ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка застосунку для обліку товарів у продуктовому міні-маркеті мовою Python»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Дар'я КОНОНЕНКО
(підпис)

Виконав: здобувачка вищої освіти групи ПД-41

Дар'я КОНОНЕНКО

Керівник: Олег ІЛЬЇН
д.т.н., професор

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кононенко Дар'ї Анатоліївні

1. Тема кваліфікаційної роботи: «Розробка застосунку для обліку товарів у продуктовому міні-маркеті мовою Python»
керівник кваліфікаційної роботи д.т.н., проф., професор кафедри ІПЗ Олег ІЛЬЇН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.

2. Строк подання кваліфікаційної роботи «28» травня 2024 р.

3. Вихідні дані до кваліфікаційної роботи: технічні вимоги до додатку для обліку товарів у продуктовому міні-маркеті «Магазин "Даринка"», опис необхідних функціональних можливостей додатку, вимоги до використання баз даних та інтеграції з Телеграм-ботом.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз вимог та проектування архітектури додатку для обліку товарів у продуктовому міні-маркеті.

2. Проектування структури бази даних та моделей даних для зберігання інформації про товари, продажі, борги та боржників.

3. Програмна реалізація додатку для обліку товарів у продуктовому міні-маркеті «Магазин "Даринка"» з використанням PyQt, SQLite та інтеграцією з Телеграм-ботом.

4. Тестування додатку та Телеграм-бота.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до програмного забезпечення.
3. Програмні засоби реалізації.
4. Діаграма варіантів використання.
5. Діаграма класів.
6. Схема бази даних.
7. Граф діалогів Telegram-боту.
8. Екранні форми.
9. Апробація результатів дослідження

6. Дата видачі завдання «28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-13.03.2024	
3	Обґрунтування вибору мови програмування, бібліотек та технологій для розробки додатку	14.03-20.03.2024	
4	Проектування архітектури додатку для обліку товарів та моделювання програмного забезпечення	21.03-01.04.2024	
5	Програмна реалізація додатку	02.04-22.04.2024	
6	Тестування додатку	23.04-28.04.2024	
7	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
8	Розробка демонстраційних матеріалів	06.05-12.05.2024	
9	Попередній захист роботи	13.05-31.05.2024	

Здобувачка вищої освіти _____ Дар’я КОНОНЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Олег ІЛЬЇН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 41 стор., 4 табл., 19 рис., 15 джерел.

Мета роботи – спрощення процесу обліку товарів у продуктовому міні-маркеті за рахунок використання застосунку, розробленого мовою Python.

Об'єкт дослідження – процес обліку товарів у продуктовому міні-маркеті.

Предмет дослідження – програмне забезпечення для автоматизації обліку товарів у продуктовому міні-маркеті.

Короткий зміст роботи: Робота присвячена розробці додатку для обліку товарів у продуктовому міні-маркеті «Магазин "Даринка"» на мові Python з використанням PyQt та SQLite. У результаті виконання кваліфікаційної роботи було розроблено додаток «Магазин "Даринка"» для автоматизації обліку товарів, продажів та роботи з боржниками у продуктовому міні-маркеті. Проведено аналіз предметної області обліку товарів у продуктових міні-маркетах, визначено ключові бізнес-процеси та вимоги до додатку. Досліджено та проаналізовано існуюче програмне забезпечення для автоматизації роздрібної торгівлі, визначено їх переваги та недоліки. Розроблено архітектуру та спроектовано основні модулі і компоненти додатку. Реалізовано функціональність додатку, яка включає облік товарів, управління продажами, роботу з боржниками, ведення історії змін та формування звітів. Також реалізовано інтеграцію з Телеграм-ботом для нагадування боржникам про їхні борги. Проведено тестування додатку з використанням різних видів тестів, що забезпечує його якість та відповідність вимогам.

Сферою використання застосунку є роздрібна торгівля продуктовими товарами.

КЛЮЧОВІ СЛОВА: ОБЛІК ТОВАРІВ, ПРОДАЖІ, БОРГИ, БАЗА ДАНИХ, ТЕЛЕГРАМ-БОТ, PyQt, SQLite.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

PyQt - набір бібліотек Python, що дозволяють створювати графічний інтерфейс користувача (GUI) за допомогою інструментарію Qt.

SQLite - вбудована реляційна база даних, що реалізує більшість стандарту SQL і доступна у вигляді окремої бібліотеки.

GUI – Graphical User Interface (графічний інтерфейс користувача).

ВСТУП

У сучасному світі спостерігається зростаючий попит на автоматизовані рішення для управління різними бізнес-процесами та операціями. Одним з напрямків, який потребує впровадження інноваційних рішень, є облік товарів у роздрібній торгівлі, зокрема в продуктових міні-маркетах. Традиційні методи обліку запасів, продажів та роботи з боржниками є трудомісткими, схильними до помилок та неефективними з точки зору управління товарними потоками.

У зв'язку з цим, розробка спеціалізованого програмного забезпечення для автоматизації обліку товарів у продуктових міні-маркетах стає актуальним завданням. Такий додаток має потенціал значно спростити процес обліку надходження товарів, відстеження залишків, здійснення продажів, роботи з боржниками та генерації звітності. Це дозволить підвищити ефективність управління міні-маркетом, знизити ризики втрати даних та помилок, а також забезпечити зручний інтерфейс для співробітників.

Об’єкт дослідження – процес обліку товарів у продуктовому міні-маркеті.

Предмет дослідження – програмне забезпечення для автоматизації обліку товарів у продуктовому міні-маркеті.

Мета роботи – спрощення процесу обліку товарів у продуктовому міні-маркеті за рахунок використання застосунку, розробленого мовою Python.

Для досягнення поставленої мети необхідно виконати наступні задачі:

1. Провести аналіз процесів обліку товарів у продуктових міні-маркетах.

2. Виконати огляд існуючих програмних рішень для управління запасами та обліку продажів у роздрібній торгівлі. Визначити їх переваги та недоліки.

3. Провести огляд та в вибір засобів реалізації програмного забезпечення, інструментів та технологій розробки.

4. Спроектувати та розробити програмний додаток для обліку товарів у продуктовому міні-маркеті мовою Python з використанням PyQt та SQLite:

• реалізувати ключову функціональність внесення, редагування, видалення товарів, обліку надходжень та залишків;

• забезпечити можливість продажу товарів з підтримкою різних способів оплати та роботою з боржниками;

• інтегрувати додаток з Telegram Bot API для надсилання нагадувань боржникам;

• реалізувати формування звітності про продажі, оплати, залишки товарів та управління змінами роботи.

5. Виконати тестування та налагодження розробленого програмного забезпечення.

6. Розробити інструкцію користувача для ефективного використання створеного додатку.

Практична новизна роботи полягає у наданні специфічного функціоналу, відсутнього в аналогічних застосунках, зокрема, робота з боржниками та нагадування клієнтам магазину про борг через Телеграм-бот.

Робота пройшла апробацію на всеукраїнських конференціях: Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ», (24.04.2024, ДУІКТ, м. Київ); Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті» (15.05.2024, ДУІКТ, м. Київ). За результатами участі опубліковано тези доповідей.

1 АНАЛІЗ ПРОЦЕСІВ ОБЛІКУ ТОВАРІВ У ПРОДУКТОВИХ МІНІ-МАРКЕТАХ ТА ОГЛЯД ІСНУЮЧИХ ДОДАТКІВ ДЛЯ ОБЛІКУ ТОВАРІВ

1.1 Особливості обліку товарів у продуктовому міні-маркеті

У продуктовому міні-маркеті відбувається низка процесів, пов'язаних з обліком товарів. Перш за все, необхідно забезпечити надходження товарів на склад від постачальників. Це включає приймання товарів, перевірку їх кількості та якості, внесення інформації про нові надходження до системи обліку.

Після надходження товарів на склад їх необхідно належним чином зберігати та контролювати залишки. Це передбачає регулярну інвентаризацію для виявлення розбіжностей між фактичними залишками та записами в системі обліку. Також важливо відстежувати терміни придатності продуктів та вживати заходів для своєчасної реалізації товарів, що наближаються до закінчення терміну придатності.

Ключовим процесом є здійснення продажів товарів покупцям. Це включає пошук необхідних товарів, фіксацію кількості проданих одиниць, розрахунок загальної вартості, прийняття оплати та видачу решти. Важливо також враховувати різні способи оплати, такі як готівка, безготівковий розрахунок або надання товару в борг.

Робота з боржниками є специфічною особливістю продуктових міні-маркетів. Необхідно вести облік осіб, які взяли товари в борг, відстежувати суми їхніх боргів та забезпечувати своєчасне погашення заборгованості. Це може включати нагадування боржникам про необхідність оплати та застосування відповідних заходів у разі прострочення боргу.

Наприкінці робочого дня або зміни важливо підбивати підсумки та формувати звітність. Це включає підрахунок загальної виручки, розподіл її за способами оплати, зіставлення продажів зі змінами залишків на складі та виявлення можливих розбіжностей чи помилок.

Усі ці процеси традиційно виконуються вручну або з використанням простих інструментів, таких як таблиці, що створює ризики помилок, втрати даних та ускладнює управління товарними потоками. Саме тому розробка спеціалізованого програмного забезпечення для автоматизації цих процесів є актуальним завданням для підвищення ефективності роботи міні-маркету.

1.2 Специфіка додатків для обліку товарів у продуктовому міні-маркеті

Додаток для обліку товарів у продуктовому міні-маркеті - це спеціалізоване програмне забезпечення, призначене для автоматизації та спрощення процесів управління товарними запасами, продажами та роботою з боржниками в невеликих торгових точках.

Ключовими функціями такого додатку є можливість вносити, редагувати та видаляти інформацію про товари в базі даних, включаючи назву, тип, штрих-код, ціну та акцизний збір. Додаток дозволяє відстежувати надходження товарів на склад та контролювати поточні залишки.

Одним з основних модулів є система продажів, яка забезпечує пошук товарів за назвою або скануванням штрих-коду, вибір необхідної кількості та способу оплати (готівка, картка або борг). Додаток також має функціонал для ведення бази даних боржників, внесення нових записів, погашення боргів та інтеграцію з Телеграм-ботом для надсилання нагадувань.

Крім того, додаток для обліку товарів у міні-маркеті дозволяє формувати звіти про продажі та перегляд історії змін.

Загалом, додаток для обліку товарів у продуктовому міні-маркеті є комплексним рішенням, що автоматизує ключові бізнес-процеси, пов'язані з торгівлею, запасами та роботою з клієнтами. Його застосування дозволяє підвищити ефективність управління міні-маркетом, знизити ризики помилок та забезпечити точність і актуальність облікових даних.

1.3 Аналіз існуючих додатків для обліку товарів у продуктовому міні-маркеті

Аналіз існуючого програмного забезпечення для обліку товарів у продуктових міні-маркетах є важливим етапом у розробці нового додатку. Це дозволяє зрозуміти, які функції та можливості вже реалізовані в інших системах, а також виявити їхні недоліки та проблеми, щоб врахувати їх під час створення власного рішення. Найбільш поширеними додатками для автоматизації роздрібної торгівлі є Dilovod, HugeProfit та “Квітка Торгівля”.

Ретельне вивчення наявних на ринку програмних продуктів дає змогу визначити їхні сильні та слабкі сторони, функціональні можливості та обмеження.

Це допоможе сформувати чітке бачення необхідних характеристик майбутнього додатку для обліку товарів, щоб він міг ефективно задовольнити потреби міні-маркетів та усунути виявлені недоліки існуючих рішень.

Крім того, аналіз конкурентів дозволить виділити унікальні риси нового програмного забезпечення, які допоможуть йому вигідно відрізнятися на ринку та привернути увагу потенційних користувачів.

Таким чином, ретельне опрацювання вже наявних програмних рішень для автоматизації роздрібної торгівлі є необхідним кроком для створення якісного та конкурентоспроможного продукту, який враховуватиме як сильні сторони, так і недоліки існуючих аналогів.

1.3.1 Додаток Dilovod

Додаток Dilovod[1], як і всі програми для обліку товарів, має простий та зручний інтерфейс. Є можливість завантажити номенклатуру з файлу Excel. Додаток дозволяє вести повний облік товарних залишків, відстежувати надходження та реалізацію товарів, здійснювати інвентаризацію. Функціонал програми включає ведення клієнтської бази, формування замовлень, рахунків та накладних. Dilovod також надає аналітичні інструменти у вигляді різноманітних

звітів та графіків для відстеження продажів, найбільш популярних товарів та іншого.

Перевагами Dilovod є підтримка різних схем оподаткування, можливість роботи з торговими точками, підтримка штрих-кодів та терміналів збору даних. Додаток інтегрується з фіскальними реєстраторами, підтримує експорт/імпорт даних. Екранна форма додатку Dilovod наведена на рисунку 1.1.

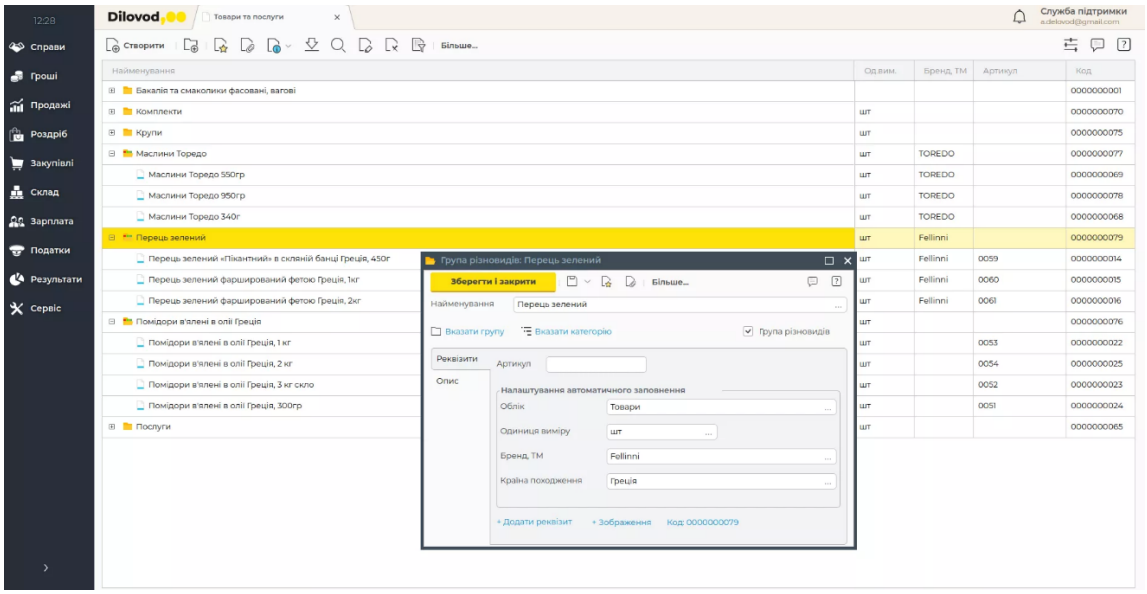


Рис. 1.1 Приклад екранної форми додатку Dilovod

1.3.2 Додаток HugeProfit

HugeProfit[2] - це програмне забезпечення для управління товарним бізнесом. Воно пропонує інструменти для контролю запасів, обробки замовлень, продажів, фінансового обліку та роботи з клієнтською базою.

Основні можливості:

- Управління запасами на складах та торгових майданчиках, оформлення поставок, інвентаризація.
- Централізована обробка замовлень з різних джерел, формування супровідних документів.
- Облік доходів, витрат, рахунків, переказів коштів.
- Ведення бази клієнтських даних, підтримка програм лояльності.

– Управління співробітниками, контроль продуктивності.
– Звіти по продажах, прибутковості, співробітниках.

Програма має стандартний функціонал для автоматизації товарного бізнесу. Інтерфейс виглядає типовим для такого роду ПЗ. Підтримуються штрих-коди та фіскальні принтери.

Перевагами можна назвати централізоване управління, але є і недоліки - складність інтеграції з іншими системами, необхідність навчання персоналу. Екранна форма додатку HugeProfit наведена на рисунку 1.2.

