Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Сергієнка Олександра Романовича*
(ПІБ)

академічної групи *122-20-4*
(шифр)

освітньої програми *Комп'ютерні науки*
(код і назва напряму підготовки)

на тему: Розробка автоматизованої інформаційної системи для підприємства з прийому та продажу металу

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Кабак Л.В*			
розділів:				
спеціальний	*доц. Кабак Л.В*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент	*доц. Каштан В.Ю.*			
Нормоконтролер	*доц. Гуліна І. Г.*			

Дніпро
2024

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » 2024 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-20-4* *Сергієнко О. Р.*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи Розробка автоматизованої інформаційної системи для підприємства з прийому та продажу металу

затверджена наказом ректора НТУ «ДП» від *23 . 05 .2024 р.* № *470 -с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2024 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2024 р.***

Завдання видав _____ *доц.Кабак Л.В.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Сергієнко О.Р.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2024 р.*
Термін подання кваліфікаційної роботи до ЕК: *10.06.2024 р.*

РЕФЕРАТ

Пояснювальна записка: 96 с., 15 рис., 3 дод., 20 джерел.

Об’єктом розробки є програмне забезпечення для управління продажами та покупками металів на підприємствах металургійної галузі.

Метою роботи є створення інформаційної системи, яка дозволяє автоматизувати облік продажів та покупок металів, забезпечити зручність і точність введення та обробки даних, а також підвищити ефективність управління підприємством.

Для досягнення поставлених цілей використовувалися методи системного аналізу, програмування на мові Python, робота з базами даних MongoDB, а також інтерфейсні фреймворки Qt.

Розроблена інформаційна система включає такі функціональні можливості: введення та зберігання даних про продажі та покупки металів; генерація звітів за заданим періодом часу; управління бюджетом на день; забезпечення безпеки даних шляхом автентифікації користувачів та шифрування паролів; заміна пароля адміністратора; перегляд наявних продуктів та їх кількість; видалення транзакцій; пошук продуктів у системі.

Практичне значення розробленого програмного забезпечення полягає у можливості його впровадження на підприємствах металургійної галузі для автоматизації процесів управління продажами та покупками металів. Це дозволить зменшити обсяг ручної роботи, підвищити точність обліку даних та оперативність доступу до необхідної інформації. Система також забезпечує збереження даних та їх захист, що є критично важливим для підприємств, що працюють з великими обсягами інформації.

Подальший розвиток розробленого програмного забезпечення може включати додавання нових функцій, таких як інтеграція з іншими системами управління підприємством (ERP), розширення можливостей аналізу даних та прогнозування, впровадження мобільних додатків для доступу до системи з будь-якого місця. Крім того, можливо розширення сфери застосування програмного забезпечення на інші галузі промисловості, що потребують ефективного управління ресурсами та даними.

Список ключових слів: ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МЕТАЛУРГІЯ, УПРАВЛІННЯ ПРОДАЖАМИ, АВТОМАТИЗАЦІЯ, ОБЛІК ДАНИХ, БАЗИ ДАНИХ, PYTHON, MONGODB, QT, ІНФОРМАЦІЙНА СИСТЕМА, ЗАХИСТ ДАНИХ, АНАЛІЗ ДАНИХ, ERP.

ABSTRACT

Explanatory note: 96 p., 15 figures, 3 appendix, 20 sources.

The object of development is software for managing sales and purchases of metals at metallurgical enterprises.

The aim of the work is to create an information system that allows automating the accounting of sales and purchases of metals, ensuring the convenience and accuracy of data entry and processing, as well as increasing the efficiency of enterprise management.

To achieve these goals, we used the methods of system analysis, Python programming, working with MongoDB databases, and Qt interface frameworks. The developed information system includes the following functionalities: entering and storing data on sales and purchases of metals; generating reports for a given period of time; managing the budget for the day; ensuring data security by authenticating users and encrypting passwords; changing the administrator password; viewing available products and their quantity; deleting transactions; searching for products in the system.

The practical significance of the developed software lies in the possibility of its implementation at metallurgical enterprises to automate the processes of managing sales and purchases of metals. This will reduce the amount of manual work, improve the accuracy of data accounting and the speed of access to the necessary information. The system also ensures data security and protection, which is critical for companies working with large amounts of information.

Further development of the developed software may include the addition of new features, such as integration with other enterprise resource planning (ERP) systems, expanding data analysis and forecasting capabilities, and the introduction of mobile applications to access the system from anywhere. In addition, it is possible to expand the scope of the software to other industries that require efficient resource and data management.

List of keywords: SOFTWARE, METALLURGY, SALES MANAGEMENT, AUTOMATION, DATA ACCOUNTING, DATABASES, PYTHON, MONGODB, QT, INFORMATION SYSTEM, DATA PROTECTION, DATA ANALYSIS, ERP.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ПЗ – програмне забезпечення;

МБ – мегабайти;

ГБ – гігабайти;

GUI – графічний інтерфейс користувача;

UI – користувацький інтерфейс (User Interface);

ERP – система управління підприємством (Enterprise Resource Planning);

AES – стандарт шифрування даних;

UTF-8 – формат кодування символів;

XLSX – формат файлів Excel.

ЗМІСТ

ВСТУП

Сучасний розвиток інформаційних технологій значно вплинув на різні галузі промисловості, включаючи металургійну. Програмне забезпечення стало невід'ємною частиною багатьох бізнес-процесів, забезпечуючи автоматизацію обліку, управління даними та підвищення ефективності управління. В умовах зростаючих обсягів інформації та постійного розвитку технологій, підприємства металургійної галузі стикаються з новими викликами, пов'язаними з обробкою великих масивів даних, забезпеченням їх безпеки та оперативного доступу до них.

Актуальність даної кваліфікаційної роботи полягає в розробці програмного забезпечення для автоматизації процесів управління продажами та покупками металу, що є важливою складовою ефективного функціонування металургійних підприємств. Сучасні інформаційні системи дозволяють підвищити точність обліку, зменшити обсяги ручної роботи та забезпечити швидкий доступ до необхідної інформації, що в кінцевому підсумку сприяє підвищенню конкурентоспроможності підприємств.

Метою цієї роботи є розробка інформаційної системи, яка забезпечить автоматизацію обліку продажів та покупок металу, підвищить зручність та точність введення і обробки даних, а також підвищить ефективність управління підприємством.

Для досягнення поставленої мети визначено наступні завдання:

1. Провести аналіз існуючих рішень та технологій у сфері автоматизації обліку продажів та покупок металу.
2. Розробити структуру та функціональні можливості інформаційної системи.
3. Реалізувати інформаційну систему за допомогою сучасних технологій програмування.
4. Провести тестування та валідацію розробленого програмного забезпечення.

5. Оцінити економічну ефективність впровадження розробленої системи.

Об’єктом дослідження є процеси обліку та управління продажами та покупками металу на підприємствах металургійної галузі.

Предметом дослідження є інформаційна система для автоматизації обліку продажів та покупок металу.

Методологія дослідження базується на системному підході та включає методи аналізу, проектування, програмування на мові Python, роботу з базами даних MongoDB, а також використання інтерфейсних фреймворків Qt.

Наукова новизна роботи полягає у розробці інноваційного програмного забезпечення, яке забезпечує комплексний підхід до автоматизації обліку продажів та покупок металу, включаючи функції генерації звітів, управління бюджетом та забезпечення безпеки даних.

Практична значимість розробленого програмного забезпечення полягає у можливості його впровадження на підприємствах металургійної галузі для автоматизації процесів управління продажами та покупками металу, що дозволить зменшити обсяги ручної роботи, підвищити точність обліку даних та оперативність доступу до необхідної інформації.

У подальшому розробка може бути доповнена новими функціями, такими як інтеграція з іншими системами управління підприємством (ERP), розширення можливостей аналізу даних та прогнозування, а також впровадження мобільних додатків для доступу до системи з будь-якого місця. Це дозволить значно підвищити ефективність управління ресурсами та даними, забезпечуючи стабільний розвиток підприємств.

Таким чином, розробка інформаційної системи для управління продажами та покупками металу є актуальною задачею, яка сприятиме підвищенню ефективності та конкурентоспроможності підприємств металургійної галузі в сучасних умовах.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної області

Металургійна галузь є однією з найстаріших галузей промисловості, що з'явилася ще в давні часи. Вона має велике значення для розвитку економіки та технологічного прогресу. Металургійна галузь включає в себе різні етапи виробництва, такі як видобуток сировини, її переробка, виробництво металевих виробів та їх реалізація. Незважаючи на те, що металургійна галузь України є важливим учасником глобального ринку чорних металів та сировини для їх виготовлення, має певні переваги у їх виробництві та експорті, вона, як і вся національна економіка, є відкритою та за макроекономічними рисами – малою. Це дає підстави віднести металургійну галузь України до категорії "малої відкритої галузі", яка характеризується експортною орієнтованістю, незначною часткою випуску у світовому виробництві, експорті, імпорті, внутрішньому споживанні продукції і, що найголовніше, – відсутністю визначального впливу на формування світової ціни[1].

Основні види металів, які виробляються в металургійній галузі, включають сталь, чавун, алюміній, мідь, свинець, цинк та інші. Кожен вид металу має свої унікальні властивості, що роблять його придатним для різних застосувань.

Продаж та закупівля металів є важливим елементом бізнес-процесів в металургійній галузі. Це включає в себе відстеження витрат на закупівлю сировини, виробництво та продаж готової продукції.

Управління продажами та закупівлями металів вимагає від підприємств використання спеціалізованого програмного забезпечення, яке забезпечує автоматизацію процесів введення даних про продажі та покупки, а також генерацію звітів та управління бюджетом.

Програмне забезпечення для управління продажами та закупівлями металів повинно включати в себе наступні функції:

- Введення та зберігання даних про продажі та покупки металів.
- Автоматичний розрахунок сумарної вартості на основі введеної ваги та ціни.
- Генерація звітів за заданим періодом.
- Управління бюджетом на день.
- Забезпечення безпеки даних.

Розробка програмного забезпечення для управління продажами та закупівлями металів з використанням сучасних технологій та фреймворків дозволить створити ефективний та зручний інструмент для управління бізнес-процесами в галузі металургії.

Основні вимоги до програмного забезпечення для управління продажами та закупівлями металів включають:

1. Функціональність: ПЗ повинно забезпечувати можливість введення та зберігання даних про продажі та покупки металів, автоматичний розрахунок сумарної вартості на основі введеної ваги та ціни, генерацію звітів за заданим періодом, управління бюджетом на день та забезпечення безпеки даних.
2. Інтерфейс користувача: ПЗ повинно мати інтуїтивно зрозумілий та зручний інтерфейс користувача, що дозволяє легко виконувати всі необхідні операції.
3. БД: ПЗ повинно працювати з базою даних, що забезпечує зберігання всіх необхідних даних про продажі та покупки металів.
4. Технології: ПЗ повинно бути розроблене з використанням сучасних технологій та фреймворків, що забезпечує його ефективну роботу та швидке виконання всіх необхідних операцій.
5. Безпека: ПЗ повинно забезпечувати безпеку даних, що зберігаються в базі даних, за допомогою аутентифікації та авторизації, шифрування конфіденційної інформації та резервного копіювання даних.

Усі ці вимоги повинні бути враховані при розробці програмного забезпечення для управління продажами та закупівлями металів, щоб забезпечити його ефективну роботу та забезпечення всіх необхідних функцій для управління бізнес-процесами в галузі металургії.

1.2. Призначення розробки та галузь застосування

ПЗ для управління продажами та закупівлями металів призначене для автоматизації процесів введення даних про продажі та покупки металів, генерації звітів та управління бюджетом.

Ця програма є важливим інструментом для металургійних підприємств, торговельних компаній, що займаються продажем та закупівлею металів. Вона допомагає управляти складом, відстежувати фінансові операції та генерувати аналітичні звіти.

ПЗ для управління продажами та закупівлями металів може бути використане в різних галузях, таких як:

- Металургійна промисловість: для управління продажами та закупівлями металів, відстеження витрат на сировину та виробництво, генерації звітів про продажі та покупки.
- Торговельні компанії: для управління продажами та закупівлями металів, відстеження витрат на закупівлю та продаж, генерації звітів про продажі та покупки.
- Логістика: для управління складом металів, відстеження витрат на транспортування та зберігання, генерації звітів про витрати на логістику.

ПЗ для управління продажами та закупівлями металів є важливим інструментом для підвищення ефективності роботи підприємств та забезпечення їх конкурентоспроможності на ринку. Вона дозволяє автоматизувати процеси введення даних про продажі та покупки металів, генерацію звітів та управління

бюджетом, що дозволяє підприємствам приймати обґрунтовані рішення щодо управління своїм бізнесом.

1.3. Підстави для розробки

Необхідність автоматизації процесів управління продажами та закупівлями металів для підвищення ефективності роботи підприємства, зменшення помилок при введенні даних, пришвидшення роботи персоналу, запобігання фінансового шахрайства з боку персоналу та забезпечення оперативного доступу до інформації.

1.4. Постановка завдання

Розробити ПЗ, яке забезпечує:

– Введення та зберігання даних про продажі та покупки металів.

– Генерацію звітів за заданим періодом часу.

– Управління бюджетом на день.

– Забезпечення безпеки даних.

– Заміна пароля адміністратора.

– Перегляд наявних продуктів які є в наявності та їх кількість.

– Видалення транзакцій.

– Пошук продуктів в системі.

– Показ суми за уведену вагу.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Інформаційна система повинна надавати такий функціонал:

1. Додавання в базу даних назву продукту.
2. Додавати в базу даних ціну за кілограм або кількість та сумарну ціну ваги.
3. Розраховувати сумарну вагу металу.
4. Показувати суму при покупці чи продажі товару.
5. Давати можливість власнику підприємства вибирати період для звітності.
6. Формування Exel-звіту про прибуток і проданих металів.
7. Надання окремого функціоналу для власника підприємства.
8. Можливість заміни пароля для адміністратора.
9. Можливість видалення транзакцій.
10. Показ списку товарів які є у наявності.
11. Пошук назви товарів та пошук товарів на складі.

1.5.2. Вимоги до інформаційної безпеки

Основними вимогами до інформаційної безпеки є:

– забезпечення цілісності даних;

– встановлення базового антивіруса Windows Defender

Для того, щоб ПЗ надійно функціонувало необхідно дотримуватися наступних вимог:

– використання лише ліцензійного програмного забезпечення;

– встановлення блоків безперебійного живлення;

– захищеність від зловмисних програм, які можуть завдати шкоди ПЗ.

1.5.3. Вимоги до складу та параметрів технічних засобів

– Програма може працювати на операційних системах: Windows, MacOS, Linux.

– Мінімальні вимоги до комп'ютера:

- Процесор: Intel Xeon E5 2670 або AMD Ryzen 3
- Оперативна пам'ять: 2 ГБ
- Відеокарта: інтегрована
- Місце на жорсткому диску: 500 МБ

1.5.4. Вимоги до інформаційної та програмної сумісності

Для нормального функціонування програми необхідно ПЗ персонального комп’ютера, на якому буде встановлено Windows 10 та вище. Застосунок реалізовано на мові програмування Python з використанням фраймворку PyQt5 та бібліотеки openpyxl та pymongo з використанням БД MongoDB.

РОЗДІЛ 2.
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1 Функціональне призначення системи

Цей застосунок є системою управління продуктами, яка дозволяє користувачам здійснювати покупку та продаж товарів, встановлювати та контролювати бюджет, а також надає адміністративні функції для управління даними та конфігурації системи. Основні можливості системи включають:

Купівля та продаж продуктів Головне вікно (MainWindow) дозволяє користувачам вводити деталі продукту (назва, ціна, вага/кількість та одиниця виміру) та здійснювати операції купівлі або продажу. Система розраховує загальну вартість, оновлює бюджет та додає/видаляє запаси продуктів відповідно до транзакцій.

Управління бюджетом Система дозволяє користувачам встановлювати щоденний бюджет через вікно BudgetWindow. Бюджет оновлюється на основі операцій купівлі та продажу.

Пошук та перегляд продуктів Головне вікно містить функцію пошуку продуктів за назвою та відображає список наявних продуктів разом з їх поточними запасами.

Адміністративна панель Вікно AdminWindow забезпечує адміністративні функції, такі як генерування звітів про транзакції за вказаний період часу, видалення продуктів, зміна пароля адміністратора та видалення окремих транзакцій.

Інтеграція з базою даних Система використовує MongoDB як базу даних для зберігання транзакцій, бюджетів, продуктів, щоденних бюджетів, назв продуктів та пароля адміністратора.

Автентифікація користувачів Для доступу до вікна AdminWindow потрібен пароль адміністратора, що забезпечує безпечний доступ до адміністративних функцій.

Загалом, ця Система управління продуктами надає графічний користувацький інтерфейс (GUI) для управління продуктами, відстеження запасів, обробки транзакцій (купівлі та продажу), встановлення та моніторингу бюджетів, а також генерування звітів. Вона також включає адміністративні функції для управління даними та конфігурації системи.

Користувачі можуть легко вводити деталі продуктів, здійснювати покупки та продажі, відстежувати наявні запаси та контролювати свій бюджет. Адміністратори мають доступ до додаткових функцій, таких як генерування звітів, видалення продуктів, зміна пароля та видалення окремих транзакцій.

Система інтегрована з базою даних MongoDB, що забезпечує зберігання та відновлення даних транзакцій, бюджетів, продуктів та інших відомостей. Це дозволяє підтримувати цілісність даних та забезпечувати безперебійну роботу системи.

Загалом, ця Система управління продуктами є потужним інструментом для ефективного управління запасами, контролю бюджету та відстеження транзакцій. Вона пропонує зручний інтерфейс для користувачів та розширені можливості для адміністраторів, забезпечуючи всебічне рішення для управління продуктами.

Експлуатаційне призначення даного програмного коду полягає в створенні та підтримці функціонування Системи управління продуктами, яка призначена для використання в різноманітних комерційних середовищах, таких як магазини роздрібної торгівлі, склади, або малі підприємства. Ця система забезпечує ефективне управління запасами, контроль за транзакціями та моніторинг бюджету.

