Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

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

Пояснювальна записка

до дипломної роботи
ОКР – бакалавр
(освітньо-кваліфікаційний рівень)

на тему: *Розроблення клієнт-серверної інформаційної системи складського обліку товарів (BackEnd).*

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

Малахівський В.І.
(прізвище та ініціали)

Керівник Сторожук О.Л.
(прізвище та ініціали)

Рецензент Крошний І.М.
(прізвище та ініціали)

Львів – 2023

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревообробних та комп'ютерних технологій і дизайну
Кафедра інформаційних систем та комп'ютерного моделювання
Рівень вищої освіти перший (бакалаврський)
Спеціальність 126 " Інформаційні системи та технології"
(шифр і назва)

ЗАТВЕРДЖУЮ
В.о.завідувача кафедри
Сторожук О.Л.
"12" 06 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Малахівському Володимиру Ігоровичу
(прізвище, ім'я, по батькові)

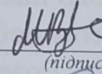
1. Тема роботи *«Розроблення клієнт-серверної інформаційної системи складського обліку товарів (BackEnd)»*
керівник роботи *Сторожук Олександр Леонідович, к.т.н., доцент*,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від *"21"11 2022 року № С-521*
2. Термін подання студентом роботи *12.06.2023р.*
3. Вихідні дані до роботи: Розробити програмний застосунок клієнт-серверної інформаційної системи складського обліку товарів. Застосунок має мати зручний та комфортний інтерфейс для користувача цієї програми. Для реалізації роботи використати мову програмування "Python", а також бібліотеки PyQt5, та інші.
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити): *1) Стан проблемної області; 2) Інформаційне та математичне забезпечення; 3) Програмне та технічне забезпечення.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) *Презентація до диплому*
6. Консультанти розділів проекту (роботи)

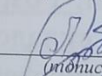
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *23 листопада 2022 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд літературних даних.*	*23.11.2022р. – 21.12.2022р.*	викон
2.	*Розділ 1. Стан проблемної області.*	*21.12.2022р. – 05.01.2023р.*	викон
3.	*Розділ 2. Інформаційне та математичне забезпечення.*	*05.01.2023р. – 02.02.2023р.*	викон
4.	*Розділ 3. Програмне та технічне забезпечення.*	*02.02.2023р. – 12.05.2023р.*	викон
9.	*Аналіз отриманих результатів та написання висновків. Оформлення дипломної роботи.*	*12.05.2023р. – 06.06.2023р.*	викон
10.	*Здача пояснювальної записки на перевірку керівнику, виправлення помилок та здача роботи рецензенту.*	*12.06.2023 р.*	викон

Студент  (підпис) *Малахівський В.І.* (прізвище та ініціали)

Керівник роботи  (підпис) *Сторожук О.Л.* (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота містить *43* сторінок пояснювальної записки, *51* рисунків, *1* додаток, *15* джерел.

Дипломна робота присвячена розробці клієнт-серверної інформаційної системи складського обліку товарів. Для реалізації завдання використано модуль PyQT5. PyQT – оболонка на мові програмування Python для бібліотеки Qt.Для бази даних взято SQLite. SQLite – полегшена реляційна система керування базами даних. Інтерфейс реалізовано у простому для користування вигляді. Передбачено можливість керування працівниками складу та редагування записів про товар.

Ключові слова: PyQT, SQLite, Інтерфейс, товар.

ABSTRACT

Thesis contains *43* pages of explanatory note, *51* drawings, *1* appendix, *15* sources.

The thesis is devoted to development of the client-server informative system of ware-house account of commodities. For realization of task it is used module of PyQT5. PyQT is a shell in programming of Python language for the library of Qt.For a database it is taken SQLite. SQLite – relational control system by the bases of data is facilitated. An interface is realized in a simple for the use kind. Possibility of management of composition and adjusting of records workers is envisaged about a commodity.

Keywords: PyQT, SQLite, Interface, commodity.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити клієнт-серверну інформаційну систему складського обліку товарів (BackEnd). Використати SQLite для створення бази даних. Ознайомитись із специфікою даної сфери для розробки якісної програми. При розробці використати мову програмування Python і бібліотеку PyQt5. Передбачити додавання, видалення, а також редагування товарів через бібліотеку sqlite3. Використати socketserver для написання мережевого серверу.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 7
ВСТУП 8
РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ 9
1.1. Огляд проблемної області. 9
1.2. Актуальність розроблення застосунку. 10
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ 11
2.1. Python 11
2.2. PyQt5 12
2.3. SqlitE3 13
РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ 14
3.1. Послідовна розробка програми 14
3.2. Тестування програми 40
ВИСНОВКИ 43
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 44
ДОДАТКИ 46

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface, інтерфейс програмування застосунків.

DB – Database, база даних.

GUI – Graphical User Interface, графічний інтерфейс користувача.

IDE – Integrated Development Environment, інтегроване середовище розробки програмного забезпечення.

SQL – Structured Query Language, мова структурованих запитів до баз даних.

UTF – Unicode Transformation Format, формат кодування символів Unicode.

ООП – об'єктно-орієнтоване програмування.

СУБД – система управління базами даних.

ВСТУП

Навіщо потрібен облік товару?

Облік товарів - це відстеження шляху товару від постачальника до клієнта та фіксування операцій на всіх етапах, такі як надходження та розміщення товару на складі, переміщення товару, продаж товару та списання товару зі складу.

Облік товарів допомагає зрозуміти, що відбувається з товаром наприклад у магазині, оптимізувати процеси та бюджет на закупівлі, сформувати привабливий асортимент та уникнути крадіжки.

Без обліку неможливо машстабувати магазин. Відсутність обліку веде за собою значну кількість проблем такі як помилки постачання або неефективна робота з постачальниками. Без обліку неможливо побачити чи прийшло продукції менше ніж було замовлено. Очевидною проблемою будуть порожні полиці та низький товарообіг. Контролювати залишковий товар неможливо, а деякі товари лежатимуть місяцями через неправильно сформований асортимент. Звичайно неможливо не зачепити теми крадіжки. Постійні недостачі, брак коштів у касах, зникнення товарів зі складу неможливо помітити без обліку.

Об'єктом дослідження є використання SQLite для створення обліку ведення товарів.

Метою роботи є створення клієнт-серверної інформаційної системи для зручного ведення обліку товарів.

Предметом дослідження є використання модуля SQLite для інформаційної системи обліку.

Практичне значення – розроблена інформаційна система полегшує роботу працівникам складу та дозволяє вести складський облік товарів.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1. Огляд проблемної області.

Складський облік товарів є важливим елементом управління запасами. Він включає в себе процеси отримання, зберігання та розподілу товарів на складах. Основною метою складського обліку є забезпечення належного контролю за запасами, їх точністю та ефективним використанням.

Недостатня автоматизація: Багато компаній все ще використовують застарілий ручний підхід до складського обліку, що може призводити до помилок, затримок та незручностей. Недостатня автоматизація може ускладнювати контроль за запасами, ведення журналів і відстеження переміщення товарів.

Помилки в даних: Неправильне введення або запис даних можуть призвести до серйозних проблем у складському обліку. Наприклад, неправильний запис кількості товару, помилкове визначення його місцезнаходження або невідповідність даних в реальному часі.

Недостатня інтеграція з іншими системами: складський облік часто пов'язаний з іншими бізнес-процесами, такими як закупівля, продаж та бухгалтерія. Недостатня інтеграція між цими системами може призводити до непослідовності даних і ускладнювати звітність.

Ускладнення ланцюга постачання: Недостатній контроль за запасами може впливати на ефективність ланцюга постачання. Наприклад, затримки у виявленні недостатніх запасів можуть призвести до несвоєчасного замовлення нових товарів, що впливає на виробництво та доставку.

Нестача розуміння попиту: Несправна аналітика попиту і передбачення може призвести до неправильного планування запасів. Якщо компанія недооцінює попит, це може призвести до недостачі товарів, втрати продажів і незадоволеності клієнтів. З іншого боку, якщо попит переоцінюється, можуть виникати проблеми зі зберіганням та звільненням непроданих запасів.

1.2. Актуальність розроблення застосунку.

Застосунок для ведення складського обліку товарів є важливим і корисним інструментом для підприємств у сучасному бізнес-середовищі з наступних причин:

Автоматизація процесів: Застосунок для складського обліку дозволяє автоматизувати багато повсякденних процесів, пов'язаних з управлінням запасами. Це включає в себе приймання товарів, розміщення на складі, замовлення, відправку, інвентаризацію тощо. Автоматизація допомагає зменшити ручну працю, помилки та затримки, покращує швидкість і точність операцій.

Відстеження запасів в реальному часі: Застосунок дозволяє вести точний облік запасів в реальному часі. Це означає, що ви завжди знаєте, скільки товарів у вас є на складі, де вони знаходяться і які їх характеристики. Це сприяє покращенню планування постачання, уникненню недостачі або переобрання товарів і забезпеченню оптимального рівня запасів.

Оптимізація планування та прогнозування: Застосунки для складського обліку можуть містити аналітичні інструменти, які допомагають у плануванні попиту, передбаченні тенденцій і оптимізації запасів. Вони використовують алгоритми і статистичні методи для прогнозування майбутнього попиту, що дозволяє підприємствам здійснювати точніші прогнози і зменшувати ризики недостачі або надлишків товарів.

Інтеграція з іншими системами: Сучасні застосунки для складського обліку можуть легко інтегруватися з іншими системами, такими як система управління виробництвом, система управління продажами або система електронного обліку. Це сприяє покращенню координації між різними бізнес-процесами і забезпечує єдину базу даних для керівників та співробітників.

Звітність та аналітика: Застосунки для складського обліку надають можливість отримувати детальні звіти і аналізи про ефективність складського управління. Вони допомагають виявляти потенційні проблеми, здійснювати моніторинг показників продуктивності, аналізувати витрати та вдосконалювати стратегії управління запасами.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Python

Python - це високорівнева інтерпретована мова програмування загального призначення. Вона була створена в 1991 році Гвідо ван Россумом і має прихильну та активну спільноту розробників [9]. Характеристики та особливості мови Python:

Синтаксис: Python має простий та лаконічний синтаксис, що дозволяє розробникам писати зрозумілий та читабельний код. Вона використовує відступи для визначення блоків коду, що сприяє його структурованості.

