань
"24" 03 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Алгоритмічні та програмні засоби оптимізації розміщення посилок на складі в інтелектуальних логістичних системах».

Галузь застосування – складські логістичні системи.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від 20 березня 2025 року ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є підвищення ефективності управління складським простором шляхом розробки інтелектуального програмного забезпечення, що забезпечує оптимальне розміщення посилок на основі фізичних параметрів вантажів та поточної завантаженості секторів складу.

Призначення роботи – Алгоритмічні та програмні засоби оптимізації розміщення посилок на складі

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Кислий В. М., Біловодська О. А., Олефіренко О. М., Соляник О. М. Логістика: теорія та практика. – Київ: Центр учбової літератури, 2019. – 360 с.

2. Марчук В. Є., Григорак М. Ю., Гармаш О. М. Складська логістика – Вінниця : ТОВ "ОЛДІ-ПЛЮС", 2020. – 256 с.

3. Посилкіна О. В., Деренська Я. М. Оптимізація логістичних рішень та управління логістичними ризиками. – Харків: НФаУ, 2018. – 61 с.

4. Чупріна М. О. Інформаційні технології в логістиці. – Київ: КПІ ім. Ігоря Сікорського, 2025. – 214 с.

5. Технічні вимоги

Характеристики посилок – унікальний ідентифікатор, вага, габарити, опис; дані про сектори складу – максимальна допустима вага, об’єм, поточна завантаженість; вхідні дії користувача – додавання, пошук, видалення, перегляд посилок; алгоритмічні параметри – межі допустимих характеристик, критерії оптимальності, рівень заповненості; облікові дані користувачів – логін, пароль; вихідні дані – візуальне представлення заповненості складу, таблиці посилок, рекомендовані сектори.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до БКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз та постановка задачі бакалаврської кваліфікаційної роботи	30.03.2025 – 05.04.2025
2	Аналіз розвитку систем складської логістики	06.04.2025 – 08.04.2025
3	Розробка інтерфейсу програмного додатку	09.04.2025 – 15.04.2025
4	Розробка моделі та алгоритмів роботи системи	16.04.2025 – 21.04.2025
5	Реалізація модуля керування посилками	22.04.2025 – 30.04.2025
6	Реалізація алгоритму визначення оптимального сектора	01.05.2025 – 08.05.2025
7	Тестування системи	09.05.2025 – 13.05.2025
8	Створення пояснювальної записки	14.05.2025 – 18.05.2025

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

61

Додаток Б. Протокол перевірки БКР на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Алгоритмічні та програмні засоби оптимізації розміщення посилок на складі в інтелектуальних логістичних системах

Тип роботи: бакалаврська кваліфікаційна робота

(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ, група 2ПІ-21б

(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 8/3 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) к.т.н., доц. каф. ПЗ Чехместрук Р.Ю.

(прізвище, ініціали, посада)

Здобувач (підпис) Танань А.В.

(прізвище, ініціали)