Основні аспекти експлуатаційного призначення системи включають:

1. Облік товарів: Система дозволяє користувачам вести детальний облік наявних товарів. Користувачі можуть додавати нові продукти, вказуючи їх

назву, ціну, кількість та одиницю виміру. Ця функція є критичною для підтримки актуальної інформації про наявні запаси.

2. Здійснення транзакцій: Користувачі можуть легко виконувати операції купівлі та продажу товарів. При кожній транзакції система автоматично оновлює інформацію про кількість товару та загальний бюджет. Це забезпечує точний облік руху товарів та фінансів.

3. Управління бюджетом: Система надає можливість встановлювати та контролювати щоденний бюджет. Це дозволяє користувачам ефективно планувати свої витрати та доходи, а також відстежувати фінансову ефективність бізнесу.

4. Пошук та відображення інформації: Користувачі можуть легко знаходити потрібні товари за допомогою функції пошуку. Система також відображає список наявних товарів, що дозволяє швидко оцінити поточні запаси.

5. Генерація звітів: Адміністративна панель дозволяє генерувати детальні звіти про транзакції за вказаний період часу. Це є важливим інструментом для аналізу продажів, оцінки ефективності бізнесу та прийняття управлінських рішень.

6. Адміністрування системи: Адміністратори мають доступ до додаткових функцій, таких як видалення продуктів, зміна паролю доступу та видалення окремих транзакцій. Це забезпечує необхідний рівень контролю та гнучкості в управлінні системою.

7. Забезпечення безпеки даних: Система використовує механізм автентифікації для доступу до адміністративних функцій, що захищає важливі дані та функції від несанкціонованого доступу.

8. Інтеграція з базою даних: Використання MongoDB як бази даних забезпечує надійне зберігання всіх даних системи, включаючи інформацію про продукти, транзакції, бюджети та користувачів.

Ця система призначена для щоденного використання в комерційному середовищі. Вона розроблена з урахуванням потреб користувачів різного рівня:

від звичайних працівників, які здійснюють операції купівлі-продажу, до менеджерів, які аналізують дані та приймають стратегічні рішення.

Система має інтуїтивно зрозумілий графічний інтерфейс, що дозволяє користувачам швидко освоїти її функціональність. Вона розроблена з використанням PyQt5, що забезпечує крос-платформну сумісність та можливість роботи на різних операційних системах.

Важливим аспектом експлуатаційного призначення є можливість системи адаптуватися до різних типів бізнесу. Гнучкість у визначенні типів продуктів, одиниць виміру та форматів звітності дозволяє використовувати систему в різноманітних комерційних контекстах.

Система також розроблена з урахуванням можливості майбутнього розширення. Модульна структура коду дозволяє легко додавати нові функції або модифікувати існуючі без суттєвих змін в архітектурі системи.

2.2. Опис застосованих математичних методів

У розробленій Системі управління продуктами застосовано ряд математичних методів для забезпечення коректної роботи різних функцій. Розглянемо основні математичні методи, що використовуються в системі:

1. Арифметичні операції

Базові арифметичні операції широко використовуються в системі для різних обчислень:

a) Додавання та віднімання:
   - Оновлення кількості товару при купівлі чи продажу: Нова_кількість = Поточна_кількість ± Кількість_транзакції
   - Оновлення бюджету: Новий_бюджет = Поточний_бюджет ± Сума_транзакції

b) Множення:

- Розрахунок загальної вартості транзакції: Загальна_вартість = Ціна_за_одиницю * Кількість

Приклад застосування:

```
def handle_transaction(self, action):
    price_per_unit = float(self.price_input.text())
    weight_or_quantity = float(self.weight_input.text())
    total = price_per_unit * weight_or_quantity

    if action == 'Куплено':
        current_budget = get_daily_budget()
        if total > current_budget:
            QMessageBox.warning(self, "Помилка",
"Неможливо купити продукт. Сума перевищує бюджет.")
            return
```

2. Порівняння

Операції порівняння використовуються для прийняття рішень у різних частинах системи:

a) Перевірка достатності бюджету:

```
if total > current_budget:
    QMessageBox.warning(self, "Помилка", "Неможливо
купити продукт. Сума перевищує бюджет.")
    return
```

b) Перевірка наявності достатньої кількості товару для продажу:

```
if not existing_product or
existing_product["weight_or_quantity"] <
weight_or_quantity:
    QMessageBox.warning(self, "Помилка", "Неможливо
здійснити продаж. На складі недостатньо товару.")
    return
```

3. Округлення

Метод округлення використовується для забезпечення точності фінансових розрахунків:

```
self.total_label.setText(f"Сума: {total:.2f} грн")
```

Тут `.2f` вказує на округлення до двох десяткових знаків.

4. Фільтрація даних

При генерації звітів використовується математичний метод фільтрації даних на основі дат:

```
filtered_transactions = [t for t in transactions if
start_date <= t['datetime'].date() <= end_date]
```

Це дозволяє вибрати тільки ті транзакції, які відбулися в заданому діапазоні дат.

5. Агрегація даних

При підрахунку загального прибутку та загального бюджету використовується метод агрегації даних:

```
total_profit = 0
for transaction in filtered_transactions:
    if transaction.get("action") == 'Продано':
        total_profit += transaction.get("total", 0)
    else:
        total_profit -= transaction.get("total", 0)
```

```
total_budget = 0
for budget in budgets:
    total_budget += budget.get("budget", 0)
```

6. Хешування

Для забезпечення безпеки при зберіганні паролю адміністратора використовується метод хешування:

```
key = hashlib.sha256("секретний ключ".encode()).digest()
```

Цей метод перетворює вхідний текст у фіксований набір байтів, що забезпечує однонапрямлене шифрування.

7. Шифрування та дешифрування

Для захисту паролю адміністратора використовуються математичні методи симетричного шифрування (AES):

```
def encrypt_password(password):
    key = hashlib.sha256("секретний
ключ".encode()).digest()
```

```
    cipher = AES.new(key, AES.MODE_ECB)
    encrypted_password =
cipher.encrypt(password.rjust(32).encode())
    return base64.b64encode(encrypted_password).decode()

def decrypt_password(encrypted_password):
    key = hashlib.sha256("секретний
ключ".encode()).digest()
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_password =
cipher.decrypt(base64.b64decode(encrypted_password))
    return decrypted_password.decode().strip()
```

Ці методи використовують складні математичні операції для перетворення тексту в зашифрований формат і назад.

Всі ці математичні методи є не від'ємною для функціонування системи, забезпечуючи точність розрахунків, правильність прийняття рішень та безпеку даних. Вони застосовуються в різних частинах системи відповідно до потреб конкретних функцій та операцій.

2.3. Опис використаних технологій та мов програмування

У розробці системи управління покупок та прийому металу було використано ряд сучасних технологій та мов програмування. Вибір технологій та мов програмування для розробки Системи управління продуктами був зроблений на основі ретельного аналізу вимог проекту, порівняння різних альтернатив та оцінки їх переваг і недоліків. Комбінація Python, PyQt5, MongoDB, PyCryptodome та openpyxl забезпечує оптимальне рішення для створення ефективної, гнучкої та безпечної системи з зручним користувацьким інтерфейсом.

1. Мова програмування Python

Python був обраний як основна мова програмування для розробки системи. Це високорівнева мова програмування загального призначення, яка відома своєю

простотою, читабельністю та великою кількістю бібліотек. Ця мова програмування, яка завоювала популярність завдяки своїй простоті, універсальності та потужності. Вона використовується в широкому колі завдань, від розробки веб-сайтів до машинного навчання[2].

Обґрунтування вибору:

- Простота та читабельність: Python має чистий і зрозумілий синтаксис, що полегшує розробку та подальшу підтримку коду.
- Багата екосистема: Python має велику кількість бібліотек та фреймворків, які можна використовувати для різних завдань.
- Крос-платформність: Python працює на різних операційних системах, що робить додаток більш універсальним.
- Продуктивність розробки: Python дозволяє швидко створювати прототипи та розробляти повноцінні додатки.

Приклад використання:

```
def main():
    import sys
    app = QApplication(sys.argv)
    main_window = MainWindow()
    admin_window = AdminWindow()
    budget_window = BudgetWindow(main_window)

    database.set_main_window(main_window)

    main_window.admin_button.clicked.connect(admin_window.show)
    main_window.budget_button.clicked.connect(budget_window.show)

    main_window.show()
    sys.exit(app.exec_())
```

2. Фреймворк PyQt5

PyQt5 — це оболонка графічного інтерфейсу Python для створення програм з графічним інтерфейсом за допомогою інструментарію Qt[3], який є одним з найпопулярніших фреймворків для створення графічних користувацьких інтерфейсів (GUI).

Обґрунтування вибору:

- Багатий набір віджетів: PyQt5 надає широкий спектр готових компонентів інтерфейсу.
- Крос-платформність: додатки, створені з PyQt5, можуть працювати на різних операційних системах.
- Продуктивність: PyQt5 забезпечує швидку роботу інтерфейсу.
- Гнучкість дизайну: можливість створювати складні та інтерактивні інтерфейси.

Приклад використання:

```
class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.init_ui()

    def init_ui(self):
        product_name_label = QLabel("Назва продукту")
        self.product_name_input = QLineEdit()

self.product_name_input.textChanged.connect(self.capitali
ze_first_letter)

self.product_name_input.setFocusPolicy(Qt.StrongFocus)

self.product_name_input.setStyleSheet("background-color:
#FFDAB9;")
```

3. База даних MongoDB

MongoDB - це документоорієнтована БД, яка забезпечує високу продуктивність, високу доступність та легку масштабованість[4].

Обґрунтування вибору:

- Гнучка схема даних: MongoDB дозволяє зберігати документи з різною структурою в одній колекції.
- Висока продуктивність: забезпечує швидкі операції читання та запису.
- Масштабованість: легко масштабується горизонтально.

- Підтримка складних запитів: дозволяє виконувати складні запити та агрегації.

Приклад використання:

```
def add_transaction(product_name, price_per_unit,
weight_or_quantity, unit, action, total):
    transaction = {
        "transaction_number": next_transaction_number,
        "product_name": product_name,
        "price_per_unit": price_per_unit,
        "weight_or_quantity": weight_or_quantity,
        "unit": unit,
        "action": action,
        "total": total,
        "datetime": datetime.now()
    }
    transactions.insert_one(transaction)
```

4. Бібліотека PyCryptodome

PyCryptodome - це бібліотека криптографічних примітивів для Python. Вона використовується в системі для шифрування та дешифрування паролів[5].

Обґрунтування вибору:

- Безпека: реалізує стандартні криптографічні алгоритми.
- Швидкість: оптимізована для швидкої роботи.
- Простота використання: має зрозумілий API.

Приклад використання:

```
def encrypt_password(password):
    key = hashlib.sha256("секретний
ключ".encode()).digest()
    cipher = AES.new(key, AES.MODE_ECB)
    encrypted_password =
cipher.encrypt(password.rjust(32).encode())
    return base64.b64encode(encrypted_password).decode()
```

5. Бібліотека openpyxl

openpyxl - це бібліотека Python для читання та запису файлів Excel (xlsx/xlsm/xltx/xltm)[6].

Обґрунтування вибору:

- Підтримка сучасних форматів Excel: працює з файлами xlsx, які є стандартом де-факто.
- Гнучкість: дозволяє створювати складні звіти з форматуванням.
- Легкість у використанні: має інтуїтивно зрозумілий API.

Приклад використання:

```
workbook = Workbook()
sheet = workbook.active
sheet.title = "Звіт"

headers = ["Номер транзакції", "Назва продукту", "Ціна за
одиницю", "Кількість/Вага", "Одиниця виміру", "Дія",
"Сума", "Дата і час"]
sheet.append(headers)
```

6. Стандартні бібліотеки Python

У проекті також широко використовуються стандартні бібліотеки Python, такі як datetime для роботи з датами та часом, os для роботи з файловою системою, sys для взаємодії з інтерпретатором Python[7].

Обґрунтування вибору:

- Надійність: стандартні бібліотеки добре протестовані та підтримуються.
- Портативність: працюють на всіх платформах, де встановлений Python.
- Ефективність: оптимізовані для швидкої роботи.

Приклад використання:

```
from datetime import datetime, date

def add_transaction(product_name, price_per_unit,
weight_or_quantity, unit, action, total):
    transaction = {
        "datetime": datetime.now()
    }
    # ...
```

```
today = date.today().isoformat()
```

Аналіз та обґрунтування вибору технологій

1. Вибір Python та PyQt5: Аналіз показав, що комбінація Python та PyQt5 забезпечує оптимальний баланс між швидкістю розробки, продуктивністю додатку та можливостями створення зручного користувацького інтерфейсу. Альтернативи, такі як Java з JavaFX або C++ з Qt, були розглянуті, але відкинуті через більшу складність розробки та меншу гнучкість.

2. Вибір MongoDB: Порівняльний аналіз з реляційними базами даних (наприклад, PostgreSQL) показав, що для даного проекту MongoDB забезпечує кращу продуктивність при роботі з неструктурованими даними та більшу гнучкість при зміні схеми даних. Це особливо важливо для системи управління продуктами, де можуть з'являтися нові типи товарів з різними атрибутами.

3. Вибір PyCryptodome: Аналіз різних криптографічних бібліотек показав, що PyCryptodome забезпечує найкращий баланс між безпекою, швидкодією та простотою використання. Альтернативи, такі як cryptography, були розглянуті, але PyCryptodome виявилася більш підходящою для потреб проекту.

4. Вибір openpyxl: Порівняння з іншими бібліотеками для роботи з Excel файлами (наприклад, xlsxwriter) показало, що openpyxl надає більше можливостей для форматування та роботи з існуючими файлами, що важливо для створення детальних звітів.

Ця комбінація технологій дозволяє швидко розробляти та легко підтримувати систему, забезпечуючи при цьому високу продуктивність, безпеку даних та можливість подальшого розширення функціональності. Вибрані технології також забезпечують крос-платформність, що дозволяє використовувати систему на різних операційних системах без необхідності значних змін у коді.

2.4. Опис структури системи та алгоритмів її функціонування

1. Логічна структура системи

Система управління продуктами складається з наступних основних компонентів:

1.1. Головне вікно (MainWindow) Це основний інтерфейс користувача, який містить наступні функціональні елементи:

- Поля введення для деталей продукту (назва, ціна, кількість/вага, одиниця виміру)
- Кнопки для здійснення операцій купівлі та продажу
- Відображення поточного бюджету
- Поле пошуку продуктів
- Список наявних продуктів
- Кнопки для доступу до вікна бюджету та адміністративної панелі

1.2. Вікно бюджету (BudgetWindow) дозволяє користувачам встановлювати та змінювати щоденний бюджет.

1.3. Адміністративна панель (AdminWindow) надає доступ до розширених функцій системи:

- Генерація звітів
- Видалення продуктів
- Зміна паролю адміністратора
- Видалення окремих транзакцій

1.4. Діалог паролю (PasswordDialog) забезпечує автентифікацію для доступу до адміністративних функцій.

1.5. Модуль бази даних (database.py) відповідає за всі операції з базою даних MongoDB, включаючи:

- Додавання та видалення транзакцій
- Оновлення інформації про продукти
- Управління бюджетом

- Пошук та фільтрацію даних

1.6. Модуль шифрування (в рамках password_dialog.py) забезпечує шифрування та дешифрування паролю адміністратора.

Зв'язки між компонентами:

- MainWindow взаємодіє з BudgetWindow та AdminWindow через сигнали та слоти Qt.
- Всі вікна взаємодіють з модулем бази даних для отримання та збереження даних.
- AdminWindow використовує PasswordDialog для автентифікації.
- Модуль шифрування використовується PasswordDialog для безпечного зберігання паролю.

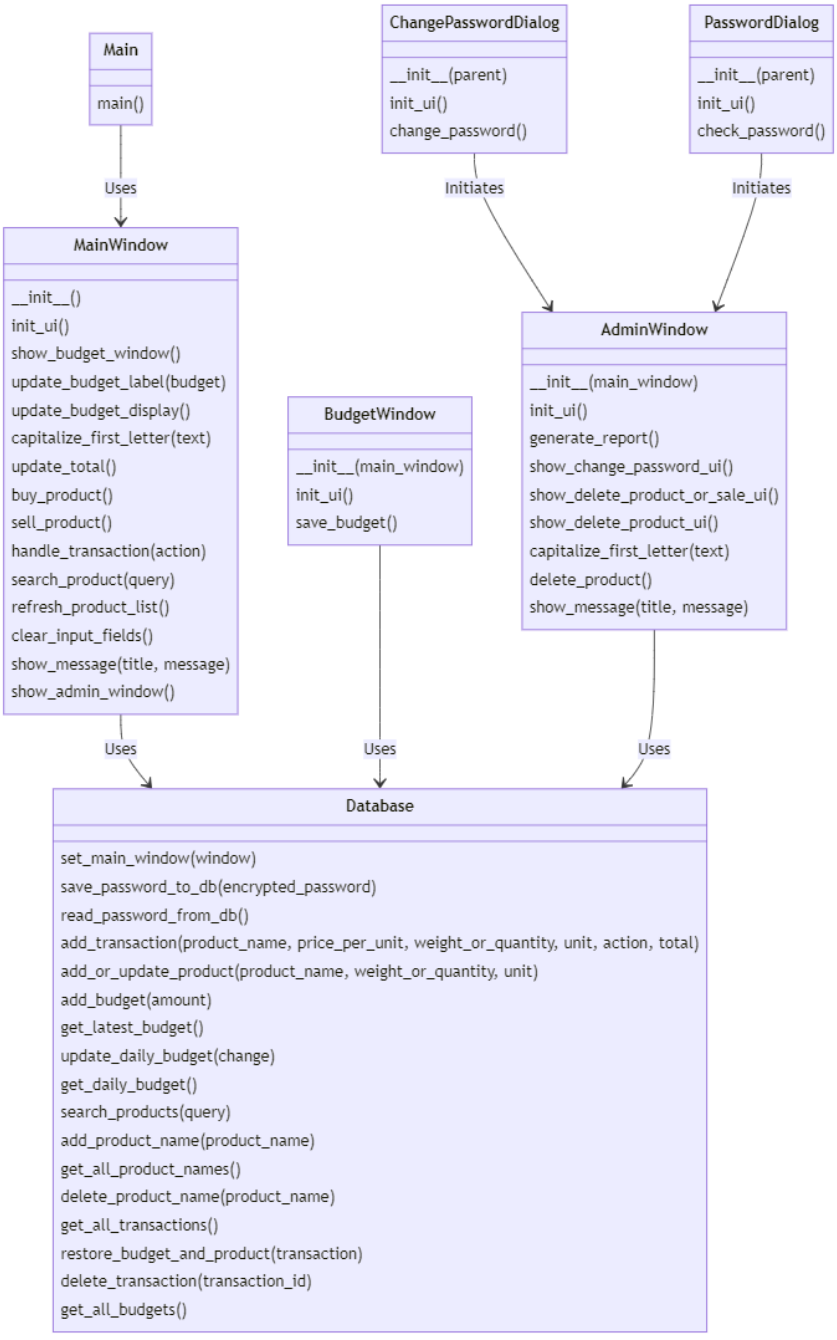

Рис. 2.1 Блок-схема структури системи

2. Алгоритми функціонування програми

2.1. Алгоритм додавання транзакції (купівля/продаж)

1. Користувач вводить дані про продукт (назва, ціна, кількість/вага, одиниця виміру).
2. Система перевіряє коректність введених даних.