Інтерпретованість: Python є інтерпретованою мовою, що означає, що програмний код виконується рядок за рядком без необхідності компіляції. Це спрощує процес розробки та тестування програм.

Платформонезалежність: Python підтримує різні операційні системи, такі як Windows, macOS та Linux. Це дозволяє розробникам писати програми на Python, які працюватимуть на різних платформах без змін вихідного коду.

Багатий набір бібліотек: Python має велику кількість стандартних бібліотек, які включають різноманітні функціональні можливості, такі як робота з рядками, мережами, базами даних, обробкою зображень і багато іншого. Крім того, існує широкий спектр сторонніх бібліотек, які розширюють функціональність мови.

Об'єктно-орієнтований підхід: Python підтримує об'єктно-орієнтований підхід до програмування, що дозволяє розробникам створювати класи, об'єкти та спадковість. Це сприяє модульності та повторному використанню коду.

Широке застосування: Python використовується в різних сферах, таких як веб-розробка, наукові дослідження, аналіз даних, штучний інтелект, машинне навчання та багато іншого. Вона має велику популярність серед програмістів через свою простоту та потужність [1,2,4].

2.2. PyQt5

PyQt5 - це набір інструментів для розробки графічного інтерфейсу користувача (GUI) на мові програмування Python. Він є надбудовою над Qt, потужною багатоплатформовою бібліотекою для створення програмного забезпечення.

Особливості та характеристики PyQt5:

Багатоплатформовість: PyQt5 підтримує різні операційні системи. Це дозволяє створювати кросплатформові програми.

Повна підтримка Qt: PyQt5 надає доступ до повного функціоналу Qt, включаючи його розширені можливості управління графікою, мультимедіа, мережею, базами даних, анімації та багато іншого. Qt має широкий набір вбудованих віджетів та інструментів для створення потужних GUI додатків.

Легкість використання: PyQt5 має зрозумілий та лаконічний синтаксис, який дозволяє розробникам швидко створювати інтерактивні та естетично привабливі користувацькі інтерфейси. Вона надає велику кількість вбудованих методів та функцій для роботи з віджетами і обробки подій.

Гнучкість та розширюваність: PyQt5 дозволяє розробникам створювати власні віджети та розширювати функціонал інструментів Qt за допомогою Python. Вона також підтримує взаємодію з іншими бібліотеками та інструментами Python, що дозволяє використовувати їх разом з PyQt5.

Документація та спільнота: PyQt5 має добре документовану інструкцію та довідку, що сприяє вивченню та розумінню функціоналу [8]. Крім того, вона має активну спільноту розробників, яка надає підтримку, допомогу та поширює корисні ресурси.

Ліцензія: PyQt5 доступна під подвійною ліцензією: GNU General Public License (GPL) для вільного та відкритого програмного забезпечення та комерційної ліцензії для використання в комерційних проектах.[3]

2.3. SqlitE3

SQLite3 - це вбудована реляційна база даних, яка працює на принципах SQL (Structured Query Language). Вона є частиною бібліотеки SQLite, яка забезпечує незалежну від платформи і вбудовану систему управління базами даних [10].

Особливості та характеристики SQLite3:

Реляційна база даних: SQLite3 підтримує основні принципи реляційної моделі даних, такі як таблиці зі стовпцями та рядками. Вона дозволяє використовувати SQL для створення, зміни та запитування даних.

Транзакційність та безпека: SQLite3 підтримує транзакції, що дозволяє забезпечити цілісність та безпеку даних. Вона має механізм блокування, який дозволяє уникнути конфліктів доступу до даних.

Широкий функціонал: SQLite3 підтримує багато функцій, таких як індекси, підзапити, тригери, керування доступом та багато іншого. Вона також підтримує різні типи даних, включаючи числа, рядки, дати, час та більш складні структури даних.

Невеликі вимоги до ресурсів: SQLite3 є легким та ефективним, вона має невисокі вимоги до пам'яті та обробки, що робить її популярною для вбудованих систем, мобільних додатків та простих веб-сайтів [5].

РОЗДІЛ 3. ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Послідовна розробка програми

SQLite

Для реалізаії бази даних було використано sqlite3 - це компактна, вбудована реляційна база даних, яка використовується для зберігання і управління даними. Вона є одним з найпоширеніших і популярних вбудованих систем управління базами даних(СУБД) і використовується в багатьох програмних продуктах, веб-сайтах та мобільних додатках. Особливістю SQLite3 є те, що вона не працює як окремий сервер баз даних, а замість цього вбудовується безпосередньо в програму, що використовує її [11]. Це означає, що вам непотрібно окремо налаштовувати сервер баз даних або встановлювати складні залежності. Ви можете просто додати бібліотеку SQLite3 до своєї програми і почати працювати з базою даних.

Має маленький розмір, що робить її ефективним варіантом для вбудовування в обмежені ресурси, такі як мобільні пристрої. Підтримується на багатьох операційних системах. Розповсюджується під ліцензією Public Domain, що дозволяє використовувати, модифікувати і поширювати її безкоштовно.

SQLite3 пропонує швидкий доступ до даних завдяки використанню простих SQL-запитів і оптимізації внутрішніх структур даних. Підтримує транзакції ACID (атомарність, констистентність, ізольованість, стійкість), що забезпечує цілісність та безпеку даних [12]. SQLite3 підтримує повний набір SQL- операцій, що дозволяє виконувати створення, зчитування, оновлення та видалення даних за допомогою стандартних SQL-запитів.

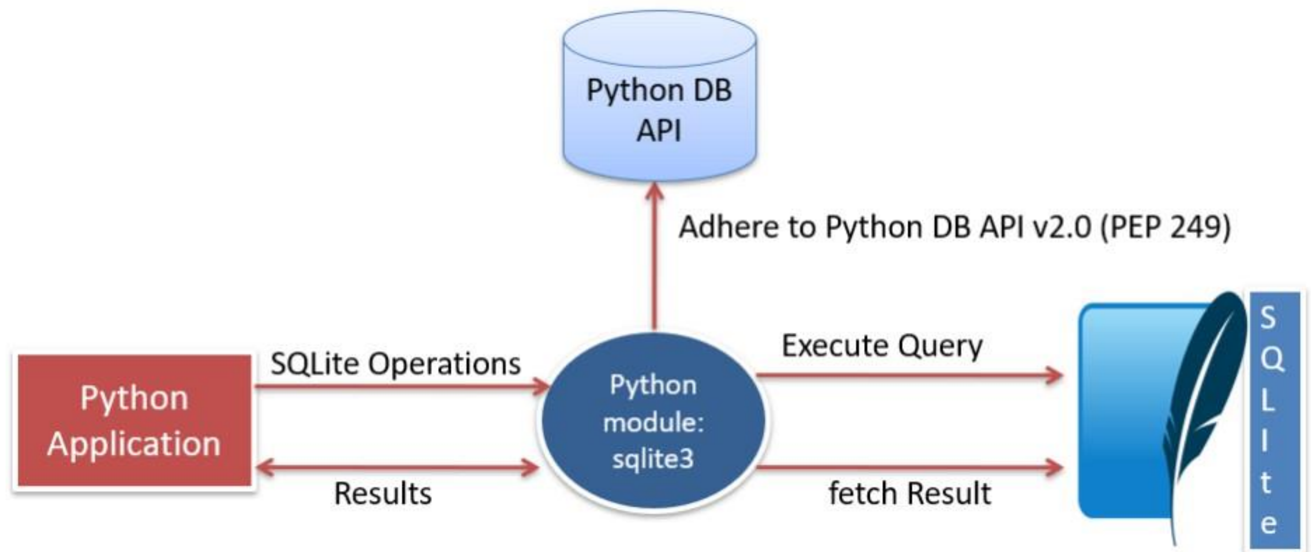


Рис. 3.1. Робота модуля SQLite3 у python.

Перш ніж почати використовувати SQLite3 у програмі треба розібратися з типами даних. Потрібно знати типи даних SQLite і їх відповідники у Python. Механізм бази даних SQLite3 має кілька класів для зберігання значень [13]. Кожне значення, що зберігається в базі даних SQLite, має один із наведених нижче типів даних:

1. NULL: - значенням є null.
2. INTEGER: - для збереження числового значення. Ціле число, що зберігається в 1, 2, 3, 4, 6 або 8 байтах в залежності від величини числа.
3. REAL: - це значення з плаваючою комою, наприклад значення числа Pi - 3,14...
4. TEXT: - значення є текстовим рядком збереженим з використанням кодування UTF-8, UTF-16BE або UTF-16LE.
5. BLOB: - значення є двійковий тип даних.Використовується для зберігання зображень та файлів.

Щоб використовувати їх коректно, потрібно запам’ятати аналоги з Python. У цьому допоможе дана таблиця:

Python Types	SQLite types
None	NULL
int	INTEGER
float	REAL
str	TEXT
bytes	BLOB

Рис. 3.2. Відповідники типам даних у Python та SQLite.

Щоб використовувати SQLite3 у нашій програмі, перш за все потрібно прописати:

import sqlite3. На зображенні знизу продемонстрована ця строка:

```
import sys
from PyQt5.QtCore import *
import csv
import os
from PyQt5.QtWidgets import *
from PyQt5.QtWidgets import QMessageBox
from PyQt5.QtGui import *
from PyQt5.QtCore import Qt, QDateTime
import sqlite3
from PIL import Image
import tkinter as tk
from PIL import ImageGrab
from PyQt5 import QtWidgets, QtCore, QtGui
from PyQt5.QtCore import Qt
import re
from shutil import copyfile
import xlwt
from xlwt import Workbook
```

Рис. 3.3. Список під’єднаних бібліотек.