Додаток В. Лістинг програми

```
import sys
import os
import sqlite3
import hashlib
import time

from datetime import datetime
from PyQt6.QtWidgets import (QApplication, QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
                             QLabel, QLineEdit, QPushButton, QTabWidget, QTableWidget,
                             QTableWidgetItem, QMessageBox, QComboBox, QSpinBox, QDoubleSpinBox,
                             QGridLayout, QGroupBox, QScrollArea, QSplitter, QFrame, QProgressBar,
                             QDialog, QFormLayout, QCheckBox, QHeaderView)
from PyQt6.QtCore import Qt, QSize, pyqtSignal, QTimer
from PyQt6.QtGui import QIcon, QFont, QColor
class LoginWindow(QWidget):
    login_successful = pyqtSignal(dict)

    def __init__(self):
        super().__init__()
        self.setWindowTitle('Вхід до системи управління складом')
        self.setMinimumSize(400, 250)

        main_layout = QVBoxLayout()

        title_label = QLabel('Система оптимізації розміщення')
        title_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        title_font = QFont()
        title_font.setPointSize(16)
        title_font.setBold(True)
        title_label.setFont(title_font)
        main_layout.addWidget(title_label)

        form_layout = QFormLayout()
        self.username_input = QLineEdit()
        self.username_input.setPlaceholderText('Введіть логін')
        form_layout.addRow('Логін:', self.username_input)
        self.password_input = QLineEdit()
        self.password_input.setPlaceholderText('Введіть пароль')
        self.password_input.setEchoMode(QLineEdit.EchoMode.Password)
        form_layout.addRow('Пароль:', self.password_input)

        main_layout.addLayout(form_layout)

        login_button = QPushButton('Увійти')
        login_button.clicked.connect(self.attempt_login)
        login_button.setStyleSheet("""
            QPushButton {
                background-color: #4CAF50;
                color: white;
                border: none;
                padding: 8px 16px;
                border-radius: 4px;
```

```
            font-weight: bold;
        }
        QPushButton:hover {
            background-color: #45a049;
        }
    """)
    main_layout.addWidget(login_button)

    info_label = QLabel('За замовчуванням: admin/admin або user/user')
    info_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
    main_layout.addWidget(info_label)

    main_layout.addStretch()
    self.setLayout(main_layout)

    self.username_input.returnPressed.connect(self.password_input.setFocus)
    self.password_input.returnPressed.connect(login_button.click)

  def attempt_login(self):
    username = self.username_input.text().strip()
    password = self.password_input.text()

    if not username or not password:
        QMessageBox.warning(self, "Помилка", "Введіть логін та пароль")
        return
    try:
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()

        password_hash = hashlib.sha256(password.encode()).hexdigest()
        cursor.execute("SELECT id, username, full_name, role FROM users WHERE username = ? AND
password_hash = ?",
                       (username, password_hash))
        user_data = cursor.fetchone()

        if user_data:
            cursor.execute("UPDATE users SET last_login = ? WHERE id = ?",
                           (datetime.now().strftime('%Y-%m-%d %H:%M:%S'), user_data[0]))

            cursor.execute('''
            INSERT INTO activity_logs (user_id, action, details, timestamp)
            VALUES (?, ?, ?, ?)
            ''', (user_data[0], 'login', f'Успішний вхід користувача {username}',
                  datetime.now().strftime('%Y-%m-%d %H:%M:%S')))
            conn.commit()
            user_info = {
                'id': user_data[0],
                'username': user_data[1],
                'full_name': user_data[2],
                'role': user_data[3]
            }
            self.username_input.clear()
            self.password_input.clear()

            self.login_successful.emit(user_info)
            self.hide()
```

```
            else:
                QMessageBox.warning(self, "Помилка", "Невірний логін або пароль")
        except sqlite3.Error as e:
            QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
        finally:
            conn.close()

class AddPackageDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Додати нову посилку")
        self.setMinimumWidth(500)
        self.sectors = self.get_sectors()
        layout = QVBoxLayout()
        form_layout = QFormLayout()
        self.tracking_input = QLineEdit()
        self.tracking_input.setPlaceholderText("Наприклад: TR12345678")
        form_layout.addRow("Номер відстеження:", self.tracking_input)
        self.sender_input = QLineEdit()
        form_layout.addRow("Відправник:", self.sender_input)
        self.recipient_input = QLineEdit()
        form_layout.addRow("Отримувач:", self.recipient_input)
        self.weight_input = QDoubleSpinBox()
        self.weight_input.setRange(0.1, 100.0)
        self.weight_input.setSingleStep(0.1)
        self.weight_input.setValue(1.0)
        self.weight_input.setSuffix(" кг")
        form_layout.addRow("Вага:", self.weight_input)
        dimensions_layout = QHBoxLayout()
        self.length_input = QDoubleSpinBox()
        self.length_input.setRange(1, 300)
        self.length_input.setSingleStep(1)
        self.length_input.setValue(20)
        self.length_input.setSuffix(" см")
        dimensions_layout.addWidget(QLabel("Довжина:"))
        dimensions_layout.addWidget(self.length_input)
        self.width_input = QDoubleSpinBox()
        self.width_input.setRange(1, 300)
        self.width_input.setSingleStep(1)
        self.width_input.setValue(15)
        self.width_input.setSuffix(" см")
        dimensions_layout.addWidget(QLabel("Ширина:"))
        dimensions_layout.addWidget(self.width_input)
        self.height_input = QDoubleSpinBox()
        self.height_input.setRange(1, 300)
        self.height_input.setSingleStep(1)
        self.height_input.setValue(10)
        self.height_input.setSuffix(" см")
        dimensions_layout.addWidget(QLabel("Висота:"))
        dimensions_layout.addWidget(self.height_input)

        form_layout.addRow("Розміри:", dimensions_layout)

        self.manual_sector_checkbox = QCheckBox("Вказати сектор вручну")
        self.manual_sector_checkbox.stateChanged.connect(self.toggle_manual_sector)
        form_layout.addRow("", self.manual_sector_checkbox)
```

```
        self.sector_combo = QComboBox()
        for sector in self.sectors:
            self.sector_combo.addItem(f"{sector['name']} ({sector['current_fill']}/{sector['capacity']})",
sector['id'])
        self.sector_combo.setEnabled(False)
        form_layout.addRow("Сектор:", self.sector_combo)

        self.shelf_input = QLineEdit()
        self.shelf_input.setPlaceholderText("Наприклад: A1, B5, C2")
        form_layout.addRow("Позиція на полиці:", self.shelf_input)

        self.description_input = QLineEdit()
        self.description_input.setPlaceholderText("Додаткова інформація про посилку")
        form_layout.addRow("Опис:", self.description_input)

        layout.addLayout(form_layout)

        info_box = QGroupBox("Рекомендований сектор")
        info_layout = QVBoxLayout()
        self.recommended_sector_label = QLabel("Натисніть 'Розрахувати рекомендований сектор'")
        info_layout.addWidget(self.recommended_sector_label)

        calculate_btn = QPushButton("Розрахувати рекомендований сектор")
        calculate_btn.clicked.connect(self.calculate_recommended_sector)
        info_layout.addWidget(calculate_btn)

        info_box.setLayout(info_layout)
        layout.addWidget(info_box)

        button_layout = QHBoxLayout()
        save_btn = QPushButton("Зберегти")
        save_btn.clicked.connect(self.accept)
        cancel_btn = QPushButton("Скасувати")
        cancel_btn.clicked.connect(self.reject)

        button_layout.addWidget(save_btn)
        button_layout.addWidget(cancel_btn)
        layout.addLayout(button_layout)

        self.setLayout(layout)

    def toggle_manual_sector(self, state):
        self.sector_combo.setEnabled(state == Qt.CheckState.Checked)

    def get_sectors(self):
        sectors = []
        try:
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            cursor.execute("""
                SELECT id, name, min_weight, max_weight, min_volume, max_volume,
                    capacity, current_fill, description
                FROM sectors
                ORDER BY id
            """)
```

```
            for row in cursor.fetchall():
                sectors.append({
                    'id': row[0],
                    'name': row[1],
                    'min_weight': row[2],
                    'max_weight': row[3],
                    'min_volume': row[4],
                    'max_volume': row[5],
                    'capacity': row[6],
                    'current_fill': row[7],
                    'description': row[8]
                })

        except sqlite3.Error as e:
            QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
        finally:
            conn.close()

        return sectors

    def calculate_recommended_sector(self):
        weight = self.weight_input.value()
        length = self.length_input.value() / 100
        width = self.width_input.value() / 100
        height = self.height_input.value() / 100

        volume = length * width * height

        best_sector = None
        best_match_score = float('-inf')

        for sector in self.sectors:
            if sector['current_fill'] >= sector['capacity']:
                continue

            weight_match = (sector['min_weight'] <= weight <= sector['max_weight'])
            volume_match = (sector['min_volume'] <= volume <= sector['max_volume'])

            if weight_match and volume_match:
                fill_factor = 1 - (sector['current_fill'] / sector['capacity'])

                weight_center = (sector['min_weight'] + sector['max_weight']) / 2
                volume_center = (sector['min_volume'] + sector['max_volume']) / 2

                weight_fit = 1 - abs(weight - weight_center) / (sector['max_weight'] - sector['min_weight'] +
0.001)
                volume_fit = 1 - abs(volume - volume_center) / (sector['max_volume'] - sector['min_volume'] +
0.001)

                match_score = (weight_fit * 0.4) + (volume_fit * 0.4) + (fill_factor * 0.2)

                if match_score > best_match_score:
                    best_match_score = match_score
                    best_sector = sector
```

```
        if best_sector is None:
            best_distance = float('inf')

            for sector in self.sectors:
                if sector['current_fill'] >= sector['capacity']:
                    continue

                weight_distance = 0
                if weight < sector['min_weight']:
                    weight_distance = sector['min_weight'] - weight
                elif weight > sector['max_weight']:
                    weight_distance = weight - sector['max_weight']

                volume_distance = 0
                if volume < sector['min_volume']:
                    volume_distance = sector['min_volume'] - volume
                elif volume > sector['max_volume']:
                    volume_distance = volume - sector['max_volume']

                total_distance = (weight_distance / max(1, sector['max_weight'])) + (volume_distance / max(0.1,
sector['max_volume']))

                if total_distance < best_distance:
                    best_distance = total_distance
                    best_sector = sector

        if best_sector:
            index = self.sector_combo.findData(best_sector['id'])
            if index >= 0:
                self.sector_combo.setCurrentIndex(index)

            self.recommended_sector_label.setText(
                f"<b>Рекомендований сектор:</b> {best_sector['name']}<br>"
                f"<b>Заповненість:</b> {best_sector['current_fill']}/{best_sector['capacity']}<br>"
                f"<b>Опис:</b> {best_sector['description']}"
            )
        else:
            self.recommended_sector_label.setText("Не вдалося знайти відповідний сектор. Перевірте
параметри.")

    def accept(self):
        if not self.tracking_input.text().strip():
            QMessageBox.warning(self, "Помилка", "Введіть номер відстеження")
            return
        if not self.sender_input.text().strip():
            QMessageBox.warning(self, "Помилка", "Введіть відправника")
            return
        if not self.recipient_input.text().strip():
            QMessageBox.warning(self, "Помилка", "Введіть отримувача")
            return