3. При натисканні кнопки "Купити" або "Продати":
    - Система перевіряє наявність достатньої кількості товару (для продажу) або достатнього бюджету (для купівлі).
    - Якщо умови виконані, система створює нову транзакцію в базі даних.
    - Оновлюється інформація про кількість товару та бюджет.
    - Оновлюється відображення на головному вікні.

Обґрунтування вибору алгоритму: Цей алгоритм забезпечує послідовну та безпечну обробку транзакцій, запобігаючи помилкам, пов'язаним з недостатньою кількістю товару або перевищенням бюджету.

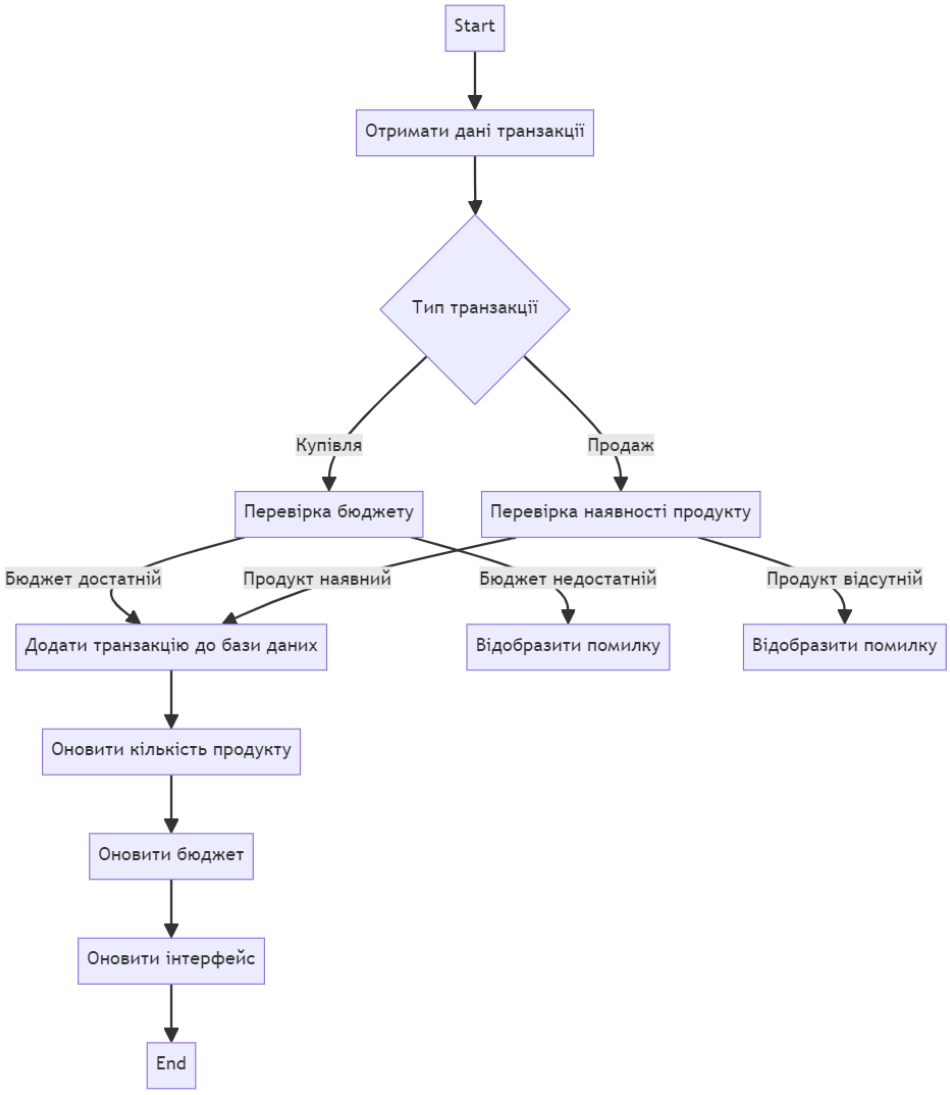

Рис. 2.2 Блок-схема алгоритма додавання транзакції (купівля/продаж)

2.2. Алгоритм пошуку продуктів

1. Користувач вводить частину назви продукту в поле пошуку.
2. Система реагує на кожну зміну тексту в полі пошуку.
3. Виконується запит до бази даних з використанням регулярного виразу для пошуку продуктів, назва яких містить введений текст.
4. Результати пошуку відображаються в списку продуктів.

Обґрунтування: Цей алгоритм забезпечує миттєвий відгук системи на дії користувача, покращуючи користувацький досвід та швидкість роботи з системою.

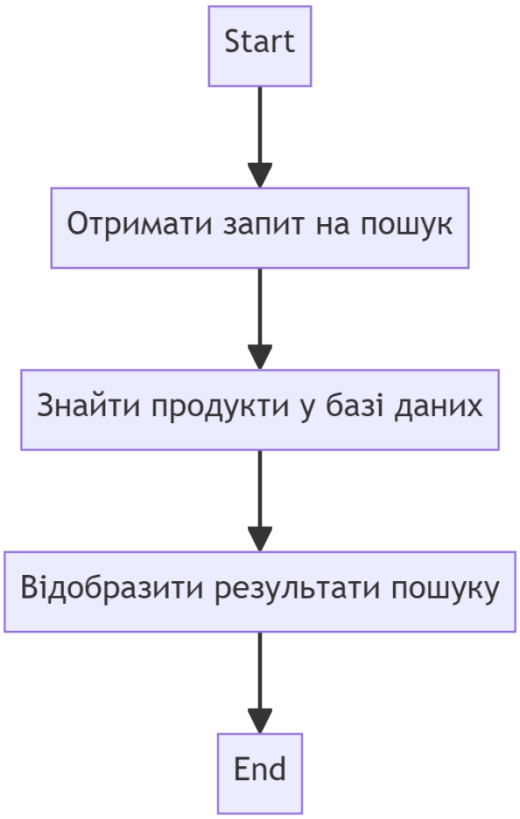

Рис. 2.3 Блок-схема алгорита пошуку продуктів

2.3. Алгоритм генерації звіту:

a. Адміністратор вводить діапазон дат для звіту.

b. Система виконує запит до бази даних для отримання всіх транзакцій за вказаний період.
c. Дані обробляються для обчислення загального прибутку та інших метрик.
d. Створюється новий Excel-файл з використанням бібліотеки openpyxl.
e. Дані записуються в файл з відповідним форматуванням.
f. Файл зберігається на диску.
g. Обґрунтування: Цей алгоритм дозволяє генерувати детальні та легко читаємі звіти, які можна далі аналізувати в Excel.

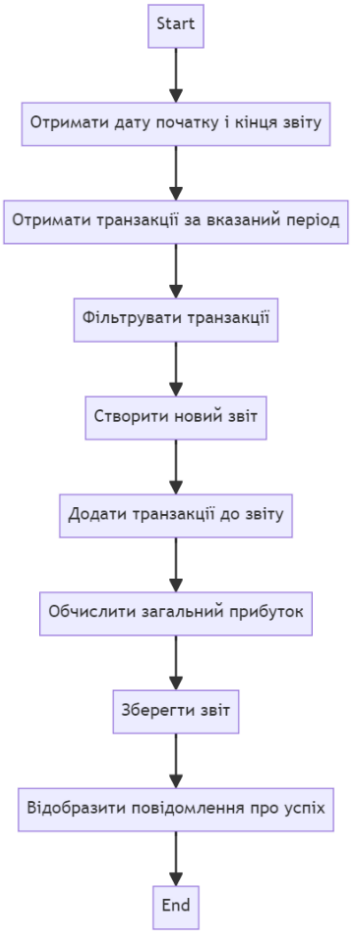

Рис. 2.4 Блок-схема алгоритма генерації звіту

2.4. Алгоритм автентифікації адміністратора

a) При спробі доступу до адміністративної панелі відкривається діалог введення паролю.
b) Введений пароль шифрується.
c) Зашифрований пароль порівнюється зі збереженим у базі даних.
d) Якщо паролі співпадають, надається доступ до адміністративної панелі.

Обґрунтування: Цей алгоритм забезпечує безпеку адміністративних функцій, використовуючи надійне шифрування для зберігання та перевірки паролю.

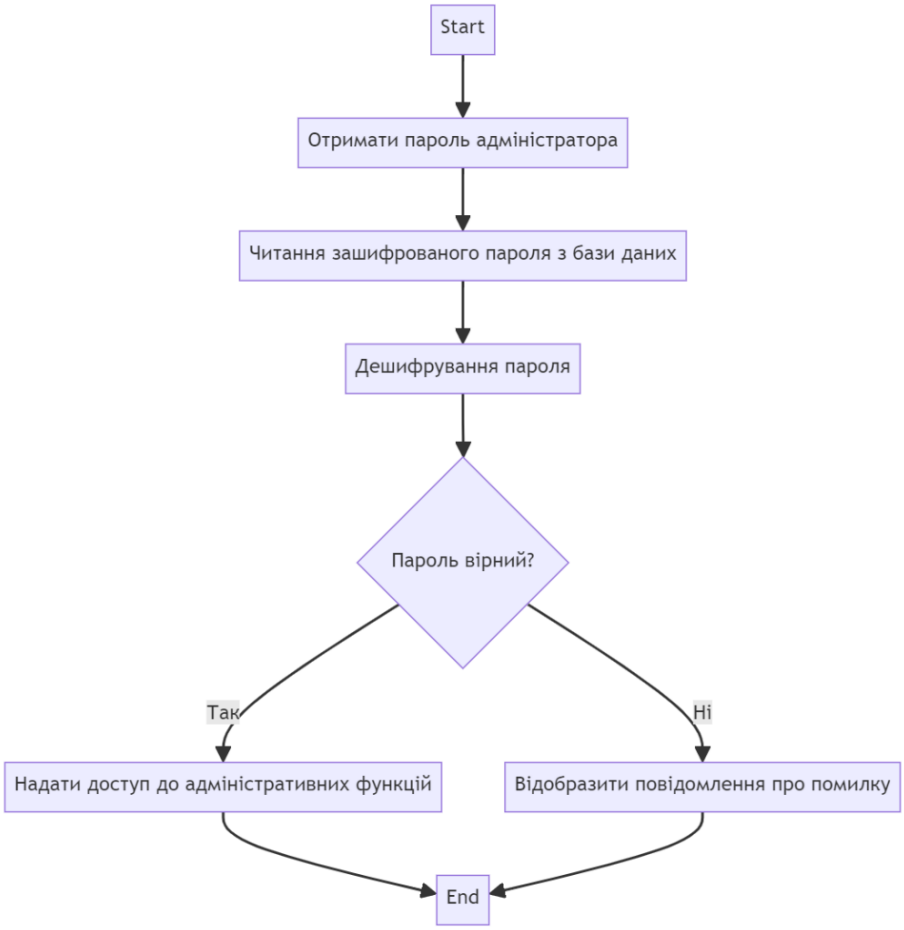

Рис. 2.5 Блок-схема алгоритма автентифікації адміністратора

3. Структура бази даних

БД MongoDB складається з наступних колекцій:

3.1. Колекція 'transactions' Зберігає інформацію про всі транзакції. Структура документа:

```
transaction = {
        "transaction_number": next_transaction_number,
        "product_name": product_name,
        "price_per_unit": price_per_unit,
        "weight_or_quantity": weight_or_quantity,
        "unit": unit,
        "action": action,
        "total": total,
        "datetime": datetime.now()
    }
```

3.2. Колекція 'budgets' Зберігає інформацію про встановлені бюджети. Структура документа:

```
budgets = {
    "_id": ObjectId,
    "budget": Float,
    "datetime": Date
}
```

3.3. Колекція 'products' Зберігає інформацію про наявні продукти. Структура документа:

```
products = {
    "_id": ObjectId,
    "product_name": String,
    "weight_or_quantity": Float,
    "unit": String
}
```

3.4. Колекція 'daily_budgets' Зберігає інформацію про щоденні бюджети. Структура документа:

```
daily_budgets = {
    "_id": ObjectId,
    "date": String,
    "budget": Float
}
```

3.5. Колекція 'product_names' Зберігає унікальні назви продуктів для автозаповнення. Структура документа:

```
product_names = {
    "_id": ObjectId,
    "product_name": String
}
```

3.6. Колекція 'passwords' Зберігає зашифрований пароль адміністратора. Структура документа:

```
passwords = {
    "_id": ObjectId,
    "password": String
}
```

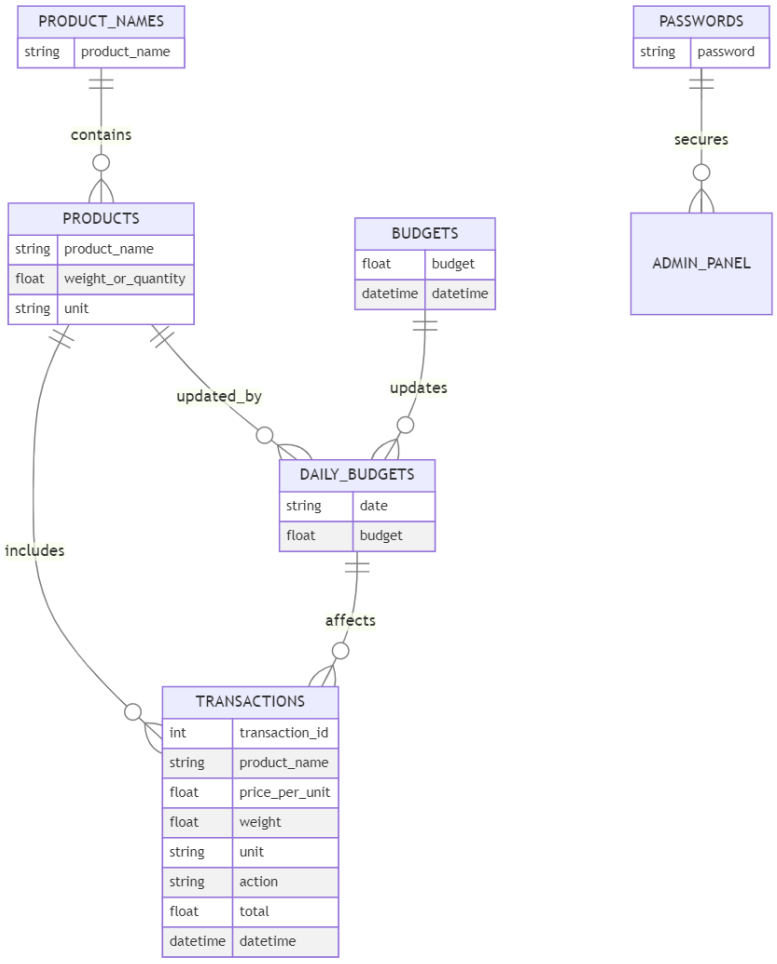

Рис. 2.6 Блок-схема структури бази даних

Обґрунтування структури бази даних:

- Використання окремих колекцій для різних типів даних забезпечує швидкий доступ та легке масштабування.
- Структура документів дозволяє зберігати всю необхідну інформацію в зручному форматі.

- Використання індексів (наприклад, за полями 'product_name' та 'datetime') оптимізує швидкість пошуку та фільтрації.

4. Зв'язок системи з іншими програмами

Система управління продуктами в основному є автономною, але має кілька точок взаємодії з зовнішніми системами:

4.1. Експорт даних в Excel Система використовує бібліотеку openpyxl для створення звітів у форматі Excel. Ці файли можуть бути відкриті та оброблені будь-якою програмою, яка підтримує формат .xlsx (наприклад, Microsoft Excel, Google Sheets, LibreOffice Calc).

4.2. Взаємодія з базою даних MongoDB Система взаємодіє з MongoDB, яка може бути встановлена локально або на віддаленому сервері. Це дозволяє, при необхідності, інтегрувати систему з іншими додатками, які використовують ту ж базу даних.

5. Детальний опис ключових алгоритмів

5.1. Алгоритм оновлення запасів при транзакції

1. Отримання даних про транзакцію (продукт, кількість, тип операції).
2. Пошук продукту в базі даних.
3. Якщо продукт знайдено:
   a) Для купівлі: додавання кількості до існуючого запасу.
   b) Для продажу:
      - Перевірка, чи достатньо запасів.
      - Якщо достатньо, віднімання кількості від існуючого запасу.
4. Якщо продукт не знайдено (тільки для купівлі):
      - Створення нового запису продукту.
5. Оновлення запису в базі даних.
6. Оновлення відображення в головному вікні.