Тепер можемо використовувати модуль SQLite3 у нашій програмі. Використовуючи класи та методи визначені у ньому, можемо комунікувати з базою даних SQLite.

Щоб під'єднатись до бази даних потрібно виконати метод connect() з назвою бд.

Якщо вказати назву файлу з існуючою базою даних, яка уже присутня на диску, він підключиться до цього файлу. Але якщо вказаний файл бази даних SQLite не існує, то SQLite створить для вас нову базу даних. Метод повертає об'єкт з'єднання SQLite, якщо з'єднання вдалось.

```
21
22    con = sqlite3.connect("db_database/main_database.db")
```

Рис. 3.4. Метод connect().

Далі використовуємо метод cursor класу з'єднання для виконання команд запитів SQLite з Python.

```
23    cur = con.cursor()
```

Рис. 3.5. Метод cursor().

```
        if self.update_to_DB.text() == 'Оновити':
            result = QMessageBox.question(self, "Підтвердити вихід?", "Ви не зберегли зміни в базі даних\nБудь ласка, збережіть зміни"
            event.ignore()
            if result == QMessageBox.Save:
                self.update_DB_database()
                #event.accept()
            elif result == QMessageBox.No:
                file = open('database_backup/state.txt', 'w')
                file.write('state')
                event.accept()
```

Рис. 3.6. Вихід з програми

Даний фрагмент коду відповідає за вихід з програми без збереження змін.

Якщо нажати 'Save' таблиця баз даних обновиться і виконається вихід з програми. Нажавши 'No' вихід виконається без обновлення бази даних. При натисканні 'Cancel' дане вікно закриється.

Також необхідно використовувати метод execute(). Він виконує SQL-запит і повертає результат. Ось кілька прикладів використання цього методу:

```
def displayProducts(self):
    self.productsTable.setFont(QFont("Times", 10))
    for i in reversed(range(self.productsTable.rowCount())):
        self.productsTable.removeRow(i)
    query = cur.execute("SELECT product_id,description,product
```

Рис. 3.7. Метод execute().

```
    QMessageBox.information(self, "Увага!", "П
else:
    self.searchEntry.setText("")
    #query = cur.execute("")
    query = ('SELECT product_id,description,pr
    results = cur.execute(query, ('%'+value+'%
    if results == []:
        QMessageBox.information(self, "Увага!"
```

Рис. 3.8. Метод execute()(1)

```
elif self.notAvailableProducts.isChecked() == True:
    self.productsTable.sortItems(10, Qt.AscendingOrder)
    query = ("SELECT product_id,description,product_manufacturer,product_name,product_quota,product_availability,supplier,date_add
    products = cur.execute(query).fetchall()
```

Рис. 3.9. Метод execute()(2)

Важливі моменти під час підключення до SQLite:

1. Об’єкт підключення не є потоково безпечним. Модуль sqlite3 не дозволяє ділитися з’єднаннями між потоками. Якщо ви все ж спробуєте це зробити, ви отримаєте exception під час виконання.

2. Метод connect() приймає різні аргументи. У нашому прикладі ми передали аргумент імені бази даних для підключення.

3. Використовуючи об’єкт підключення, ми можемо створити об’єкт cursor, який дозволяє нам виконувати команду/запити SQLite через Python.

4. Ми можемо створити скільки завгодно cursor з одного об’єкта підключення. Як і об’єкт з’єднання, цей об’єкт-курсор також не є потокобезпечним. Модуль sqlite3 не дозволяє обмінюватися курсорами між потоками. Якщо ви все ж спробуєте це зробити, ви отримаєте виняток під час виконання.

5. Використовуючи клас Error, ми можемо обробляти будь-які помилки та винятки бази даних, які можуть виникнути під час роботи з SQLite з Python.

6. Клас Error допомагає нам детально зрозуміти помилку. Він повертає повідомлення про помилку та код помилки.

7. Завжди добре закривати курсор і об’єкт підключення після завершення роботи, щоб уникнути проблем із базою даних.

```
def clear_all_items(self):
    self.password_window = password.PasswordInitializer()
    if self.password_window.exec_() == QDialog.Accepted:

        mbox = QMessageBox.warning(self, "Увага", "Видалити всі елементи?", QMessageBox.Yes | QMessageBox.No, QMessageBox.No)

        if mbox == QMessageBox.Yes:

            try:
                cur.execute("DELETE FROM products")
                cur.execute("DELETE FROM members")
                cur.execute("DELETE FROM take_product")
                cur.execute("DELETE FROM transaction_history")
                cur.execute("DELETE FROM transactions")
                con.commit()
```

Рис. 3.10. Видалення всіх елементів.

Даний фрагмент коду дозволяє нам моментально очистити всю базу даних шляхом видалення всіх елементів.

Приклад витягнення інформаії з бази даних за допомогою звичайного SQL-запиту:

```
query = ("SELECT * FROM products WHERE product_id=?")
```

Рис. 3.11. SQL-запит.

SQL-запит `SELECT * FROM` є одним з найпростіших та найбільш загальних запитів у мові SQL. Він використовується для вибору всіх стовпців з таблиці бази даних без будь-яких обмежень або умов.

Варто зауважити, що запит `SELECT * FROM` може повернути велику кількість даних, що може бути недоцільним у великих таблицях. У таких випадках краще вказувати конкретні стовпці, які вам потрібні, замість використання `*`.

Наприклад тут для видалення фото ми вказуємо стовпець, який стосується лише фото:

```
if (name and manufacturer and price and quota != ""):
    img_to_delete = cur.execute("SELECT product_img FROM products WHERE product_id=?", (productId,)).fetchone()

    if len(os.listdir('src/current_pictures/')) == 0:
        print("Порожній каталог")
        defaulImgSrc = img_to_delete[0]
        print("defaulImgSrc->> ", defaulImgSrc)
    else:
        print("Ось деякі предмети")
        os.unlink(img_to_delete[0])
```

Рис. 3.12. Видалення фото з БД.

```
def displayMembers(self):
    self.membersTableWidgets.setFont(QFont("Times", 12))

    for i in reversed(range(self.membersTableWidgets.rowCount())):
        self.membersTableWidgets.removeRow(i)

    query_members = cur.execute("SELECT * FROM members")
    for row_data in query_members:
        row_number = self.membersTableWidgets.rowCount()
        self.membersTableWidgets.insertRow(row_number)
        for column_number, data in enumerate(row_data):
            self.membersTableWidgets.setItem(row_number, column_number, QTableWidgetItem(str(data)))
```

Рис. 3.13. Перегляд користувачів.

Реалізувано перегляд користувачив шляхом звичайного SQL-запиту SELECT з таблиці ‘members’.

```
ef search_members_btn(self):
    value = self.memberSearchEntry.text()
    if value == "":
        QMessageBox.information(self, "Увага", "Пошуковий запит порожній!")
    else:
        self.memberSearchEntry.setText("")

        query = ('SELECT * FROM members WHERE member_name LIKE ? or member_surname LIKE ? or member_phone LIKE
        results = cur.execute(query, ('%'+value+'%','%'+value+'%','%'+value+'%')).fetchall()
        if results == []:
            QMessageBox.information(self, "Увага!", "Працівника немає. Шукайте іншого.")
        else:
            for i in reversed(range(self.membersTableWidgets.rowCount())):
                self.membersTableWidgets.removeRow(i)
            for row_data in results:
                row_number = self.membersTableWidgets.rowCount()
                self.membersTableWidgets.insertRow(row_number)
                for column_number, data in enumerate(row_data):
                    self.membersTableWidgets.setItem(row_number, column_number, QTableWidgetItem(str(data)))
```

Рис. 3.14. Пошук користувача.

Реалізація пошуку користувача за ім’ям, прізвищем або телефоном. Звертаємось до таблиці за допомогою SQL-запиту ‘SELECT FROM’ де завдяки ‘WHERE’ встановлюємо фільтрацію даних. Беремо стовпці в яких відповідно записані ім’я, прізвища та телефони користувачів. Щоб знайти саме конкретну інформацію, в нашому випадку користувача, потрібно скористуватись SQL-запитом ‘LIKE’. Він дозволяє виконувати пошук за текстовим шаблоном в рядках стовпців. Ще тут використано метод fetchall(). Цей метод належить sqlite3 і використовується для отримання результату запиту.

```
if column == 10:
    query = ("SELECT * FROM products WHERE product_po=?")
    product = cur.execute(query, (self.id,)) # single item tuple=(1,)

    for i in product:
        if i[10] == 'Вибрано':
            self.newMemberWindow = takedItem.TakedItem(self.id)
            self.newMemberWindow.displayProducts()
        else:
            QMessageBox.information(self, "Увага", "Не було вибрано жодного товару {}")
```

Рис. 3.15. Вибір товару.