        if not self.manual_sector_checkbox.isChecked():
            self.calculate_recommended_sector()
        sector_id = self.sector_combo.currentData()
        for sector in self.sectors:
            if sector['id'] == sector_id and sector['current_fill'] >= sector['capacity']:
                QMessageBox.warning(self, "Помилка", f"Сектор '{sector['name']}' повністю заповнений!
```

```
Виберіть інший сектор.")
                return
        super().accept()

    def get_package_data(self):
        return {
            'tracking_number': self.tracking_input.text().strip(),
            'sender_name': self.sender_input.text().strip(),
            'recipient_name': self.recipient_input.text().strip(),
            'weight': self.weight_input.value(),
            'length': self.length_input.value(),
            'width': self.width_input.value(),
            'height': self.height_input.value(),
            'sector_id': self.sector_combo.currentData(),
            'shelf_position': self.shelf_input.text().strip(),
            'description': self.description_input.text().strip()
        }

class PackageInfoDialog(QDialog):
    def __init__(self, package_id, parent=None):
        super().__init__(parent)
        self.setWindowTitle("Інформація про посилку")
        self.setMinimumWidth(500)
        self.package_id = package_id
        layout = QVBoxLayout()
        self.package_data = self.load_package_data()
        if not self.package_data:
            layout.addWidget(QLabel("Посилку не знайдено"))
            self.setLayout(layout)
            return
        info_layout = QGridLayout()
        row = 0
        for label, value in [
            ("Номер відстеження:", self.package_data.get('tracking_number', '')),
            ("Відправник:", self.package_data.get('sender_name', '')),
            ("Отримувач:", self.package_data.get('recipient_name', '')),
            ("Вага:", f"{self.package_data.get('weight', 0)} кг"),
            ("Розміри:", f"{self.package_data.get('length', 0)} x {self.package_data.get('width', 0)} x
{self.package_data.get('height', 0)} см"),
            ("Сектор:", self.package_data.get('sector_name', '')),
            ("Позиція на полиці:", self.package_data.get('shelf_position', '')),
            ("Дата прибуття:", self.package_data.get('arrival_date', '')),
            ("Статус:", self.package_data.get('status', '')),
            ("Опис:", self.package_data.get('description', ''))
        ]:
            label_widget = QLabel(f"<b>{label}</b>")
            value_widget = QLabel(str(value))

            info_layout.addWidget(label_widget, row, 0)
            info_layout.addWidget(value_widget, row, 1)
            row += 1
        layout.addLayout(info_layout)
        close_button = QPushButton("Закрити")
        close_button.clicked.connect(self.accept)
        layout.addWidget(close_button)
        self.setLayout(layout)
```

```
    def load_package_data(self):
        try:
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            cursor.execute("""
                SELECT p.id, p.tracking_number, p.sender_name, p.recipient_name,
                       p.weight, p.length, p.width, p.height,
                       p.sector_id, s.name as sector_name, p.shelf_position,
                       p.arrival_date, p.status, p.description
                FROM packages p
                LEFT JOIN sectors s ON p.sector_id = s.id
                WHERE p.id = ?
            """, (self.package_id,))
                    row = cursor.fetchone()
            if row:
                return {
                    'id': row[0],
                    'tracking_number': row[1],
                    'sender_name': row[2],
                    'recipient_name': row[3],
                    'weight': row[4],
                    'length': row[5],
                    'width': row[6],
                    'height': row[7],
                    'sector_id': row[8],
                    'sector_name': row[9],
                    'shelf_position': row[10],
                    'arrival_date': row[11],
                    'status': row[12],
                    'description': row[13]
                }
            return None
        except sqlite3.Error as e:
            QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
            return None
        finally:
            conn.close()

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('Система оптимізації розміщення посилок')
        self.setMinimumSize(1200, 800)
        self.current_user = None
        central_widget = QWidget()
        main_layout = QVBoxLayout(central_widget)
        self.user_info_layout = QHBoxLayout()
        self.user_label = QLabel("Користувач не авторизований")
        self.logout_button = QPushButton("Вийти")
        self.logout_button.clicked.connect(self.logout)
        self.logout_button.setVisible(False)
        self.user_info_layout.addWidget(self.user_label)
        self.user_info_layout.addStretch()
        self.user_info_layout.addWidget(self.logout_button)
        main_layout.addLayout(self.user_info_layout)
```

```
        self.tabs = QTabWidget()
        self.warehouse_tab = QWidget()
        self.setup_warehouse_tab()
        self.tabs.addTab(self.warehouse_tab, "Огляд складу")
        self.packages_tab = QWidget()
        self.setup_packages_tab()
        self.tabs.addTab(self.packages_tab, "Посилки")
        self.sectors_tab = QWidget()
        self.setup_sectors_tab()
        self.tabs.addTab(self.sectors_tab, "Сектори")

        main_layout.addWidget(self.tabs)
        self.setCentralWidget(central_widget)
        self.login_window = LoginWindow()
        self.login_window.login_successful.connect(self.on_login_successful)
        self.login_window.show()
        self.refresh_timer = QTimer()
        self.refresh_timer.timeout.connect(self.refresh_data)
        self.refresh_timer.start(60000)

    def on_login_successful(self, user_info):
        self.current_user = user_info
        self.user_label.setText(f"Користувач: {user_info['full_name']} ({user_info['username']}) | Роль:
{user_info['role']}")
        self.logout_button.setVisible(True)
        self.show()
        self.refresh_data()

    def logout(self):
        self.hide()
        self.current_user = None
        self.user_label.setText("Користувач не авторизований")
        self.logout_button.setVisible(False)
        self.clear_tab_data()
        self.login_window.show()

    def clear_tab_data(self):
        self.packages_table.setRowCount(0)
        self.packages_search_input.clear()

    def refresh_data(self):
        if not self.current_user:
            return
        self.load_warehouse_data()
        self.load_packages_data()
        self.load_sectors_data()

    def setup_warehouse_tab(self):
        layout = QVBoxLayout(self.warehouse_tab)
        stats_layout = QHBoxLayout()
        self.total_packages_box = QGroupBox("Загальна кількість посилок")
        box_layout = QVBoxLayout()
        self.total_packages_label = QLabel("0")
        self.total_packages_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        font = QFont()
        font.setPointSize(24)
```

```
        font.setBold(True)
        self.total_packages_label.setFont(font)
        self.total_packages_label.setStyleSheet(f"color: {AppStyles.PRIMARY};")
        box_layout.addWidget(self.total_packages_label)
        self.total_packages_box.setLayout(box_layout)
        stats_layout.addWidget(self.total_packages_box)
        self.warehouse_capacity_box = QGroupBox("Заповненість складу")
        box_layout = QVBoxLayout()
        self.warehouse_capacity_progress = QProgressBar()
        self.warehouse_capacity_progress.setRange(0, 100)
        self.warehouse_capacity_progress.setValue(0)
        self.warehouse_capacity_progress.setFixedHeight(25)
        box_layout.addWidget(self.warehouse_capacity_progress)
        self.warehouse_capacity_label = QLabel("0%")
        self.warehouse_capacity_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
        font = QFont()
        font.setBold(True)
        self.warehouse_capacity_label.setFont(font)
        box_layout.addWidget(self.warehouse_capacity_label)
        self.warehouse_capacity_box.setLayout(box_layout)
        stats_layout.addWidget(self.warehouse_capacity_box)
        self.sectors_packages_box = QGroupBox("Розподіл посилок за секторами")
        box_layout = QVBoxLayout()
        self.sectors_packages_label = QLabel("Завантаження...")
        self.sectors_packages_label.setWordWrap(True)
        box_layout.addWidget(self.sectors_packages_label)
        self.sectors_packages_box.setLayout(box_layout)
        stats_layout.addWidget(self.sectors_packages_box)
        layout.addLayout(stats_layout)

        warehouse_view_box = QGroupBox("Графічне представлення складу")
        box_layout = QVBoxLayout()
        self.sectors_grid = QGridLayout()
        self.sectors_grid.setSpacing(15)
        box_layout.addLayout(self.sectors_grid)

        warehouse_view_box.setLayout(box_layout)
        layout.addWidget(warehouse_view_box)

        refresh_btn = QPushButton("Оновити дані")
        refresh_btn.setStyleSheet(AppStyles.get_button_style("primary"))
        refresh_btn.setFixedWidth(200)
        refresh_btn.setIcon(QIcon("refresh.png"))
        refresh_btn.clicked.connect(self.refresh_data)

        btn_layout = QHBoxLayout()
        btn_layout.addStretch()
        btn_layout.addWidget(refresh_btn)
        btn_layout.addStretch()

        layout.addLayout(btn_layout)

    def setup_packages_tab(self):
        layout = QVBoxLayout(self.packages_tab)
        search_layout = QHBoxLayout()
        search_layout.addWidget(QLabel("Пошук:"))
```

```
        self.packages_search_input = QLineEdit()
        self.packages_search_input.setPlaceholderText("Введіть номер відстеження")
        self.packages_search_input.textChanged.connect(self.filter_packages)
        search_layout.addWidget(self.packages_search_input)
        search_btn = QPushButton("Пошук")
        search_btn.clicked.connect(self.filter_packages)
        search_layout.addWidget(search_btn)
        add_package_btn = QPushButton("Додати посилку")
        add_package_btn.clicked.connect(self.add_new_package)
        search_layout.addWidget(add_package_btn)

        layout.addLayout(search_layout)

        self.packages_table = QTableWidget()
        self.packages_table.setColumnCount(8)
        self.packages_table.setHorizontalHeaderLabels([
            "ID", "Номер відстеження", "Відправник", "Отримувач",
            "Вага (кг)", "Сектор", "Статус", "Дії"
        ])
        self.packages_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)
        self.packages_table.setSelectionBehavior(QTableWidget.SelectionBehavior.SelectRows)
        layout.addWidget(self.packages_table)

        control_layout = QHBoxLayout()

        refresh_btn = QPushButton("Оновити")
        refresh_btn.clicked.connect(lambda: self.load_packages_data())
        control_layout.addWidget(refresh_btn)

        layout.addLayout(control_layout)

    def setup_sectors_tab(self):
        layout = QVBoxLayout(self.sectors_tab)

        self.sectors_table = QTableWidget()
        self.sectors_table.setColumnCount(6)
        self.sectors_table.setHorizontalHeaderLabels([
            "ID", "Назва", "Діапазон ваги (кг)", "Діапазон об'єму (м³)",
            "Заповненість", "Опис"
        ])
        self.sectors_table.horizontalHeader().setSectionResizeMode(QHeaderView.ResizeMode.Stretch)
        layout.addWidget(self.sectors_table)
        control_layout = QHBoxLayout()
        refresh_btn = QPushButton("Оновити")
        refresh_btn.clicked.connect(lambda: self.load_sectors_data())
        control_layout.addWidget(refresh_btn)

        layout.addLayout(control_layout)

    def load_warehouse_data(self):
        try:
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()

            cursor.execute("SELECT COUNT(*) FROM packages")
            total_packages = cursor.fetchone()[0]
```

```
self.total_packages_label.setText(str(total_packages))

cursor.execute("""
    SELECT id, name, capacity, current_fill, description
    FROM sectors
    ORDER BY id
""")
sectors_data = cursor.fetchall()
total_capacity = sum(sector[2] for sector in sectors_data)
total_fill = sum(sector[3] for sector in sectors_data)
if total_capacity > 0:
    fill_percentage = int((total_fill / total_capacity) * 100)
    self.warehouse_capacity_progress.setValue(fill_percentage)
    self.warehouse_capacity_label.setText(f"{fill_percentage}%")
else:
    self.warehouse_capacity_progress.setValue(0)
    self.warehouse_capacity_label.setText("0%")
while self.sectors_grid.count():
    item = self.sectors_grid.takeAt(0)
    if item.widget():
        item.widget().deleteLater()