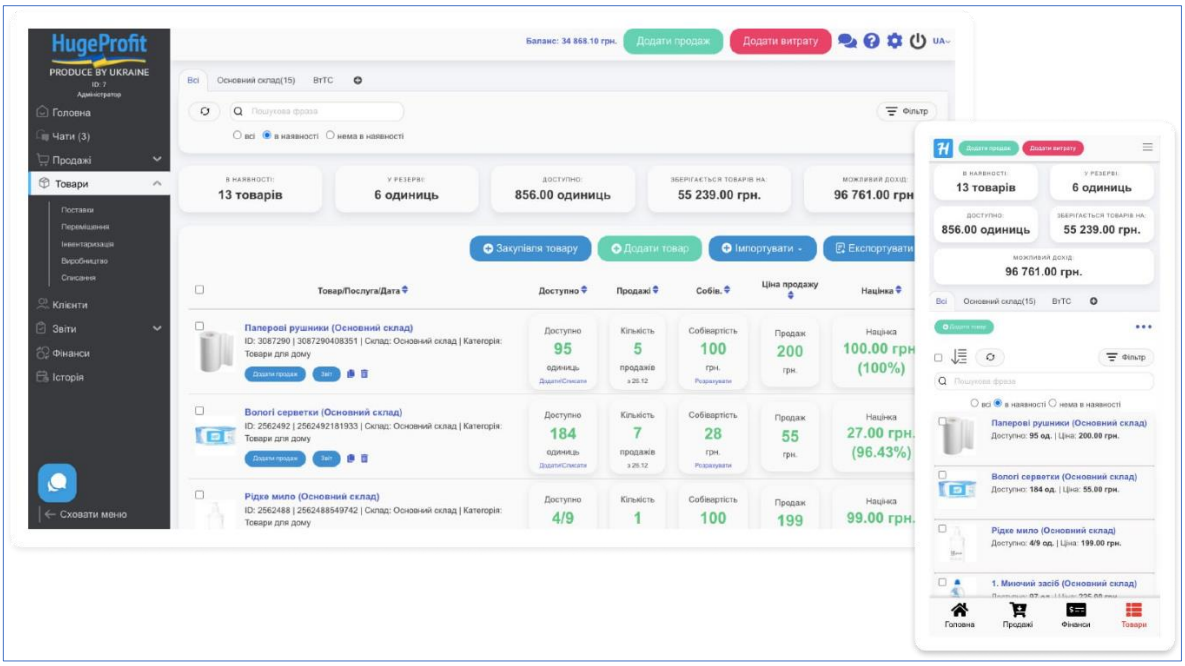


Рис. 1.2 Приклад екранної форми додатку HugeProfit

1.3.3 Додаток «Квітка Торгівля»

Додаток «Квітка Торгівля»[3] призначений для автоматизації управління торговельною діяльністю в роздрібних магазинах, офісах та на складах. Серед ключових можливостей:

– Підтримка роботи з декількома торговими точками через єдину базу даних.
– Функціонал для обліку витрат та формування аналітичних звітів за продажами.

– Інтеграція зі сканерами штрих-кодів, фіскальними принтерами та принтерами етикеток для зручності роботи з товарами.

Програма позиціонується як зручний інструмент автоматизації та управління торговими процесами для бізнесів різного масштабу - від невеликих магазинів до великих торгових мереж. Інтерфейс видається типовим для подібних програмних рішень в торгівлі.

Ключовою перевагою є можливість централізованого управління кількома торговими точками через єдину систему. Водночас, функціонал щодо обліку витрат та звітності видається доволі стандартним. Екранна форма додатку «Квітка Торгівля» наведена на рисунку 1.3.

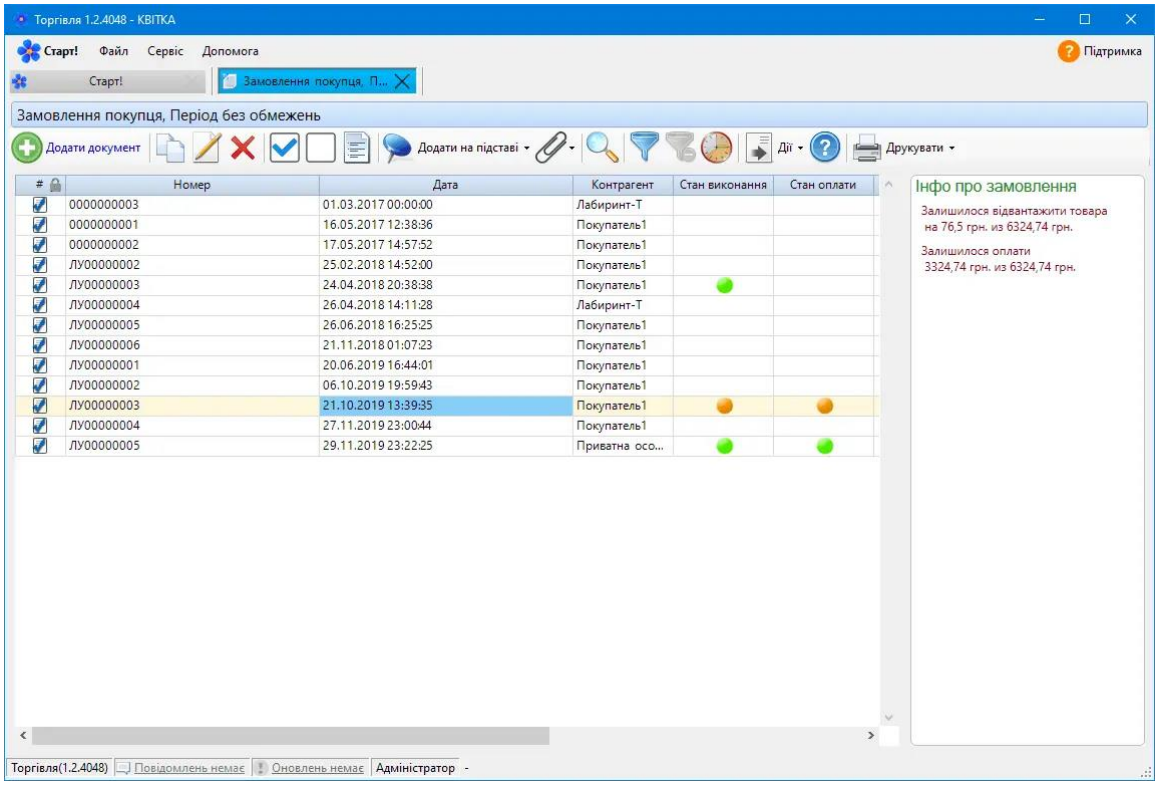


Рис. 1.3 Приклад екранної форми додатку «Квітка Торгівля»

Зведені результати аналізу характеристик розглянутих додатків наведено у таблиці 1.1

Таблиця 1.1

Порівняння існуючих програм-аналогів

Показник/функція	**Dilovod**	**HugeProfit**	**“Квітка Торгівля”**	**Магазин «Даринка»**
Платформа	Web	Android, iOs, Web	Windows	Windows
Облік товарів	+	+	+	+
Завантаження номенклатури	+ (з Excel)	+	+	+
Ведення клієнтської бази	+	+	-	+
Формування замовлень/рахунків	+	+	+	+
Облік продажів	+	+	+	+
Інвентаризація	+	+	+	+
Аналітичні звіти	+	+	+	+
Підтримка штрих-кодів	+	+	+	+
Інтеграція з фіскальними принтерами	+	+	+	Зайва для стейкхолдерів функція
Адаптація під різні типи магазинів	-	+	+	Зайва для стейкхолдерів функція
Робота з боржниками	-	-	-	+
Нагадування про борг через Телеграм-бот	-	-	-	+

2 ПРОЕКТУВАННЯ ТА РОЗРОБКА ДОДАТКУ ДЛЯ ОБЛІКУ ТОВАРІВ У ПРОДУКТОВОМУ МІНІ-МАРКЕТІ

2.1 Моделювання вимог до додатку

З урахуванням особливостей роботи продуктового міні-маркету, загальний перелік функціональних вимог включає:

– внесення, редагування та видалення інформації про товари;

– облік надходження товарів на склад;

– відстеження залишків товару;

– продаж товарів з можливістю пошуку, сканування штрих-коду, вибору кількості та способу оплати;

– ведення бази даних боржників, внесення нових боржників, погашення боргів;

– надсилання нагадувань боржникам;

– формування звітів про продажі, оплати та залишки товарів;

– управління змінами роботи міні-маркету та перегляд історії змін.

Нефункціональні вимоги описуються наступним переліком:

– інформація про товари повинна включати поля: назва, тип, штрих-код, ціна, акциз;

– система повинна підтримувати фіксацію продажів з наступними типами плати: готівка, картка, борг;

– надсилання нагадувань боржникам повинно виконуватись через інтеграцію з телеграм-ботом.

Для моделювання вимог додатку для обліку товарів було використано уніфіковану мову моделювання UML (Unified Modeling Language) [4]. Це стандартизована графічна мова, яка дозволяє візуалізувати різні аспекти системи, такі як діаграми класів, діаграми варіантів використання, діаграми послідовності та інші. Це допомагає краще зрозуміти взаємозв'язки між компонентами системи та

спланувати її архітектуру. На рисунку 2.1 наведена діаграма варіантів використання додатку для обліку товарів, яка дозволяє змоделювати функціональні вимоги до додатку.

Рисунок 2.1 Діаграма варіантів використання

Діаграма містить основні аспекти роботи роздрібного підприємства, такі як управління товарними потоками, здійснення продажів, роботи з боржниками, формування звітів та закриття зміни.

В системі присутні 3 актори: продавець, покупець та телеграм-бот.

1. Продавець:
    1) Управління товарами:
        - Налаштування бази даних товарів (додавання, редагування, видалення товарів);
        - Внесення товару в базу даних;

- Видалення товару з бази даних;
- Редагування інформації про товар.

2) Продаж товарів:
   - Оплата товару (готівкою, карткою);
   - Взяття товару в борг.
3) Облік надходжень товару:
   - Відстеження залишків товарів на складі.
4) Робота з боржниками:
   - Нагадування боржникам про борг;
   - Погашення боргу боржниками.
5) Перегляд історії змін;
6) Формування звітів про продажі, надходження коштів тощо;
7) Закриття робочої зміни.

2. Покупець:
   - Отримання нагадувань про борг.
3. Телеграм-бот:
   - Надсилання нагадувань боржникам про борг.

Продавець є основним користувачем системи, який здійснює всі операції з управління товарами, продажу, обліку надходжень, роботи з боржниками, перегляду історії та звітності, а також закриття робочих змін. Покупець взаємодіє з системою лише для отримання нагадувань про борг через Телеграм-бота, який виконує функцію надсилання цих нагадувань.

2.2Обґрунтування вибору програмних засобів розробки

Щоб створити додаток для обліку товарів у продуктовому міні-маркеті, було вибрано мову програмування Python. Python [5] є мовою програмування високого рівня, яка має простий і зрозумілий синтаксис, що дозволяє швидко розробляти додатки. Це покращує продуктивність розробників і полегшує подальше супроводження та розширення функціональності програм.

Python включає безліч фреймворків і бібліотек, які можна використовувати для роботи з базами даних, створення графічних інтерфейсів користувача (GUI), розробки додатків і інтеграції з різними сервісами, такими як Telegram.

Об’єктно-орієнтований підхід дозволяє структурувати програмний код у вигляді окремих класів і об’єктів. Це забезпечує модульність, повторне використання коду та полегшує подальшу розширення функціоналу програми.

Зокрема, для розробки програмного забезпечення було використано наступні бібліотеки:

1. Бібліотека PyQt[6] призначена для створення графічних інтерфейсів користувача (GUI). PyQt є обгорткою Python для популярної бібліотеки Qt і пропонує широкий набір інструментів для розробки GUI, таких як віджети, макети, обробка подій та інші функції.

2. SQLite[7] є вбудованою реляційною базою даних, яка підтримує більшість стандартів SQL. Завдяки своїй компактності, вбудованості та підтримці більшості функцій SQLite є кращим варіантом для зберігання даних про товари, продажі, боржників та історію змін.

3. Бібліотека Python-telegram-bot[8] призначена для створення Telegram-ботів і взаємодії з API Telegram. У цій бібліотеці можна інтегрувати програму з месенджером Telegram, щоб надсилати нагадування боржникам.

2.3 Моделювання бази даних та використання SQLite

Для зберігання даних про товари, продажі, боржників та історію змін було обрано систему управління базами даних SQLite. SQLite є компактною, вбудованою реляційною базою даних, яка не вимагає окремого сервера для роботи. Це спрощує розгортання та використання додатку на різних платформах.

SQLite підтримує більшість стандарту SQL, що полегшує роботу з даними та забезпечує їх цілісність. У додатку для обліку товарів SQLite використовується для зберігання наступної інформації :

1. Таблиця «Товари» містить наступну інформацію: назва, одиниця виміру (штучний, ваговий), штрих-код, акциз, ціна.

2. Таблиця «Продажі»: дата, загальна сума, спосіб оплати (готівка, картка або борг), список товарів в продажах, який пов’язаний з відповідною таблицею зовнішнім ключем.

3. Таблиця «Товари в продажах» містить товари з кожного продажу, його назву, кількість та ціну.

4. Таблиця «Склад» відображає поточні залишки товарів на складі та пов'язана з таблицею "Товари".

5. Таблиця «Боржники» зберігає інформацію про осіб, які мають борги за товари: ім'я, прізвище, телефон, сума боргу, унікальний код боржника, його ідентифікатор у Telegram.

6. Таблиця «Історія змін» фіксує всі зміни продажів, що відбулися протягом робочого дня: дата, загальна сума продажів, суми, отримані готівкою, карткою та як борги.

На рисунку 2.2 зображена схема бази даних.

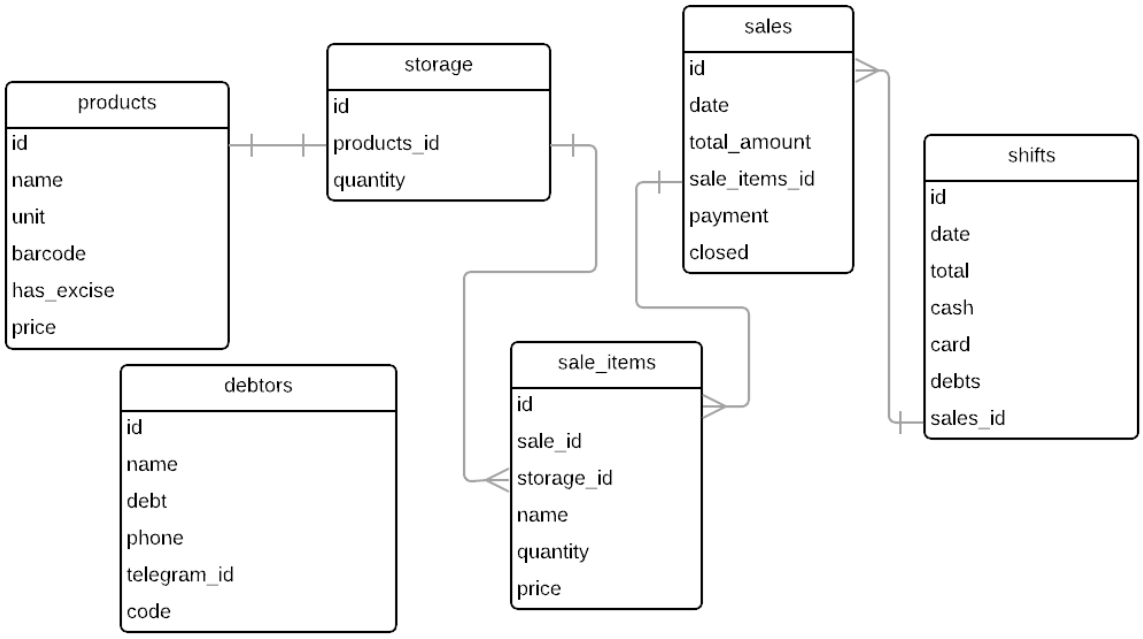


Рис. 2.2 Схема бази даних

Для моделювання бази даних було використано концепцію реляційних баз даних, де дані зберігаються у вигляді таблиць з рядками (записами) та стовпцями (полями). Таблиці пов'язані між собою через зовнішні ключі, що забезпечує цілісність даних та можливість встановлювати зв'язки між сутностями.