Тут продемонстрована реалізація вибору товару завдяки SQL-запитам до таблиці.

```
query_selected = ("SELECT product_id,description,product_manufacturer,product_name,product_quota,product_availability,
supplier,date_adding,product_price,product_po,picked_by FROM products WHERE picked_by='Picked'")

products = cur.execute(query_selected).fetchall()
```

Рис. 3.16. Вибір обраного продукту

```
query = ("SELECT product_id,description,product_manufacturer,product_name,product_quota,product_availability,supplier,
date_adding,product_price,product_po,picked_by FROM products WHERE product_availability='Available'")
products = cur.execute(query).fetchall()
```

Рис. 3.17. Доступний продукт

```
query = ("SELECT product_id,description,product_manufacturer,product_name,product_quota,product_availability,supplier,
date_adding,product_price,product_po,picked_by FROM products WHERE product_availability='UnAvailable'")
products = cur.execute(query).fetchall()
```

Рис. 3.18. Недоступний продукт

Запити для обраних продуктів у базі даних. Якщо ми оберемо продукт з поміткою ‘available’ то нам висвітить, що він доступний , у разі помітки ‘unavailable’ ми отримаємо “Не доступно”. Ось повні фрагменти коду включаючи запит:

```
def list_products(self):
    if self.allProducts.isChecked() == True:
        self.productsTable.sortItems(10, Qt.AscendingOrder)
        self.displayProducts()
    elif self.picked_items.isChecked() == True:
        self.productsTable.sortItems(10, Qt.AscendingOrder)
        query_selected = ("SELECT product_id,description,product_manufacturer,product_name,product_quota,product_availability,
        supplier,date_adding,product_price,product_po,picked_by FROM products WHERE picked_by='Picked'")

        products = cur.execute(query_selected).fetchall()

        for i in reversed(range(self.productsTable.rowCount())):
            self.productsTable.removeRow(i)
        for row_data in products:
            row_number = self.productsTable.rowCount()
            self.productsTable.insertRow(row_number)
            for column_number, data in enumerate(row_data):
                if str(row_data[5]) == "Не доступно":
                    self.productsTable.setItem(row_number, column_number, QTableWidgetItem(str(data)))

                    if column_number == 10:
                        continue
                    self.productsTable.item(row_number, column_number).setBackground(QColor(255, 204, 204))
                elif str(row_data[10]) == 'Вибрано' and str(row_data[5]) == "Доступно":
                    print("Picked")
                    self.productsTable.setItem(row_number, column_number, QTableWidgetItem(str(data)))
                    if column_number == 10:
                        continue
                    self.productsTable.item(row_number, column_number).setBackground(QColor(180, 190, 204))
```

Рис. 3.19. Picked.

```
def sell_picked_clicked(self, row, column):
    some = self.productsTable.item(row, 9)
    self.id = some.text()

    if column == 10:
        query = ("SELECT * FROM products WHERE product_po=?")
        product = cur.execute(query, (self.id,)) # single item tuple=(1,)

        for i in product:
            if i[10] == 'Вибрано':
                self.newMemberWindow = takedItem.TakedItem(self.id)
                self.newMemberWindow.displayProducts()
            else:
                QMessageBox.information(self, "Увага", "Не було вибрано жодного товару {}")
```

Рис. 3.20. Невибраний товар.

Exception на випадок невибраного товару.

```
elif self.availableProducts.isChecked() == True:
    self.productsTable.sortItems(10, Qt.AscendingOrder)
    query = ("SELECT product_id,description,product_manufacturer,product_name,product_quota,product_availability,supplier,
    date_adding,product_price,product_po,picked_by FROM products WHERE product_availability='Available'")
    products = cur.execute(query).fetchall()


    for i in reversed(range(self.productsTable.rowCount())):
        self.productsTable.removeRow(i)
    for row_data in products:
        row_number = self.productsTable.rowCount()
        self.productsTable.insertRow(row_number)
        for column_number, data in enumerate(row_data):
            if str(row_data[5]) == "НЕ доступно":
                self.productsTable.setItem(row_number, column_number, QTableWidgetItem(str(data)))
                if column_number == 10:
                    continue
                self.productsTable.item(row_number, column_number).setBackground(QColor(255, 204, 204))
            elif str(row_data[10]) == 'Вибрано' and str(row_data[5]) == "Доступно":
                print("Вибрано")
                self.productsTable.setItem(row_number, column_number, QTableWidgetItem(str(data)))
                if column_number == 10:
                    continue
                self.productsTable.item(row_number, column_number).setBackground(QColor(180, 190, 204))

            else:
                self.productsTable.setItem(row_number, column_number, QTableWidgetItem(str(data)))
```

Рис. 3.21. Available.

```
elif self.notAvailableProducts.isChecked() == True:
    self.productsTable.sortItems(10, Qt.AscendingOrder)
    query = ("SELECT product_id,description,product_manufacturer,product_name,product_quota,product_availability,supplier,
    date_adding,product_price,product_po,picked_by FROM products WHERE product_availability='UnAvailable'")
    products = cur.execute(query).fetchall()


    for i in reversed(range(self.productsTable.rowCount())):
        self.productsTable.removeRow(i)
    for row_data in products:
        row_number = self.productsTable.rowCount()
        self.productsTable.insertRow(row_number)
        for column_number, data in enumerate(row_data):
            self.productsTable.setItem(row_number, column_number, QTableWidgetItem(str(data)))
            if column_number == 10:
                continue
            self.productsTable.item(row_number, column_number).setBackground(QColor(255, 204, 204))
```

Рис. 3.22. Unavailable.

```
def delete_member(self):
    global memberId
    mbox = QMessageBox.information(self, "Увага!", "Ви впевнені, що хочете видалити цього учасника?", QMessageBox.Yes|QMessageBox.
    No, QMessageBox.No)

    if mbox == QMessageBox.Yes:
        try:
            query = "DELETE FROM members WHERE member_id=?"
            cur.execute(query, (memberId,))
            con.commit()
            self.main.displayMembers()
            QMessageBox.information(self, "Інформація!", "Користувача успішно видалено!")
            self.close()
        except:
            QMessageBox.information(self, "Інформація!", "Користувача не видалено")
```

Рис. 3.23. Delete member.

Видалення користувача з інформаційної системи за допомогою SQL-запиту DELETE FROM. Цей запит використовується для видалення одного або більше рядків з таблиці бази даних. Він дозволяє видалити вказані рядки, які відповідають певним умовам. В даному фрагменті продемонстровано реалізацію видалення користувача за його ID. Також тут задіяний метод ‘commit’. Він належить бібліотеці sqlite3 і використовується для підтвердження (збереження) змін, зроблених у поточній транзакції бази даних.

Коли ви виконуєте ряд операторів SQL (INSERT, UPDATE, DELETE тощо) в SQLite3, ці зміни не відразу відображаються у базі даних. Замість цього вони зберігаються в буфері. Команда commit використовується для фізичного збереження цих змін у базі даних [15]. Після команди commit зміни стають постійними і доступними для інших користувачів або процесів, які працюють з базою даних.

```
def update_member(self):
    global memberId
    name = self.nameEntry.text()
    surname = self.surnameEntry.text()
    phone = self.phoneEntry.text()

    if (name and surname and phone != ""):
        try:
            query = 'UPDATE members set member_name=?, member_surname=?, member_phone=? WHERE member_id=?'
            cur.execute(query, (name, surname,phone,memberId))
            con.commit()
            QMessageBox.information(self, "Інформація", "Продукт оновлено!")
        except:
            QMessageBox.information(self, "Інформація", "Продукт не оновлювався!")
    else:
        QMessageBox.information(self, "Інформація", "Будь ласка, заповніть усі поля!")
```

Рис. 3.24. UPDATE MEMBERS.

Цей код відповідає за обновлення даних про користувача у інформаційній системі. Використовується SQL-запит ‘UPDATE’, який призначено для обновлення полів.Якщо потрібно буде змінити ім’я, прізвище або телефон користувача то це можливо завдяки запиту ‘UPDATE’, який використовується з запитом ‘WHERE’, щоб звернутись до стовпця саме з користувачами.

```
def productDetails(self):
    global productId
    query = ("SELECT * FROM products WHERE product_id=?")
    product = cur.execute(query, (productId,)).fetchone()  # single item tuple=(1,)
    print(product)
```

Рис. 3.25. Деталі про продукт.

Даний уривок коду витягує дані про товар з таблиці та виводить на екран щоб користувач зміг побачити всі дані про вибраний ним товар. Використовується ‘SELECT’ для вибору даних і ‘WHERE’ для звернення до стовпця з товаром.

```
query = 'UPDATE products set description=?, supplier=?, product_name=?, product_manufacturer=?, product_price=?,
product_quota=?, product_img=?, product_availability=? WHERE product_id=?'
cur.execute(query, (description, supplier, name, manufacturer, price, quota, defaulImgSrc, status, productId))
cur.execute("UPDATE products set product_po=? WHERE product_id=?", (productId_po, productId))
con.commit()
print("Виконано!")
self.main.productsTable.sortItems(10, Qt.AscendingOrder)
self.main.displayProducts()
QMessageBox.information(self, "Інформація", "Продукт оновлено!")
```

Рис. 3.26. Оновлення продукту.

Запит для обновлення опису про товар. Тут до ‘UPDATE’ додано ще SQL-запит ‘SET’, який використовується для вказівки стовпців та їх нових значень, які потрібно оновити в базі даних.

```
def delete_product(self):
    global productId

    mbox = QMessageBox.question(self, "Увага!", "Ви впевнені, що хочете видалити цей продукт?", QMessageBox.Yes|QMessageBox.No,
    QMessageBox.No)

    if (mbox == QMessageBox.Yes):

        try:
            img_to_delete = cur.execute("SELECT product_img FROM products WHERE product_id=?",(productId,)).fetchone()
            print(img_to_delete[0])
            if img_to_delete[0] == "src/img/upload_new_img.png":
                pass
            else:
                os.unlink(img_to_delete[0])
            cur.execute("DELETE FROM products WHERE product_id=?", (productId,))
            con.commit()
            self.main.productsTable.sortItems(10, Qt.AscendingOrder)
            self.main.displayProducts()
```

Рис. 3.27. Видалення продукту.