          max_columns = 3
row, col = 0, 0
for sector_id, name, capacity, fill, description in sectors_data:
    sector_frame = QFrame()
    sector_frame.setFrameShape(QFrame.Shape.Box)
    sector_frame.setLineWidth(1)

    if capacity > 0:
        fill_percentage = (fill / capacity) * 100
    else:
        fill_percentage = 0
    if fill_percentage < 50:
        color = "#4CAF50"
    elif fill_percentage < 80:
        color = "#FF9800"
    else:
        color = "#F44336"

    sector_layout = QVBoxLayout(sector_frame)

    title_label = QLabel(f"<b>Сектор {name}</b>")
    title_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
    sector_layout.addWidget(title_label)

    progress = QProgressBar()
    progress.setRange(0, 100)
    progress.setValue(int(fill_percentage))
    progress.setStyleSheet(f"QProgressBar::chunk {{ background-color: {color}; }}")
    sector_layout.addWidget(progress)

    info_label = QLabel(f"Заповненість: {fill}/{capacity} ({int(fill_percentage)}%)")
    info_label.setAlignment(Qt.AlignmentFlag.AlignCenter)
    sector_layout.addWidget(info_label)

    self.sectors_grid.addWidget(sector_frame, row, col)
```

```
            col += 1
            if col >= max_columns:
                col = 0
                row += 1
        sectors_info = []
        for sector_id, name, capacity, fill, description in sectors_data:
            if capacity > 0:
                percentage = f"{int((fill / capacity) * 100)}%"
            else:
                percentage = "0%"
            sectors_info.append(f"{name}: {fill}/{capacity} ({percentage})")

        self.sectors_packages_label.setText("\n".join(sectors_info))

    except sqlite3.Error as e:
        QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
    finally:
        conn.close()

def load_packages_data(self):
    try:
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()

        cursor.execute("""
            SELECT p.id, p.tracking_number, p.sender_name, p.recipient_name,
                   p.weight, s.name as sector_name, p.status
            FROM packages p
            LEFT JOIN sectors s ON p.sector_id = s.id
            ORDER BY p.id DESC
        """)
        packages = cursor.fetchall()
        self.packages_table.setRowCount(len(packages))

        for row, (pkg_id, tracking, sender, recipient, weight, sector, status) in enumerate(packages):
            self.packages_table.setItem(row, 0, QTableWidgetItem(str(pkg_id)))
            self.packages_table.setItem(row, 1, QTableWidgetItem(tracking))
            self.packages_table.setItem(row, 2, QTableWidgetItem(sender))
            self.packages_table.setItem(row, 3, QTableWidgetItem(recipient))
            self.packages_table.setItem(row, 4, QTableWidgetItem(str(weight)))
            self.packages_table.setItem(row, 5, QTableWidgetItem(sector if sector else ""))
            self.packages_table.setItem(row, 6, QTableWidgetItem(status))

            actions_widget = QWidget()
            actions_layout = QHBoxLayout(actions_widget)
            actions_layout.setContentsMargins(0, 0, 0, 0)

            view_btn = QPushButton("Перегляд")
            view_btn.setStyleSheet(AppStyles.get_button_style("success"))
            pkg_id_copy = pkg_id
            view_btn.clicked.connect(lambda _, pid=pkg_id_copy: self.view_package(pid))
            actions_layout.addWidget(view_btn)
            delete_btn = QPushButton("Видалити")
            delete_btn.setStyleSheet(AppStyles.get_button_style("danger"))
            delete_btn.clicked.connect(lambda _, pid=pkg_id_copy: self.delete_package(pid))
```

```
            actions_layout.addWidget(delete_btn)

            self.packages_table.setCellWidget(row, 7, actions_widget)

        except sqlite3.Error as e:
            QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
        finally:
            conn.close()

    def load_sectors_data(self):
        try:
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            cursor.execute("""
                SELECT id, name, min_weight, max_weight, min_volume, max_volume,
                       capacity, current_fill, description
                FROM sectors
                ORDER BY id
            """)
            sectors = cursor.fetchall()
            self.sectors_table.setRowCount(len(sectors))
            for row, (id, name, min_weight, max_weight, min_volume, max_volume,
                      capacity, current_fill, description) in enumerate(sectors):

                self.sectors_table.setItem(row, 0, QTableWidgetItem(str(id)))
                self.sectors_table.setItem(row, 1, QTableWidgetItem(name))
                self.sectors_table.setItem(row, 2, QTableWidgetItem(f"{min_weight} - {max_weight}"))
                self.sectors_table.setItem(row, 3, QTableWidgetItem(f"{min_volume} - {max_volume}"))
                fill_widget = QWidget()
                fill_layout = QHBoxLayout(fill_widget)
                fill_layout.setContentsMargins(5, 2, 5, 2)
                progress = QProgressBar()
                progress.setRange(0, capacity)
                progress.setValue(current_fill)
                fill_percentage = (current_fill / capacity) * 100 if capacity > 0 else 0
                if fill_percentage < 50:
                    progress.setStyleSheet("QProgressBar::chunk { background-color: #4CAF50; }")
                elif fill_percentage < 80:
                    progress.setStyleSheet("QProgressBar::chunk { background-color: #FF9800; }")
                else:
                    progress.setStyleSheet("QProgressBar::chunk { background-color: #F44336; }")
                fill_layout.addWidget(progress)
                fill_layout.addWidget(QLabel(f"{current_fill}/{capacity}"))
                self.sectors_table.setCellWidget(row, 4, fill_widget)
                self.sectors_table.setItem(row, 5, QTableWidgetItem(description))
        except sqlite3.Error as e:
            QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
        finally:
            conn.close()

    def filter_packages(self):
        search_text = self.packages_search_input.text().strip().lower()
        if not search_text:
            self.load_packages_data()
            return
```

```
        try:
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            cursor.execute("""
                SELECT p.id, p.tracking_number, p.sender_name, p.recipient_name,
                       p.weight, s.name as sector_name, p.status
                FROM packages p
                LEFT JOIN sectors s ON p.sector_id = s.id
                WHERE LOWER(p.tracking_number) LIKE ? OR
                      LOWER(p.sender_name) LIKE ? OR
                      LOWER(p.recipient_name) LIKE ?
                ORDER BY p.id DESC
            """, (f'%{search_text}%', f'%{search_text}%', f'%{search_text}%'))
            packages = cursor.fetchall()
            self.packages_table.setRowCount(len(packages))
            for row, (pkg_id, tracking, sender, recipient, weight, sector, status) in enumerate(packages):
                self.packages_table.setItem(row, 0, QTableWidgetItem(str(pkg_id)))
                self.packages_table.setItem(row, 1, QTableWidgetItem(tracking))
                self.packages_table.setItem(row, 2, QTableWidgetItem(sender))
                self.packages_table.setItem(row, 3, QTableWidgetItem(recipient))
                self.packages_table.setItem(row, 4, QTableWidgetItem(str(weight)))
                self.packages_table.setItem(row, 5, QTableWidgetItem(sector if sector else ""))
                self.packages_table.setItem(row, 6, QTableWidgetItem(status))

                actions_widget = QWidget()
                actions_layout = QHBoxLayout(actions_widget)
                actions_layout.setContentsMargins(0, 0, 0, 0)
                view_btn = QPushButton("Перегляд")
                view_btn.setStyleSheet(AppStyles.get_button_style("success"))
                pkg_id_copy = pkg_id
                view_btn.clicked.connect(lambda _, pid=pkg_id_copy: self.view_package(pid))
                actions_layout.addWidget(view_btn)
                delete_btn = QPushButton("Видалити")
                delete_btn.setStyleSheet(AppStyles.get_button_style("danger"))
                delete_btn.clicked.connect(lambda _, pid=pkg_id_copy: self.delete_package(pid))
                actions_layout.addWidget(delete_btn)
                self.packages_table.setCellWidget(row, 7, actions_widget)
        except sqlite3.Error as e:
            QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
        finally:
            conn.close()

    def add_new_package(self):
        dialog = AddPackageDialog(self)
        if dialog.exec() == QDialog.DialogCode.Accepted:
            package_data = dialog.get_package_data()
            self.save_new_package(package_data)

    def save_new_package(self, package_data):
        try:
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()

            cursor.execute("SELECT id FROM packages WHERE tracking_number = ?",
(package_data['tracking_number'],))
            if cursor.fetchone():
```

```
            QMessageBox.warning(self, "Помилка", f"Посилка з номером відстеження
{package_data['tracking_number']} вже існує")
            return

        cursor.execute('''
            INSERT INTO packages (tracking_number, sender_name, recipient_name, weight, length, width,
height,
                                  sector_id, shelf_position, arrival_date, status, description)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ''', (
            package_data['tracking_number'],
            package_data['sender_name'],
            package_data['recipient_name'],
            package_data['weight'],
            package_data['length'],
            package_data['width'],
            package_data['height'],
            package_data['sector_id'],
            package_data['shelf_position'],
            datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            'на складі',
            package_data['description']
        ))

        cursor.execute("UPDATE sectors SET current_fill = current_fill + 1 WHERE id = ?",
(package_data['sector_id'],))
        cursor.execute('''
            INSERT INTO activity_logs (user_id, action, details, timestamp)
            VALUES (?, ?, ?, ?)
        ''', (
            self.current_user['id'],
            'add_package',
            f"Додано нову посилку {package_data['tracking_number']}",
            datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        ))
        conn.commit()
        QMessageBox.information(self, "Успіх", "Посилку успішно додано")

        self.refresh_data()

    except sqlite3.Error as e:
        QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
    finally:
        conn.close()

def view_package(self, package_id):
    dialog = PackageInfoDialog(package_id, self)
    dialog.exec()

def delete_package(self, package_id):
    confirm = QMessageBox.question(
        self, "Підтвердження видалення",
        "Ви впевнені, що хочете видалити цю посилку?",
        QMessageBox.StandardButton.Yes | QMessageBox.StandardButton.No
    )
    if confirm == QMessageBox.StandardButton.No:
```

```
            return

        try:
            conn = sqlite3.connect(DB_PATH)
            cursor = conn.cursor()
            cursor.execute("SELECT sector_id, tracking_number FROM packages WHERE id = ?",
(package_id,))
            package_data = cursor.fetchone()

            if not package_data:
                QMessageBox.warning(self, "Помилка", "Посилку не знайдено")
                return

            sector_id, tracking_number = package_data

            cursor.execute("DELETE FROM packages WHERE id = ?", (package_id,))
            if sector_id:
                cursor.execute("UPDATE sectors SET current_fill = current_fill - 1 WHERE id = ?", (sector_id,))

            cursor.execute('''
                INSERT INTO activity_logs (user_id, action, details, timestamp)
                VALUES (?, ?, ?, ?)
            ''', (
                self.current_user['id'],
                'delete_package',
                f"Видалено посилку {tracking_number}",
                datetime.now().strftime('%Y-%m-%d %H:%M:%S')
            ))

            conn.commit()
            QMessageBox.information(self, "Успіх", "Посилку успішно видалено")
            self.refresh_data()

        except sqlite3.Error as e:
            QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
        finally:
            conn.close()

def main():
    create_database()
    app = QApplication(sys.argv)
    app.setStyle('Fusion')
    app.setStyleSheet(AppStyles.setup_global_style())
      window = MainWindow()
    sys.exit(app.exec())

if __name__ == "__main__":
    main()
```