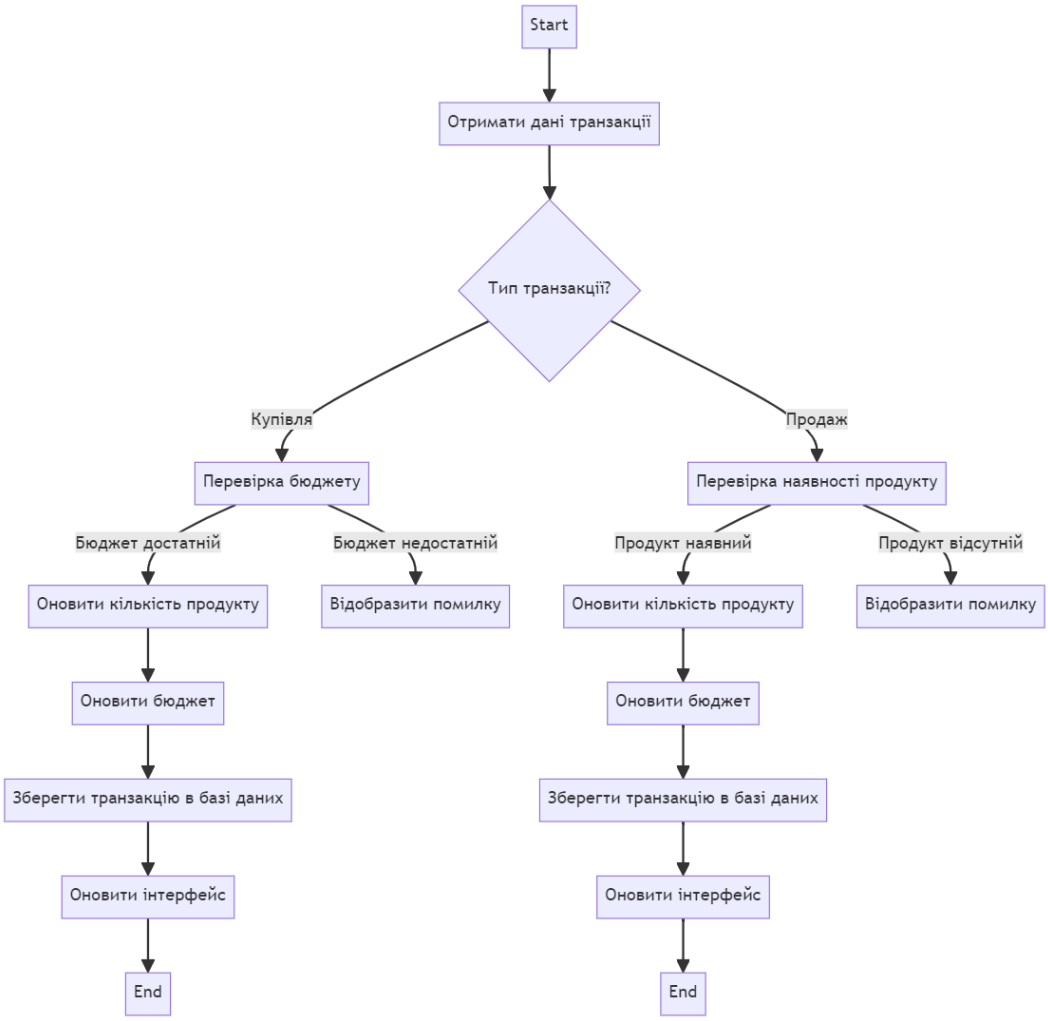

Рис. 2.7 Блок-схема алгоритма оновлення запасів при транзакції

Цей алгоритм забезпечує точне відстеження запасів, запобігаючи від'ємним значенням кількості товару та автоматично додаючи нові продукти при необхідності.

5.2. Алгоритм розрахунку щоденного бюджету

1. Отримання поточної дати.
2. Пошук запису бюджету для поточної дати в колекції 'daily_budgets'.
3. Якщо запис знайдено:
    - Повернення значення бюджету.
4. Якщо запис не знайдено:

- Отримання останнього встановленого бюджету з колекції 'budgets'.
- Створення нового запису в 'daily_budgets' з цим значенням.
- Повернення значення бюджету.

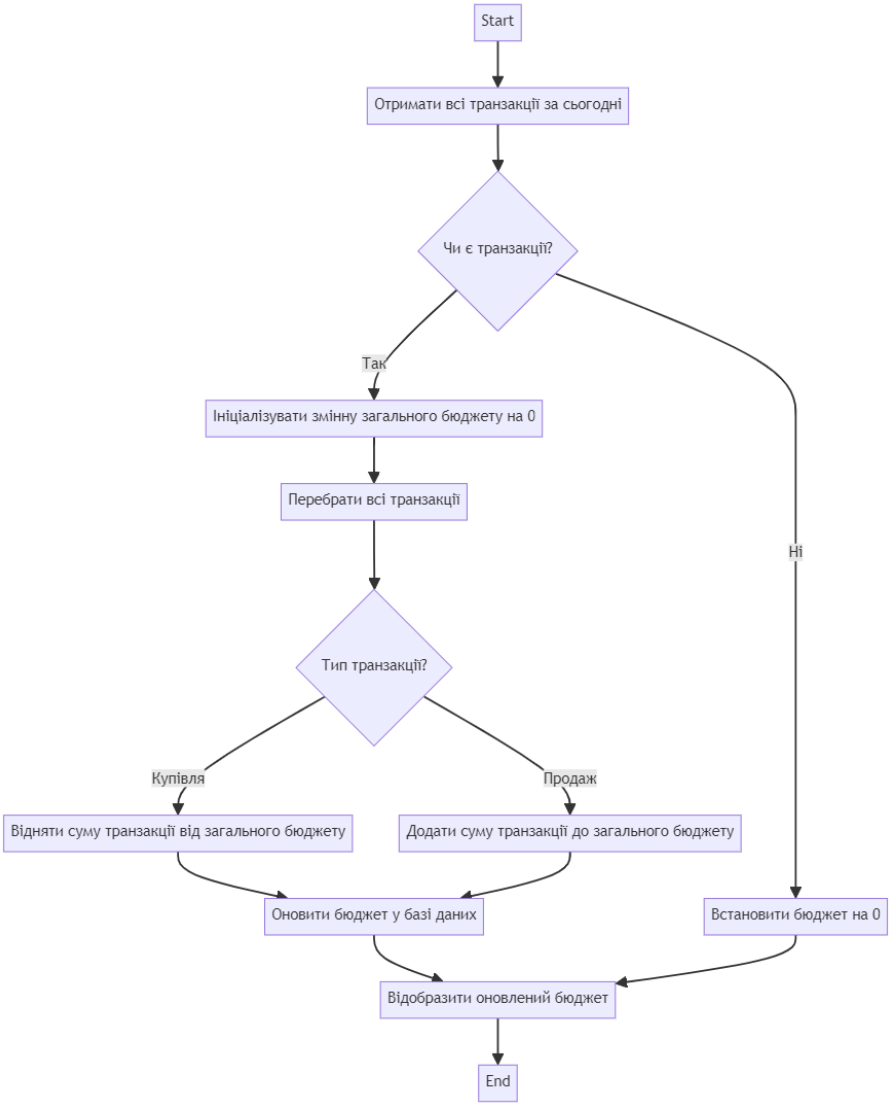

Рис. 2.8 Блок-схема алгоритма розрахунку щоденного бюджету

Цей алгоритм дозволяє гнучко керувати бюджетом, автоматично встановлюючи щоденний бюджет на основі останнього введеного значення.

5.3. Алгоритм генерації звіту про прибуток

5. Отримання діапазону дат від користувача.

6. Запит до бази даних для отримання всіх транзакцій у вказаному діапазоні.
7. Ініціалізація змінних для загального доходу та витрат.
8. Для кожної транзакції:
    - Якщо це продаж, додавання суми до доходу.
    - Якщо це купівля, додавання суми до витрат.
9. Розрахунок прибутку як різниці між доходом та витратами.
10. Створення нового аркуша Excel.
11. Запис заголовків та даних про транзакції.
12. Додавання підсумкової інформації (загальний дохід, витрати, прибуток).
13. Форматування аркуша (кольори, шрифти, ширина стовпців).
14. Збереження файлу.

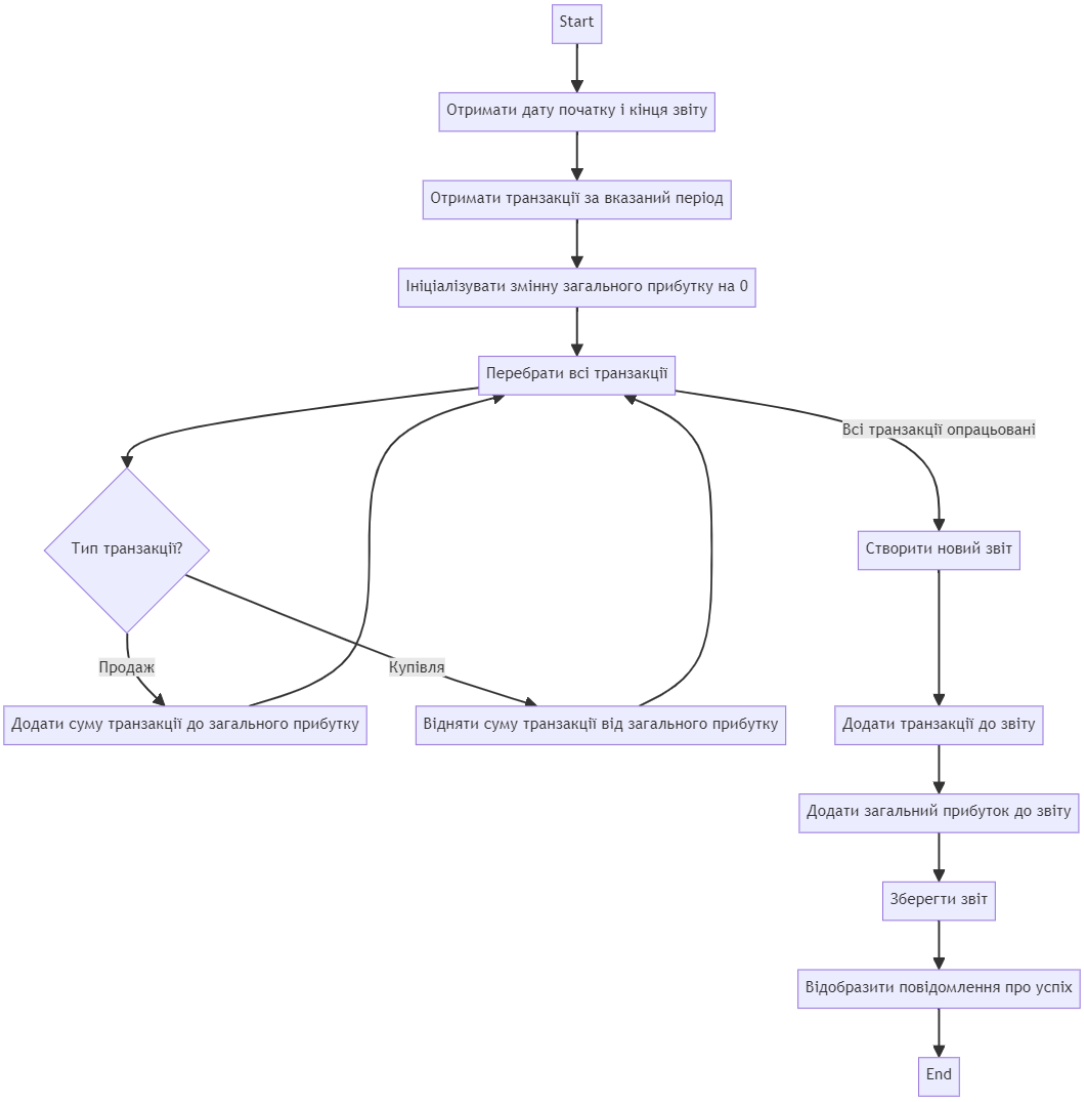

Рис. 2.9 Блок-схема алгоритма генерації звіту про прибуток

Цей алгоритм забезпечує створення детального та інформативного звіту, який допомагає в аналізі фінансової діяльності.

5.4. Алгоритм автодоповнення при введенні назви продукту

1. Отримання частини назви продукту, введеної користувачем.
2. Запит до колекції 'product_names' для отримання всіх назв, що починаються з введеного тексту.
3. Сортування отриманих назв за алфавітом.
4. Відображення списку можливих варіантів завершення назви.

5. При виборі користувачем варіанту зі списку, автоматичне заповнення поля повною назвою.

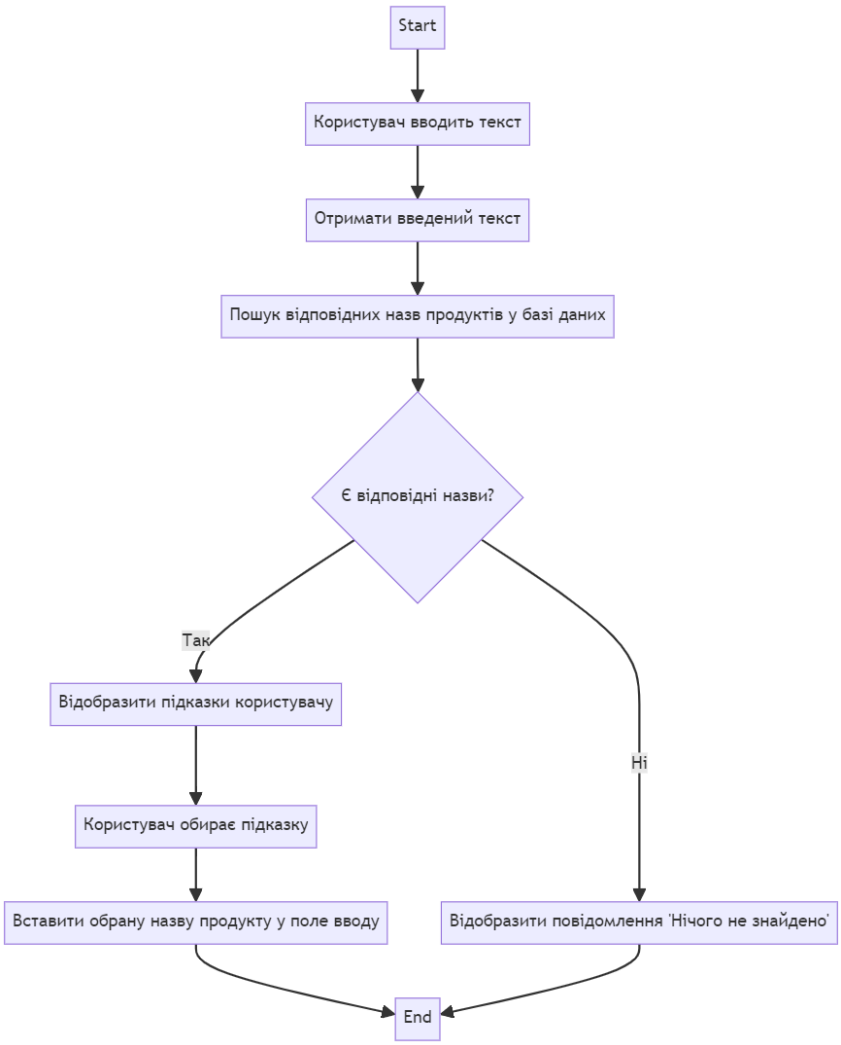

Рис. 2.10 Блок-схема алгоритма автодоповнення при введенні назви продукту

Цей алгоритм покращує користувацький досвід, прискорюючи введення даних та зменшуючи ймовірність помилок при введенні назв продуктів.

6. Механізми забезпечення цілісності даних

6.1. Транзакційність операцій Всі операції з базою даних, які змінюють стан системи (наприклад, додавання транзакції та оновлення запасів), виконуються в

рамках однієї атомарної операції. Це забезпечує, що у випадку помилки система залишиться в узгодженому стані.

6.2. Валідація вхідних даних Перед виконанням будь-яких операцій з базою даних, система перевіряє коректність вхідних даних. Наприклад, перевіряється, чи є ціна та кількість додатними числами, чи не перевищує сума покупки поточний бюджет.

6.3. Обробка виключних ситуацій Система включає механізми обробки виключень на всіх рівнях. Це дозволяє грацерним чином обробляти помилки, такі як втрата з'єднання з базою даних чи некоректні дані, не допускаючи краху програми.

6.4. Регулярне резервне копіювання Хоча це не є частиною самого програмного коду, система передбачає можливість регулярного резервного копіювання бази даних для захисту від втрати даних.

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Організація вхідних та вихідних даних у програмі "Product Management System" спроектована таким чином, щоб забезпечити максимальну зручність для користувачів, мінімізувати можливість помилок при введенні даних та надати широкі можливості для аналізу та звітності.

Використання стандартних форматів даних (UTF-8, XLSX,) забезпечує сумісність з іншими системами та можливість подальшого розширення функціональності програми. Механізми валідації та забезпечення цілісності даних гарантують надійність та точність інформації в системі.

Подальший розвиток системи може включати розширення можливостей інтеграції з іншими системами, впровадження додаткових методів збору даних (наприклад, сканування штрих-кодів) та розробку більш складних аналітичних інструментів.

1. Характер, організація і попередня підготовка вхідних даних

1.1. Загальна характеристика вхідних даних

Програма "Product Management System" призначена для управління товарами, транзакціями та бюджетом. Вхідні дані для цієї системи можна розділити на кілька категорій:

- Дані про товари
- Дані про транзакції (купівля/продаж)
- Дані про бюджет d) Адміністративні дані

1.2. Організація введення даних

Введення даних у систему здійснюється користувачем в діалоговому режимі через графічний інтерфейс користувача (GUI). Це забезпечує зручність та інтуїтивність при роботі з програмою. Для кожного типу даних передбачені відповідні форми введення з необхідними полями.