2.4 Створення графічного інтерфейсу користувача з використанням PyQt

Для створення графічного інтерфейсу користувача було використано бібліотеку PyQt, яка є Python-обгорткою для популярної бібліотеки Qt. PyQt надає широкий набір інструментів для розробки GUI, включаючи віджети, макети, обробку подій та інші функції. Ця бібліотека забезпечує створення зручного та інтуїтивно зрозумілого інтерфейсу для користувачів додатку.

Графічний інтерфейс користувача додатку для обліку товарів у продуктовому міні-маркеті складається з декількох основних вікон та вкладок:

1. Головне вікно з меню, що містить наступні розділи:
   - Продаж (екран для здійснення продажів товарів);
   - Склад (перегляд та редагування товарів на складі);
   - Боржники (перегляд, редагування та додавання боржників);
   - Історія (перегляд історії змін та звітів);
   - Налаштування (редагування та додавання товарів у базі даних);
   - Закрити зміну (формування звіту та закриття поточної зміни).
2. Вікно продажу, де відображаються додані товари, їх кількість та ціна. Тут також присутні елементи для вибору способу оплати (готівка, картка, борг), введення суми оплати та розрахунку решти.
3. Вікно складу, що відображає список наявних товарів та дозволяє редагувати їх кількість.
4. Вікно боржників, де відображається список боржників з інформацією про їхні борги та можливістю переглядати деталі, вносити оплату або надсилати нагадування через Telegram.

5. Вікно історії змін, що містить перелік попередніх змін з датами, сумами продажів та можливістю переглянути детальний звіт по кожній зміні.
6. Вікно налаштувань, де можна редагувати базу даних товарів, додавати нові товари, змінювати їх характеристики (назва, тип, штрих-код, ціна, акциз).
7. Діалогові вікна для підтвердження дій, введення даних та повідомлень.

На рисунку 2.3 зображено схему роботи додатку.

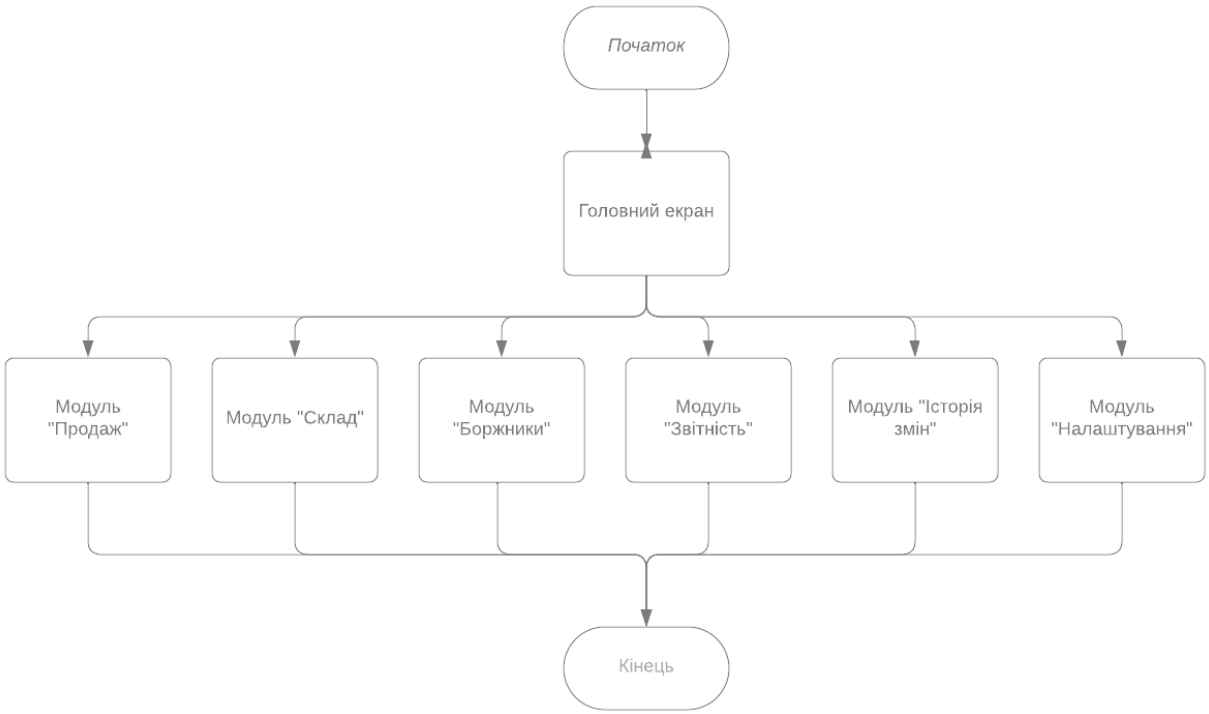


Рис. 2.3 Схема роботи додатку

Розглянемо основні компоненти додатку:

1. Клас QMainWindow використовується для створення головного вікна додатку з файлу main_window.py. Він містить верхню панель з кнопками для переходу між вкладками та область для відображення різних вкладок або віджетів (QStackedWidget).

2. QWidget успадковує окремий віджет для кожної вкладки програми. Наприклад, SaleWidget відповідає за продаж товарів, StockWidget керує складом, DebtorsWidget обслуговує боржників і так далі.

3. Віджети, такі як QLineEdit (для введення тексту), QListWidget (для відображення списку елементів), QComboBox (для вибору значення зі списку), QSpinBox (для введення цілочисельних значень) та інші, використовуються для введення та відображення даних.

4. Для виконання додаткових дій, таких як редагування або додавання нових елементів, використовуються діалогові вікна QDialog і QInputDialog.

2.5 Інтеграція з Телеграм-ботом

Для зручності нагадування боржникам про їхні борги та підвищення зручності використання додатку, було реалізовано інтеграцію з Телеграм-ботом за допомогою бібліотеки Python-telegram-bot.

Телеграм-бот додатку «Магазин "Даринка"» дозволяє виконувати наступні функції:

1. Нагадування боржникам: користувач може використовувати програму для відправки нагадувань конкретному боржнику про необхідність погасити його борг. Нагадування надсилається ботом в Телеграм.
2. Сповіщення про новий борг: якщо боржник набуває нового боргу, йому буде надіслано сповіщення про його розмір.
3. Сповіщення про сплату: коли боржник сплачує борг, йому надсилається сповіщення про сплату та його поточний баланс.
4. Перевірка стану боргу: боржник може використовувати Телеграм-бота для перегляду поточного стану свого боргу.

На рисунку 2.4 зображено граф діалогів, який визначає послідовність повідомлень та дій, що можуть бути виконані користувачем або ботом.

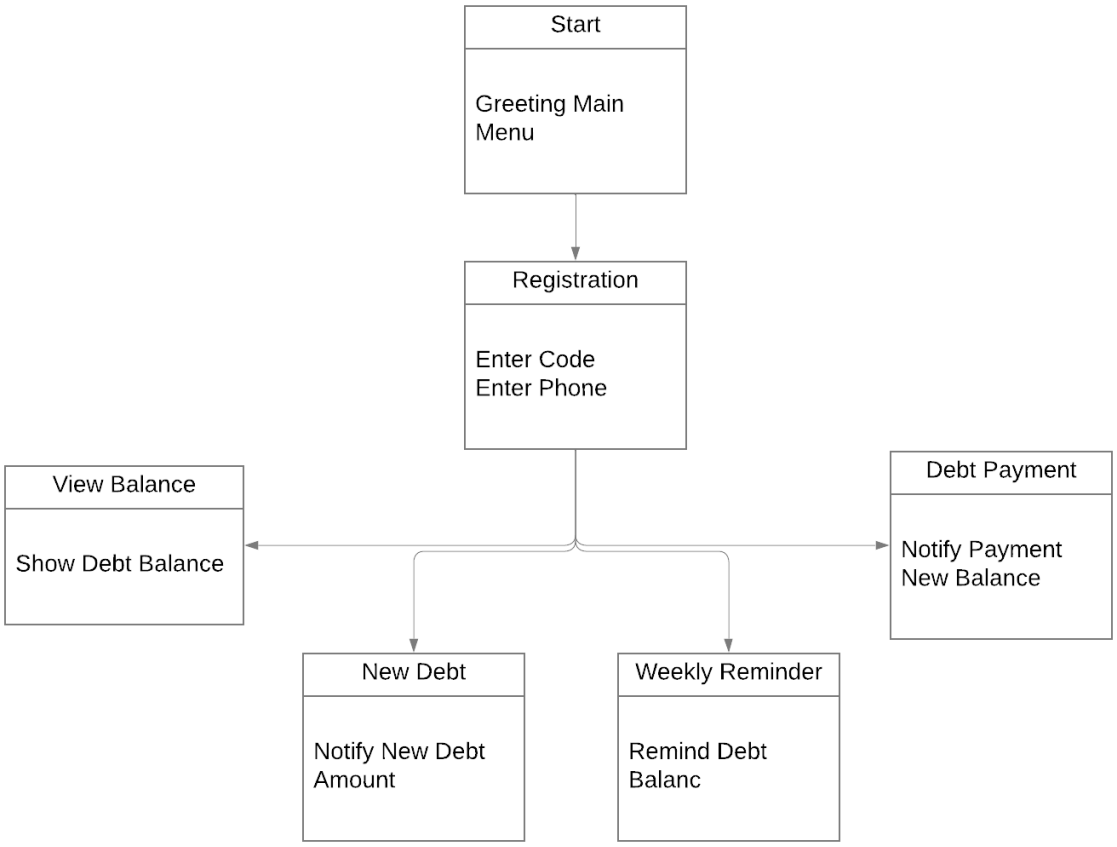


Рис. 2.4 Граф діалогів телеграм-бота

Поєднання з Телеграм-ботом полегшує взаємодію з боржниками та дозволяє отримувати нагадування. Це полегшує співпрацю з боржниками та сприяє своєчасному погашенню боргів.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОБЛІКУ ТОВАРІВ У ПРОДУКТОВОМУ МІНІ-МАРКЕТІ

3.1 Діаграма класів

Для створення функцій додатку було застосовано об'єктно-орієнтований підхід до програмування. На рисунку 3.1 зображена діаграма класів, що відображає основні сутності та їх взаємозв'язки.

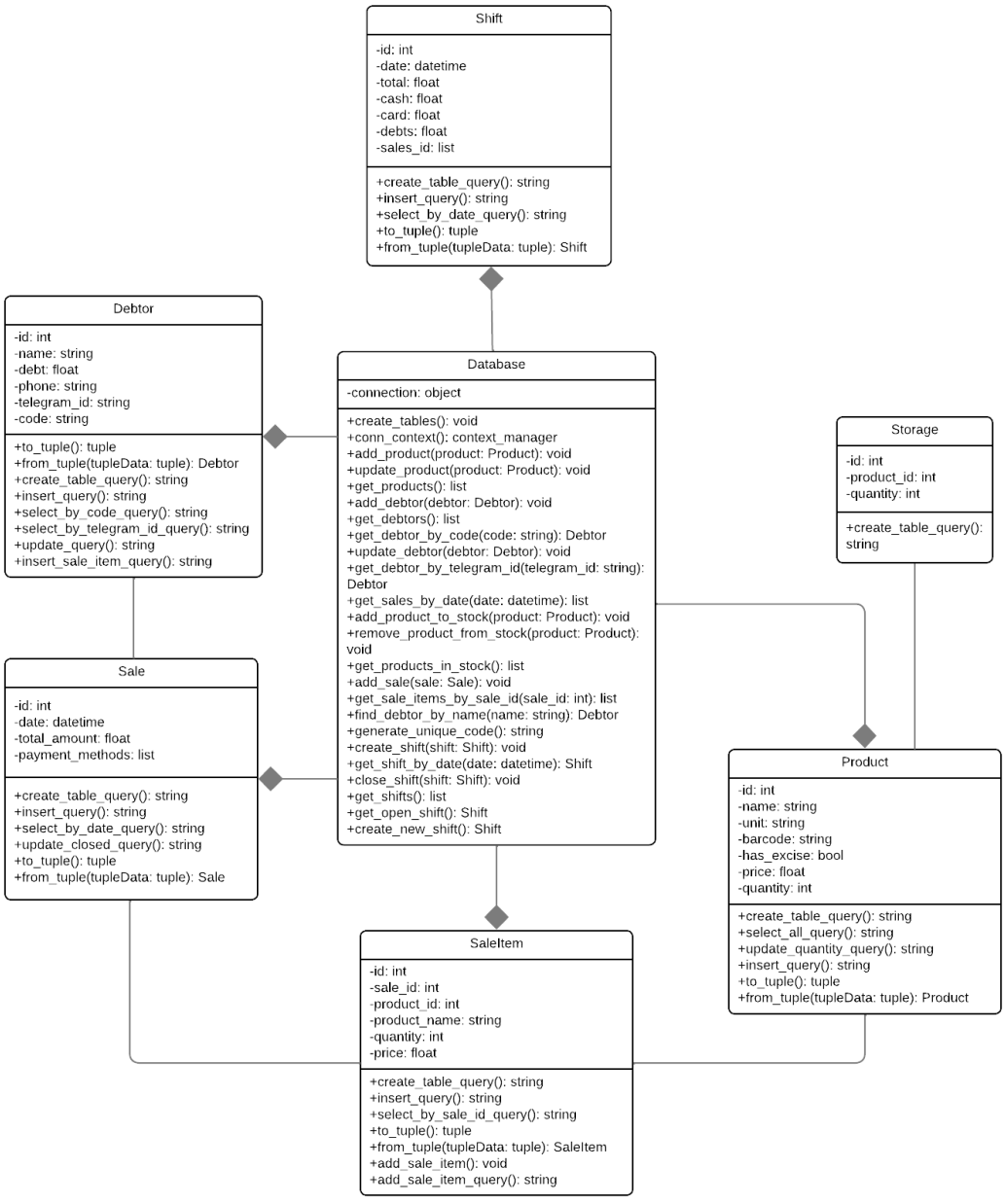


Рис. 3.1 Діаграма класів

Ця діаграма описує основні класи та методи для системи управління товарами та продажами в роздрібних магазинах. Система може вести облік запасів, продажів, боржників, товарів і змін. Нижче наведено опис і функції основних класів.

3.2 Опис ключових класів та їх методів

1. Клас Product:

   Атрибути: id, name, unit, barcode, has_excise, price, quantity;

   Методи:

   1) create_table_query(): повертає SQL-запит для створення таблиці продуктів;
   2) select_all_query(): повертає SQL-запит для вибірки всіх продуктів;
   3) update_quantity_query(): повертає SQL-запит для оновлення кількості продуктів;
   4) insert_query(): повертає SQL-запит для вставки нового продукту;
   5) to_tuple(): повертає кортеж з атрибутами продукту для вставки в базу даних;
   6) from_tuple(): створює екземпляр класу Product з кортежу, отриманого з бази даних.

2. Клас Debtor:

   Атрибути: id, name, debt, phone, telegram_id, code;

   Методи:

   1) to_tuple(): повертає кортеж з атрибутами боржника для вставки в базу даних;
   2) from_tuple(): створює екземпляр класу Debtor з кортежу, отриманого з бази даних;
   3) create_table_query(): повертає SQL-запит для створення таблиці боржників;
   4) insert_query(): повертає SQL-запит для вставки нового боржника;

5) select_by_code_query(): повертає SQL-запит для вибірки боржника за кодом;
6) select_by_telegram_id_query(): повертає SQL-запит для вибірки боржника за ідентифікатором Telegram;
7) update_query(): повертає SQL-запит для оновлення інформації про боржника;
8) insert_sale_item_query(): повертає SQL-запит для вставки деталей продажу.