Додаток Г. Реалізація модулів та алгоритмів

```
def accept(self):
    """Валідація та збереження посилки"""
    if not self.tracking_input.text().strip():
        QMessageBox.warning(self, "Помилка", "Введіть номер відстеження")
        return

    if not self.sender_input.text().strip():
        QMessageBox.warning(self, "Помилка", "Введіть відправника")
        return

    if not self.recipient_input.text().strip():
        QMessageBox.warning(self, "Помилка", "Введіть отримувача")
        return

    if not self.manual_sector_checkbox.isChecked():
        self.calculate_recommended_sector()

    sector_id = self.sector_combo.currentData()
    for sector in self.sectors:
        if sector['id'] == sector_id and sector['current_fill'] >= sector['capacity']:
            QMessageBox.warning(self, "Помилка", f"Сектор '{sector['name']}' повністю заповнений! Виберіть інший сектор.")
            return

    super().accept()
```

Рисунок Г.1 – Метод «accept»

```
def calculate_recommended_sector(self):
    """Розрахунок рекомендованого сектора за параметрами посилки"""
    weight = self.weight_input.value()
    length = self.length_input.value()
    width = self.width_input.value()
    height = self.height_input.value()

    volume = length * width * height
```

Рисунок Г.2 – Перетворення розмірів

```
best_sector = None
best_match_score = float('-inf')

for sector in self.sectors:
    if sector['current_fill'] >= sector['capacity']:
        continue

    weight_match = (sector['min_weight'] <= weight <= sector['max_weight'])
    volume_match = (sector['min_volume'] <= volume <= sector['max_volume'])

    if weight_match and volume_match:
        fill_factor = 1 - (sector['current_fill'] / sector['capacity'])

        weight_center = (sector['min_weight'] + sector['max_weight']) / 2
        volume_center = (sector['min_volume'] + sector['max_volume']) / 2

        weight_fit = 1 - abs(weight - weight_center) / (sector['max_weight'] - sector['min_weight'] + 0.001)
        volume_fit = 1 - abs(volume - volume_center) / (sector['max_volume'] - sector['min_volume'] + 0.001)

        match_score = (weight_fit * 0.4) + (volume_fit * 0.4) + (fill_factor * 0.2)

        if match_score > best_match_score:
            best_match_score = match_score
            best_sector = sector
```

Рисунок Г.3 – Пошук посилки за критеріями

```
if best_sector:
    index = self.sector_combo.findData(best_sector['id'])
    if index >= 0:
        self.sector_combo.setCurrentIndex(index)

    self.recommended_sector_label.setText(
        f"<b>Рекомендований сектор:</b> {best_sector['name']}<br>"
        f"<b>Заповненість:</b> {best_sector['current_fill']}/{best_sector['capacity']}<br>"
        f"<b>Опис:</b> {best_sector['description']}"
    )
else:
    self.recommended_sector_label.setText("Не вдалося знайти відповідний сектор. Перевірте параметри.")
```

Рисунок Г.4 – Автоматичне відображення рекомендації

```python
def save_new_package(self, package_data):
    """Збереження нової посилки у базі даних"""
    try:
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()

        cursor.execute("SELECT id FROM packages WHERE tracking_number = ?", (package_data['tracking_number'],))
        if cursor.fetchone():
            QMessageBox.warning(self, "Помилка", f"Посилка з номером відстеження {package_data['tracking_number']} вже існує")
            return

        cursor.execute('''
            INSERT INTO packages (tracking_number, sender_name, recipient_name, weight, length, width, height,
                                  sector_id, shelf_position, arrival_date, status, description)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ''', (
            package_data['tracking_number'],
            package_data['sender_name'],
            package_data['recipient_name'],
            package_data['weight'],
            package_data['length'],
            package_data['width'],
            package_data['height'],
            package_data['sector_id'],
            package_data['shelf_position'],
            datetime.now().strftime('%Y-%m-%d %H:%M:%S'),
            'на складі',
            package_data['description']
        ))

        cursor.execute("UPDATE sectors SET current_fill = current_fill + 1 WHERE id = ?", (package_data['sector_id'],))

        cursor.execute('''
            INSERT INTO activity_logs (user_id, action, details, timestamp)
            VALUES (?, ?, ?, ?)
        ''', (
            self.current_user['id'],
            'add_package',
            f"Додано нову посилку {package_data['tracking_number']}",
            datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        ))

        conn.commit()
        QMessageBox.information(self, "Успіх", "Посилку успішно додано")

        self.refresh_data()

    except sqlite3.Error as e:
        QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
    finally:
        conn.close()
```

Рисунок Г.5 – Метод «save_new_package»

```
def filter_packages(self):
    """Фільтрація посилок за пошуковим запитом"""
    search_text = self.packages_search_input.text().strip().lower()

    if not search_text:
        self.load_packages_data()
        return

    try:
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()

        cursor.execute("""
            SELECT p.id, p.tracking_number, p.sender_name, p.recipient_name,
                   p.weight, s.name as sector_name, p.status
            FROM packages p
            LEFT JOIN sectors s ON p.sector_id = s.id
            WHERE LOWER(p.tracking_number) LIKE ? OR
                  LOWER(p.sender_name) LIKE ? OR
                  LOWER(p.recipient_name) LIKE ?
            ORDER BY p.id DESC
        """, (f'%{search_text}%', f'%{search_text}%', f'%{search_text}%'))

        packages = cursor.fetchall()

        self.packages_table.setRowCount(len(packages))

        for row, (pkg_id, tracking, sender, recipient, weight, sector, status) in enumerate(packages):
            self.packages_table.setItem(row, 0, QTableWidgetItem(str(pkg_id)))
            self.packages_table.setItem(row, 1, QTableWidgetItem(tracking))
            self.packages_table.setItem(row, 2, QTableWidgetItem(sender))
            self.packages_table.setItem(row, 3, QTableWidgetItem(recipient))
            self.packages_table.setItem(row, 4, QTableWidgetItem(str(weight)))
            self.packages_table.setItem(row, 5, QTableWidgetItem(sector if sector else ""))
            self.packages_table.setItem(row, 6, QTableWidgetItem(status))

            actions_widget = QWidget()
            actions_layout = QHBoxLayout(actions_widget)
            actions_layout.setContentsMargins(0, 0, 0, 0)

            view_btn = QPushButton("Перегляд")
            view_btn.setStyleSheet(AppStyles.get_button_style("success"))
            pkg_id_copy = pkg_id
            view_btn.clicked.connect(lambda _, pid=pkg_id_copy: self.view_package(pid))
            actions_layout.addWidget(view_btn)

            delete_btn = QPushButton("Видалити")
            delete_btn.setStyleSheet(AppStyles.get_button_style("danger"))
            delete_btn.clicked.connect(lambda _, pid=pkg_id_copy: self.delete_package(pid))
            actions_layout.addWidget(delete_btn)

            self.packages_table.setCellWidget(row, 7, actions_widget)

    except sqlite3.Error as e:
        QMessageBox.critical(self, "Помилка бази даних", f"Помилка: {e}")
    finally:
        conn.close()
```