1.3. Попередня підготовка вхідних даних

Перед введенням даних у систему користувач повинен виконати наступні кроки підготовки:

- Збір інформації про товари (назва, ціна, кількість, одиниця виміру)
- Підготовка даних про транзакції (дата, тип операції, сума)
- Визначення бюджету
- Підготовка адміністративної інформації (паролі, права доступу)

1.4. Детальний опис вхідних даних

a) Дані про товари:

- Назва товару (текстове поле)
- Ціна за одиницю (числове поле з плаваючою комою)
- Кількість або вага (числове поле з плаваючою комою)
- Одиниця виміру (вибір зі списку: кг або шт)

b) Дані про транзакції:

- Тип операції (вибір: купівля або продаж)
- Назва товару (вибір зі списку існуючих товарів)
- Кількість або вага (числове поле з плаваючою комою)
- Ціна за одиницю (числове поле з плаваючою комою)

c) Дані про бюджет:

- Сума бюджету (числове поле з плаваючою комою)

d) Адміністративні дані:

- Пароль адміністратора (текстове поле)

1.5. Формат та кодування вхідних даних

Всі текстові дані вводяться у форматі UTF-8 для забезпечення підтримки різних мов та символів. Числові дані вводяться у десятковому форматі з використанням крапки як роздільника десяткових знаків. Дати вводяться у форматі YYYY-MM-DD.

2. Характер і організація вихідних даних

2.1. Загальна характеристика вихідних даних

Вихідні дані програми "Product Management System" включають:

- Інформацію про поточний стан товарів
- Звіти про транзакції
- Інформацію про бюджет
- Аналітичні дані та звіти

2.2. Організація виведення даних

Виведення даних здійснюється через графічний інтерфейс користувача та у вигляді експортованих файлів. Основні способи виведення даних:

- Відображення на екрані (в реальному часі)
- Генерація звітів у форматі Excel
- Виведення повідомлень про результати операцій

2.3. Детальний опис вихідних даних

a) Інформація про поточний стан товарів:

- Список товарів з їх кількістю та ціною
- Загальна вартість товарів на складі

b) Звіти про транзакції:

- Список всіх транзакцій за вибраний період
- Сумарна інформація про купівлі та продажі

c) Інформація про бюджет:

- Поточний стан бюджету
- Історія змін бюджету

d) Аналітичні дані та звіти:

- Прибуток за вибраний період
- Найбільш продавані товари
- Динаміка зміни цін на товари

2.4. Формат та кодування вихідних даних

Вихідні дані представляються у наступних форматах:

- Текстові дані на екрані: UTF-8
- Числові дані на екрані: десятковий формат з двома знаками після коми
- Дати на екрані: формат DD.MM.YYYY
- Експортовані звіти: формат XLSX (Microsoft Excel)

3. Обґрунтування вибору формату вхідних та вихідних даних

3.1. Вхідні дані

Вибір формату вхідних даних обґрунтований наступними факторами:

- Простота введення: використання текстових полів та випадаючих списків забезпечує інтуїтивно зрозумілий інтерфейс для користувача.
- Мінімізація помилок: використання випадаючих списків для вибору одиниць виміру та типу операції зменшує ймовірність помилок при введенні.
- Універсальність: використання UTF-8 кодування дозволяє вводити дані різними мовами без проблем з відображенням символів.
- Точність: використання чисел з плаваючою комою для цін та кількості забезпечує необхідну точність для фінансових розрахунків.

3.2. Вихідні дані

Вибір формату вихідних даних обґрунтований наступними факторами:

- Читабельність: відображення даних на екрані у зрозумілому для користувача форматі (наприклад, використання роздільників тисяч у великих числах).
- Сумісність: використання формату XLSX для експорту звітів забезпечує сумісність з широко розповсюдженим програмним забезпеченням Microsoft Excel.
- Аналітичні можливості: формат XLSX дозволяє користувачам легко аналізувати дані, створювати графіки та діаграми.
- Універсальність: UTF-8 кодування забезпечує коректне відображення даних на різних пристроях та в різних операційних системах.

4. Способи збору та підготовки даних

4.1. Збір даних про товари

Дані про товари збираються наступним чином:

- Ручне введення інформації про нові товари через інтерфейс програми.
- Імпорт даних з електронних таблиць (наприклад, Excel файлів) для масового додавання товарів.
- Сканування штрих-кодів товарів (для майбутніх версій програми).

4.2. Збір даних про транзакції

Дані про транзакції збираються в режимі реального часу при здійсненні операцій купівлі або продажу:

- Вибір товару зі списку наявних товарів.
- Введення кількості товару та ціни (якщо вона відрізняється від стандартної).
- Автоматичне збереження дати та часу транзакції.

4.3. Збір даних про бюджет

Дані про бюджет вводяться вручну адміністратором системи:

- Введення початкового бюджету при першому запуску програми.
- Регулярне оновлення бюджету (наприклад, щомісяця) через інтерфейс адміністратора.

4.4. Підготовка даних для звітів

Для генерації звітів система виконує наступні кроки підготовки даних:

- Агрегація даних про транзакції за вибраний період.
- Розрахунок сумарних показників (загальна сума продажів, прибуток тощо).
- Сортування та фільтрація даних відповідно до параметрів звіту.

5. Забезпечення якості та цілісності даних

5.1. Валідація вхідних даних

Для забезпечення якості вхідних даних програма використовує наступні методи валідації:

- Перевірка типів даних (наприклад, чи є введене значення числом).
- Перевірка діапазонів значень (наприклад, чи не є ціна від'ємною).
- Перевірка унікальності (наприклад, чи не існує вже товар з такою назвою).
- Перевірка обов'язкових полів (чи заповнені всі необхідні поля).

5.2. Забезпечення цілісності даних

Для забезпечення цілісності даних програма використовує наступні механізми:

- Транзакційність операцій в базі даних.
- Обмеження доступу до даних через систему аутентифікації та авторизації.

6. Інтеграція з іншими системами

6.1. Експорт даних

Система підтримує експорт Excel файлів для забезпечення швидкого підрахунку витрат та прибутку, і для зручного ведення обліку прийому та продаж металургійних товарів.

2.6. Опис розробленої системи

Розроблена система управління прийому та продажу металу - це комплексне програмне рішення, призначене для ефективного контролю за товарообігом та фінансами малого бізнесу або домашнього господарства. Система надає користувачам зручний інтерфейс для відстеження покупок, продажів, управління запасами та контролю бюджету.

Основні функції системи включають:

1. Реєстрація покупок та продажів: користувачі можуть легко вводити інформацію про придбані або продані товари, включаючи назву продукту, ціну, кількість та одиницю виміру.
2. Управління запасами: система автоматично оновлює інформацію про наявність товарів на складі після кожної транзакції.
3. Бюджетний контроль: користувачі можуть встановлювати денний бюджет та відстежувати його використання. Система попереджає, якщо покупка перевищує доступний бюджет.
4. Пошук продуктів: реалізовано функцію швидкого пошуку товарів за назвою.
5. Адміністративний модуль: включає функції для генерації звітів, видалення продуктів з бази даних, зміни паролю доступу та видалення транзакцій.
6. Звітність: система дозволяє генерувати детальні звіти про транзакції та стан бюджету за вибраний період.

Інтерфейс системи розроблено з урахуванням принципів user-friendly дизайну, що забезпечує інтуїтивно зрозуміле використання навіть для користувачів без спеціальної підготовки. Всі елементи управління логічно розташовані та мають чіткі позначення.

Система працює з локальною базою даних, що забезпечує швидкий доступ до інформації та можливість роботи без підключення до інтернету. Це робить її особливо зручною для використання в малих магазинах або домашніх умовах.

Безпека даних забезпечується системою паролів, при цьому паролі зберігаються в зашифрованому вигляді, що захищає від несанкціонованого доступу.

Розроблена система є гнучкою та може бути легко адаптована під специфічні потреби користувача. Вона може бути корисною для широкого кола користувачів: від власників малого бізнесу до людей, які хочуть ефективно керувати своїм домашнім бюджетом.

2.6.1. Використані технічні засоби

Конфігурація технічних засобів в цій інформаційній автоматичній системі забезпечує стабільну роботу системи та можливість її подальшого розвитку. Вимоги до обладнання є помірними, що робить систему доступною для широкого кола користувачів. Водночас, більш потужне обладнання може покращити швидкодію системи, особливо при роботі з великими обсягами даних. Для розробки та функціонування системи управління прийому та продажу металу були використані наступні технічні засоби:

1. Комп'ютерне обладнання:
    - Персональний комп'ютер або ноутбук з процесором Intel Core i5 або аналогічним AMD Ryzen 5 (або вище)
    - Оперативна пам'ять: мінімум 8 ГБ RAM
    - Жорсткий диск: мінімум 1 ТБ (рекомендовано SSD для швидкої роботи з базою даних)
    - Монітор з роздільною здатністю не менше 1920x1080 для комфортної роботи з інтерфейсом
2. Операційна система:
    - Windows 10 або новіша версія
    - macOS Catalina (10.15) або новіша версія
    - Linux (Ubuntu 20.04 LTS або аналогічний дистрибутив)

3. Програмне забезпечення для розробки:
    - Python 3.8 або новіша версія
    - PyQt5 для створення графічного інтерфейсу
    - MongoDB для управління базою даних
    - PyMongo для взаємодії Python з MongoDB
    - Openpyxl для генерації звітів у форматі Excel
    - PyCryptodome для шифрування паролів
4. Середовище розробки:
    - PyCharm Professional або Community Edition
    - Visual Studio Code з відповідними розширеннями для Python
5. Система контролю версій:
    - Git для управління версіями коду
6. Додаткове обладнання:
    - Принтер для друку звітів (опціонально)
    - Сканер штрих-кодів для швидкого введення інформації про товари (опціонально, для майбутніх розширень системи)
7. Мережеве обладнання:
    - Локальна мережа або Wi-Fi для майбутньої можливості багатокористувацького доступу
    - Маршрутизатор для забезпечення мережевого з'єднання
8. Резервне копіювання:
    - Зовнішній жорсткий диск або хмарне сховище для регулярного резервного копіювання бази даних

Варто зазначити, що система розроблена з урахуванням можливості майбутнього масштабування. При необхідності обробки більших обсягів даних або збільшення кількості одночасних користувачів, може знадобитися більш потужне серверне обладнання та оптимізація бази даних.

2.6.2. Використані програмні засоби

Всі використані у створені програмні засоби працюють разом, забезпечуючи надійне та ефективне функціонування системи. Вибір кожного компонента був зроблений з урахуванням його стабільності, продуктивності та сумісності з іншими елементами системи.

Для забезпечення функціонування розробленої системи прийрму та продажу металу були використані наступні програмні засоби:

1. Операційна система: Система розроблена з можливістю функціонування на різних операційних системах, включаючи:
    - Windows 10 або новіша версія
    - macOS Catalina (10.15) або новіша версія
    - Linux (Ubuntu 20.04 LTS або аналогічні дистрибутиви)

Це забезпечує гнучкість у виборі платформи для кінцевого користувача.

2. Python: Версія 3.8 або новіша. Python є основною мовою програмування, на якій розроблено систему. Вибір Python обумовлений його простотою, потужністю та великою кількістю доступних бібліотек.

3. PyQt5: Бібліотека для створення графічного інтерфейсу користувача. PyQt5 забезпечує створення нативних віджетів для різних операційних систем, що покращує користувацький досвід.

4. MongoDB: Версія 4.4 або новіша. NoSQL БД, яка використовується для зберігання інформації про продукти, транзакції та бюджет. MongoDB обрана через її гнучкість у роботі з неструктурованими даними та високу продуктивність.

5. PyMongo: Бібліотека для взаємодії Python з MongoDB. PyMongo дозволяє ефективно виконувати операції з базою даних безпосередньо з Python-коду.

6. Openpyxl: Бібліотека для роботи з Excel-файлами. Використовується для генерації звітів у форматі .xlsx, що забезпечує зручність аналізу даних користувачами.

7. PyCryptodome: Бібліотека для криптографічних операцій. Використовується для шифрування паролів адміністратора, що підвищує безпеку системи.

8. Pip: Система управління пакетами Python. Використовується для встановлення та управління залежностями проекту.

9. Віртуальне середовище Python (venv): Використовується для створення ізольованого середовища Python, що дозволяє уникнути конфліктів між залежностями різних проектів.

10. JSON: Формат даних, який використовується для серіалізації та десеріалізації даних при обміні інформацією між різними компонентами системи.

11. Datetime: Вбудований модуль Python для роботи з датами та часом. Використовується для точного відстеження часу транзакцій та генерації звітів.

12. OS: Вбудований модуль Python для взаємодії з операційною системою. Використовується для операцій з файловою системою, наприклад, при збереженні звітів.

13. Sys: Вбудований модуль Python, який надає доступ до деяких змінних та функцій, що взаємодіють з інтерпретатором Python. Використовується для управління аргументами командного рядка та виходу з програми.

Важливо відзначити, що всі використані програмні засоби є відкритими та безкоштовними, що знижує загальну вартість розробки та впровадження системи. Крім того, активна спільнота розробників навколо цих технологій забезпечує постійну підтримку та оновлення, що гарантує довгострокову життєздатність розробленої системи.

2.6.3. Виклик та завантаження програми

Розроблена система управління прийому та продажу металу є настільним додатком, який може бути встановлений та запущений на персональному комп'ютері користувача. Процес виклику та завантаження програми складається з декількох етапів, які забезпечують коректну ініціалізацію всіх компонентів системи та підготовку робочого середовища.

1. Інсталяція: Перед першим запуском програми необхідно провести її інсталяцію. Система поставляється у вигляді інсталяційного пакету, який містить всі необхідні компоненти та залежності. Процес інсталяції включає наступні кроки: a) Запуск інсталятора (setup.exe для Windows або .pkg для macOS). b) Вибір директорії для встановлення програми. c) Встановлення необхідних компонентів, включаючи Python, MongoDB та всі потрібні бібліотеки. d) Створення ярликів на робочому столі та в меню "Пуск" (для Windows). Після завершення інсталяції система готова до першого запуску.

2. Запуск програми: Виклик програми може здійснюватися кількома способами:

a) Через ярлик на робочому столі (подвійний клік).

b) Через меню "Пуск" (для Windows) або Launchpad (для macOS).

c) Запуском виконуваного файлу з директорії встановлення.

d) Через командний рядок, перейшовши до директорії з програмою та виконавши команду:

```
python main.py
```

3. Процес завантаження: При запуску програми відбуваються наступні процеси: a) Ініціалізація Python-інтерпретатора. b) Завантаження основного модуля програми (main.py). c) Імпорт необхідних бібліотек та модулів. d) Встановлення з'єднання з базою даних MongoDB. e) Ініціалізація графічного інтерфейсу користувача (GUI). f) Завантаження збережених налаштувань та даних користувача. Цей процес зазвичай займає від 2 до 5 секунд, залежно від потужності комп'ютера та швидкості жорсткого диска.

4. Використання оперативної пам'яті: Система розроблена з урахуванням ефективного використання ресурсів комп'ютера. Вимоги до оперативної пам'яті є наступними:

- Мінімальний обсяг: 256 МБ
- Рекомендований обсяг: 512 МБ
- Оптимальний обсяг: 1 ГБ

При запуску програма займає близько 100-150 МБ оперативної пам'яті. Цей обсяг може збільшуватися до 200-300 МБ при активній роботі з великими обсягами даних або при генерації складних звітів.

5. Обсяг програми: Загальний обсяг встановленої програми на жорсткому диску складає приблизно:

- Основні програмні файли: 50-70 МБ
- БД MongoDB (початковий розмір): 100-150 МБ
- Додаткові бібліотеки та залежності: 200-250 МБ

Загальний обсяг може варіюватися від 350 до 500 МБ, залежно від конфігурації та обсягу даних користувача. Варто зазначити, що розмір бази даних буде збільшуватися з часом, в залежності від кількості збережених транзакцій та продуктів.

6. Умови завантаження: Для коректного завантаження та функціонування програми необхідно забезпечити наступні умови: a) Наявність встановленої операційної системи (Windows 10+, macOS 10.15+, або Linux). b) Достатній обсяг вільного місця на жорсткому диску (мінімум 1 ГБ). c) Наявність прав адміністратора для першого запуску та налаштування. d) Активний антивірус не повинен блокувати роботу програми або доступ до бази даних. e) Відкриті необхідні порти для роботи MongoDB (за замовчуванням 27017).

7. Оновлення програми: Система підтримує автоматичне оновлення. При наявності підключення до інтернету, програма при запуску перевіряє наявність нових версій. Якщо оновлення доступне, користувачу пропонується

його встановити. Процес оновлення відбувається автоматично і зазвичай не вимагає перезапуску програми.

8. Завершення роботи: При закритті програми відбувається коректне завершення всіх процесів: a) Збереження незбережених даних. b) Закриття з'єднання з базою даних. c) Вивільнення зайнятої оперативної пам'яті. d) Збереження налаштувань користувача.

Важливо зазначити, що система розроблена з урахуванням можливості роботи на комп'ютерах з різною конфігурацією. У випадку недостатньої кількості оперативної пам'яті або повільного процесора, програма може працювати дещо повільніше, але залишається функціональною. Для оптимальної роботи рекомендується використовувати комп'ютер, який відповідає або перевищує мінімальні системні вимоги.

Завдяки використанню Python та крос-платформних бібліотек, програма може бути легко адаптована для роботи на різних операційних системах без значних змін у коді. Це забезпечує гнучкість у виборі робочого середовища для кінцевого користувача.

2.6.4. Опис інтерфейсу користувача

Інтерфейс користувача розробленої інформаційної системи для прийому та продажу металу складається з кількох основних вікон та діалогів, які забезпечують зручну та ефективну роботу з програмою.