3. Клас Sale:

   Атрибути: id, date, total_amount, payment_methods;

   Методи:

   1) create_table_query(): повертає SQL-запит для створення таблиці продажів;
   2) insert_query(): повертає SQL-запит для вставки нового продажу;
   3) select_by_date_query(): повертає SQL-запит для вибірки продажів за датою;
   4) update_closed_query(): повертає SQL-запит для оновлення статусу продажу як закритого;
   5) to_tuple(): повертає кортеж з атрибутами продажу для вставки в базу даних;
   6) from_tuple(): створює екземпляр класу Sale з кортежу, отриманого з бази даних.

4. Клас SaleItem:

   Атрибути: id, sale_id, product_id, product_name, quantity, price;

   Методи:

   1) create_table_query(): повертає SQL-запит для створення таблиці деталей продажу;
   2) insert_query(): повертає SQL-запит для вставки нової деталі продажу;

3) select_by_sale_id_query(): повертає SQL-запит для вибірки деталей продажу за ідентифікатором продажу;
4) to_tuple(): повертає кортеж з атрибутами деталі продажу для вставки в базу даних;
5) from_tuple(): створює екземпляр класу SaleItem з кортежу, отриманого з бази даних;
6) add_sale_item(): додає деталь продажу до бази даних;
7) add_sale_item_query(): повертає SQL-запит для додавання деталі продажу.

5. Клас Shift:

   Атрибути: id, date, total, cash, card, debts, sales_id;

   Методи:

   1) create_table_query(): повертає SQL-запит для створення таблиці змін;
   2) insert_query(): повертає SQL-запит для вставки нової зміни;
   3) select_by_date_query(): повертає SQL-запит для вибірки зміни за датою;
   4) to_tuple(): повертає кортеж з атрибутами зміни для вставки в базу даних;
   5) from_tuple(): створює екземпляр класу Shift з кортежу, отриманого з бази даних.

6. Клас Storage:

   Атрибути: id, product_id, quantity;

   Метод:

   1) create_table_query(): повертає SQL-запит для створення таблиці складу.

7. Клас Database:

   Атрибут: connection;

   Методи:

   1) create_tables(): створює всі необхідні таблиці в базі даних;

2) conn_context(): контекстний менеджер для керування транзакціями;
3) add_product(): додає новий продукт до бази даних;
4) update_product(): оновлює інформацію про продукт в базі даних;
5) get_products(): повертає список всіх продуктів з бази даних;
6) add_debtor(): додає нового боржника до бази даних;
7) get_debtors(): повертає список всіх боржників з бази даних;
8) get_debtor_by_code(): повертає боржника за його унікальним кодом;
9) update_debtor(): оновлює інформацію про боржника в базі даних;
10) get_debtor_by_telegram_id(): повертає боржника за його ідентифікатором Telegram;
11) get_sales_by_date(): повертає список продажів за датою;
12) add_product_to_stock(): додає продукт до складу або оновлює його кількість;
13) remove_product_from_stock(): видаляє продукт зі складу;
14) get_products_in_stock(): повертає список всіх продуктів, що є на складі;
15) add_sale(): додає новий продаж до бази даних;
16) get_sale_items_by_sale_id(): повертає деталі продажу за ідентифікатором продажу;
17) find_debtor_by_name(): знаходить боржника за його іменем;
18) generate_unique_code(): генерує унікальний код для боржника;
19) create_shift(): створює нову зміну в базі даних;
20) get_shift_by_date(): повертає зміну за датою;
21) close_shift(): закриває зміну та зберігає звіт;
22) get_shifts(): повертає список всіх змін;
23) get_open_shift(): повертає відкриту зміну за датою;
24) `create_new_shift(): створює нову зміну.

3.3 Опис функціонування програми

Програма для обліку товарів у продуктовому міні-маркеті «Магазин "Даринка"» має головне вікно з меню, що надає доступ до різних функцій системи. Усі функції працюють з базою даних SQLite та графічним інтерфейсом користувача, створеним за допомогою PyQt.

При запуску додатку, якщо попередньо була зміна закрита або це перший запуск, відкривається нова зміна з поточною датою. Якщо зміна не була закрита, то відкривається остання незакрита зміна. Зміна - це сесія, де зберігається вся інформація продажів за день. В кожній зміні зберігається історія продажів. Є кнопка закриття зміни, яка закриває поточну зміну та формує звіт. У звіті міститься сума продажів (без врахування боргів), скільки оплат було готівкою, скільки карткою і скільки взято у борг. Всі зміни з історією продажів та звітами зберігаються.

Після запуску на головному екрані відразу відображається вкладка «Продаж». Зверху знаходиться меню синього кольору, яке відображається на всіх сторінках. Меню містить наступні пункти: Продаж, Склад, Боржники, Історія, Налаштування, Закрити зміну. Детальне зображення на рисунку 3.2.

На вкладці «Продаж» реалізований пошук товару по назві (товари беруться зі складу). Після додавання товару відображається його назва, кількість, одиниця виміру та ціна. Кількість можна редагувати, а товар можна видалити. Праворуч відображається сума всіх вибраних товарів, є вибір способу оплати (готівка, картка або борг), поле для введення суми від покупця та кнопка розрахувати, після натискання якої виводиться решта. Також реалізована можливість комбінувати способи оплати (частково готівкою, частково карткою, частково у борг). Якщо вибрано оплату боргом, то з'являється поле для введення імені та прізвища боржника. При введенні відображаються співпадіння з існуючими боржниками в базі даних. Для нового боржника можна ввести номер телефону. Після вибору боржника та натискання кнопки «Взяти в борг» інформація про борг додається в базу даних. Детально зображено на рисунках 3.3 – 3.5.

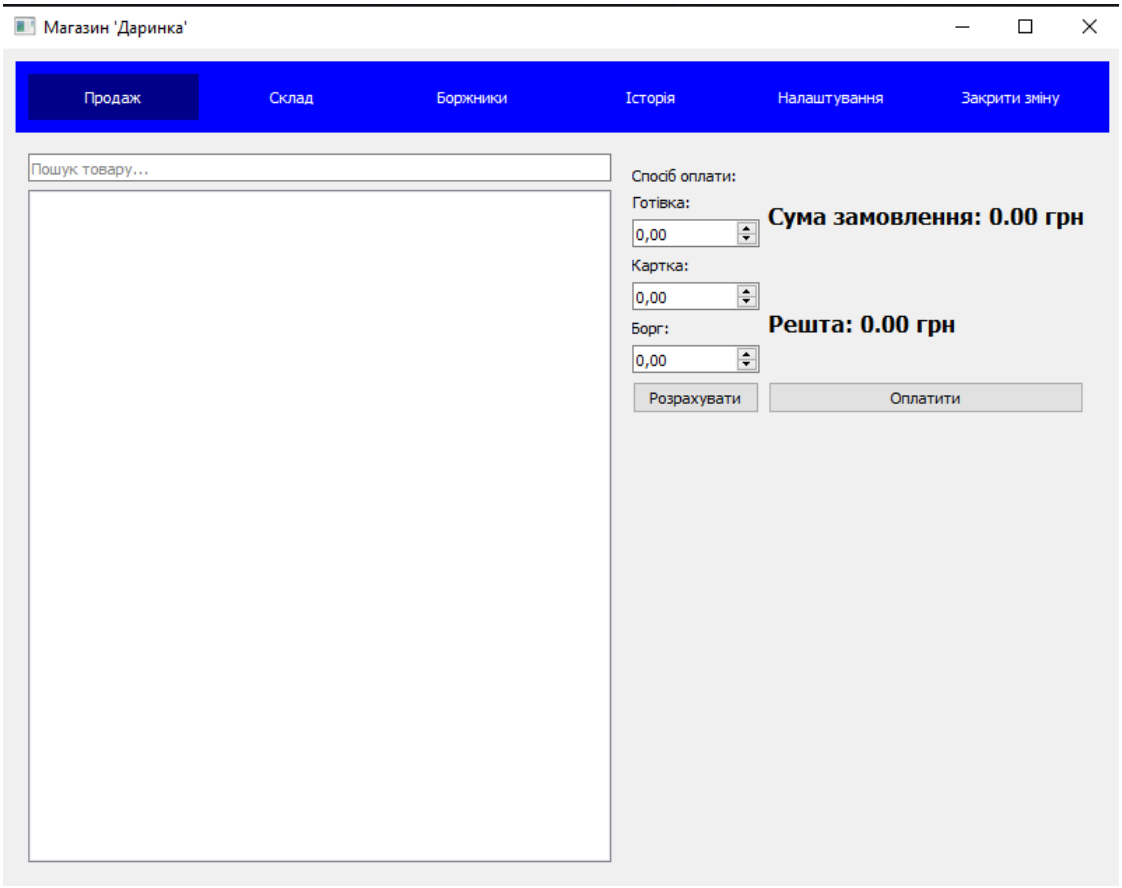


Рис. 3.2 Головний екран додатка. Вкладка «Продаж»

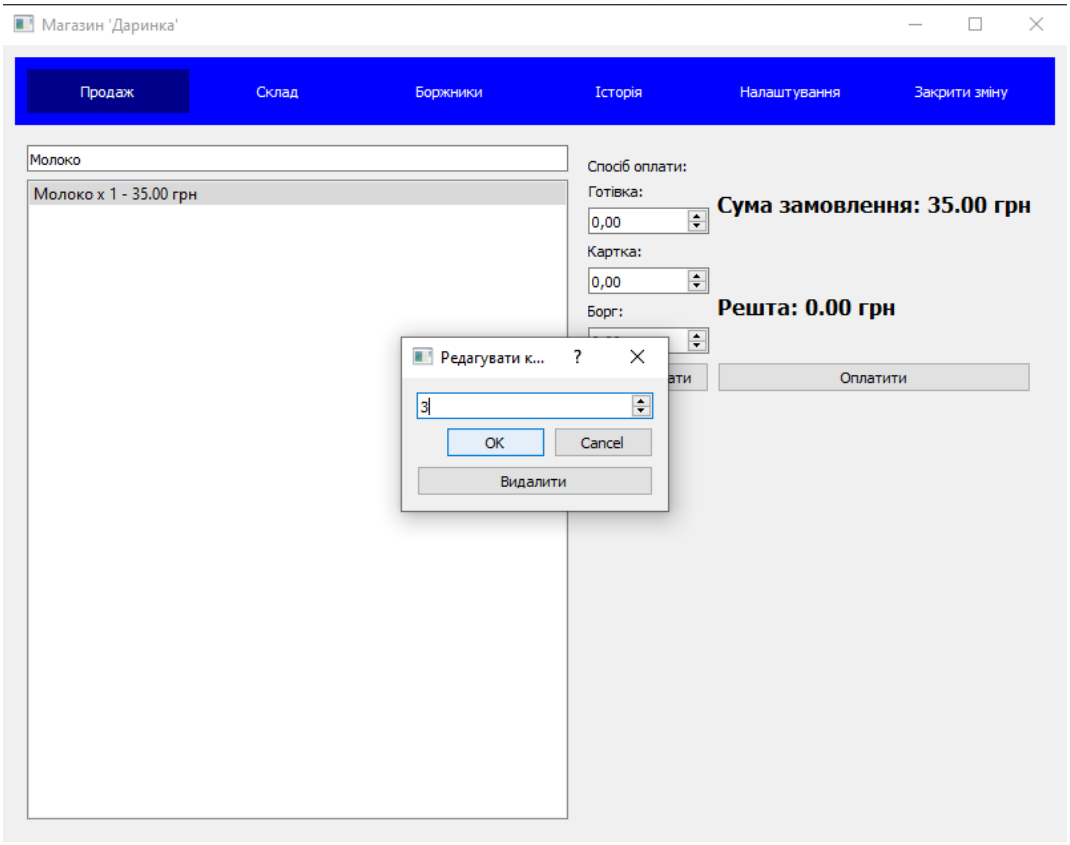


Рис. 3.3 Додавання товару та зміна кількості

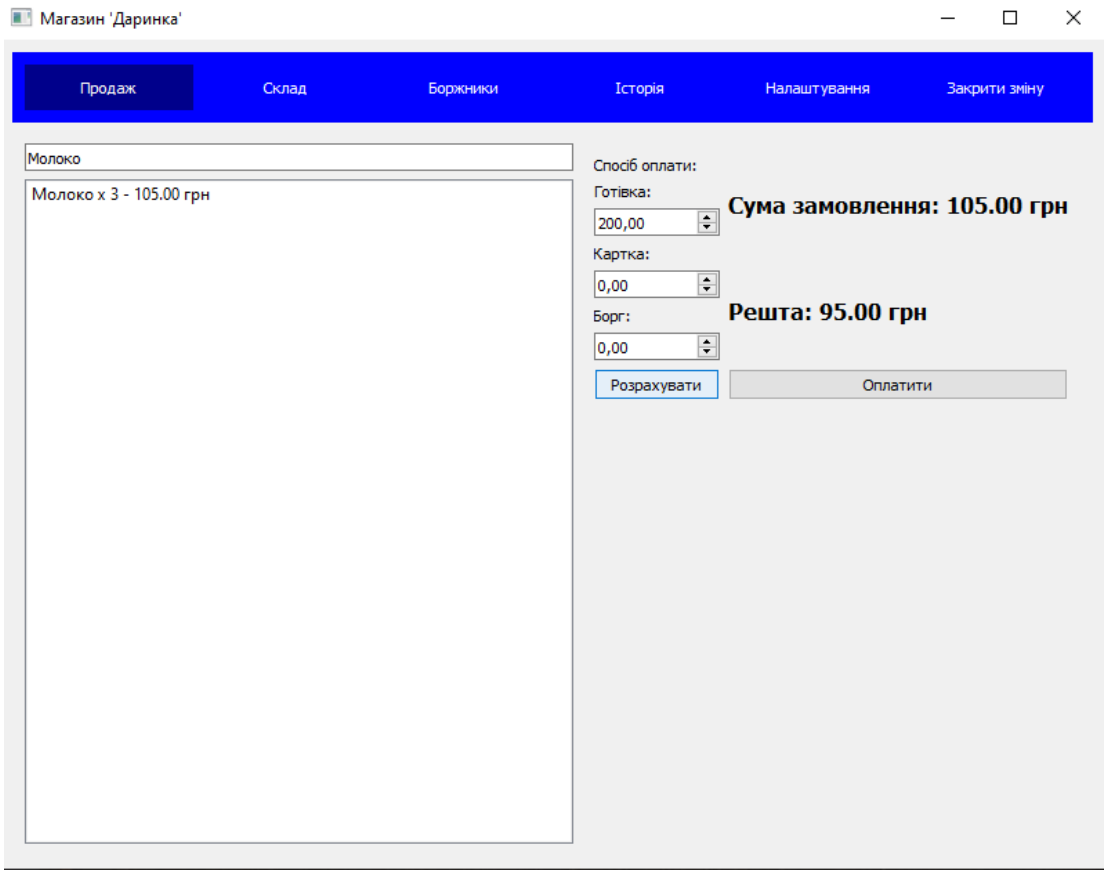


Рис. 3.4 Розрахунок решти

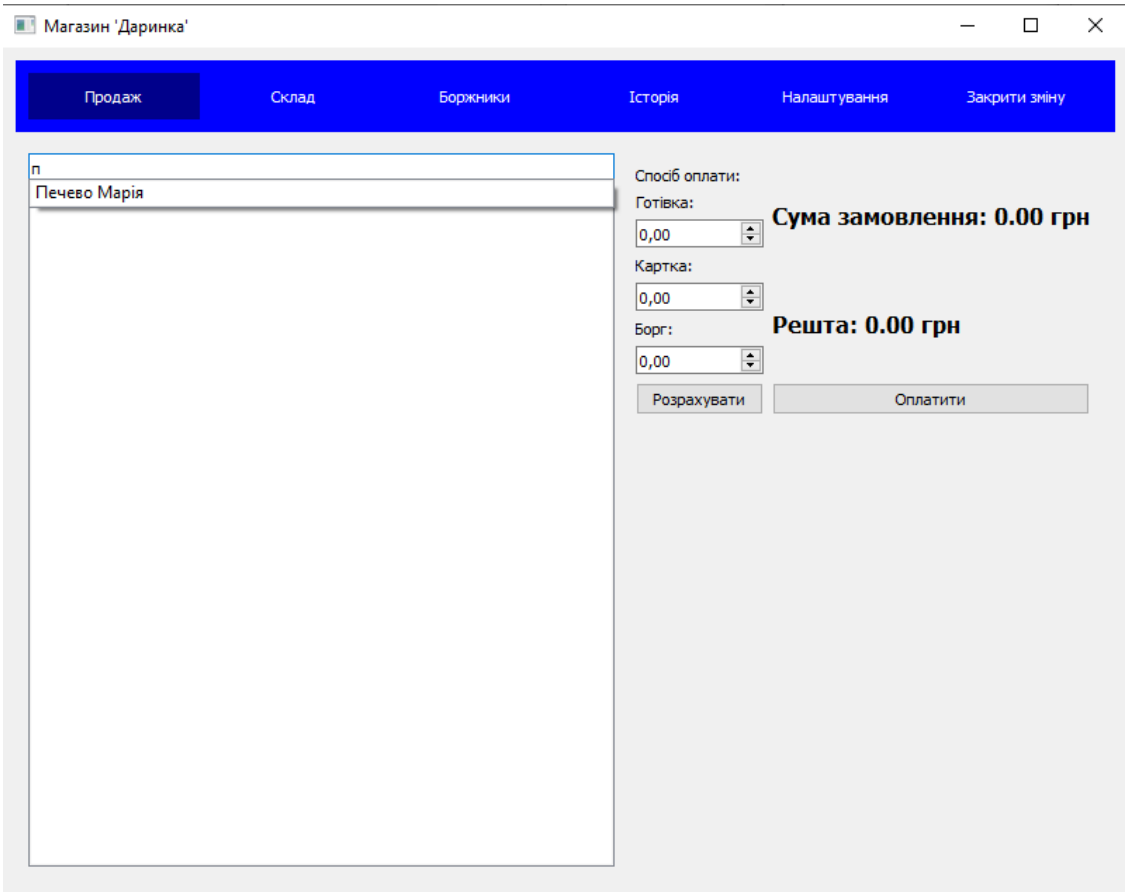


Рис. 3.5 Пошук товару

На вкладці «Склад» відображаються наявні товари. Товари додаються на склад з бази даних всіх товарів за допомогою пошуку або сканування штрих-коду. При продажу товару його кількість на складі зменшується. Приклади роботи наведено на рисунках 3.6 – 3.8.