Створено запит для видалення вибраного продукту з бази даних. Реалізовано обновлення після видалення, щоб не доводилось перезапускати програму для обновлення даних.

```
                QMessageBox.information(self, "Information!", "Product has been deleted!")
                self.main.update_to_DB.setIcon(QIcon('src/icons/updateToServer.png'))
                self.main.update_to_DB.setText("Оновити")
                self.close()

            except:
                QMessageBox.information(self, "Інформація!", "Товар не видалено!")
        else:
            QMessageBox.information(self, "Інформація!", "Товар не видалено!")
```

Рис. 3.28. Оновлення запису після видалення.

```
def add_member_btn(self):
    global defaulImg
    name = self.nameEntry.text()
    surname = self.surnameEntry.text()
    phone = self.phonenumberEntry.text()

    if (name and surname and phone != ""):
        print("Добре")
        self.chek_empl()
        try:
            existing_members = []
            chech_mem = cur.execute("SELECT member_name FROM members")
            for i in chech_mem:
                existing_members.append(str(i[0]))
            if str(name) in existing_members:
                self.nameEntry.setStyleSheet(style.qLineEditRed())
                QMessageBox.warning(self, "Увага!","Користувач {} ім'я вже зайнято. \nБудь ласка вибери
                ім'я".format(name))


            elif str(name) in str(self.empl_dc.keys()) and str(phone) in str(self.empl_dc.values()):
                print("User Exist")
                self.nameEntry.setStyleSheet(style.qLineEditRed())
                self.phonenumberEntry.setStyleSheet(style.qLineEditRed())
                QMessageBox.warning(self, "Увага!", "Користувач з ім'ям ='{}' і ID='{}' вже зайнято.
                \nБудь ласка виберіть унікальне Id і Ім'я.".format(name, phone))
```

```
            else:
                print("New User")
                query = "INSERT INTO 'members' (member_name,member_surname,member_phone,member_photo)
                VALUES(?,?,?,?)"
                cur.execute(query, (name, surname, phone, defaulImg))
                con.commit()
                self.main.displayMembers()
                self.nameEntry.setStyleSheet(style.search_btn_style_2())
                self.phonenumberEntry.setStyleSheet(style.search_btn_style_2())
                QMessageBox.information(self, "Інформація", "Користувача додано!")
                self.nameEntry.setText("")
                self.surnameEntry.setText("")
                self.phonenumberEntry.setText("")


        except:
            QMessageBox.warning(self, "Увага", "Користувача НЕ додано!")
    else:
        QMessageBox.warning(self, "Увага", "Поля не можуть бути пустими!")

    self.addMemberImg.setPixmap(QPixmap('src/icons/add_engineer.png'))

def chek_empl(self):
    self.empl_dc.clear()
    self.sq = cur.execute("SELECT member_name,member_phone FROM members").fetchall()
    for i in self.sq:
        self.empl_dc[i[0]] = i[1]
    print(self.empl_dc)
```

Рис. 3.29 і Рис. 3.30. Створення нового користувача.

Створення нового користувача з перевіркою на уже існуючих даних для запобігання повтору запису.

Змінна defaulImg використовується для доступу до зображення за замовчуванням. Отримуються значення з трьох полів введення nameEntry, surnameEntry і phonenumberEntry та зберігаються у відповідних змінних name, surname і phone. Перевіряється, чи всі три поля (name, surname, phone) не є порожніми рядками за допомогою умови name and surname and phone != "". Виконується перевірка наявності в базі даних імені name. Запит до бази даних cur.execute("SELECT member_name FROM members") виконується для отримання списку існуючих учасників. Якщо ім'я name вже міститься в списку existing_members, тоді встановлюється червоний стиль для nameEntry, а також виводиться попередження QMessageBox.warning() про те, що ім'я вже зайняте. Також виконується перевірка, чи ім'я name та номер телефону phone вже містяться в словнику self.empl_dc. Якщо ці дані вже існують, то встановлюються червоні стилі для nameEntry і phonenumberEntry, а також виводиться попередження QMessageBox.warning() про те, що вже існує користувач з таким ім'ям та ID. Формується запит query для вставки нового користувача в базу даних "members". Запит виконується за допомогою cur.execute() з передачею значень name, surname, phone та defaulImg в якості параметрів. Зберігаються зміни в базі даних за допомогою commit(). Викликається метод displayMembers() об'єкта self.main для оновлення відображення учасників. Виводиться інформаційне повідомлення про те, що користувача додано. Очищаються поля введення за допомогою self.nameEntry.setText(""), self.surnameEntry.setText("") і self.phonenumberEntry.setText(""). Виконується метод setPixmap() для об'єкта self.addMemberImg, що встановлює відповідне зображення. Метод chek_empl() очищає словник self.empl_dc і отримує дані з бази даних для заповнення цього словника.

```python
        defaulImg = self.unique_img_name

def add_product(self):
    global defaulImg
    print("Стандартне: \n",defaulImg)
    img_name = os.path.basename(defaulImg)
    defaulImgSrc = 'src/img/{}'.format(img_name)

    try:
        description = self.descriptionEntry.text()
        name = self.nameEntry.text()
        manufacturer = self.manufacturerEntry.text()
        supplier = self.supplier.text()
        #poNumber = int(self.PoNumber.text()) #TODO change
        #price = int(self.priceEntry.text()) #TODO change
        price = str(self.priceEntry.text()) #TODO change
        poNumber = str(self.PoNumber.text()) #TODO change
        qouta = self.quotaEntry.text()
        dayOfAdding = self.timeAndDateEdit.text()

        # Checking if fields are empty or not
        if (name and manufacturer and price and qouta and poNumber != ""):
            try:
                #price = int(price) #TODO Price integer now
                existing_products = []
                query = "INSERT INTO 'products' (description,product_manufacturer,product_name,supplier,product_price,product_quota,
                product_img,product_po,date_adding) VALUES(?,?,?,?,?,?,?,?,?)"
                check = cur.execute("SELECT product_po FROM products").fetchall()
```

```python
                for i in check:
                    existing_products.append(str(i[0]))
                if str(poNumber) in existing_products:
                    self.PoNumber.setStyleSheet(style.qLineEditRed())
                    QMessageBox.warning(self, "Увага!", "Продукт з {} id вже є. Будь ласка введіть унікальний Id".format(poNumber))
                else:
                    self.PoNumber.setStyleSheet(style.search_btn_style_2())
                    cur.execute(query, (description,manufacturer,name, supplier, price, qouta, defaulImgSrc, poNumber, dayOfAdding))
                    con.commit()
                    self.main.productsTable.sortItems(10, Qt.AscendingOrder)
                    self.main.displayProducts()
                    QMessageBox.information(self, "Інформація", "Продукт було додано")
                    self.main.update_to_DB.setIcon(QIcon('src/icons/updateToServer.png'))
                    self.main.update_to_DB.setText("Оновити")
                    self.nameEntry.setText("")
                    self.manufactuterEntry.setText("")
                    self.priceEntry.setText("")
                    self.PoNumber.setText("")
                    self.quotaEntry.setText("")
                    self.descriptionEntry.setText("")
                    self.manufacturerEntry.setText("")
                    self.supplier.setText("")
                    self.addProductImg.setPixmap(QPixmap('src/icons/upload_new_img.png'))

            except:
                QMessageBox.warning(self, "Інформація", "Правильно перевірте всі поля!")
        else:
            QMessageBox.warning(self, 'Інформація', "Будь ласка, заповніть усі поля. Поля не можуть бути порожніми!")
    except:
```

Рис. 3.31 і Рис. 3.32. Створення нового запису про товар.

```
        QMessageBox.warning(self, 'Інформація', "Будь ласка, введіть ціле число")

    print(defaulImg)
    #defaulImg = 'src/icons/upload_new_img.png'
    if defaulImg == 'src/icons/upload_new_img.png':
        copyfile(defaulImg, 'src/img/{}'.format(img_name))
        copyfile('src/icons/default_img.png', 'src/img/default_img.png')
        print("Copying: ", img_name)
    else:
        print("Else")
        try:
            img_name = os.path.basename(defaulImg)
            copyfile("db_database/semiland_database.db","database_backup/semiland_database.db")
            copyfile(defaulImg,'src/img/{}'.format(img_name))

            folder = 'src/current_pictures'
            for the_file in os.listdir(folder):
                file_path = os.path.join(folder, the_file)
                try:
                    if os.path.isfile(file_path):
                        os.unlink(file_path)
                except Exception as e:
                    QMessageBox.warning(self, "Увага!", "{}".format(e))

        except Exception as e:
            copyfile('src/icons/upload_new_img.png', 'src/img/upload_new_img.png')

    defaulImg = 'src/icons/upload_new_img.png'
```

Рис. 3.33. Створення нового запису про товар(1).

Даний фрагмент коду відповідає за обробку події додавання нового продукту в програмі.

Глобальна змінна defaulImg використовується для доступу до шляху до зображення за замовчуванням. Початкове значення defaulImg виводиться за допомогою функції print(). За допомогою функції os.path.basename() визначається ім'я файлу зображення за замовчуванням. Отримуються значення з полів введення descriptionEntry, nameEntry, manufacturerEntry, supplier, priceEntry, PoNumber, quotaEntry, timeAndDateEdit та зберігаються у відповідних змінних description, name, manufacturer, supplier, price, poNumber, quota, dayOfAdding. Перевіряється, чи всі поля заповнені (не є порожніми рядками) за допомогою умови name and manufacturer and price and quota and poNumber != "". Виконується внутрішня перевірка наявності у базі даних значення poNumber. Запит cur.execute("SELECT product_po FROM products") виконується для отримання списку існуючих номерів poNumber. Якщо poNumber вже міститься в списку existing_products, тоді встановлюється червоний стиль для PoNumber, а

також виводиться попередження про те, що продукт з таким номером вже існує. Виконується запит query для вставки нового продукту в базу даних "products" з використанням значень description, manufacturer, name, supplier, price, quota, defaulImgSrc, poNumber та dayOfAdding як параметрів. Зміни зберігаються в базі даних за допомогою con.commit(). Метод sortItems() викликається для відсортування таблиці productsTable за колонкою 10 в порядку зростання. Викликається метод displayProducts() об'єкта self.main для оновлення відображення продуктів.Виводиться інформаційне повідомлення про успішне додавання продукту. Поля введення nameEntry, manufactuterEntry, priceEntry, PoNumber, quotaEntry, descriptionEntry, manufacturerEntry, supplier очищаються. Метод setPixmap() викликається для об'єкта self.addProductImg, що встановлює відповідне зображення.

```
def displayProducts(self):

    for i in reversed(range(self.statisticTable.rowCount())):
        self.statisticTable.removeRow(i)

    query = cur.execute("SELECT * FROM transaction_history")
    for row_data in query:
        print(row_data[6])
        row_number = self.statisticTable.rowCount()
        self.statisticTable.insertRow(row_number)
```

Рис. 3.34 Історія транзакцій.