1. Головне вікно (MainWindow)

Головне вікно програми є центральним елементом інтерфейсу, через яке здійснюється більшість операцій. Воно містить наступні елементи:

а) Поля вводу:

- "Назва продукту" - для введення назви металу
- "Ціна за одиницю" - для введення ціни за кілограм або штуку

- "Вага або кількість" - для введення ваги або кількості металу
- "кг/шт" - випадаючий список для вибору одиниці виміру

b) Кнопки:

- "Купити" - для реєстрації покупки металу
- "Продати" - для реєстрації продажу металу
- "Бюджет" - для переходу до вікна управління бюджетом
- "Адмін" - для переходу до адміністративної панелі

c) Інформаційні поля:

- "Сума" - відображає загальну суму операції
- "Бюджет" - показує поточний бюджет

d) Поле пошуку та список продуктів:

- Поле для введення пошукового запиту
- Список наявних продуктів, який оновлюється при пошуку

Управління діалогом:

2. Користувач вводить дані про метал у відповідні поля.
3. При зміні ціни або кількості автоматично оновлюється сума операції.
4. Кнопки "Купити" і "Продати" стають активними після заповнення всіх необхідних полів.
5. При натисканні на "Купити" або "Продати" виконується відповідна операція, оновлюється список продуктів та бюджет.
6. Поле пошуку дозволяє швидко знайти потрібний метал у списку.
7. Вікно управління бюджетом (BudgetWindow)

Це вікно відкривається при натисканні кнопки "Бюджет" і містить:

- Поле для введення нового значення бюджету
- Кнопку "Зберегти" для підтвердження зміни бюджету
- Кнопку "Назад" для повернення до головного вікна

Управління діалогом:

- Користувач вводить нове значення бюджету.

- При натисканні "Зберегти" бюджет оновлюється, і зміни відображаються в головному вікні.

8. Адміністративна панель (AdminWindow)

Доступ до цього вікна захищений паролем. Воно містить наступні елементи:

- Поля для вибору дат початку і кінця періоду для звіту
- Кнопки для різних адміністративних функцій:
- "Звіт" - для генерації звіту за вибраний період
- "Видалення продуктів" - для видалення продуктів з бази даних
- "Заміна ключа доступу" - для зміни адміністративного пароля
- "Видалення покупок чи продажу" - для видалення окремих транзакцій
- "Назад" - для повернення до головного вікна

Управління діалогом:

- Адміністратор вводить пароль для доступу до панелі.
- Вибирає потрібну функцію, натискаючи відповідну кнопку.
- Для генерації звіту вибирає період і натискає "Звіт".
- Для видалення продукту або транзакції використовує відповідні діалогові вікна.

Ілюстрації результатів роботи:

1. Головне вікно з заповненими полями та списком продуктів

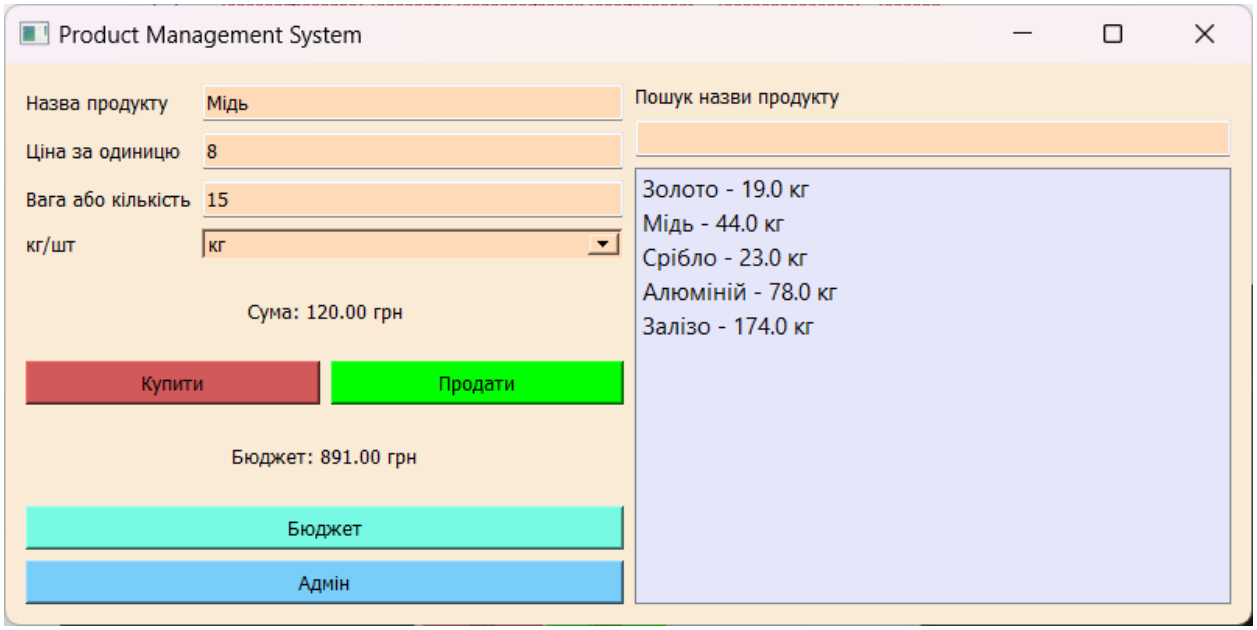

Рис. 2.11 Інтерфейс головного вікна

2. Вікно управління бюджетом

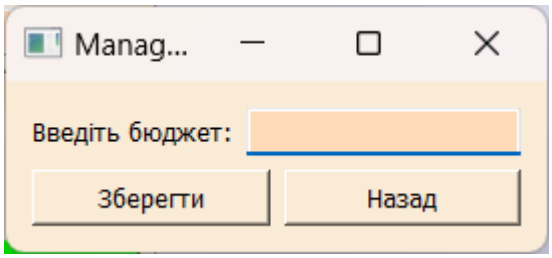

Рис. 2.12 вікно управління бюджетом

3. Вікно адміністратора

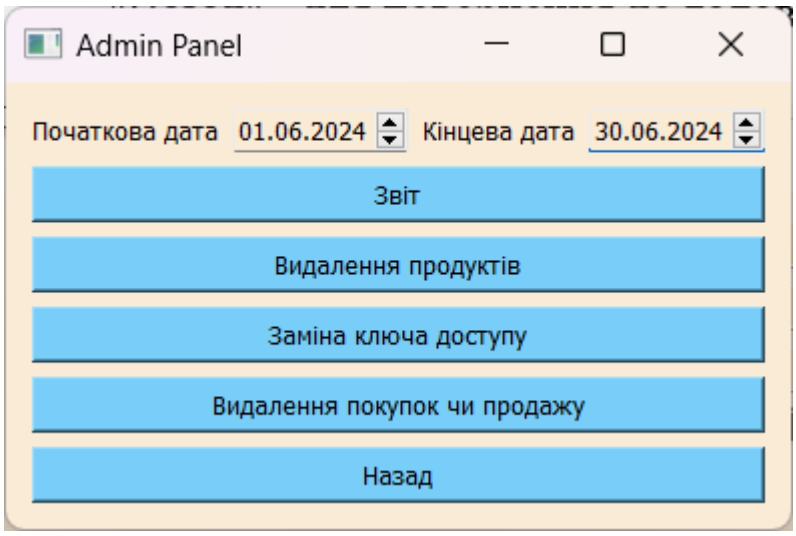

Рис. 2.13 вікно адміністратора

4. Діалогове вікно видалення продукту

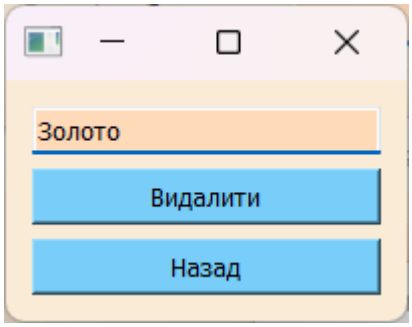

Рис. 2.14 діалогове вікно видалення продукту

5. Приклад згенерованого звіту

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Номер тра	Назва про	Ціна за од	Кількість/І	Одиниця	Дія	Сума	Дата і час				Бюджет	Дата	
2	1	Картошка	12	4	кг	Куплено	48	2024-06-13	03:16:50			1000	2024-06-08	
3	4	Мідь	11	1	кг	Куплено	11	2024-06-13	03:22:18			100	2024-06-08	
4	5	Мідь	1	1	кг	Куплено	1	2024-06-13	03:22:49			1	2024-06-08	
5	7	Крісло	100	3	шт	Куплено	300	2024-06-17	13:55:04			1001	2024-06-09	
6	8	Молока	10	7	кг	Куплено	70	2024-06-17	13:58:08			1012	2024-06-09	
7	9	Кермо	2	100	шт	Куплено	200	2024-06-17	14:02:10			1022	2024-06-09	
8	10	Кермо	100	3	шт	Куплено	300	2024-06-17	14:02:38			1033	2024-06-09	
9	11	Срібло	23	1	шт	Куплено	23	2024-06-17	14:03:03			1044	2024-06-09	
10	12	Срібло	10	3	кг	Куплено	30	2024-06-17	14:04:31			1045	2024-06-09	
11	13	Срібло	11	0,5	кг	Куплено	5,5	2024-06-17	14:17:33			56	2024-06-09	
12	14	Срібло	12	1	кг	Куплено	12	2024-06-17	14:40:56			1256	2024-06-10	
13	15	Срібло	12	1	кг	Куплено	12	2024-06-17	14:43:54			1356	2024-06-10	
14	16	Срібло	15	1	кг	Куплено	15	2024-06-17	14:47:08			100	2024-06-10	
15	17	Мідь	12	3	кг	Куплено	36	2024-06-17	14:47:28			300	2024-06-13	
16	19	Срібло	12	3	кг	Куплено	36	2024-06-17	14:51:17			300	2024-06-13	
17	20	Срібло	12	1	кг	Куплено	12	2024-06-17	14:53:55			111	2024-06-15	
18	21	Срібло	12	1	кг	Куплено	12	2024-06-17	14:54:13			1000	2024-06-17	
19	22	Срібло	12	1	кг	Куплено	12	2024-06-17	14:55:44			993	2024-06-17	
20	23	Срібло	12	1	кг	Куплено	12	2024-06-17	14:58:05			1000	2024-06-18	
21	24	Молоко	12	10	кг	Куплено	120	2024-06-17	16:57:53			13	2024-06-18	
22			Сумарний	13743								Прибуток:	-1597,5	

Рис. 2.15 згенерований звіт в Exel-файлі

Порядок роботи з програмою:

1. Запуск програми:
    - Користувач запускає executable файл програми.
    - Відкривається головне вікно програми.
2. Введення даних про операцію:
    - Користувач вводить назву металу, ціну, кількість та вибирає одиницю виміру.
    - Система автоматично розраховує загальну суму операції.
3. Виконання операції купівлі/продажу:
    - Користувач натискає кнопку "Купити" або "Продати".
    - Система оновлює базу даних, список продуктів та бюджет.
4. Управління бюджетом:
    - Користувач натискає кнопку "Бюджет".
    - У новому вікні вводить нове значення бюджету і зберігає зміни.
5. Адміністративні функції:
    - Адміністратор натискає кнопку "Адмін" і вводить пароль.

- У панелі адміністратора виконує необхідні дії: генерація звіту, видалення продуктів або транзакцій, зміна пароля.

6. Завершення роботи:
    - Користувач закриває програму, всі дані автоматично зберігаються в базі даних.

Повідомлення:

1. "Неможливо купити продукт. Сума перевищує бюджет."
    - Для кого: Користувач
    - Ситуація: Спроба купівлі металу на суму, що перевищує поточний бюджет
    - Дії: Перевірити суму операції та поточний бюджет, за необхідності збільшити бюджет
2. "Неможливо здійснити продаж. На складі недостатньо товару."
    - Для кого: Користувач
    - Ситуація: Спроба продажу більшої кількості металу, ніж є в наявності
    - Дії: Перевірити наявну кількість металу в списку продуктів
3. "Невірний пароль!"
    - Для кого: Адміністратор
    - Ситуація: Введено неправильний пароль при спробі доступу до адміністративної панелі
    - Дії: Перевірити правильність введення пароля, за необхідності звернутися до технічної підтримки
4. "Продукт '[назва]' не знайдено."
    - Для кого: Адміністратор
    - Ситуація: Спроба видалення неіснуючого продукту
    - Дії: Перевірити правильність введення назви продукту
5. "Не вдалося видалити транзакцію."
    - Для кого: Адміністратор

- Ситуація: Помилка при спробі видалення транзакції
- Дії: Перевірити наявність транзакції в базі даних, спробувати ще раз, за повторення помилки звернутися до технічної підтримки

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Під час розробки автоматизованої інформаційної системи для підприємства, яке займається прийомом і продажу металу, було проведено розрахунок трудомісткості та вартості з урахуванням наступних вихідних даних:

1. Передбачуване число операторів програми: 804;
2. Коефіцієнт складності програми: 1,3;
3. Коефіцієнт корекції програмного продукту в ході його розробки: 0,08;
4. Годинна заробітна плата розробника рівня Junior: 316,48 грн/год. Згідно зі статистикою з сайту Jooble: Junior python developer зарплата - Перевірити середню junior python developer ставку на Jooble. 50638 грн за місяць, тобто за 160 робочих годин, приблизно 8 доларів за годину.
5. Коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі: 1,4;
6. Коефіцієнт кваліфікації програміста, залежний від стажу роботи: 0,8;
7. Вартість машино-години ЕОМ: 3,4 грн/год. Розрахунок був проведений за наступним алгоритмом: загалом написання кваліфікаційної роботи разом з написанням коду зайняло приблизно 1 місяці, де в середньому на роботу витрачалось 2 години кожен день, без урахуванням вихідних днів. Тобто за 1 місяці на написання роботи було витрачено 40 годин безперервного використання ноутбуку. Згідно тарифного плану https://yasno.com.ua/b2c-tariffs кВт/год коштує 4,32 грн з урахуванням ПДВ. Ноутбук використаної моделі витрачає 60 Вт на годину під час активної роботи, тобто в цілому було витрачено

2,400 Вт, або 2,4 кВт. Загальна сума витрат на електроенергію становить 10,4 гривень.

У ході виконання кваліфікаційної роботи, були використані різні інтернет джерела. Для забезпечення безперервного та стабільного інтернет з'єднання протягом 8 місяців використовувався тариф від компанії «Київстар» "Все разом WOW+ТБ" https://www.connect.net.ua/provayder/kyivstar, де місяць надання послуг коштує 125 грн.

Трудомісткість розробки програмного продукту можна розрахувати за допомогою формули:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{ – людино-годин,} \tag{3.1}$$

де t_o - витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a - витрати праці на розробку блок-схеми алгоритму;

t_n - витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ - витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p) \tag{3.2}$$

де q - передбачуване число операторів;

C - коефіцієнт складності програми;

p - коефіцієнт кореляції програми в ході її розробки.

Таким чином, умовне число операторів становить:

$$Q = 804 \cdot 1{,}3 \cdot (1 + 0{,}08) = 1128{,}8$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75..85) \cdot k} \tag{3.3}$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

k – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

З урахуванням коефіцієнта кваліфікації k = 1, отримуємо витрати праці на вивчення опису завдання:

t_u = (1128,8 · 1,4) / (80 · 0,8) = 24,7 людино-годин.

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20..25) \cdot k} \text{ - людино-годин,} \tag{3.4}$$

де Q – умовне число операторів програми;

k – коефіцієнт кваліфікації програміста.

Підставивши відповідні значення в формулу (3.4), отримаємо:

t_a = 1128,8 / (20 · 0,8) = 70,55 людино-годин.

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20..25) \cdot k} \tag{3.5}$$

t_n = 1128,8 / (25 · 0,8) = 56,44 людино-годин.

Витрати праці на налагодження програми на ЕОМ:

− за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4..5) \cdot k} \text{ – людино-годин,}$$

(3.6)

$$t_{отл} = 1128{,}8 \,/\, (4 \cdot 0{,}8) = 352{,}75 \text{людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t^{k}_{отл} = 1{,}5 \cdot t_{отл}$$

(3.7)

$$t^{k}_{отл} = 1{,}5 \cdot 352{,}75 = 529{,}125 \text{ людино-годин.}$$

Витрати праці на підготовку документації:

$$t_{д} = t_{др} + t_{до}$$

(3.8)

$$t_{д} = 78{,}4 + 58{,}8 = 137{,}2 \text{ людино-годин.}$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису;

$$t_{др} = \frac{Q}{(15-20) \cdot k}$$

(3.9)

$$t_{др} = 1128{,}8 \,/\, (18 \cdot 0{,}8) = 78{,}4 \text{ людино-годин.}$$

Підставляючи відповідні значення, отримаємо:

$$t_{до} = 0{,}75 \cdot t_{др}$$

(3.10)

$$t_{до} = 0{,}75 \cdot 78{,}4 = 58{,}8 \text{ людино-годин.}$$

Де $t_{до}$- трудомісткість редагування, печатки й оформлення документації:

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

t = 50 + 24,7 + 70,55 + 56,44+ 529,125 + 137,2 = 868 людино-годин.

За результатами розрахунків, загальна трудомісткість розробки даного програмного продукту складає 1946,8 людино-годин.

3.2 Розрахунок витрат на створення програми

Витрати на створення ПЗ $К_{ПО}$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.} \tag{3.11}$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн,} \tag{3.12}$$

де: $З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

t - загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата розробника становить 316,48 грн / год, отримуємо:

$$З_{ЗП} = 868 \cdot 316{,}48 = 274\,705 \text{ грн.}$$

Вартість машинного часу $З_{МВ}$, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \cdot С_{мч}, \text{ грн} \tag{3.13}$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

$С_{мч}$ – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} = 329{,}1 \cdot 3{,}4 = 1119 \text{ грн.}$$

Звідси витрати на створення програмного продукту:

$$К_{ПО} = 274\,705 + 1119 = 275\,824 \text{ грн.}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ міс,}$$

(3.14)

де B_k– число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні F_p= 176 годин).