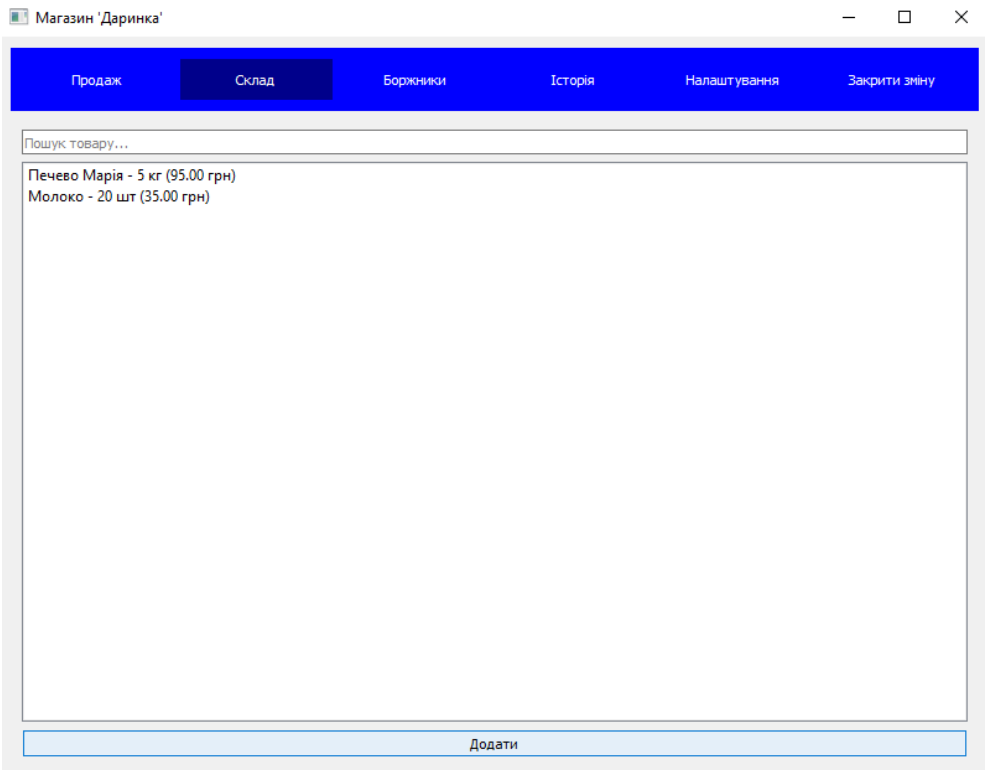


Рис. 3.6 Склад

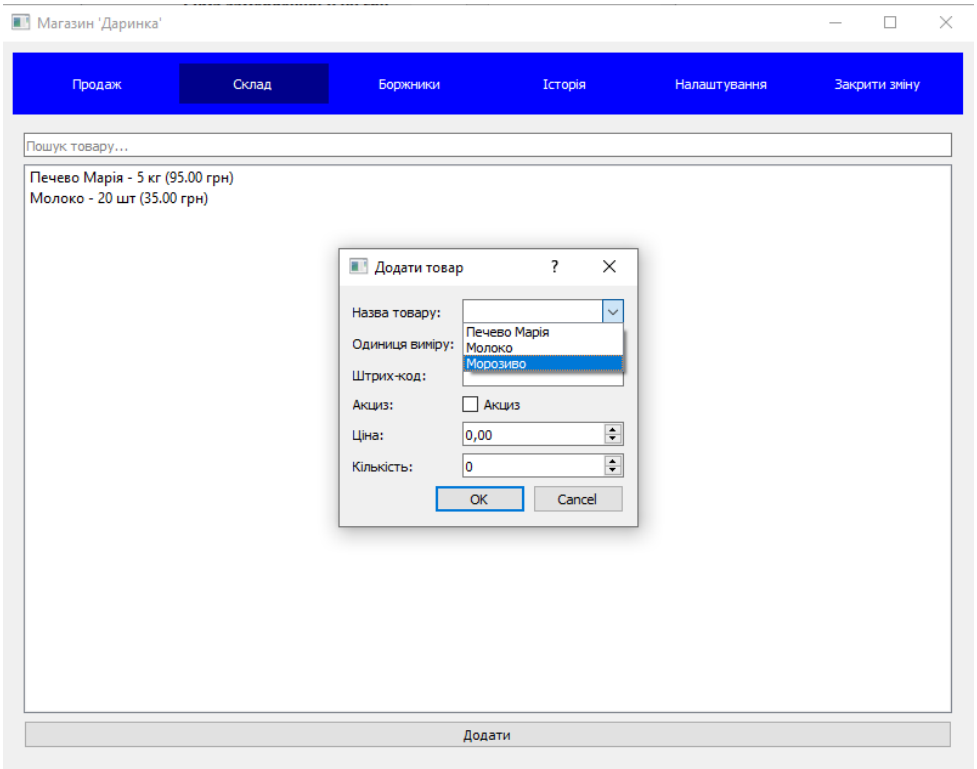


Рис. 3.7 Додавання товару до складу, вибір товару

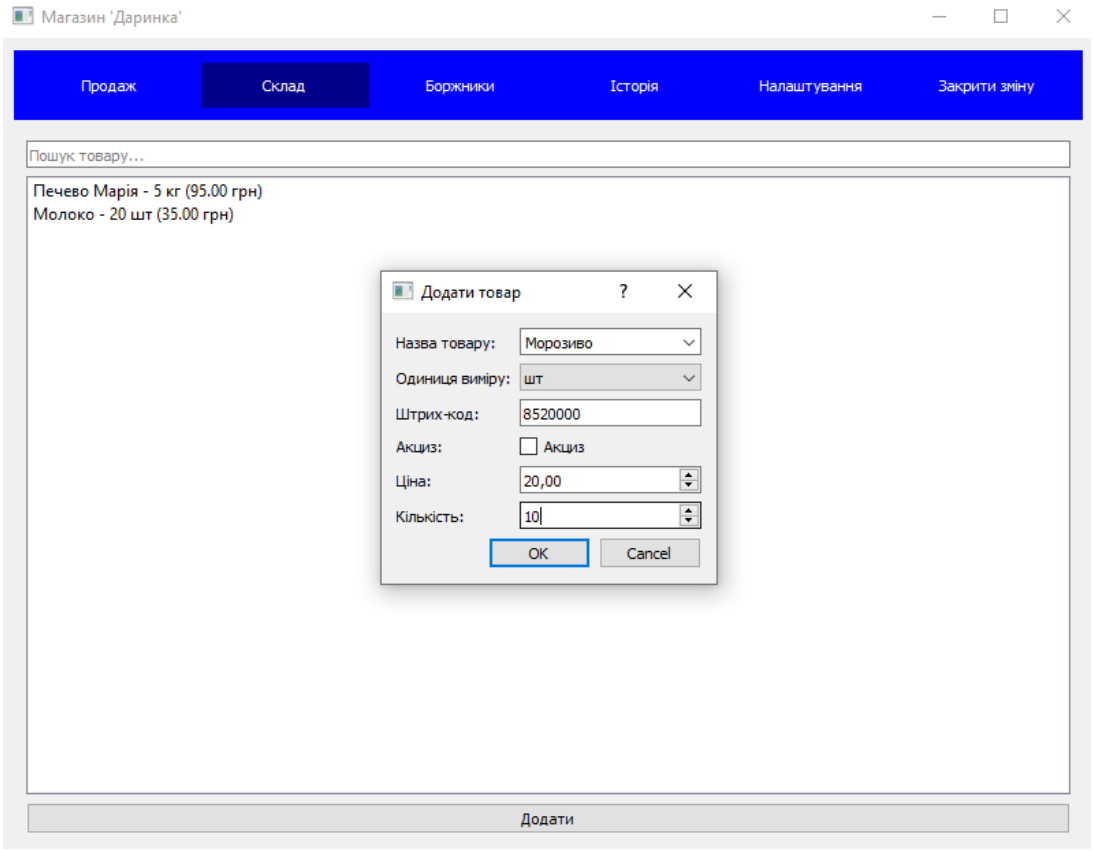


Рис. 3.8 Додавання товару до складу

На вкладці «Боржники» відображається список всіх боржників з інформацією про їх борги. Реалізований пошук боржників по імені та прізвищу, додавання нових боржників, редагування існуючих. При виборі боржника є кнопки «Сплатити» (для сплати боргу) та «Нагадати за борг» (для надсилання нагадування через Телеграм-бота). На рисунку 3.9 зображено екран вкладки «Боржники».

На вкладці «Історія» відображаються всі попередні зміни в хронологічному порядку (від поточної дати до минулих). Для кожної зміни відображається дата та сума продажів (без врахування боргів). При виборі зміни відкривається детальна інформація, де видно всі продажі за вибраний день та детальний звіт (загальна сума продажів, суми отримані готівкою, карткою та як борги).

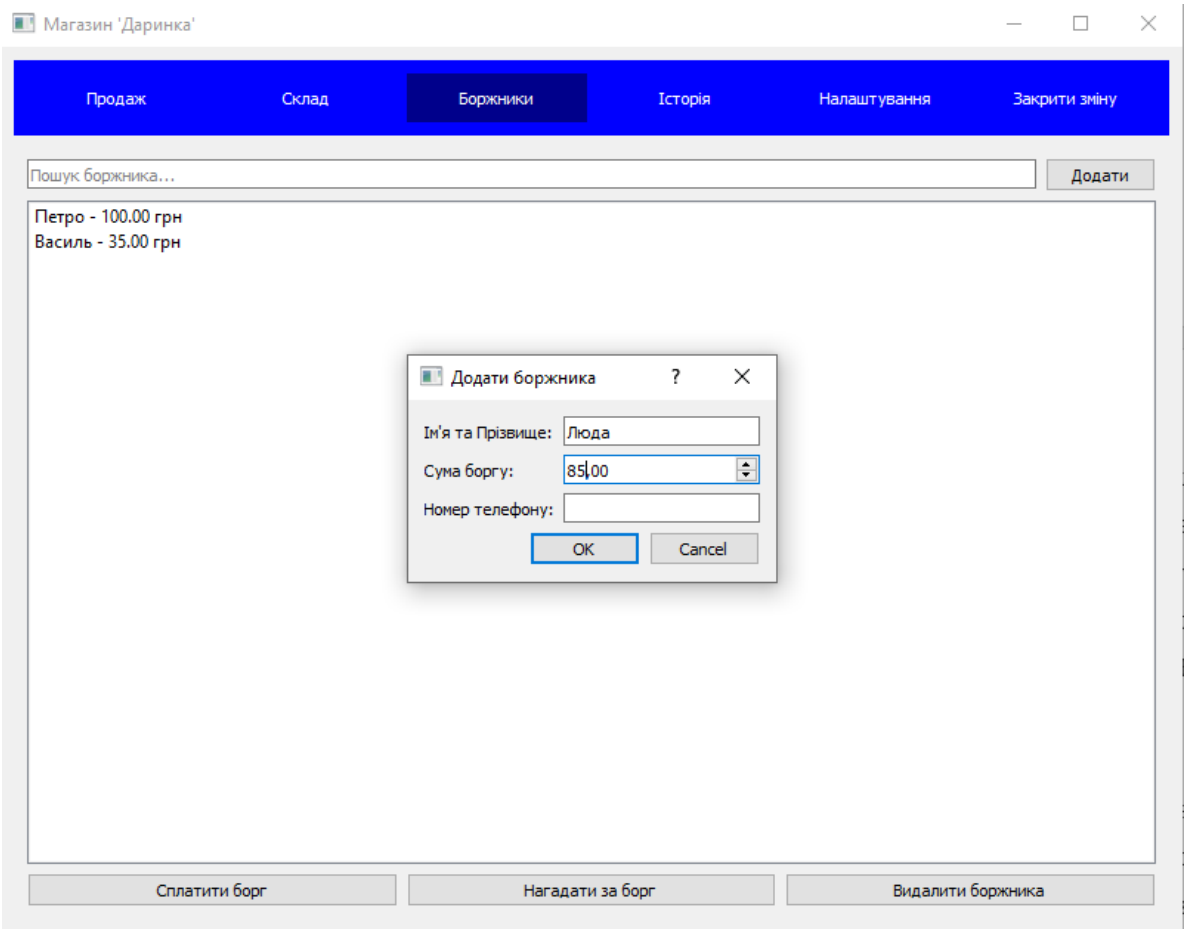


Рис. 3.9 Боржники

На вкладці «Налаштування» відбувається налаштування бази даних товарів, які потім будуть додаватися на склад. Реалізований пошук товару по назві для редагування існуючого товару (назва, ціна, штрих-код тощо). Є кнопка для додавання нового товару в базу даних відповідно до вимог (назва, одиниця виміру, штрих-код, акциз, ціна). На рисунках 3.10 – 3.11 приклад додавання товару та редагування.

На вкладці «Закрити зміну» відкривається вікно для підтвердження закриття поточної зміни. Після підтвердження формується звіт зміни та закривається поточна зміна. Відкривається нове вікно, де виведений звіт (сума продажів без врахування боргів, скільки оплат було готівкою, скільки карткою і скільки взято у борг). Цей звіт також можна переглянути у вкладці «Історія».