Рисунок Г.6 – Метод «filter_packages»

Додаток Д. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

АЛГОРИТМІЧНІ ТА ПРОГРАМНІ ЗАСОБИ ОПТИМІЗАЦІЇ РОЗМІЩЕННЯ ПОСИЛОК НА СКЛАДІ В ІНТЕЛЕКТУАЛЬНИХ ЛОГІСТИЧНИХ СИСТЕМАХ

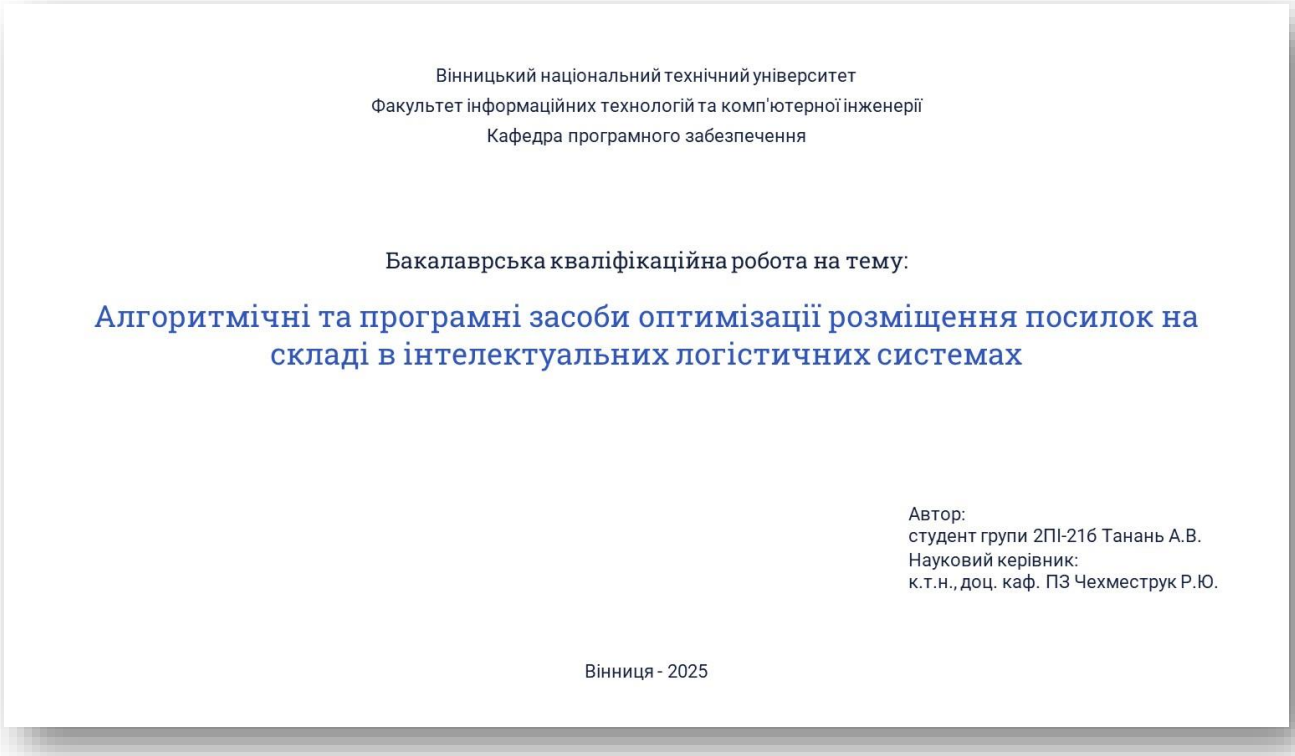

Рисунок Д.1 – Титульний слайд

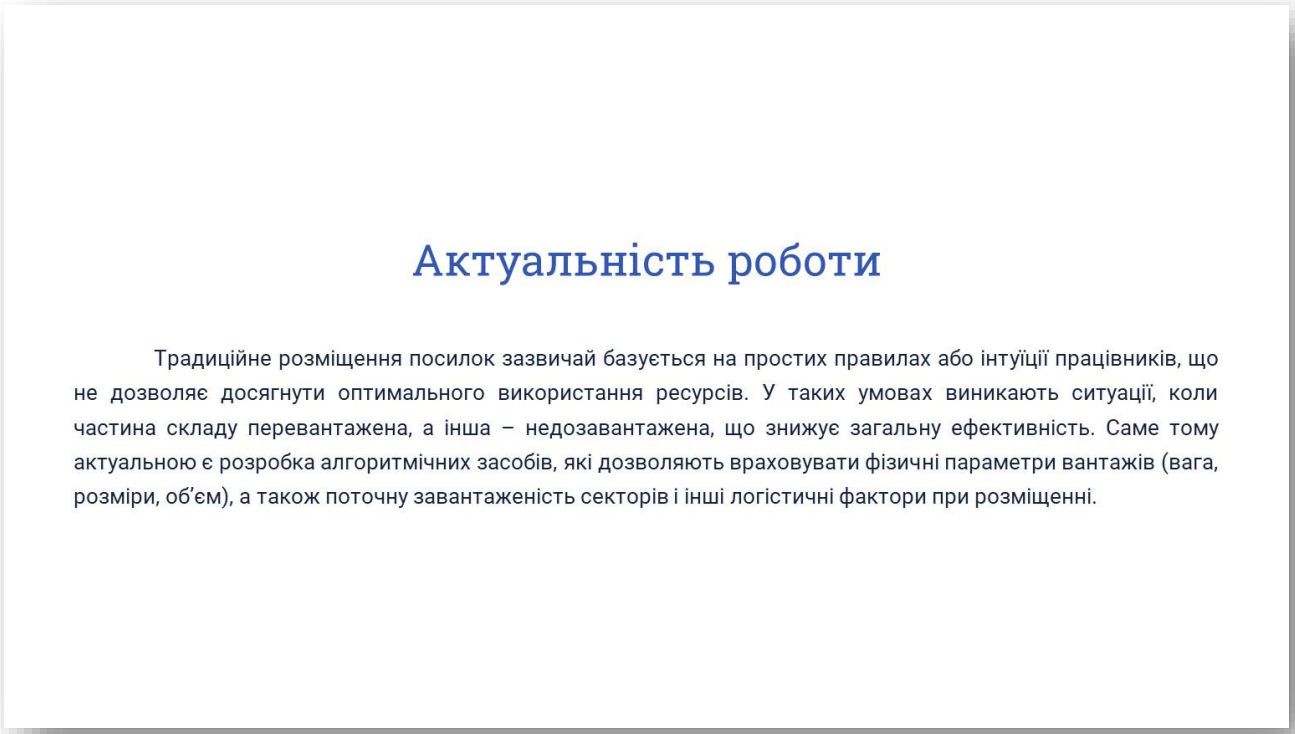

Рисунок Д.2 – Актуальність роботи

Рисунок Д.3 – Мета, предмет та об’єкт дослідження

Порівняльний аналіз аналогів

Критерії	Warehouse Expert	SAP EWM	IT-Enterprise WMS	SmartStock WMS	Odoo Inventory	Власна програма
Автоматичне визначення сектору	-	+	+	-	-	+
Адаптованість для малого бізнесу	+	-	-	+	+	+
Вартість та доступність	+	-	-	+	+	+
Інтуїтивність інтерфейсу	+	-	-	+	+	+
Візуалізація заповненості складу	-	+	+	+	-	+
Загальна оцінка	60%	40%	40%	80%	60%	100%

Рисунок Д.4 – Порівняльний аналіз аналогів

Постановка задачі

1. Аналіз розвитку систем складської логістики
2. Розробка моделі та алгоритмів роботи системи
3. Реалізація модуля керування посилками
4. Реалізація алгоритму визначення оптимального сектора
5. Розробка інтерфейсу програмного застосунку
6. Тестування системи

Рисунок Д.5 – Постановка задачі

Новизна

Комбінований алгоритм оптимізації

Удосконалено комбінований алгоритм оптимізації розміщення посилок на складі, який, на відміну від відомих, враховує обмеження за вагою, об'ємом та рівнем заповненості секторів, що дозволяє уникати перевантаження окремих зон та покращити просторову рівномірність розподілу вантажів.