Звідси, за формулою (3.14) витрати на створення програмного продукту:

$$T = 868 / (1 \cdot 176) = 5 \text{ міс.}$$

Таким чином, розробка автоматизованої інформаційної системи для підприємства яке займається прийомом і продажу металу буде коштувати 275 824 грн з урахуванням розробки однією людиною, яка має рівень Junior та один рік комерційного досвіду розробки програмного забезпечення. Приблизний час, потрібний для розробки складає 5 місяців при стандартному 40 годинному робочому тижні. Загальна кількість людино-годин, яка буде витрачена на розробку, складає 868 грн.

Висновки

У цій кваліфікаційній роботі було розглянуто актуальну проблему автоматизації процесів управління продажами та покупками металу на підприємствах металургійної галузі. На основі проведеного дослідження було розроблено програмне забезпечення, яке дозволяє вирішити поставлені завдання та досягти визначеної мети.

Основні результати роботи включають:

1. Аналіз предметної області та існуючих рішень:
    - Було проведено детальний аналіз сучасних методів та засобів автоматизації обліку продажів та покупок металу.
    - Визначено основні вимоги до інформаційної системи, що враховують специфіку металургійних підприємств.
2. Розробка структури та функціональних можливостей системи:
    - Сформовано архітектуру інформаційної системи, що включає модулі для введення та зберігання даних, генерації звітів, управління бюджетом та забезпечення безпеки даних.
    - Описано функціональні можливості системи, які включають введення та зберігання даних про продажі та покупки металів, генерацію звітів, управління бюджетом, забезпечення безпеки даних та інші.
3. Реалізація програмного забезпечення:
    - Виконано розробку програмного забезпечення за допомогою сучасних технологій, таких як мова програмування Python, база даних MongoDB та інтерфейсні фреймворки Qt.
    - Проведено тестування та валідацію програмного забезпечення, що підтвердило його відповідність встановленим вимогам та коректність функціонування.
4. Економічне обґрунтування:

- Проведено розрахунок трудомісткості розробки програмного забезпечення, витрат на його створення та тривалості розробки.
- Оцінено економічну ефективність впровадження розробленої системи, яка полягає у зменшенні обсягу ручної роботи, підвищенні точності обліку даних та оперативності доступу до необхідної інформації.

Наукова новизна роботи полягає у розробці комплексної інформаційної системи, яка забезпечує автоматизацію ключових процесів управління продажами та покупками металів на металургійних підприємствах. Запропоноване рішення включає використання сучасних методів та технологій, що забезпечують високу ефективність та надійність системи.

Практична значимість розробленого програмного забезпечення полягає у можливості його впровадження на підприємствах металургійної галузі для автоматизації процесів управління продажами та покупками металів. Це дозволить значно підвищити продуктивність праці, точність обліку даних та забезпечити швидкий доступ до необхідної інформації, що є критично важливим для підприємств, що працюють з великими обсягами інформації.

Подальший розвиток розробленого програмного забезпечення може включати додавання нових функцій, таких як інтеграція з іншими системами управління підприємством (ERP), розширення можливостей аналізу даних та прогнозування, впровадження мобільних додатків для доступу до системи з будь-якого місця. Крім того, можливе розширення сфери застосування програмного забезпечення на інші галузі промисловості, що потребують ефективного управління ресурсами та даними.

Таким чином, розробка інформаційної системи для управління продажами та покупками металу є важливим кроком у напрямку підвищення ефективності та конкурентоспроможності металургійних підприємств в умовах сучасного розвитку технологій. Впровадження цієї системи сприятиме зменшенню ручної роботи, підвищенню точності обліку даних та забезпеченню оперативного

доступу до необхідної інформації, що в кінцевому підсумку сприятиме стабільному розвитку підприємств.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Оцінка ризиковості підприємств із використанням інструментарію корпоративних фінансів в Україні [Електронний ресурс] // Економіка і прогнозування. – Режим доступу: http://eip.org.ua/docs/EP_21_1_07_uk.pdf – Дата звернення: 30.05.2024.
2. Які переваги та недоліки мови програмування Python? [Електронний ресурс] // Розумні ідеї. – Режим доступу: https://rozumni-ideyi.com/posts/view/yaki-perevahy-ta-nedoliky-movy-prohramuvannya-python – Дата звернення: 01.06.2024.
3. Створення графічних інтерфейсів Python за допомогою PyQt5 — Прості графічні інтерфейси для повноцінних програм [Електронний ресурс] // Python GUIs. – Режим доступу: https://www.pythonguis.com/pyqt5/ – Дата звернення: 01.06.2024.
4. MongoDB: The Developer Data Platform [Електронний ресурс] // MongoDB. – Режим доступу: https://www.mongodb.com/ – Дата звернення: 01.06.2024.
5. PyCryptodome — PyCryptodome 3.210b0 документація [Електронний ресурс]. – Режим доступу: https://www.pycryptodome.org/src/introduction – Дата звернення: 04.06.2024.
6. openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files — openpyxl 3.1.3 documentation [Електронний ресурс]. – Режим доступу: https://openpyxl.readthedocs.io/en/stable – Дата звернення: 02.06.2024.
7. Стандартна бібліотека Python — Python 3.12.4 documentation [Електронний ресурс]. – Режим доступу: https://docs.python.org/uk/3/library/index.html – Дата звернення: 01.06.2024.
8. Копець Л. Класичні задачі Computer Science на мові Python. - К.: Фабула, 2020.

9. Hoberman, S. Data Modeling for MongoDB. – Technics Publications, 2019.

10. Лаврищева К.М., Слабоспицька О.О. Підхід до побудови об'єктно-компонентної моделі сімейства програмних систем. – Проблеми програмування, 2020.

11. Summerfield M. Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming. – Prentice Hall, 2015.

12. Chodorow K. MongoDB: The Definitive Guide: Powerful and Scalable Data Storage. 3rd Edition. – O'Reilly Media, 2019.

13. Phillips D., Giridhar C. Python Data Structures and Algorithms. – Packt Publishing, 2015.

14. Martin R.C. Clean Code: A Handbook of Agile Software Craftsmanship. – Prentice Hall, 2008.

15. Hunt A., Thomas D. The Pragmatic Programmer: Your Journey to Mastery, 20th Anniversary Edition. – Addison-Wesley Professional, 2019.

16. Lutz M. Learning Python, 5th Edition. – O'Reilly Media, 2019.

17. Bischof, S., Böhm, K., & Harth, A. MongoDB and Python: Patterns and processes for the popular document-oriented database. – O'Reilly Media, 2020.

18. Тюріна В.М., Федорова Н.В. Інтегровані інформаційні системи в промисловості. – К.: Центр учбової літератури, 2016. – 372 с.

19. Бернал Р. Програмування на Python для науки та техніки. – Львів: Новий Світ-2000, 2019. – 416 с. – Дата звернення: 10.06.2024.

20. Смирнов А.В. Ефективність використання MongoDB в сучасних інформаційних системах – Сучасні інформаційні технології. – 2021. – № 2. – С. 78-85. – Дата звернення: 10.06.2024.

Додаток А

ЛІСТИНГ ПРОГРАМИ

Файл main.py:

```
from PyQt5.QtWidgets import QApplication
from main_window import MainWindow
from admin_window import AdminWindow
from budget_window import BudgetWindow
import database
def main():
    import sys
    app = QApplication(sys.argv)
    main_window = MainWindow()
    admin_window = AdminWindow()
    budget_window = BudgetWindow(main_window)
    database.set_main_window(main_window)
    main_window.admin_button.clicked.connect(admin_window.show)
    main_window.budget_button.clicked.connect(budget_window.show)
    main_window.show()
    sys.exit(app.exec_())
if __name__ == '__main__':
    main()
```

Файл main_window.py:

```
from PyQt5.QtWidgets import (QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout,
QGridLayout,
                             QComboBox, QListWidget, QMessageBox, QCompleter)
from database import (add_transaction, search_products, get_daily_budget, get_all_product_names)
from password_dialog import PasswordDialog
from PyQt5.QtGui import QDoubleValidator
from budget_window import BudgetWindow
from admin_window import AdminWindow
from PyQt5.QtCore import Qt, QSize
class MainWindow(QWidget):
    def __init__(self):
```

```
        super().__init__()
        self.init_ui()
        self.refresh_product_list()
        self.update_budget_display()
        self.resize(QSize(800, 360))
        self.setStyleSheet("background-color: #FAEBD7;")
    def init_ui(self):
        product_name_label = QLabel("Назва продукту")
        self.product_name_input = QLineEdit()
        self.product_name_input.textChanged.connect(self.capitalize_first_letter)
        self.product_name_input.setFocusPolicy(Qt.StrongFocus)
        self.product_name_input.setStyleSheet("background-color: #FFDAB9;")
        product_names = get_all_product_names()
        completer = QCompleter(product_names, self)
        completer.setCaseSensitivity(Qt.CaseInsensitive)
        self.product_name_input.setCompleter(completer)
        price_label = QLabel("Ціна за одиницю")
        self.price_input = QLineEdit()
        self.price_input.setValidator(QDoubleValidator(0.00, 10000.00, 2))
        self.price_input.setFocusPolicy(Qt.StrongFocus)
        self.price_input.setStyleSheet("background-color: #FFDAB9;")
        weight_label = QLabel("Вага або кількість")
        self.weight_input = QLineEdit()
        self.weight_input.setValidator(QDoubleValidator(0.00, 10000.00, 2))
        self.weight_input.setFocusPolicy(Qt.StrongFocus)
        self.weight_input.setStyleSheet("background-color: #FFDAB9;")
        unit_label = QLabel("кг/шт")
        self.unit_combobox = QComboBox()
        self.unit_combobox.addItems(["кг", "шт"])
        self.unit_combobox.setFocusPolicy(Qt.StrongFocus)
        self.unit_combobox.setStyleSheet("background-color: #FFDAB9;")
        self.buy_button = QPushButton("Купити")
        self.sell_button = QPushButton("Продати")
        self.budget_button = QPushButton("Бюджет")
        self.admin_button = QPushButton("Адмін")
```

```
self.buy_button.setStyleSheet("background-color: #D15959;")
self.sell_button.setStyleSheet("background-color: #00FF00;")
self.budget_button.setStyleSheet("background-color: #78F9E3;")
self.admin_button.setStyleSheet("background-color: #78CEF9;")
self.total_label = QLabel("Сума: 0 грн")
self.total_label.setAlignment(Qt.AlignCenter)
self.budget_label = QLabel("Бюджет: 0 грн")
self.budget_label.setAlignment(Qt.AlignCenter)
grid_layout = QGridLayout()
grid_layout.addWidget(product_name_label, 0, 0)
grid_layout.addWidget(self.product_name_input, 0, 1)
grid_layout.addWidget(price_label, 1, 0)
grid_layout.addWidget(self.price_input, 1, 1)
grid_layout.addWidget(weight_label, 2, 0)
grid_layout.addWidget(self.weight_input, 2, 1)
grid_layout.addWidget(unit_label, 3, 0)
grid_layout.addWidget(self.unit_combobox, 3, 1)
buy_sell_layout = QHBoxLayout()
buy_sell_layout.addWidget(self.buy_button)
buy_sell_layout.addWidget(self.sell_button)
info_layout = QVBoxLayout()
info_layout.addWidget(self.total_label)
info_layout.addLayout(buy_sell_layout)
info_layout.addWidget(self.budget_label)
left_layout = QVBoxLayout()
left_layout.addLayout(grid_layout)
left_layout.addLayout(info_layout)
left_layout.addWidget(self.budget_button)
left_layout.addWidget(self.admin_button)
search_label = QLabel("Пошук назви продукту")
self.search_input = QLineEdit()
self.search_input.textChanged.connect(self.search_product)
self.search_input.setFocusPolicy(Qt.StrongFocus)
self.product_list = QListWidget()
self.product_list.setStyleSheet("background-color: #E6E6FA;")
```

```
        self.search_input.setStyleSheet("background-color: #FFDAB9;")
        right_layout = QVBoxLayout()
        right_layout.addWidget(search_label)
        right_layout.addWidget(self.search_input)
        right_layout.addWidget(self.product_list)
        main_layout = QHBoxLayout()
        main_layout.addLayout(left_layout)
        main_layout.addLayout(right_layout)
        self.setLayout(main_layout)
        self.setWindowTitle('Product Management System')
        self.buy_button.clicked.connect(self.buy_product)
        self.sell_button.clicked.connect(self.sell_product)
        self.price_input.textChanged.connect(self.update_total)
        self.weight_input.textChanged.connect(self.update_total)
        self.budget_button.clicked.connect(self.show_budget_window)
        self.admin_button.clicked.connect(self.show_admin_window)
    def keyPressEvent(self, event):
        if event.key == Qt.Key_Up:
            self.focusPreviousChild()
        elif event.key == Qt.Key_Down:
            self.focusNextChild()
        else:
            super().keyPressEvent(event)
    def show_budget_window(self):
        if not hasattr(self, 'budget_window') or not self.budget_window or not self.budget_window.isVisible():
            self.budget_window = BudgetWindow(self)
        else:
            self.budget_window.activateWindow()
    def update_budget_label(self, budget):
        self.budget_label.setText(f"Бюджет: {budget:.2f} грн")
    def update_budget_display(self):
        latest_daily_budget = get_daily_budget()
        self.update_budget_label(latest_daily_budget)
    def capitalize_first_letter(self, text):
        self.product_name_input.setText(text.capitalize())
```

```
        self.product_name_input.setCursorPosition(len(text))
    def update_total(self):
        try:
            price = float(self.price_input.text())
            weight = float(self.weight_input.text())
            total = price * weight
            self.total_label.setText(f"Сума: {total:.2f} грн")
        except ValueError:
            self.total_label.setText("Сума: 0 грн")
    def buy_product(self):
        self.handle_transaction('Куплено')
        self.clear_input_fields()
    def sell_product(self):
        self.handle_transaction('Продано')
        self.clear_input_fields()
    def handle_transaction(self, action):
        try:
            product_name = self.product_name_input.text()
            price_per_unit = float(self.price_input.text())
            weight_or_quantity = float(self.weight_input.text())
            unit = self.unit_combobox.currentText()
            total = price_per_unit * weight_or_quantity
            if action == 'Куплено':
                current_budget = get_daily_budget()
                if total > current_budget:
                    QMessageBox.warning(self, "Помилка", "Неможливо купити продукт. Сума перевищує бюджет.")
                    return
            if action == 'Продано':
                existing_product = search_products(product_name)
                if not existing_product or existing_product[0]["weight_or_quantity"] < weight_or_quantity:
                    QMessageBox.warning(self, "Помилка", "Неможливо здійснити продаж. На складі недостатньо
товару.")
                    return
            add_transaction(product_name, price_per_unit, weight_or_quantity, unit, action, total)
            self.total_label.setText(f"Сума: {total:.2f} грн")
            self.update_budget_display()
```

```
            self.clear_input_fields()
            self.refresh_product_list()
        except ValueError:
            QMessageBox.warning(self, "Помилка", "Будь ласка, введіть дійсну числову величину для ціни та ваги/кількості.")
    def search_product(self, query):
        self.product_list.clear()
        results = search_products(query)
        for product in results:
            self.product_list.addItem(f"{product['product_name']} - {product['weight_or_quantity']} {product['unit']}")
    def refresh_product_list(self):
        self.search_product("")
    def clear_input_fields(self):
        self.product_name_input.clear()
        self.price_input.clear()
        self.weight_input.clear()
    def show_message(self, title, message):
        msg_box = QMessageBox()
        msg_box.setIcon(QMessageBox.Warning)
        msg_box.setWindowTitle(title)
        msg_box.setText(message)
        msg_box.exec_()
    def show_admin_window(self):
        dialog = PasswordDialog(self)
        if dialog.exec_():
            AdminWindow(self)
        else:
            exit()
```

Файл budget_window.py:

```
from PyQt5.QtWidgets import QWidget, QLabel, QLineEdit, QPushButton, QVBoxLayout, QHBoxLayout, QMessageBox
from database import add_budget
class BudgetWindow(QWidget):
    def __init__(self, main_window):
        super().__init__()
```

```
        self.main_window = main_window
        self.init_ui()
    def init_ui(self):
        budget_label = QLabel("Введіть бюджет:")
        self.budget_input = QLineEdit()
        self.budget_input.setStyleSheet("background-color: #FFDAB9;")
        save_button = QPushButton("Зберегти")
        back_button = QPushButton("Назад")
        input_layout = QHBoxLayout()
        input_layout.addWidget(budget_label)
        input_layout.addWidget(self.budget_input)
        button_layout = QHBoxLayout()
        button_layout.addWidget(save_button)
        button_layout.addWidget(back_button)
        main_layout = QVBoxLayout()
        main_layout.addLayout(input_layout)
        main_layout.addLayout(button_layout)
        self.setLayout(main_layout)
        self.setWindowTitle('Manage Budget')
        self.setStyleSheet("background-color: #FAEBD7;")
        back_button.clicked.connect(self.close)
        save_button.clicked.connect(self.save_budget)
    def save_budget(self):
        try:
            budget = float(self.budget_input.text())
            if budget < 0:
                raise ValueError("Budget cannot be negative")
            add_budget(budget)
            QMessageBox.information(self, "Успіх", f"Бюджет додано: {budget:.2f} грн")
            self.main_window.update_budget_display()
            self.budget_input.clear()
            self.close()
        except ValueError:
            QMessageBox.warning(self, "Помилка", "Будь ласка, введіть дійсну числову величину для бюджету.")
```