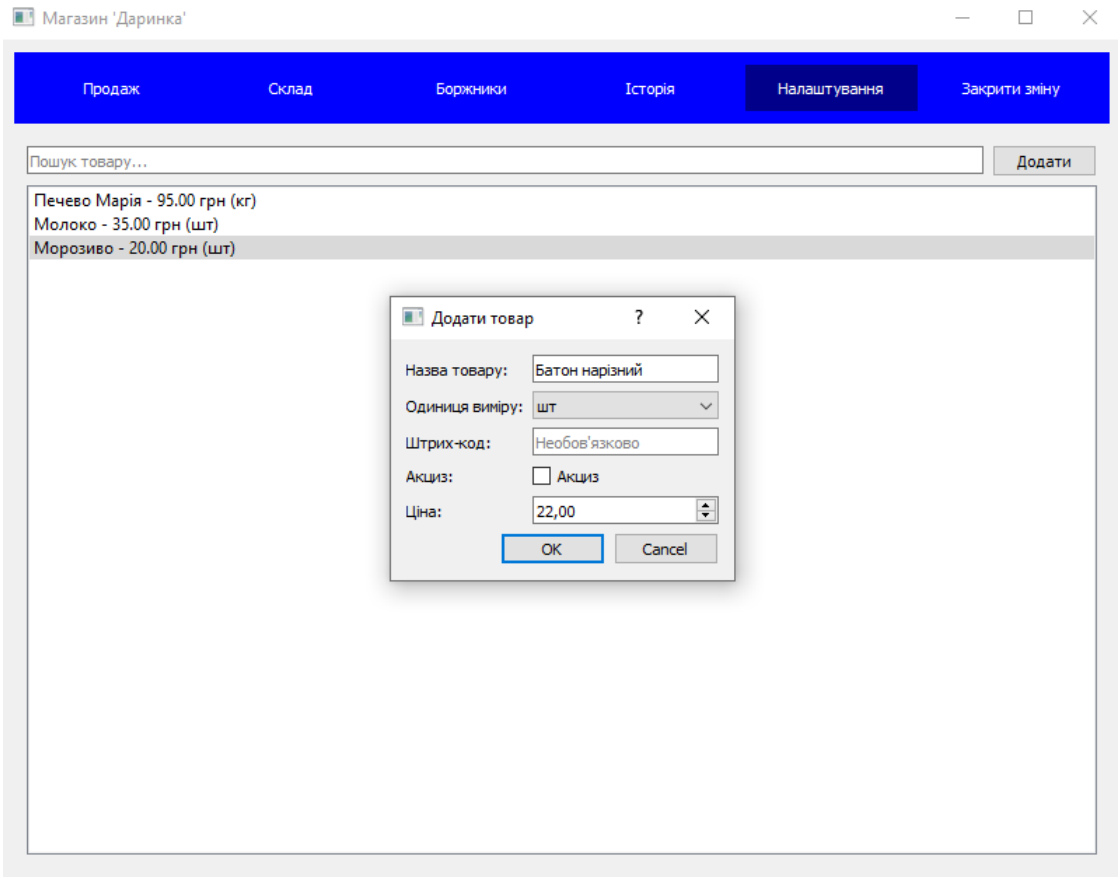


Рис.3.10 Додавання товару до бази даних

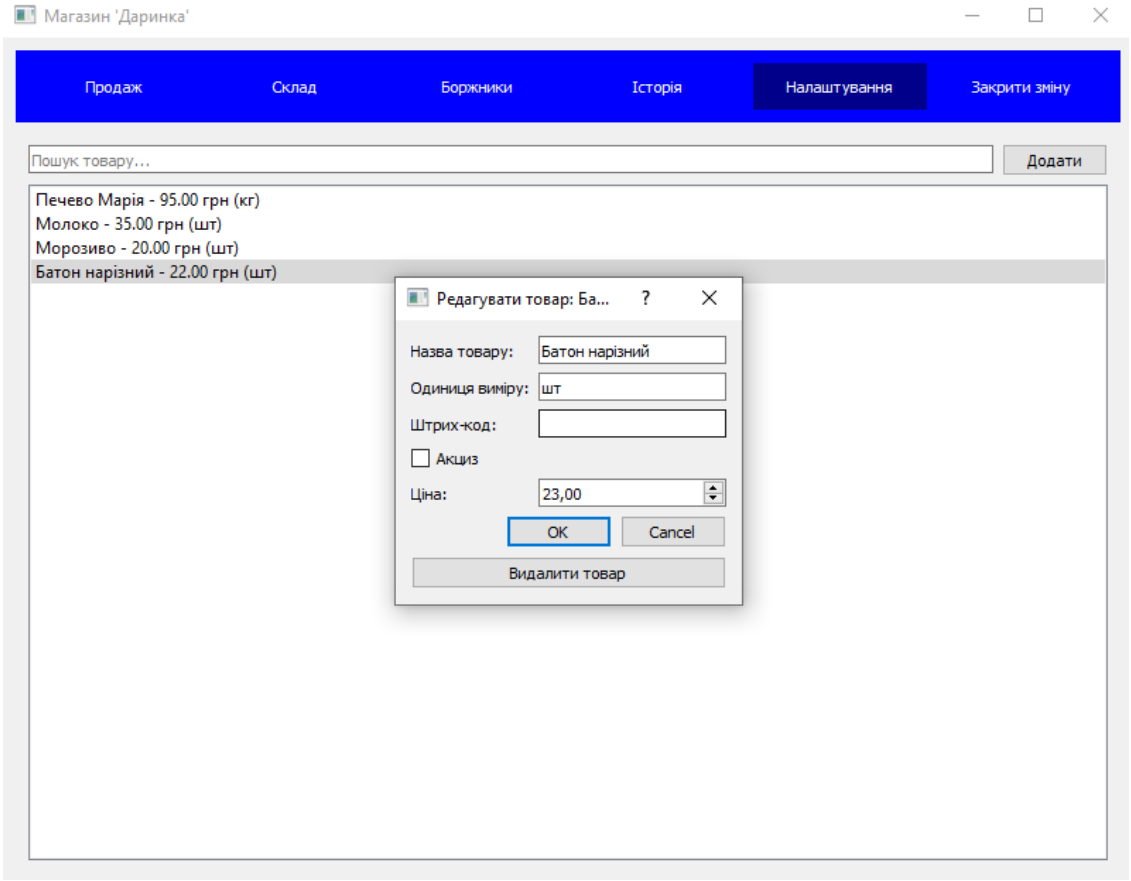


Рис. 3.11 Редагування товару

Телеграм-бот «Магазин "Даринка"». Нагадування за борг» інтегрований з додатком. Користувач може підключитися до боту, ввівши свій унікальний 6-значний код, який він отримує в магазині. Бот також запитує номер телефону користувача (якщо він його надасть, то номер зв'язується з кодом боржника в базі даних). Коли користувач бере товар в борг, бот надсилає повідомлення з сумою нового боргу. Раз на тиждень, у неділю, якщо у користувача є борг, йому надсилається нагадування з сумою боргу. Коли борг сплачується, користувачу надсилається повідомлення про сплату та поточний баланс боргу. Продавець також може надіслати нагадування боржнику про борг, натиснувши кнопку «Нагадати за борг» у вкладці «Боржники». Користувач боту має кнопку «Баланс» для перевірки суми свого боргу. На рисунку 3.12 зображений приклад роботи Телеграм-боту.

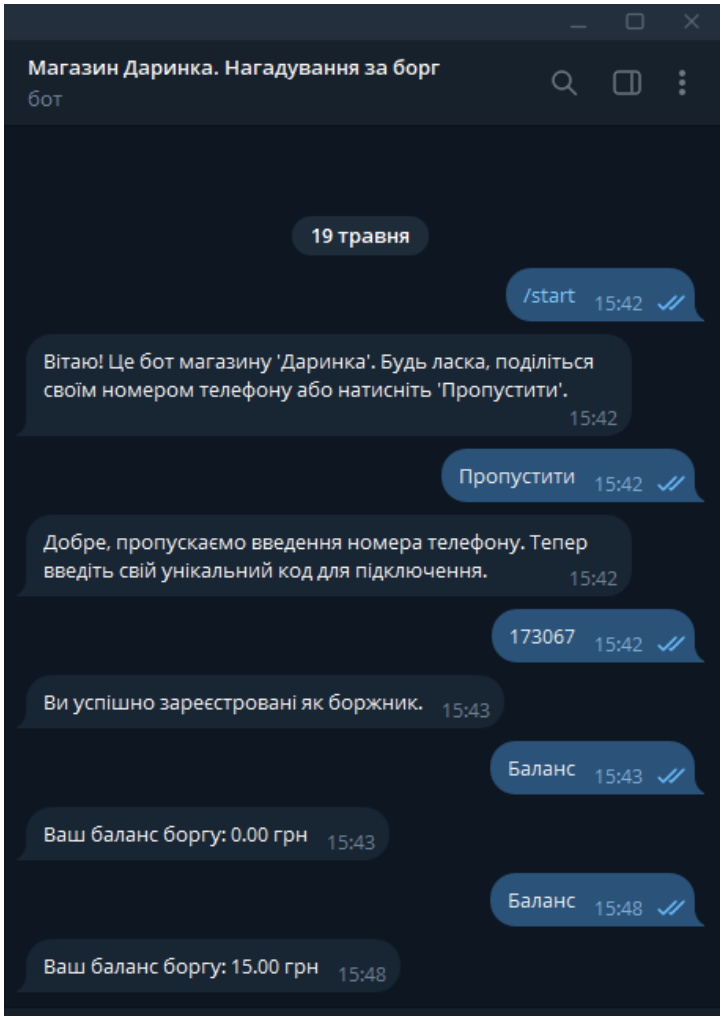


Рис. 3.12 Робота Телеграм-боту

Додаток складається з декількох модулів та файлів, зокрема main.py (головний файл), файлів графічного інтерфейсу користувача (main_window.py, sale_widget.py, stock_widget.py та інші), модуля роботи з базою даних (database.py, models.py) та модуля для Телеграм-бота (telegram_bot.py).

4 ТЕСТУВАННЯ ДОДАТКУ

Для забезпечення якості та працездатності додатку для обліку товарів у продуктовому міні-маркеті «Магазин "Даринка"» було виконане ретельне тестування з використанням різних видів тестів. Правильне тестування програмного забезпечення дозволяє виявити наявні дефекти, помилки та невідповідності на ранніх стадіях розробки, що значно знижує витрати на внесення виправлень у майбутньому.

4.1 Визначення об'єктів тестування додатку

Основним завданням тестування додатку "Магазин 'Даринка'" є перевірка його коректної роботи в нормальних умовах використання, а також правильної реакції на випадки, що відхиляються від звичайних сценаріїв користувача. У таблиці 4.1 наведені ключові характеристики, які потребують тестування в додатку, з урахуванням його функціональних можливостей.

Таблиця 4.1

Ключові характеристики для тестування додатку «Магазин "Даринка"»

Назва характеристики	**Вимоги до реалізації у додатку**
Перевірка основних функцій	У цьому етапі тестування перевіряється, чи працюють основні функції додатку згідно з очікуваннями, такі як: перегляд товарів, додавання товарів до кошика, оформлення продажу, робота з боржниками тощо.

Продовження таблиці 4.1

Ключові характеристики для тестування додатку «Магазин "Даринка"»

Назва характеристики	**Вимоги до реалізації у додатку**
Обробка різних типів операцій	Додаток повинен коректно обробляти різні типи операцій, такі як додавання товару на склад, вилучення товару зі складу, оформлення продажу, взяття/погашення боргу тощо. Тестування включає перевірку правильності виконання цих операцій.
Обробка помилок	Додаток повинен коректно обробляти помилки та відображати відповідні повідомлення або статуси помилок. Тестування включає перевірку поведінки додатку при некоректних або неповних даних, перевірку повернених повідомлень про помилки.
Робота з базою даних	Додаток використовує базу даних SQLite для зберігання даних про товари, борги, продажі тощо. Тестування включає перевірку правильності зберігання, оновлення та видалення даних в базі даних.
Інтеграція з Телеграм-ботом	Додаток інтегрований з Телеграм-ботом для роботи з боржниками. Тестування включає перевірку коректності реєстрації боржників, надсилання повідомлень та нагадувань про борги.

Зважаючи на визначені характеристики, тест-план для перевірки додатку може включати наступні види тестування:

1. Функціональне тестування – перевірка правильного виконання основних функцій додатку в звичайних умовах використання.

2. Тестування інтерфейсу користувача (GUI) – перевірка коректного відображення елементів інтерфейсу, реакції на введення даних, навігації та зручності використання.
3. Тестування бази даних – перевірка коректності операцій вставки, оновлення, видалення та вибірки даних з бази даних SQLite.
4. Тестування безпеки – перевірка додатку на вразливості, коректну обробку некоректних даних та захист від несанкціонованого доступу.
5. Інтеграційне тестування – перевірка взаємодії між різними компонентами системи, зокрема інтеграції з Телеграм-ботом.

4.2 Розробка тест-кейсів

Функціональні можливості програми тестуються за допомогою методу тестування «чорного ящика». Цей метод використовує перевірку вимог і специфікацій функціональності, не маючи доступу до внутрішньої структури коду або бази даних. Конкретні набори даних надаються на рівні звичайного користувача, а результат порівнюється з очікуваними даними.

Тест-кейси були розроблені для того, щоб переконатися, що функції системи відповідають встановленим вимогам. Тест-кейси можуть бути позитивними або негативними, залежно від очікуваного результату: тест-кейс, який є позитивним, використовує лише коректні дані і перевіряє, чи виконує додаток функцію, для якої він призначений.

Негативний тест-кейс використовує як коректні, так і некоректні дані, включаючи принаймні один неправильний параметр, щоб перевірити виняткові ситуації та переконатися, що функція не виконується при некоректних даних.

У таблицях 4.2-4.3 наведено розроблені тест-кейси для різних видів тестування додатку «Магазин "Даринка"».

Таблиця 4.2

Тест-кейси для функціонального тестування

№	ОК/ NOK	Дії	Очікуваний результат
1	ОК	1. Відкрити вкладку «Продаж» 2. Знайти товар у полі пошуку 3. Натиснути кнопку «Додати»	Товар додано до кошика
2	ОК	1. Додати товар до кошика 2. Змінити кількість товару в полі «Кількість» 3. Натиснути кнопку «Застосувати»	Кількість товару в кошику змінена
3	ОК	1. Додати товар до кошика 2. Натиснути кнопку «Видалити»	Товар видалено з кошика
4	ОК	1. Додати товари до кошика 2. . Вибрати спосіб оплати «Готівка» 3. Ввести суму отриманих грошей 4. Натиснути кнопку «Розрахувати» 5. Натиснути кнопку «Оплатити»	Продаж оформлено, відображено решту
5	ОК	1. Додати товари до кошика 2. . Вибрати спосіб оплати «Картка» 3. Ввести суму отриманих грошей 4. Натиснути кнопку «Розрахувати» 5. Натиснути кнопку «Оплатити»	Продаж оформлено
6	ОК	1. Додати товари до кошика 2. . Вибрати спосіб оплати «Борг» 3. Натиснути кнопку «Оплатити»	Продаж оформлено, боржника додано в базу даних

Продовження таблиці 4.2

Тест-кейси для функціонального тестування

№	ОК/ NOK	Дії	Очікуваний результат
7	ОК	1. Відкрити вкладку «Склад» 2. Натиснути "Додати товар" 3. Знайти товар у полі пошуку 4. Натиснути «Додати»	Товар додано до списку товарів на складі
8	ОК	1. Відкрити вкладку «Склад» 2. Знайти товар у списку 3. Змінити кількість товару 4. Натиснути «Застосувати»	Кількість товару на складі змінена
9	ОК	1. Відкрити вкладку «Боржники» 2. Натиснути «Додати» 3. Ввести ім’я та суму боргу	Боржник доданий
10	ОК	1. Відкрити вкладку «Боржники» 2. Вибрати боржника 3. Натиснути кнопку «Сплатити борг» 4. Ввести суму	Сума боргу боржника зменшується на введену суму

Таблиця 4.3

Тест-кейси для тестування інтерфейсу користувача (GUI)

№	**ОК/ NOK**	**Дії**	**Очікуваний результат**
1	ОК	1. Запустити додаток	Головне вікно відображається коректно
2	ОК	1. Запустити додаток 2. Послідовно переходити між вкладками «Продаж», «Склад», «Боржники», «Історія», «Налаштування», «Закрити зміну»	Вкладки змінюються коректно
3	ОК	1. Відкрити вкладку «Продаж» 2. Ввести текст у поле пошуку товару 3. Ввести кількість товару 4. Ввести суму оплати	Дані вводяться коректно без помилок

Всі тест-кейси, зазначені в даному розділі, пройдено успішно, що підтверджує відповідність додатку визначеним функціональним та нефункціональним вимогам.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було розроблено додаток «Магазин "Даринка"» для автоматизації обліку товарів, продажів та роботи з боржниками у продуктових міні-маркетах. Створене програмне забезпечення відповідає поставленим вимогам та задовольняє потреби користувачів.