Даний фрагмент відображає історію транзакцій з виведенням усіх даних про товар.

```python
# TODO members
query2 = ("SELECT member_id, member_name FROM members")
members = cur.execute(query2).fetchall()
for member in members:
    self.memberCombo.addItem(member[1], member[0])

# TODO products
query1 = ("SELECT DISTINCT description FROM products WHERE product_availability=?")
prod_description = cur.execute(query1, ('Доступно',)).fetchall()

self.sorting_combos(self.productCombo)

for i in prod_description:
    self.productCombo.addItem(i[0])
```

Рис. 3.35. Вибір товару для продажу.

Даний фрагмет відповідає за вибір товар який ми хочемо продати. Проводиться перевірка чи є товар у наявності на складі.

```python
current_product = self.productCombo.currentText()
current_manufacturer = self.productManufacturerCombo.currentText()
current_model = self.productModelCombo.currentText()
query = ("SELECT DISTINCT product_manufacturer FROM products WHERE description=? AND
product_availability=?")
product_manufacturer = cur.execute(query, (current_product, 'Доступно')).fetchall()
self.productManufacturerCombo.clear()
for product_manufacturer2 in product_manufacturer:
    self.productManufacturerCombo.addItem(str(product_manufacturer2[0]))
```

Рис. 3.36. Випадаючий список.

Цей фрагмент коду відповідає за заповнення випадаючого списку зі списком виробників продуктів, що відповідають певним критеріям.

```
if memberName:
    try:
        cur.execute("UPDATE products SET picked_by=? WHERE description=? AND product_manufacturer=?
        AND product_name=?", ("Picked", description, product_manuf, product_model))
        self.empl_info_data = cur.execute(
            "SELECT picked_product_name,take_by,product_id FROM take_product WHERE
            picked_product_name=? AND take_by=?",
            (uniqueId_item, memberId)).fetchone()

        if not self.empl_info_data:
            cur.execute(
                "INSERT INTO 'take_product' (picked_product_name,take_by,product_id,items_amount,
                taker_name,time_day_picking,product_personal_name) VALUES(?,?,?,?,?,?,?)",
                (uniqueId_item,memberId,productId, quantity, memberName, datetime,
                prod_personal_name))
        elif str(self.empl_info_data[0] == str(uniqueId_item)):
            if str(self.empl_info_data[1]) == str(memberId):
                some_value = cur.execute("SELECT items_amount FROM take_product WHERE
                picked_product_name=? AND take_by=?", (uniqueId_item, memberId)).fetchone()
                new_value = int(some_value[0]) + quantity
                cur.execute("DELETE FROM take_product WHERE picked_product_name=? AND take_by=?",
                (uniqueId_item, memberId))
                cur.execute(
                    "INSERT INTO 'take_product' (picked_product_name,take_by,product_id,items_amount,
                    taker_name,time_day_picking,product_personal_name) VALUES(?,?,?,?,?,?,?)",
                    (uniqueId_item,memberId,productId,new_value,memberName,datetime,
                    prod_personal_name))
```

Рис. 3.37. Оновлення бази даних для різних інженерів.

Тут обновляються записи в базі даних в залежності якого інженера ми направим, адже ми закріпляєм за ним певний товар. Додає в список товар, який використовує інженер для зручності керування ним.

```
transactionQuery = (
    "INSERT INTO 'transaction_history' (item_name_transaction, item_po_transaction,
    employee_name_transaction, amount_transaction, date_of_transaction, action_transaction)
    VALUES (?,?,?,?,?,?)")
cur.execute(transactionQuery, (prod_personal_name, uniqueId_item, memberName, quantity,
datetime, 'Picked'))

self.qouta = cur.execute("SELECT product_quota FROM products WHERE product_id=?",
                         (productId,)).fetchone()

cur.execute("UPDATE members SET took_items=?, relation_to_items='Took' WHERE member_name=?",
            (quantity, memberName))

con.commit()

if (quantity == self.qouta[0]):
    updateQoutaQuery = ("UPDATE products set product_quota=?,product_availability=? WHERE
    product_id=?")
    cur.execute(updateQoutaQuery, (0, 'Не доступно', productId))
    con.commit()

else:
    newQouta = (self.qouta[0] - quantity)
    updateQoutaQuery = ("UPDATE products set product_quota=? WHERE product_id=?")
    cur.execute(updateQoutaQuery, (newQouta, productId))
    con.commit()
self.main.productsTable.sortItems(10, Qt.AscendingOrder)
self.main.displayProducts()
```

Рис. 3.38. Перевірка наявності.

Перевірка товару на наявність і чи може інженер ним скористатись.

Реалізовано вхід для різних користувачів.

```
def handlePassword(self):
    if (self.textName.text() == 'Логін' and self.textPass.text() == 'Пароль'):
        self.accept()
    else:
        QMessageBox.warning(self, 'Помилка!', 'Неправильний користувач або пароль!')
```

Рис. 3.39. Логін меню.

Даний фрагмент коду відповідає за обробку події, яка відбувається при натисканні кнопки аутентифікації користувача.

Отримуються значення введених користувачем імені та пароля з полів введення textName та textPass відповідно. Виконується перевірка, чи введені значення імені та пароля відповідають заданим значенням ('Логін' і 'Пароль' відповідно). Якщо введені значення збігаються з заданими, викликається метод accept(), що призводить до закриття вікна аутентифікації. У протилежному випадку виводиться повідомлення про помилку за допомогою, що повідомляє користувача про неправильні дані входу.

Для створення серверу використовувався модуль socket, який застосовується у Python для написання мережевих серверів. Цей модуль забезпечує доступ до інтерфейсу BSD socket. Він доступний у всіх сучасних системах Unix, Windows, MacOS і, можливо, на додаткових платформах.

```
# Create a socket allows to connect
def socket_create():
    try:
        global host
        global port
        global s
        host = ''
        port = 9999
        s = socket.socket()
    except socket.error as msg:
        print("Помилка створення сокета: ", str(msg))
```

Рис. 3.40. Створення сокету.

Даний фрагмент коду відповідає за створення сокету для з'єднання. Спочатку оголошуються змінні host, port і s.

Змінні host та port ініціалізуються. У даному випадку host встановлюється як порожній рядок, що означає, що сокет буде приймати з'єднання на всіх доступних мережевих інтерфейсах, а port встановлюється на значення 9999. Викликається функція socket.socket(), яка створює новий сокет і присвоює його об'єкту s. Цей сокет буде використовуватися для з'єднання з іншими сокетами. У

випадку виникнення помилки при створенні сокету, виводиться повідомлення про помилку.

```
# Bind socket to port and wait for connection from client
def socket_bind():
    try:
        global host
        global port
        global s
        #print("Binding socket to port: ", str(port))
        s.bind((host, port))
        s.listen(1)
    except socket.error as msg:
        QMessageBox.warning(None, 'Увага', "Помилка зв'язування сокета: " + str(msg) +"\n"+"Повторити...")
        socket_bind()
```

Рис. 3.41. Очікування з’єднання.

Даний уривок відповідає за прив'язку сокету до порту і очікування з'єднання від клієнта.

Викликається функція bind() для прив'язки сокету s до вказаного host і port. Це дозволяє сокету приймати з'єднання на вказаному порту. Викликається функція listen(), яка встановлює сокет у режим очікування з'єднань. Параметр 1 вказує максимальну кількість з'єднань, які можуть бути прийняті. У випадку виникнення помилки при зв'язуванні сокета, виводиться повідомлення про помилку і викликається функція socket_bind() для повторної спроби зв'язування сокета.

```
# Establishing a connection with client (socket must be a listening for them)
def socket_accept():
    global conn
    global address
    conn, address = s.accept()
    #print("Connection has been established | " +" IP " + address[0] + " | Port "+str(address[1]))
    send_commands(conn)
    conn.close()

def close_server():
    global conn
    global s
    conn.close()
    s.close()
    sys.exit()
```

Рис. 3.42. Встановлення з’єднання.

Даний фрагмент коду відповідає за встановлення з'єднання з клієнтом після того, як сокет встановлено в режим прослуховування.

Оголошуються змінні conn і address. Викликається функція accept(), яка очікує з'єднання від клієнта. Функція accept() блокує виконання програми, поки не буде отримано з'єднання. Після успішного встановлення з'єднання з клієнтом, об'єкти conn і address отримують значення, пов'язані з підключенням. Об'єкт conn представляє сокет з'єднання з клієнтом, а address містить IP-адресу та порт клієнта. Викликається функція send_commands(), яка відправляє команди до клієнта через встановлене з'єднання. Після завершення обміну даними з клієнтом, з'єднання закривається за допомогою методу close().

```
# Send commands
def send_commands(conn):
    conn.send(str.encode('dsb.py'))
    conn.send(str.encode('dir'))
    client_response = str(conn.recv(1024), "utf-8")
    #print(client_response, end="")



def main_module():
    socket_create()
    socket_bind()
    socket_accept()
```

Рис. 3.43. Відправлення команд.

Викликається метод send() для відправки команди на клієнт. Команди кодуються у вигляді рядка байтів за допомогою str.encode() перед відправкою. Після відправки команди, очікується відповідь від клієнта за допомогою методу recv(). В даному випадку, очікується відповідь. Опрацьована відповідь може бути виведена на екран або використана для подальшої обробки.