Файл database.py:

```
from pymongo import MongoClient
from datetime import datetime, date
client = MongoClient('mongodb://localhost:27017/')
db = client['product_management']
transactions = db['transactions']
budgets = db['budgets']
products = db['products']
daily_budgets = db['daily_budgets']
product_names = db['product_names']
passwords = db['passwords']
def set_main_window(window):
    global main_window
    main_window = window


def save_password_to_db(encrypted_password):
    passwords.delete_many({})
    passwords.insert_one({"password": encrypted_password})
def read_password_from_db():
    password_entry = passwords.find_one()
    if password_entry:
        return password_entry['password']
    else:
        return None
def add_transaction(product_name, price_per_unit, weight_or_quantity, unit, action, total):
    existing_product = products.find_one({"product_name": product_name, "unit": unit})
    if action == 'Продано':
        if not existing_product or existing_product["weight_or_quantity"] < weight_or_quantity:
            if main_window:
                main_window.show_message("Помилка", "Неможливо здійснити продаж. На складі недостатньо
товару.")
            return
    max_transaction = transactions.find_one(sort=[("transaction_number", -1)])
    next_transaction_number = max_transaction["transaction_number"] + 1 if max_transaction else 1
    transaction = {
        "transaction_number": next_transaction_number,
```

```
        "product_name": product_name,
        "price_per_unit": price_per_unit,
        "weight_or_quantity": weight_or_quantity,
        "unit": unit,
        "action": action,
        "total": total,
        "datetime": datetime.now()
    }
    transactions.insert_one(transaction)
    if action == 'Продано':
        new_weight_or_quantity = existing_product["weight_or_quantity"] - weight_or_quantity
        products.update_one(
            {"_id": existing_product["_id"]},
            {"$set": {"weight_or_quantity": new_weight_or_quantity}}
        )
        update_daily_budget(total)
    else:
        add_or_update_product(product_name, weight_or_quantity, unit)
        update_daily_budget(-total)
    add_product_name(product_name)
def add_or_update_product(product_name, weight_or_quantity, unit):
    existing_product = products.find_one({"product_name": product_name, "unit": unit})
    if existing_product:
        new_weight_or_quantity = existing_product["weight_or_quantity"] + weight_or_quantity
        products.update_one(
            {"_id": existing_product["_id"]},
            {"$set": {"weight_or_quantity": new_weight_or_quantity}}
        )
    else:
        product = {
            "product_name": product_name,
            "weight_or_quantity": weight_or_quantity,
            "unit": unit
        }
        products.insert_one(product)
```

```
def add_budget(amount):
    budget = {
        "budget": amount,
        "datetime": datetime.now()
    }
    budgets.insert_one(budget)
    update_daily_budget(amount)


def get_latest_budget():
    latest_budget = budgets.find_one(sort=[("datetime", -1)])
    if latest_budget:
        return latest_budget["budget"]
    else:
        return 0
def update_daily_budget(change):
    today = date.today().isoformat()
    current_budget_entry = daily_budgets.find_one({"date": today})
    if current_budget_entry:
        new_budget = current_budget_entry["budget"] + change
        daily_budgets.update_one(
            {"_id": current_budget_entry["_id"]},
            {"$set": {"budget": new_budget}}
        )
    else:
        daily_budgets.insert_one({"date": today, "budget": change})
def get_daily_budget():
    today = date.today().isoformat()
    current_budget_entry = daily_budgets.find_one({"date": today})
    if current_budget_entry:
        return current_budget_entry["budget"]
    else:
        return 0
def search_products(query):
    return products.find({"product_name": {"$regex": query, "$options": "i"}, "weight_or_quantity": {"$gt": 0}})
def add_product_name(product_name):
```

```
    if not product_names.find_one({"product_name": product_name}):
        product_names.insert_one({"product_name": product_name})
def get_all_product_names():
    return [product["product_name"] for product in product_names.find()]


def delete_product_name(product_name):
    result = product_names.delete_one({"product_name": product_name})
    return result.deleted_count > 0
def get_all_transactions():
    return list(transactions.find())
def restore_budget_and_product(transaction):
    transaction_date = transaction['datetime'].date().isoformat()
    current_budget_entry = daily_budgets.find_one({"date": transaction_date})
    if current_budget_entry:
        change = transaction['total'] if transaction['action'] == 'Куплено' else -transaction['total']
        new_budget = current_budget_entry["budget"] + change
        daily_budgets.update_one(
            {"_id": current_budget_entry["_id"]},
            {"$set": {"budget": new_budget}}
        )
    existing_product = products.find_one({"product_name": transaction['product_name'], "unit": transaction['unit']})
    if existing_product:
        change = -transaction['weight_or_quantity'] if transaction['action'] == 'Куплено' else
transaction['weight_or_quantity']
        new_weight_or_quantity = existing_product["weight_or_quantity"] + change
        products.update_one(
            {"_id": existing_product["_id"]},
            {"$set": {"weight_or_quantity": new_weight_or_quantity}}
        )
def delete_transaction(transaction_id):
    transaction = transactions.find_one({"_id": transaction_id})
    if not transaction:
        return False
    result = transactions.delete_one({"_id": transaction_id})
    if result.deleted_count > 0:
        restore_budget_and_product(transaction)
```

```
        if main_window:
            main_window.update_budget_display()
            main_window.refresh_product_list()
        return True
    return False
def get_all_budgets():
    return list(budgets.find().sort("datetime", 1))
```

Файл admin_window.py:

```
from PyQt5.QtWidgets import (QWidget, QLabel, QPushButton, QVBoxLayout,
                             QHBoxLayout, QDateEdit, QInputDialog, QMessageBox,
                             QLineEdit, QDialog, QComboBox, QMessageBox)
from database import (delete_product_name, get_all_transactions, delete_transaction,
                      get_all_budgets)
from database import read_password_from_db, save_password_to_db
from password_dialog import decrypt_password, encrypt_password
from PyQt5.QtCore import QDate
from openpyxl import Workbook
from datetime import datetime
import os
class ChangePasswordDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.init_ui()
        self.setStyleSheet("background-color: #FAEBD7;")
    def init_ui(self):
        self.setWindowTitle("Заміна ключа доступу")
              old_password_label = QLabel("Старий пароль")
        self.old_password_input = QLineEdit()
        self.old_password_input.setEchoMode(QLineEdit.Password)
        self.old_password_input.setStyleSheet("background-color: #FFDAB9;")
        new_password_label = QLabel("Новий пароль")
        self.new_password_input = QLineEdit()
        self.new_password_input.setEchoMode(QLineEdit.Password)
        self.new_password_input.setStyleSheet("background-color: #FFDAB9;")
```

```
        confirm_password_label = QLabel("Повторити новий пароль")
        self.confirm_password_input = QLineEdit()
        self.confirm_password_input.setEchoMode(QLineEdit.Password)
        self.confirm_password_input.setStyleSheet("background-color: #FFDAB9;")
        self.change_button = QPushButton("Змінити")
        self.change_button.setStyleSheet("background-color: #78CEF9;")
        self.change_button.clicked.connect(self.change_password)
        self.back_button = QPushButton("Назад")
        self.back_button.setStyleSheet("background-color: #78CEF9;")
        self.back_button.clicked.connect(self.close)
        layout = QVBoxLayout()
        layout.addWidget(old_password_label)
        layout.addWidget(self.old_password_input)
        layout.addWidget(new_password_label)
        layout.addWidget(self.new_password_input)
        layout.addWidget(confirm_password_label)
        layout.addWidget(self.confirm_password_input)
        layout.addWidget(self.change_button)
        layout.addWidget(self.back_button)
        self.setLayout(layout)
    def change_password(self):
        old_password = self.old_password_input.text()
        new_password = self.new_password_input.text()
        confirm_password = self.confirm_password_input.text()
        encrypted_old_password = read_password_from_db()
        decrypted_old_password = decrypt_password(encrypted_old_password)
        if old_password != decrypted_old_password:
            QMessageBox.warning(self, "Помилка", "Невірний старий пароль!")
            return
        if new_password != confirm_password:
            QMessageBox.warning(self, "Помилка", "Новий пароль і підтвердження паролю не співпадають!")
            return
        encrypted_new_password = encrypt_password(new_password)
        save_password_to_db(encrypted_new_password)
        QMessageBox.information(self, "Успіх", "Пароль змінено успішно!")
```

```
        self.close()
class AdminWindow(QWidget):
    def __init__(self, main_window):
        super().__init__()
        self.main_window = main_window
        self.init_ui()
    def init_ui(self):
        start_date_label = QLabel("Початкова дата")
        self.start_date_edit = QDateEdit()
        self.start_date_edit.setDate(QDate.currentDate())
        end_date_label = QLabel("Кінцева дата")
        self.end_date_edit = QDateEdit()
        self.end_date_edit.setDate(QDate.currentDate())
        report_button = QPushButton("Звіт")
        delete_product_button = QPushButton("Видалення продуктів")
        change_key_button = QPushButton("Заміна ключа доступу")
        delete_transactions_button = QPushButton("Видалення покупок чи продажу")
        back_button = QPushButton("Назад")
        date_layout = QHBoxLayout()
        date_layout.addWidget(start_date_label)
        date_layout.addWidget(self.start_date_edit)
        date_layout.addWidget(end_date_label)
        date_layout.addWidget(self.end_date_edit)
        button_layout = QVBoxLayout()
        button_layout.addWidget(report_button)
        button_layout.addWidget(delete_product_button)
        button_layout.addWidget(change_key_button)
        button_layout.addWidget(delete_transactions_button)
        button_layout.addWidget(back_button)
        main_layout = QVBoxLayout()
        main_layout.addLayout(date_layout)
        main_layout.addLayout(button_layout)
        self.setLayout(main_layout)
        self.setWindowTitle('Admin Panel')
        self.setStyleSheet("background-color: #FAEBD7;")
```

```
        back_button.setStyleSheet("background-color: #78CEF9;")
        back_button.clicked.connect(self.close)
        report_button.setStyleSheet("background-color: #78CEF9;")
        report_button.clicked.connect(self.generate_report)
        delete_product_button.setStyleSheet("background-color: #78CEF9;")
        delete_product_button.clicked.connect(self.show_delete_product_ui)
        change_key_button.setStyleSheet("background-color: #78CEF9;")
        change_key_button.clicked.connect(self.show_change_password_ui)
        delete_transactions_button.setStyleSheet("background-color: #78CEF9;")
        delete_transactions_button.clicked.connect(self.show_delete_product_or_sale_ui)
    def generate_report(self):
        start_date = self.start_date_edit.date().toPyDate()
        end_date = self.end_date_edit.date().toPyDate()
        transactions = get_all_transactions()
        filtered_transactions = [t for t in transactions if start_date <= t['datetime'].date() <= end_date]
        reports_dir = "reports"
        if not os.path.exists(reports_dir):
            os.makedirs(reports_dir)
        workbook = Workbook()
        sheet = workbook.active
        sheet.title = "Звіт"
        headers = ["Номер транзакції", "Назва продукту", "Ціна за одиницю", "Кількість/Вага", "Одиниця виміру",
"Дія", "Сума", "Дата і час"]
        sheet.append(headers)
        total_profit = 0
        for transaction in filtered_transactions:
            row = [
                transaction.get("transaction_number", ""),
                transaction.get("product_name", ""),
                transaction.get("price_per_unit", ""),
                transaction.get("weight_or_quantity", ""),
                transaction.get("unit", ""),
                transaction.get("action", ""),
                transaction.get("total", ""),
                transaction.get("datetime", "").strftime("%Y-%m-%d %H:%M:%S") if transaction.get("datetime") else ""
            ]
```

```
            if transaction.get("action") == 'Продано':
                total_profit += transaction.get("total", 0)
            else:
                total_profit -= transaction.get("total", 0)
            sheet.append(row)
        sheet.append([])
        profit_cell = sheet.cell(row=sheet.max_row + 1, column=1)
        profit_cell.value = "Прибуток:"
        profit_value_cell = sheet.cell(row=sheet.max_row, column=2)
        profit_value_cell.value = total_profit
        sheet.append([])
        budget_header = ["Бюджет", "Дата"]
        sheet.append(budget_header)
        budgets = get_all_budgets()
        total_budget = 0
        for budget in budgets:
            budget_row = [
                budget.get("budget", ""),
                budget.get("datetime", "").strftime("%Y-%m-%d") if budget.get("datetime") else ""
            ]
            total_budget += budget.get("budget", 0)
            sheet.append(budget_row)
        sheet.append([])
        total_budget_cell = sheet.cell(row=sheet.max_row + 1, column=1)
        total_budget_cell.value = "Сумарний бюджет:"
        total_budget_value_cell = sheet.cell(row=sheet.max_row, column=2)
        total_budget_value_cell.value = total_budget
        report_filename = os.path.join(reports_dir,
f"звіт_{start_date.strftime('%Y%m%d')}_{end_date.strftime('%Y%m%d')}.xlsx")
        workbook.save(report_filename)
        QMessageBox.information(self, "Успіх", f"Звіт збережено як {report_filename}")
    def show_change_password_ui(self):
        self.change_password_dialog = ChangePasswordDialog(self)
        self.change_password_dialog.show()
    def show_delete_product_or_sale_ui(self):
        self.delete_product_or_sale_dialog = DeleteProductOrSaleDialog(self.main_window)
```

```
        self.delete_product_or_sale_dialog.show()
    def show_delete_product_ui(self):
        self.delete_product_ui = QWidget()
        self.delete_product_ui.setWindowTitle("Видалити продукт")
        self.delete_product_ui.setStyleSheet("background-color: #FAEBD7;")
        self.delete_product_layout = QVBoxLayout()
        self.product_name_input = QLineEdit()
        self.product_name_input.setPlaceholderText("Введіть назву продукту")
        self.product_name_input.textChanged.connect(self.capitalize_first_letter)
        self.product_name_input.setStyleSheet("background-color: #FFDAB9;")
        self.delete_product_layout.addWidget(self.product_name_input)
        self.delete_button = QPushButton("Видалити")
        self.delete_button.setStyleSheet("background-color: #78CEF9;")
        self.delete_product_layout.addWidget(self.delete_button)
        self.back_button = QPushButton("Назад")
        self.back_button.setStyleSheet("background-color: #78CEF9;")
        self.delete_product_layout.addWidget(self.back_button)
        self.delete_product_ui.setLayout(self.delete_product_layout)
        self.delete_button.clicked.connect(self.delete_product)
        self.back_button.clicked.connect(self.delete_product_ui.close)
        self.delete_product_ui.show()
    def capitalize_first_letter(self, text):
        if text:
            self.product_name_input.blockSignals(True)
            self.product_name_input.setText(text.capitalize())
            self.product_name_input.blockSignals(False)
    def delete_product(self):
        product_name = self.product_name_input.text()
        if delete_product_name(product_name):
            QMessageBox.information(self, "Успіх", f"Продукт '{product_name}' видалено.")
        else:
            QMessageBox.warning(self, "Помилка", f"Продукт '{product_name}' не знайдено.")
    def show_message(self, title, message):
        msg_box = QMessageBox()
        msg_box.setIcon(QMessageBox.Warning)
```

```
        msg_box.setWindowTitle(title)
        msg_box.setText(message)
        msg_box.exec_()
class DeleteProductOrSaleDialog(QDialog):
    def __init__(self, main_window):
        super().__init__(main_window)
        self.main_window = main_window  # Ensure main_window reference
        self.setStyleSheet("background-color: #FAEBD7;")
        self.setWindowTitle("Видалити продукт або продажу")
        list_label = QLabel("Список транзакцій")
        self.list_combobox = QComboBox()
        self.load_transactions()
        self.delete_button = QPushButton("Видалити")
        self.delete_button.setStyleSheet("background-color: #78CEF9;")
        self.delete_button.clicked.connect(self.delete_selected_transaction)
        self.back_button = QPushButton("Назад")
        self.back_button.setStyleSheet("background-color: #78CEF9;")
        self.back_button.clicked.connect(self.close)
        layout = QVBoxLayout()
        layout.addWidget(list_label)
        layout.addWidget(self.list_combobox)
        layout.addWidget(self.delete_button)
        layout.addWidget(self.back_button)
        self.setLayout(layout)
    def load_transactions(self):
        transactions = get_all_transactions()
        for transaction in transactions:
            display_text = f"{transaction['transaction_number']}: {transaction['product_name']} - {transaction['action']} -
{transaction['total']} грн"
            self.list_combobox.addItem(display_text, transaction["_id"])
    def delete_selected_transaction(self):
        selected_id = self.list_combobox.currentData()
        if delete_transaction(selected_id):
            QMessageBox.information(self, "Успіх", "Транзакція видалена.")
            self.list_combobox.clear()
            self.load_transactions()
```

```
            self.main_window.update_budget_display()  # Ensure these methods are called
            self.main_window.refresh_product_list()
        else:
            QMessageBox.warning(self, "Помилка", "Не вдалося видалити транзакцію.")
```

Файл password_dialog.py:

```
from PyQt5.QtWidgets import QDialog, QLabel, QLineEdit, QPushButton, QVBoxLayout, QMessageBox
from database import read_password_from_db
from Crypto.Cipher import AES
import hashlib
import base64
def encrypt_password(password):
    key = hashlib.sha256("секретний ключ".encode()).digest()
    cipher = AES.new(key, AES.MODE_ECB)
    encrypted_password = cipher.encrypt(password.rjust(32).encode())
    return base64.b64encode(encrypted_password).decode()
def decrypt_password(encrypted_password):
    key = hashlib.sha256("секретний ключ".encode()).digest()
    cipher = AES.new(key, AES.MODE_ECB)
    decrypted_password = cipher.decrypt(base64.b64decode(encrypted_password))
    return decrypted_password.decode().strip()
class PasswordDialog(QDialog):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.init_ui()
    def init_ui(self):
        self.setWindowTitle('Пароль адміністратора')
        self.label = QLabel('Введіть пароль:')
        self.password_input = QLineEdit()
        self.password_input.setEchoMode(QLineEdit.Password)
        self.password_input.setStyleSheet("background-color: #FFDAB9;")
        self.ok_button = QPushButton('OK')
        self.ok_button.clicked.connect(self.check_password)
        layout = QVBoxLayout()
        layout.addWidget(self.label)
```

```
        layout.addWidget(self.password_input)
        layout.addWidget(self.ok_button)
        self.setLayout(layout)
    def check_password(self):
        entered_password = self.password_input.text()
        encrypted_password = read_password_from_db()
        decrypted_password = decrypt_password(encrypted_password)
        if entered_password == decrypted_password:
            self.accept()
        else:
            QMessageBox.warning(self, 'Помилка', 'Невірний пароль!')
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
ПЗ.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
ПЗ.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_.ppt	Презентація кваліфікаційної роботи