Основними результатами даної роботи є:

1. Проведено аналіз предметної області обліку товарів у продуктових міні-маркетах, визначено ключові бізнес-процеси та вимоги до додатку.
2. Досліджено та проаналізовано існуюче програмне забезпечення для автоматизації роздрібної торгівлі, визначено їх переваги та недоліки.
3. Розроблено архітектуру та спроектовано основні модулі і компоненти додатку.
4. Реалізовано функціональність додатку, яка включає облік товарів, управління продажами, роботу з боржниками, ведення історії змін та формування звітів.
5. Інтегровано додаток з Телеграм-ботом для зручності нагадування боржникам про їхні борги.
6. Проведено тестування додатку з використанням різних видів тестів, що забезпечує його якість та відповідність вимогам.

Впровадження розробленого додатку дозволить підвищити ефективність обліку товарів у міні-маркетах, скоротити витрати часу персоналу на ручну обробку даних, забезпечить точність та актуальність інформації про товарні запаси, продажі та роботу з боржниками. Інтеграція з Телеграм-ботом спростить взаємодію з боржниками та сприятиме своєчасному погашенню боргів.

Апробація результатів дослідження:

1. Кононенко Д.А., Золотухіна О.А. Розробка додатку для автоматизації обліку товарів та продажів у продуктовому міні-маркеті. Всеукраїнська

науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24. квітня 2024 р., Київ, Державний університет інформаційно-комунікаійних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 434-435.

2. Кононенко Д.А., Ільїн О.Ю. Застосування методів штучного інтелекту для автоматизації управління боргами в роздрібній торгівлі. Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 р., Київ, Державний університет інформаційно-комунікаійних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 29-30.

ПЕРЕЛІК ПОСИЛАНЬ

1. Додаток Dilovod [Електронний ресурс] – Режим доступу до ресурсу: https://dilovod.ua/
2. Додаток HugeProfit [Електронний ресурс] – Режим доступу до ресурсу: https://h-profit.com/
3. Додаток «Квітка Торгівля» [Електронний ресурс] – Режим доступу до ресурсу: https://www.kvitkaprog.com/
4. Sundaramoorthy, Suriya. UML diagramming: a case study approach. Auerbach Publications, 2022. 430 с.
5. McKinney, Wes. Python for data analysis. " O'Reilly Media, Inc.", 2022. 582 с.
6. Willman, Joshua M. "Getting Started with PyQt." Beginning PyQt: A Hands-on Approach to GUI Programming with PyQt6. Berkeley, CA: Apress, 2022. 543 с.
7. Martelli, Alex, et al. Python in a Nutshell. " O'Reilly Media, Inc.", 2023. 738 с.
8. Toledo, Leandro. "Python-telegram-bot Documentation." (2024). 935 с.
9. Willman, Joshua M. "Presenting Data in PyQt." Beginning PyQt: A Hands-on Approach to GUI Programming with PyQt6. Berkeley, CA: Apress, 2022. 273-297.
10. Стецюра, В. О., and Т. Г. Китайчук. "ЗАГАЛЬНІ ЗАСАДИ ОБЛІКУ ТОВАРІВ НА ПІДПРИЄМСТВАХ РОЗДРІБНОЇ ТОРГІВЛІ." РЕДАКЦІЙНА КОЛЕГІЯ: 337.
11. Карнаушенко, А. С. "Сучасний стан та перспективи розвитку мінімаркетів в Україні." Підприємництво та інновації 18 (2021): 22-27.
12. Савченко, Л. А. "РОЗРОБКА АВТОМАТИЗОВАНОГО ОБЛІКУ ТА РУХУ ТОВАРІВ НА СКЛАДАХ." Проблеми та перспективи розвитку технічних та біоенергетичних систем природокористування. XX Міжнародна конференція науково-педагогічних працівників, наукових співробітників та аспірантів: збірник тез. м. Київ, Україна, 23–27 березня

2020 року. Київ. 2020. 126 с. Збірник тез рекомендовано до друку рішенням науково-технічної ради (2020): С. 69.

13. Новак, Уляна, and Марія Падюка. "ТОВАРНИЙ КРЕДИТ: ПРАВОВІ АСПЕКТИ І БУХГАЛТЕРСЬКИЙ ОБЛІК." Економіка та суспільство 34 (2021).
14. Кононенко Д.А., Золотухіна О.А. Розробка додатку для автоматизації обліку товарів та продажів у продуктовому міні-маркеті. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24. квітня 2024 р., Київ, Державний університет інформаційно-комунікаійних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 434-435.
15. Кононенко Д.А., Ільїн О.Ю. Застосування методів штучного інтелекту для автоматизації управління боргами в роздрібній торгівлі. Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 р., Київ, Державний університет інформаційно-комунікаійних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 29-30.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО -КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО -НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка застосунку для обліку товарів у продуктовому міні-маркеті мовою Python

Виконала студентка 4 курсу
Групи ПД-41
Кононенко Дар'я Анатоліївна
Керівник роботи
д.т.н., проф., професор кафедри ІПЗ Олег ІЛЬЇН

Київ – 2024

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи** - спрощення процесу обліку товарів у продуктовому міні-маркеті за рахунок використання застосунку, розробленого мовою Python.
- **Об'єкт дослідження -** процес обліку товарів у продуктовому міні-маркеті.
- **Предмет дослідження –** програмне забезпечення для автоматизації обліку товарів у продуктовому міні-маркеті.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Провести аналіз процесів обліку товарів у продуктових міні-маркетах.
2. Виконати огляд існуючих програмних рішень для управління запасами та обліку продажів у роздрібній торгівлі. Визначити їх переваги та недоліки.
3. Провести огляд та в вибір засобів реалізації програмного забезпечення, інструментів та технологій розробки.
4. Спроектувати та розробити програмний додаток для обліку товарів у продуктовому міні - маркеті мовою Python з використанням PyQt та SQLite:
   - реалізувати ключову функціональність внесення, редагування, видалення товарів, обліку надходжень та залишків;
   - забезпечити можливість продажу товарів з підтримкою різних способів оплати та роботою з боржниками ;
   - інтегрувати додаток з Telegram Bot API для надсилання нагадувань боржникам;
   - реалізувати формування звітності про продажі, оплати, залишки товарів та управління змінами роботи.
5. Виконати тестування та налагодження розробленого програмного забезпечення

АНАЛІЗ АНАЛОГІВ

Показник/функція	**Dilovod**	**HugeProfit**	**“Квітка Торгівля”**	**Магазин «Даринка»**
Платформа	Web	Android, iOs, Web	Windows	Windows
Облік товарів	+	+	+	+
Завантаження номенклатури	+ (з Excel)	+	+	+
Ведення клієнтської бази	+	+	-	+
Формування замовлень/рахунків	+	+	+	+
Облік продажів	+	+	+	+
Інвентаризація	+	+	+	+
Аналітичні звіти	+	+	+	+
Підтримка штрих -кодів	+	+	+	+
Інтеграція з фіскальними принтерами	+	+	+	Зайва для стейкхолдерів функція
Адаптація під різні типи магазинів	-	+	+	Зайва для стейкхолдерів функція
Робота з боржниками	-	-	-	+
Нагадування про борг через Телеграм-бот	-	-	-	+

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:

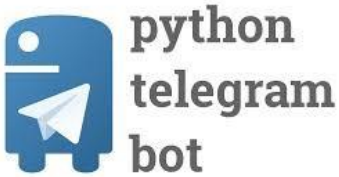
1. Внесення, редагування та видалення інформації про товари.
2. Облік надходження товарів на склад.
3. Відстеження залишків товару.
4. Продаж товарів з можливістю пошуку, сканування штрих -коду, вибору кількості та способу оплати.
5. Ведення бази даних боржників, внесення нових боржників, погашення боргів.
6. Надсилання нагадувань боржникам.
7. Формування звітів про продажі, оплати та залишки товарів.
8. Управління змінами роботи міні -маркету та перегляд історії змін.

Нефункціональні вимоги:

1. Інформація про товари повинна включати поля: назва, тип, штрих -код, ціна, акциз.
2. Система повинна підтримувати фіксацію продажів з наступними типами плати: готівка, картка, борг.
3. Надсилання нагадувань боржникам повинно виконуватись через інтеграцію з Телеграм -ботом.

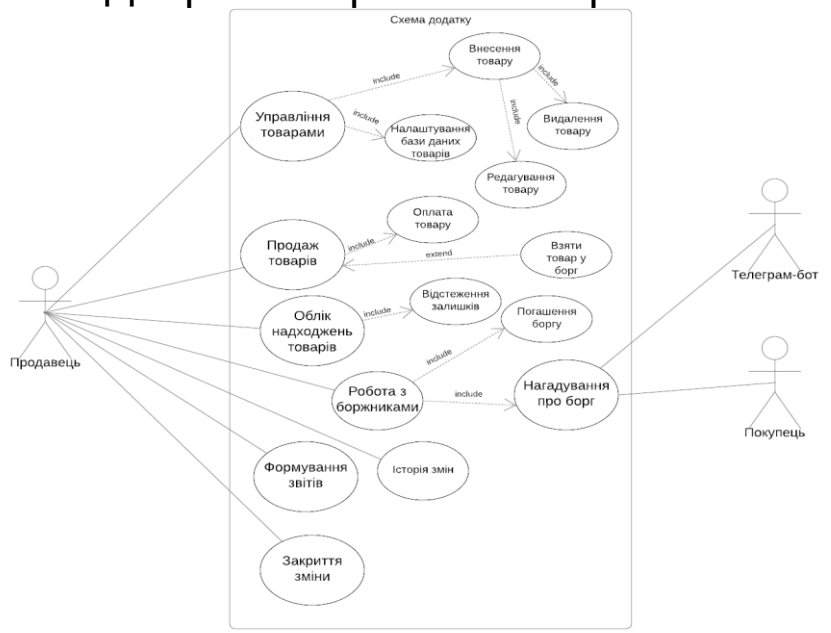
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