Для перевірки стану з'єднання з сервером створено клас ConnectingUpdating.

```
@pyqtSlot(str)
def setData(self, data):
    # print(data)
    self.spinner.stop()
    self.adjustSize()
    QMessageBox.information(self, "Успішно!", "Оновлено успішно!")
    self.main_table.update_to_DB.setIcon(QIcon('src/icons/update.png'))
    self.main_table.update_to_DB.setText("Оновлено")
    with open('database_backup/state.txt', 'w') as f:
        f.write('some')
    self.close()
```

Рис. 3.44. Оновлення сервера.

```
@pyqtSlot(str)
def setData_two(self, data):
    # print(data)
    self.spinner.stop()
    self.adjustSize()
    QMessageBox.warning(self, "Увага!", "Сервер впав дзвоніть СІСЬ адміну, а він зараз пішов на обід!")
    self.close()
```

Рис. 3.45. Exception про несправність сервера.

```python
class ConnectingUpdating(QWidget):
    def __init__(self, main_table):
        super().__init__()
        self.setWindowTitle("З'єднання з сервером")
        self.setWindowIcon(QIcon("src/icons/return.png"))
        self.setGeometry(650, 300, 250, 250)
        self.setFixedSize(self.size())
        self.spinner = QtWaitingSpinner()
        self.main_table = main_table
        self.ui()
        self.show()

    def ui(self):
        self.widgets()
        self.layouts()

    def widgets(self):
        self.ipAddressInputField = QLineEdit()
        self.ipAddressInputField.setText("192.168.1.11")

        self.ipAddressInputField.setStyleSheet('QLineEdit{border-color: #A3C1DA; font-size: 15pt; font:
        Liberation Mono}')
        self.portInputField = QLineEdit()
        self.portInputField.setText("9999")
        self.portInputField.setStyleSheet('QLineEdit{border-color: #A3C1DA; font-size: 15pt; font: Liberation
        Mono}')
        self.submitBtn = QPushButton("Submit")
        self.submitBtn.setStyleSheet('QPushButton{background-color:#ffd080;border-style:solid;
        border-width:2px;border-radius:10px;border-color:beige;font:14px;padding:6px;min-width:6em;
        font-family:Liberation Mono;}')
```

Рис. 3.46. Клас ConnectingUpdating.

3.2. Тестування програми

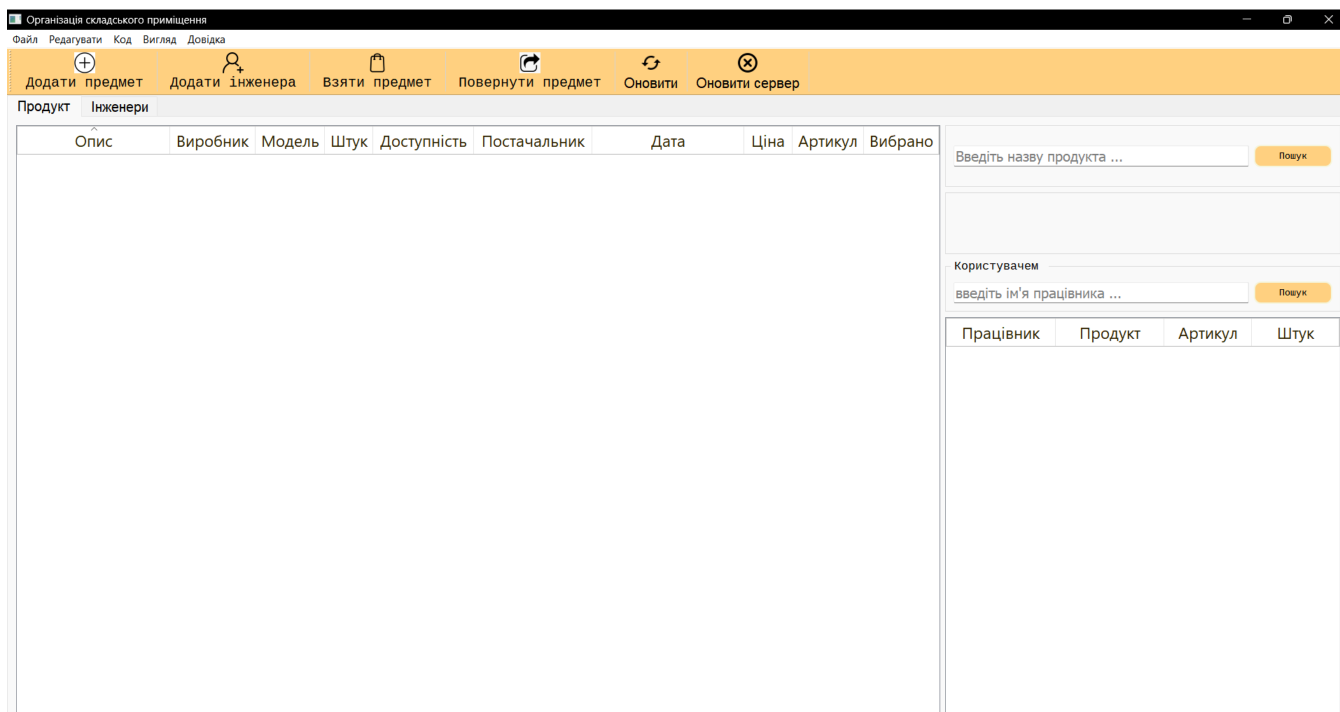


Рис. 3.47. Головний екран.

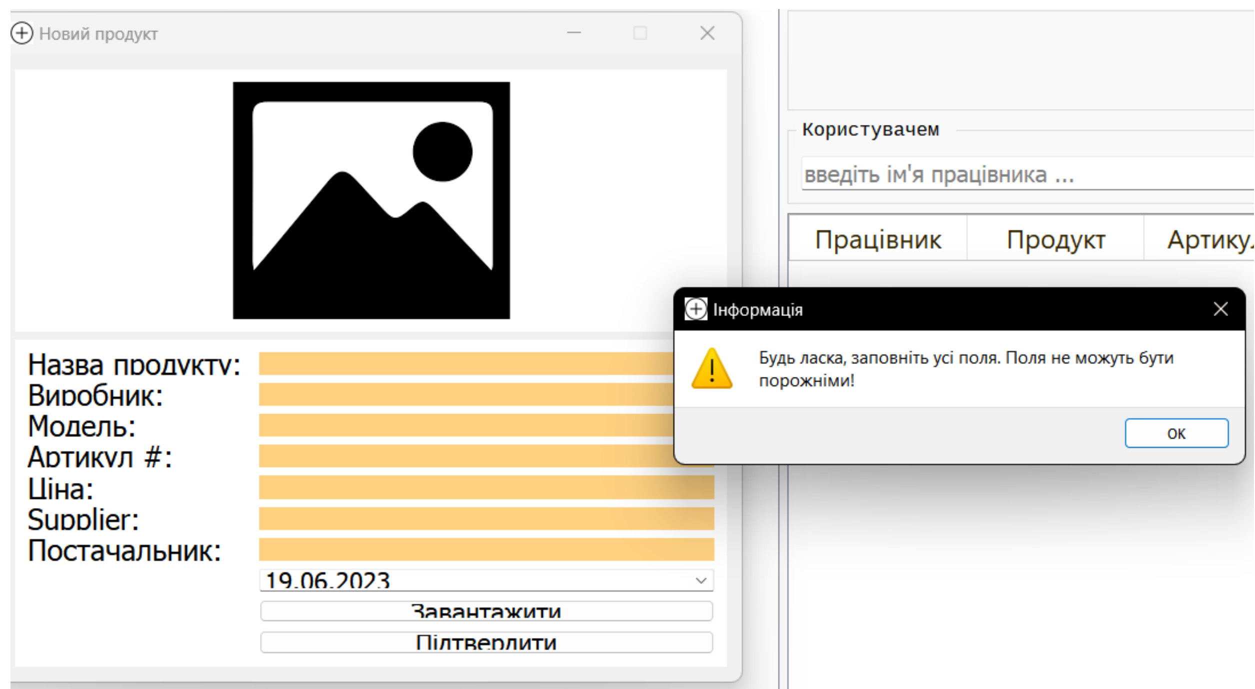


Рис. 3.48. Додавання Продукту.

Як ми бачимо якщо залишити поля пустими на не пропускає далі.

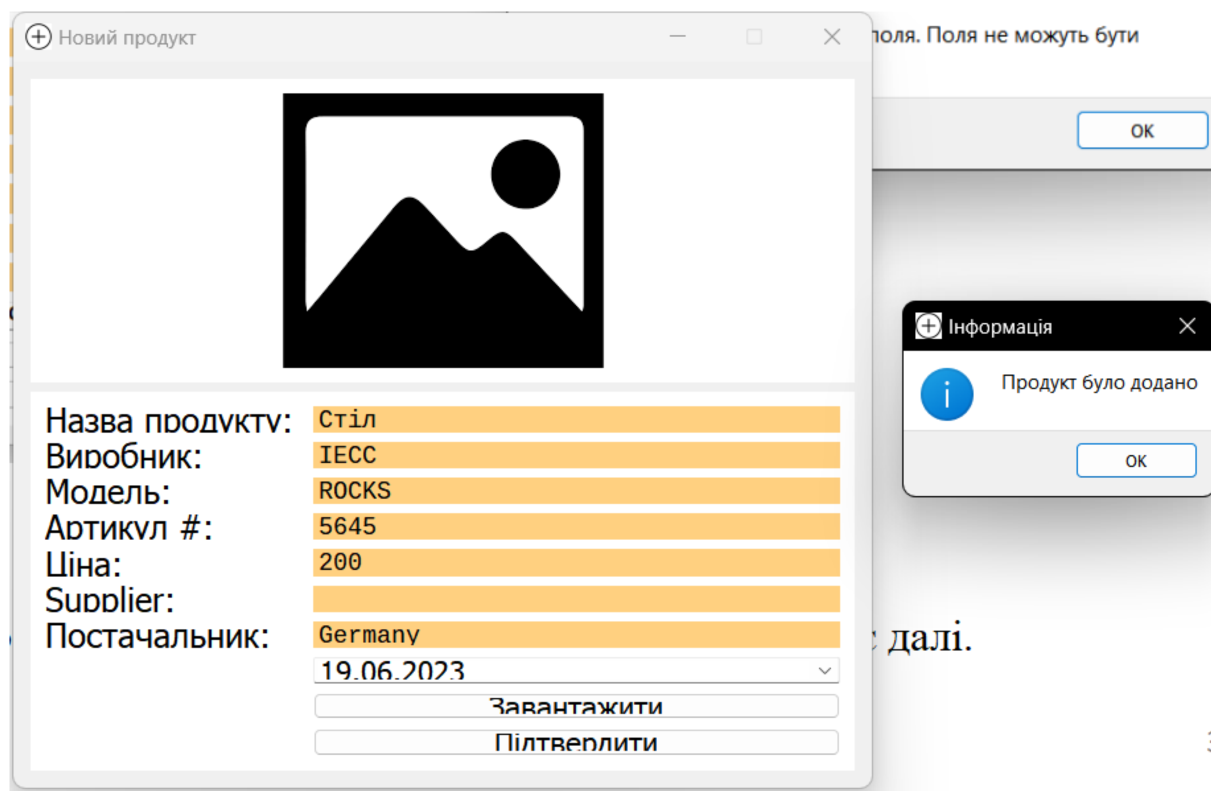


Рис. 3.49. Додавання заповненого продукту.