Модуль динамічного оновлення

Подальшого розвитку отримав модуль динамічного оновлення стану секторів, що реагує на дії користувача в режимі реального часу, завдяки чому забезпечується підвищення оперативності управління складськими ресурсами.

Архітектурно незалежне рішення

Створено архітектурно незалежне автономне програмне рішення, яке не вимагає інтеграції зі складними ERP-системами, що робить його придатним для впровадження в умовах малих і середніх логістичних підприємств.

Рисунок Д.6 – Новизна

Практична цінність

Практична цінність результатів полягає в створенні функціонального і доступного програмного рішення, яке може бути впроваджене на логістичних об'єктах з метою підвищення точності розміщення вантажів; зниження навантаження на персонал завдяки автоматизації процесів; скорочення часу обробки запитів; забезпечення автономної роботи без зовнішніх серверів; покращення обслуговування завдяки візуалізації стану складу та простому інтерфейсу.

Рисунок Д.7 – Практична цінність

Принцип роботи алгоритму визначення сектору

Метод обмежень

У межах роботи було застосовано метод обмежень у поєднанні з логікою жадібного вибору.

Аналіз посилок

Кожна посилка аналізується на відповідність доступним секторам за об'ємом і вагою.

Вибір оптимального сектора

З допустимих варіантів обирається той, який має найменший рівень завантаження. Це дозволяє уникнути як перевантаження окремих зон, так і втрати простору в менш заповнених секторах.

Реалізація алгоритму

Реалізація алгоритму виконана у вигляді окремої функції, що автоматично викликається під час додавання нової посилки та передає результат у форму збереження.

Рисунок Д.8 – Принцип роботи алгоритму визначення сектору

Рисунок Д.9 – Блок-схема роботи програми

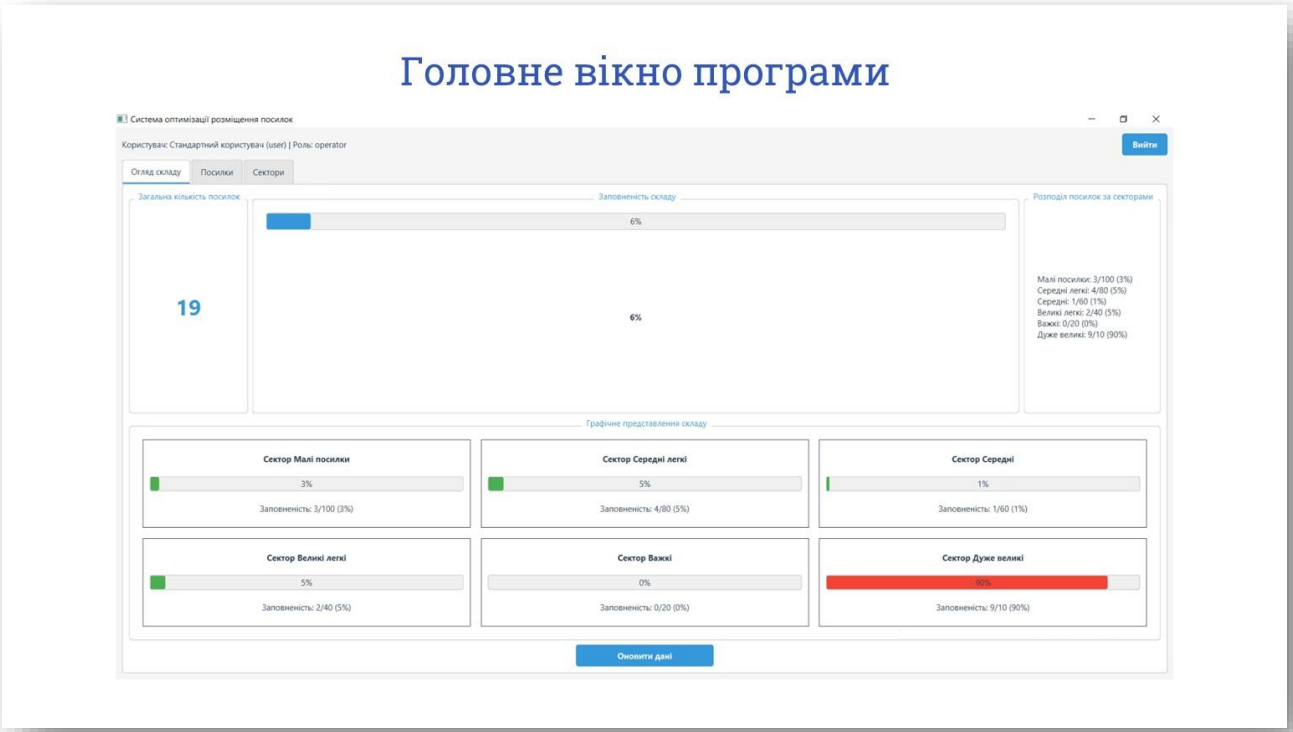

Рисунок Д.10 – Головне вікно програми

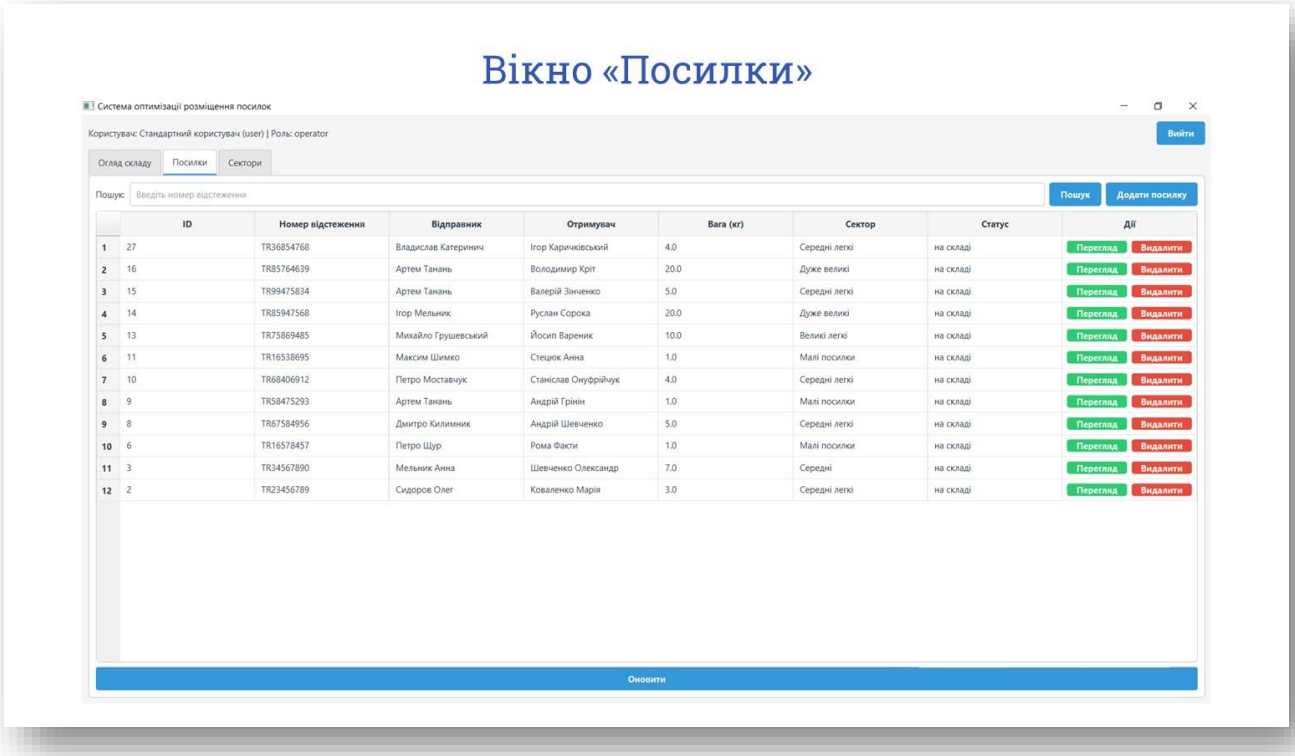

Рисунок Д.11 – Вікно «Посилки»