Продукт | Інженери

	Опис	Виробник	Модель	Штук	Доступність	Постачальник	Дата	Ціна	Артикул	Вибрано
1	Стіл	IECC	ROCKS	Germany	Available		19.06.2023	200	5645	None

Стіл | Пошук

Рис. 3.50. Реалізація пошуку.

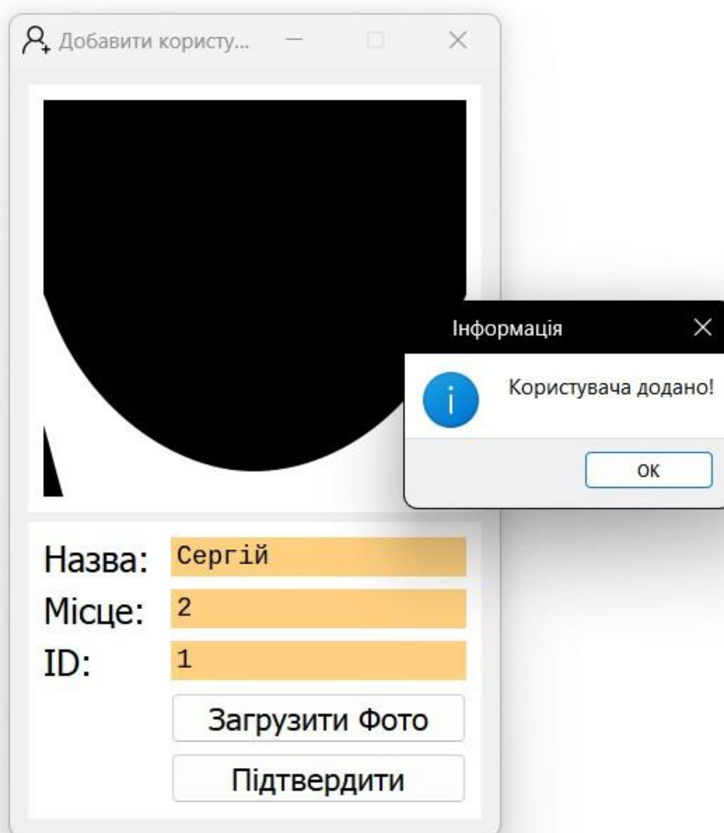


Рис. 3.51. Додавання користувача(інженера)

ВИСНОВКИ

У дипломній роботі реалізовано клієнт-серверну інформаційну систему складського обліку товарів (BackEnd). Під час реалізації було використано бібліотеку PyQt5, а також класи для неї. Базу данних було створено за допомогою sqlitE3. У програмі реалізовано редагування продуктів також є функціонал додавання працівників складу. Інтерфейс програми дозволяє у зручній формі взаємодіяти працівникам складу із продуктами та керувати ними. Реалізовано пошук по базі даних, створення і завантаження бази даних, оновлення та редагування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python SQLite tutorial using sqlite3 [Електроний ресурс] - Режим доступу до ресурсу. https://pynative.com/python-sqlite/ - доступний station на 10.06.2023.
2. socket - Low-level networking interface [Електроний ресурс] - Режим доступу до ресурсу. https://docs.python.org/uk/3.9/library/socket.html - доступний станом на 10.06.2023.
3. Mark Summerfield, Rapid GUI Programming with Python and Qt: The Definitive Guide to PyQt Programming, 2007.
4. Gerardus Blokdyk, SQLite Complete Self-Assessment Guide, 5STARCooks , 2021.
5. sqlite3 - DB-API 2.0 interface for SQLite databases [Електроний ресурс] - Режим доступу до ресурсу. https://docs.python.org/3/library/sqlite3.html - доступний станом на 10.06.2023.
6. Alan D. Moore, Python GUI Programming - A Complete Reference Guide, 2018.
7. Python Documentation [Електроний ресурс] - Режим доступу до ресурсу. https://www.python.org/ - доступний станом на 10.06.2023.
8. Burkhard Meier, Python GUI Programming Cookbook: Develop functional and responsive user interfaces with tkinter and PyQt5, 2015.
9. Lutz M., Learning Python, 5th ed, O’Reilly Media, 2022.
10. Beaulieu A., Learning SQL: Generate, Manipulate, and Retrieve Data, 3rd ed, O’Reilly Media, 2020.
11. Blokdyk G., SQLite Database Programming for Python: Build a Database Centric Application, 5STARCooks, 2021.
12. Beaulieu A., Learning SQL: Generate, Manipulate, and Retrieve Data, 3rd ed, O’Reilly Media, 2020.
13. Sweigart A., Automate the Boring Stuff with Python, 2nd ed, No Starch Press, 2020.

14. About SQLite [Електроний ресурс] - Режим доступу до ресурсу. https://www.sqlite.org/about.html - доступний станом на 10.06.2023.
15. Atomic Commit In SQLite [Електроний ресурс] - Режим доступу до ресурсу. https://sqlite.org/atomiccommit.html - доступний станом на 10.06.2023.

ДОДАТКИ

```
import socket

from PyQt5.QtWidgets import *

import sys




# Create a socket allows to connect

def socket_create():

    try:

        global host

        global port

        global s

        host = ''

        port = 9999

        s = socket.socket()

    except socket.error as msg:

        print("Помилка створення сокета: ", str(msg))


# Bind socket to port and wait for connection from client

def socket_bind():

    try:

        global host
```

```
        global port

        global s

        #print("Binding socket to port: ", str(port))

        s.bind((host, port))

        s.listen(1)

    except socket.error as msg:

        QMessageBox.warning(None, 'Увага', "Помилка зв’язування сокета: " + str(msg) +"\n"+"Повторити...")

        socket_bind()


# Establishing a connection with client (socket must be a listening for them)

def socket_accept():

    global conn

    global address

    conn, address = s.accept()

    #print("Connection has been established | " +" IP " + address[0] + " | Port "+str(address[1]))

    send_commands(conn)

    conn.close()


def close_server():

    global conn
```

```
    global s
    conn.close()
    s.close()
    sys.exit()

# Send commands
def send_commands(conn):
    conn.send(str.encode('dsb.py'))
    conn.send(str.encode('dir'))
    client_response = str(conn.recv(1024), "utf-8")
    #print(client_response, end="")


def main_module():
    socket_create()
    socket_bind()
    socket_accept()
```

```
import os
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
import sqlite3
import style


con = sqlite3.connect("db_database/main_database.db")
cur = con.cursor()


class StatisticClass(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Операції")
        self.setWindowIcon(QIcon('src/icons/power_module.png'))
        self.setGeometry(300, 200, 1500, 700)
        self.ui()
        self.show()

    def ui(self):
```

```
        self.widgets()
        self.layouts()
        self.displayProducts()

    def widgets(self):
        self.statisticTable = QTableWidget()

self.statisticTable.horizontalHeader().setStyleSheet(style.horizontalHeaderView())
        self.statisticTable.setStyleSheet(style.forQTabWidget())
        self.statisticTable.setColumnCount(7)
        self.statisticTable.setColumnHidden(0, True)
        self.statisticTable.setHorizontalHeaderItem(0, QTableWidgetItem("ID
Операції"))
        self.statisticTable.setHorizontalHeaderItem(1, QTableWidgetItem("Назва
Продукту"))
        self.statisticTable.setHorizontalHeaderItem(2,
QTableWidgetItem("Артикул"))
        self.statisticTable.setHorizontalHeaderItem(3,
QTableWidgetItem("Працівник"))
        self.statisticTable.setHorizontalHeaderItem(4, QTableWidgetItem("Сума"))
        self.statisticTable.setHorizontalHeaderItem(5, QTableWidgetItem("Дата та
Час"))
        self.statisticTable.setHorizontalHeaderItem(6, QTableWidgetItem("Дія"))
```

```
        self.statisticTable.horizontalHeader().setSectionResizeMode(5,
QHeaderView.Stretch)
        self.statisticTable.horizontalHeader().setSectionResizeMode(1),
QHeaderView.Stretch
        self.statisticTable.horizontalHeader().setSectionResizeMode(2,
QHeaderView.Stretch)
        self.statisticTable.setFont(QFont("Times",  10))



    def layouts(self):
        self.mainLayout = QHBoxLayout()
        self.childLayout = QVBoxLayout()
        self.childLayout.addWidget(self.statisticTable)
        self.mainLayout.addLayout(self.childLayout)
        self.setLayout(self.mainLayout)

    def displayProducts(self):


        for i in reversed(range(self.statisticTable.rowCount())):
            self.statisticTable.removeRow(i)


        query = cur.execute("SELECT * FROM transaction_history")
        for row_data in query:
```

```
print(row_data[6])
row_number = self.statisticTable.rowCount()
self.statisticTable.insertRow(row_number)
for column_number, data in enumerate(row_data):


    if row_data[6] == 'Вибрано':
        self.statisticTable.setItem(row_number, column_number, QTableWidgetItem(str(data)))
        self.statisticTable.item(row_number, column_number).setBackground(QColor(255, 204, 204))
    elif row_data[6] == 'Повернуто':
        self.statisticTable.setItem(row_number, column_number, QTableWidgetItem(str(data)))
        self.statisticTable.item(row_number, column_number).setBackground(QColor(204, 229, 255))
```