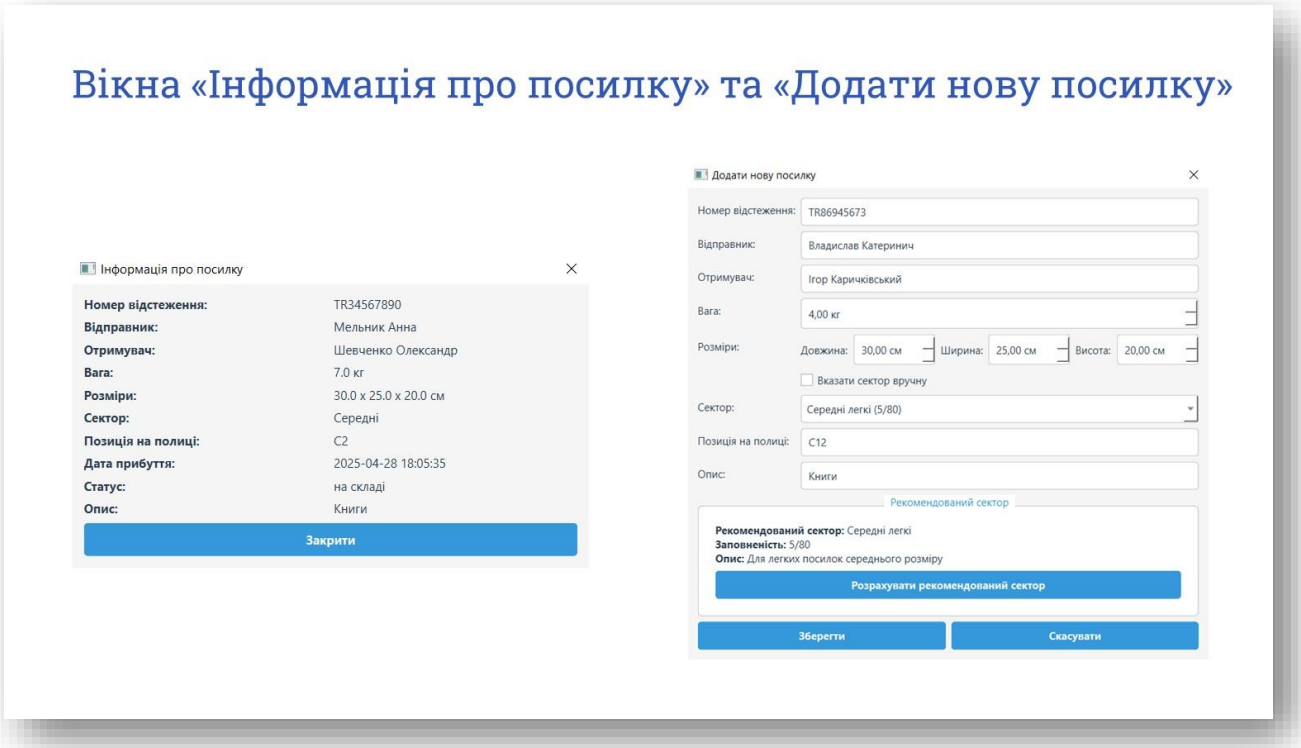

Рисунок Д.12 – Вікна «Інформація про посилку» та «Додати нову посилку»

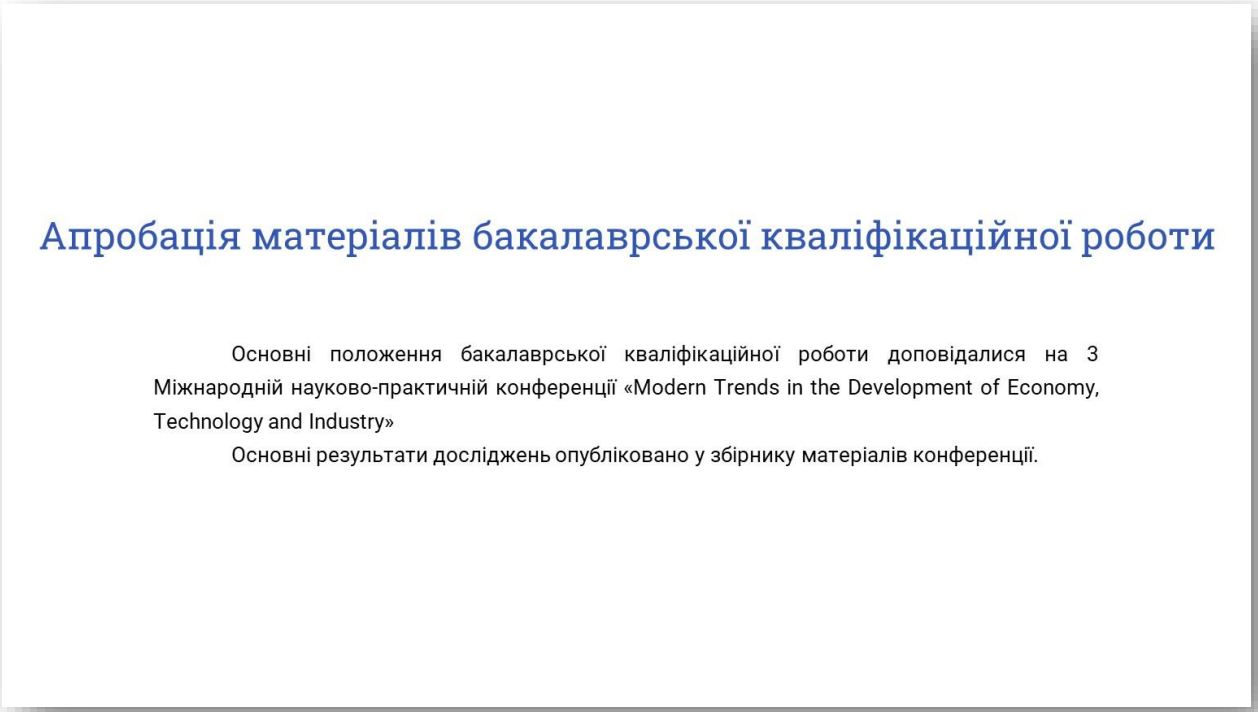

Рисунок Д.13 – Апробація матеріалів бакалаврської кваліфікаційної роботи

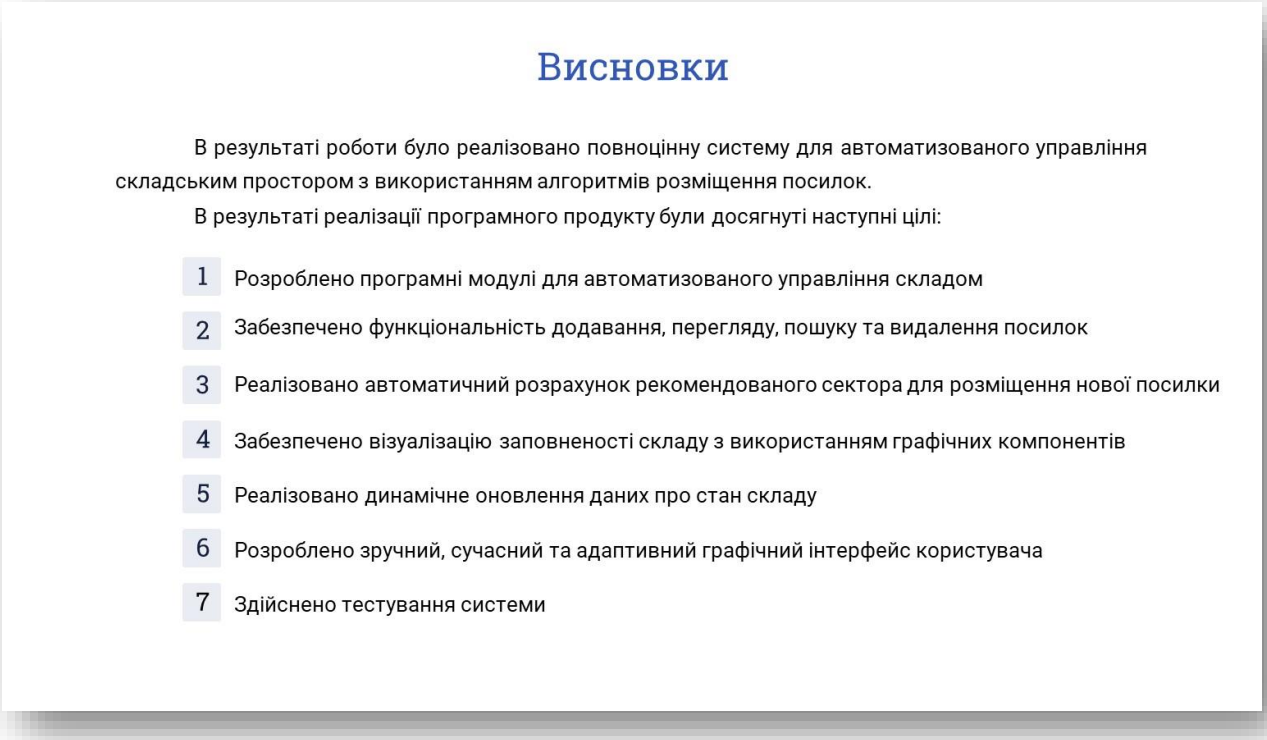

Рисунок Д.14 – Висновки