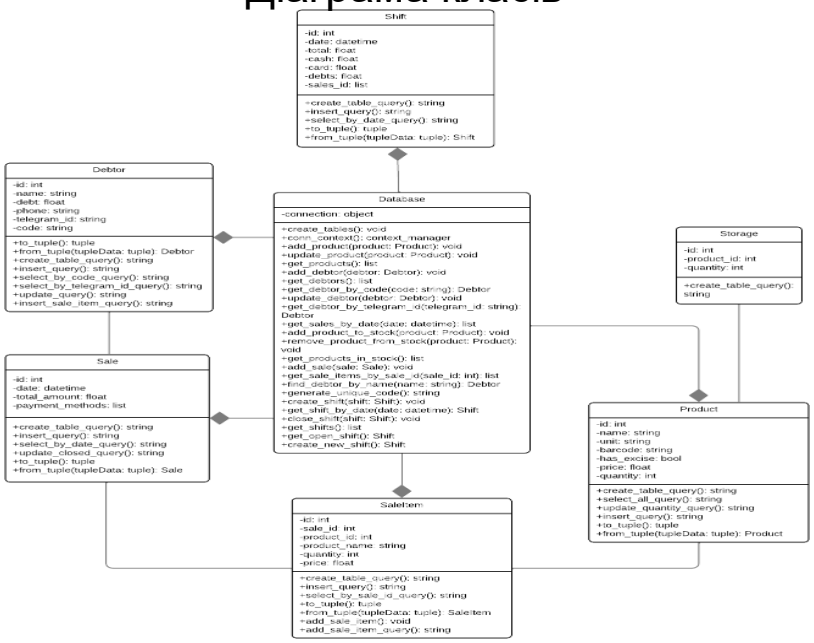
6

Діаграма варіантів використання



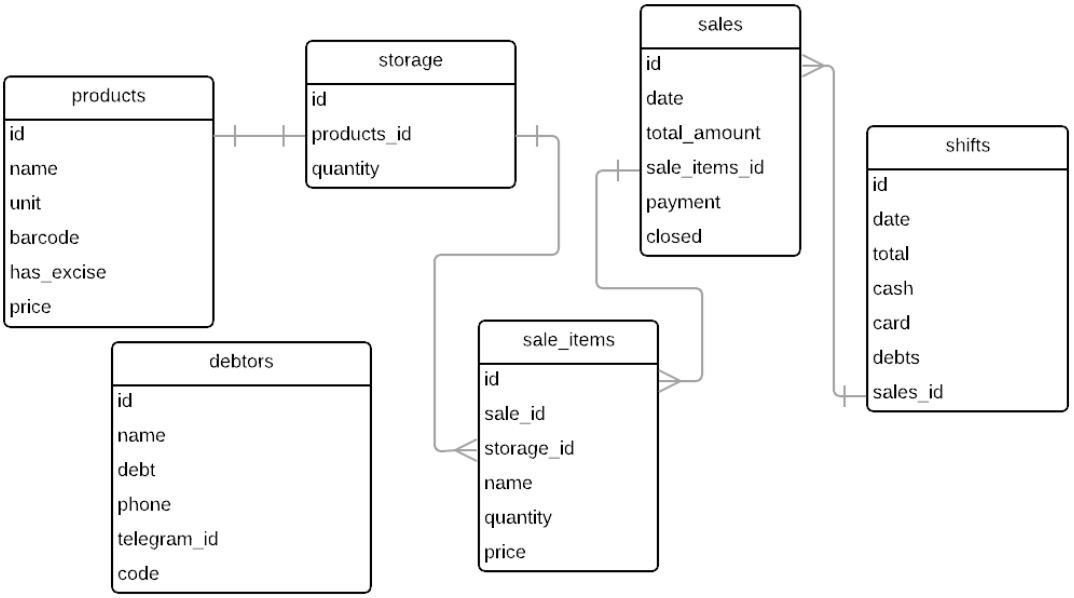
7

Діаграма класів



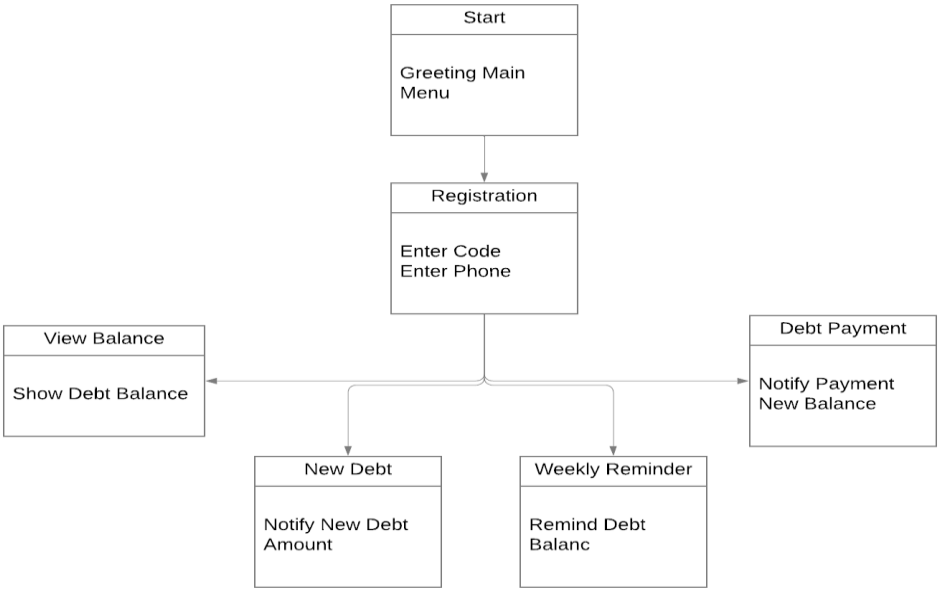
8

Схема бази даних



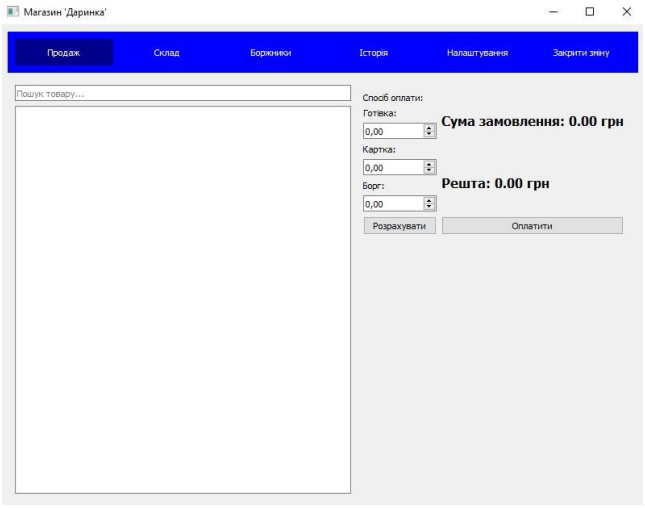
9

Граф діалогів Telegram-боту

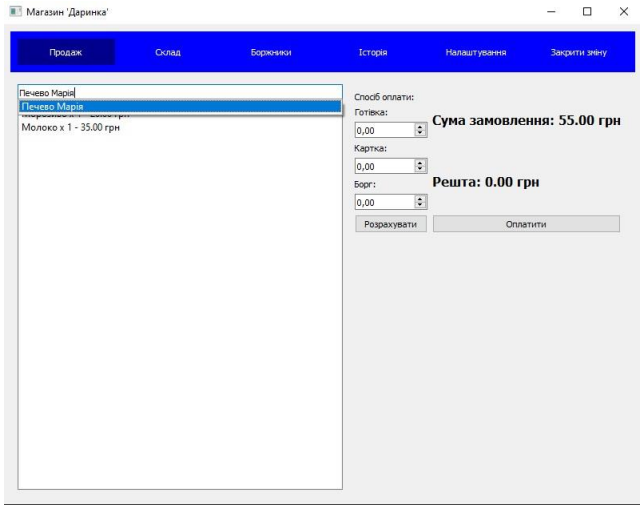


10

ЕКРАННІ ФОРМИ



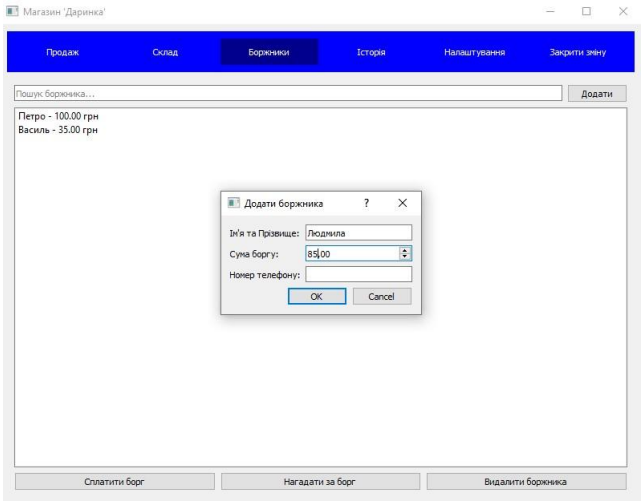
Вікно продажу



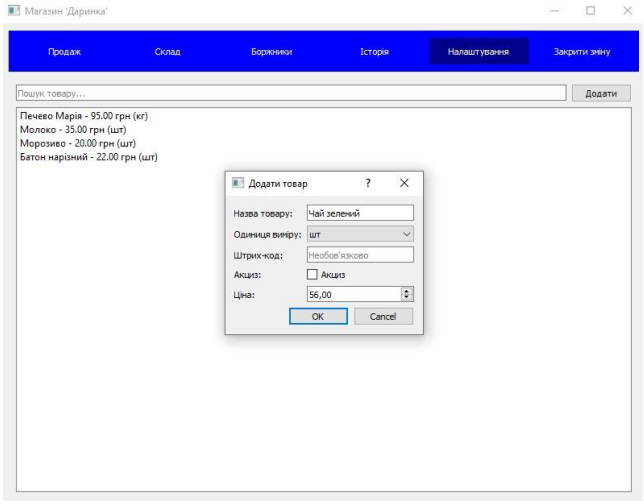
Додавання товарів у кошик

11

ЕКРАННІ ФОРМИ



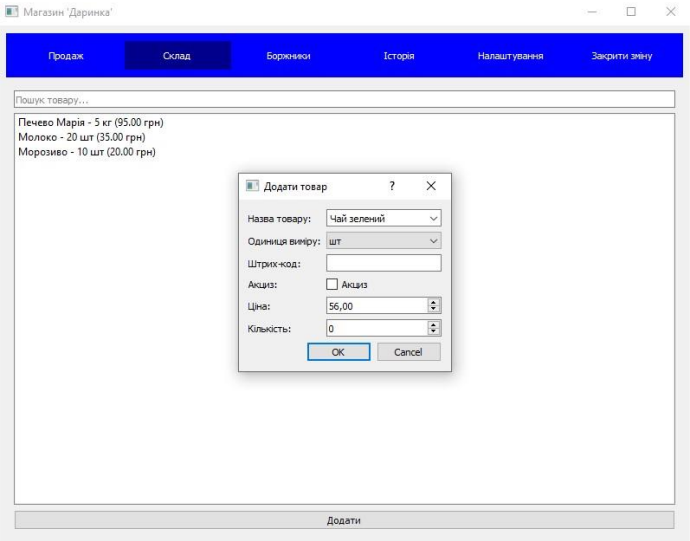
Додавання боржників



Додавання товару в налаштуваннях

12

ЕКРАННІ ФОРМИ



Додавання товару в склад

13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Кононенко Д.А., Золотухіна О.А. Розробка додатку для автоматизації обліку товарів та продажів у продуктовому міні-маркеті. Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в інформаційно-комунікаційних технологіях», 24. квітня 2024 р., Київ, Державний університет інформаційно-комунікаійних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 434-435.

2. Кононенко Д.А., Ільїн О.Ю. Застосування методів штучного інтелекту для автоматизації управління боргами в роздрібній торгівлі. Всеукраїнська науково-практична конференція «Сучасні інтелектуальні інформаційні технології в науці та освіті», 15 травня 2024 р., Київ, Державний університет інформаційно-комунікаійних технологій. Збірник тез. К.: ДУІКТ, 2024. С. 29-30.

15

ВИСНОВКИ

1. Проведено аналіз предметної області обліку товарів у продуктових мінімаркетах, визначено ключові бізнес-процеси та вимоги до додатку.
2. Досліджено та проаналізовано існуюче програмне забезпечення для автоматизації роздрібної торгівлі, визначено їх переваги та недоліки.
3. Розроблено архітектуру та спроектовано основні модулі і компоненти додатку.
4. Реалізовано функціональність додатку, яка включає облік товарів, управління продажами, роботу з боржниками, ведення історії змін та формування звітів.
5. Інтегровано додаток з Телеграм-ботом для зручності нагадування боржникам про їхні борги.
6. Проведено тестування додатку з використанням різних видів тестів, що забезпечує його якість та відповідністьвимогам.

ДОДАТОК Б. ЛІСТИНГИ ОСНОВНИХ МОДУЛІВ

1. main_window.py

```
from PyQt5.QtWidgets import QMainWindow, QStackedWidget, QWidget, QMessageBox, QPushButton, QVBoxLayout, QHBoxLayout
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import QDate

from ui.sale_widget import SaleWidget
from ui.stock_widget import StockWidget
from ui.debtors_widget import DebtorsWidget
from ui.history_widget import HistoryWidget
from ui.settings_widget import SettingsWidget

from db.database import Database

class MainWindow(QMainWindow):
    def __init__(self, db):
        super().__init__()
        self.setWindowTitle("Магазин 'Даринка'")
        self.resize(800, 600)
        self.db = db

        current_date = QDate.currentDate().toString("yyyy-MM-dd")
        open_shift = self.db.get_open_shift(current_date)

        if open_shift:
            # Якщо існує відкрита зміна, завантажити її
            self.load_open_shift(open_shift)
        else:
            # Інакше, створити нову зміну
            self.create_new_shift(current_date, 0, 0, 0, 0)

        # Створення області для відображення віджетів
        self.stacked_widget = QStackedWidget()
        self.setCentralWidget(self.stacked_widget)

        # Ініціалізація віджетів
        self.sale_widget = SaleWidget(db)
        self.stock_widget = StockWidget(self.db)
        self.debtors_widget = DebtorsWidget(self.db)
        self.history_widget = HistoryWidget(self.db)
        self.settings_widget = SettingsWidget(self.db)

        # Додавання віджетів до області відображення

self.stacked_widget.addWidget(self.sale_widget)

self.stacked_widget.addWidget(self.stock_widget)

self.stacked_widget.addWidget(self.debtors_widget)

self.stacked_widget.addWidget(self.history_widget)

self.stacked_widget.addWidget(self.settings_widget)

self.stacked_widget.setCurrentWidget(self.sale_widget)

        # Ініціалізація меню
        self.setup_menu()

    def setup_menu(self):
        button_container = QWidget()
        button_layout = QHBoxLayout()

        button_names = ["Продаж", "Склад", "Боржники", "Історія", "Налаштування", "Закрити зміну"]
```

```
        buttons = []

        for name in button_names:
            button = QPushButton(name, self)
            button.setStyleSheet("""
                QPushButton {
                    background-color: blue;
                    color: white;
                    border: none;
                    padding: 10px;
                }
                QPushButton:pressed {
                    background-color: darkblue;
                }
            """)
            button_layout.addWidget(button)
            buttons.append(button)

        buttons[0].clicked.connect(lambda:
self.show_widget(self.sale_widget, buttons[0]))
        buttons[1].clicked.connect(lambda:
self.show_widget(self.stock_widget, buttons[1]))
        buttons[2].clicked.connect(lambda:
self.show_widget(self.debtors_widget, buttons[2]))
        buttons[3].clicked.connect(lambda:
self.show_widget(self.history_widget, buttons[3]))
        buttons[4].clicked.connect(lambda:
self.show_widget(self.settings_widget, buttons[4]))
        buttons[5].clicked.connect(self.close_shift)

        button_container.setLayout(button_layout)


button_container.setStyleSheet("background-color:
blue;")

        main_layout = QVBoxLayout()
        main_layout.addWidget(button_container)

main_layout.addWidget(self.stacked_widget)

        container = QWidget()
        container.setLayout(main_layout)
        self.setCentralWidget(container)

        self.buttons = buttons

        self.show_widget(self.sale_widget,
buttons[0])

    def show_widget(self, widget, button):

self.stacked_widget.setCurrentWidget(widget)
        for btn in self.buttons:
            if btn == button:
                btn.setStyleSheet("""
                    QPushButton {
                        background-color: darkblue;
                        color: white;
                        border: none;
                        padding: 10px;
                    }
                """)
            else:
                btn.setStyleSheet("""
                    QPushButton {
                        background-color: blue;
                        color: white;
                        border: none;
                        padding: 10px;
                    }
                    QPushButton:pressed {
                        background-color: darkblue;
                    }
                """)

    def close_shift(self):
        current_date =
QDate.currentDate().toString("yyyy-MM-dd")
        sales =
self.db.get_sales_by_date(current_date)
        total_sales = sum(sale.total_amount for sale
in sales)
```

```
            cash_payments = sum(sale.total_amount for sale in sales if sale.payment_type == "Готівка")
            card_payments = sum(sale.total_amount for sale in sales if sale.payment_type == "Картка")
            debt_payments = sum(sale.total_amount for sale in sales if sale.payment_type == "Борг")

            report = f"Звіт зміни за {current_date}:\n\n"
            report += f"Сума продажів: {total_sales:.2f} грн\n"
            report += f"Оплата готівкою: {cash_payments:.2f} грн\n"
            report += f"Оплата карткою: {card_payments:.2f} грн\n"
            report += f"Борг: {debt_payments:.2f} грн"

            confirm_close = QMessageBox.question(self, "Закрити зміну", f"Ви дійсно хочете закрити зміну?\n\n{report}", QMessageBox.Yes | QMessageBox.No)

            if confirm_close == QMessageBox.Yes:
                self.db.close_shift(current_date, report)
                self.history_widget.load_shifts()
                QMessageBox.information(self, "Зміна закрита", report)

        def load_open_shift(self, shift):
            shift_id, shift_date, total, cash, cards, debts, report = shift

            # Завантажити продажі для відкритої зміни
            sales = self.db.get_sales_by_date(shift_date)

            # Відобразити продажі на віджеті продажу
            self.sale_widget.load_sales(sales)

            # Встановити поточну зміну як активну
            self.stacked_widget.setCurrentWidget(self.sale_widget)

        def create_new_shift(self, date, total_sales, cash_payments, card_payments, debts):
            with self.db.conn_context():
                self.db.connection.execute("""
                    INSERT INTO shifts (date, total, cash, card, debts)
                    VALUES (?, ?, ?, ?, ?)
                """, (date, total_sales, cash_payments, card_payments, debts))
```

2. stock_widget.py

```
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QComboBox, QHBoxLayout, QLineEdit, QListWidget, QPushButton, QSpinBox, QListWidgetItem, QDialog, QFormLayout, QCheckBox, QDoubleSpinBox, QDialogButtonBox
from PyQt5.QtCore import Qt

from db.models import Product, Storage

class StockWidget(QWidget):
    def __init__(self, database):
        super().__init__()
        self.db = database
        self.init_ui()
        self.load_products()

        # Створення атрибутів для віджетів введення
        self.unit_input = None
        self.barcode_input = None
        self.has_excise_checkbox = None
        self.price_input = None

    def init_ui(self):
        main_layout = QVBoxLayout()

        # Рядок пошуку товару
```

```
        search_layout = QHBoxLayout()
        self.search_input = QLineEdit()

self.search_input.setPlaceholderText("Пошук товару...")

self.search_input.textChanged.connect(self.filter_product
s)

search_layout.addWidget(self.search_input)
        main_layout.addLayout(search_layout)

        # Список товарів на складі
        self.product_list = QListWidget()

self.product_list.itemDoubleClicked.connect(self.edit_pr
oduct)
        main_layout.addWidget(self.product_list)

        # Рядок додавання товару
        add_layout = QHBoxLayout()
        add_button = QPushButton("Додати")

add_button.clicked.connect(self.add_product)
        add_layout.addWidget(add_button)
        main_layout.addLayout(add_layout)

        self.setLayout(main_layout)

    def load_products(self):
        self.product_list.clear()
        products = self.db.get_products_in_stock()
        if products is not None:
            for product in products:
                item =
QListWidgetItem(f"{product.name} - {product.quantity}
{product.unit} ({product.price:.2f} грн)")
                item.setData(Qt.UserRole, product)
                self.product_list.addItem(item)
        else:
            print("Failed to load products from
stock.")

    def filter_products(self, text):
        for i in range(self.product_list.count()):
            item = self.product_list.item(i)
            product = item.data(Qt.UserRole)
            if text.lower() in product.name.lower():
                item.setHidden(False)
            else:
                item.setHidden(True)

    def add_product(self):
        dialog = QDialog(self)
        dialog.setWindowTitle("Додати товар")
        form_layout = QFormLayout()

        name_input = QComboBox()
        name_input.addItems([product.name for
product in self.db.get_products()])
        name_input.setEditable(True)
        name_input.setCurrentIndex(-1)

name_input.currentIndexChanged.connect(self.fill_produ
ct_details)

        self.unit_input = QComboBox()
        self.unit_input.addItems(["шт", "кг"])

        self.barcode_input = QLineEdit()
        self.has_excise_checkbox =
QCheckBox("Акциз")
        self.price_input = QDoubleSpinBox()
        quantity_input = QSpinBox()

        form_layout.addRow("Назва товару:",
name_input)
        form_layout.addRow("Одиниця виміру:",
self.unit_input)
        form_layout.addRow("Штрих-код:",
self.barcode_input)
        form_layout.addRow("Акциз:",
self.has_excise_checkbox)
        form_layout.addRow("Ціна:",
self.price_input)
```

```
        form_layout.addRow("Кількість:",
quantity_input)

        button_box =
QDialogButtonBox(QDialogButtonBox.Ok |
QDialogButtonBox.Cancel)

button_box.accepted.connect(dialog.accept)
        button_box.rejected.connect(dialog.reject)

        main_layout = QVBoxLayout()
        main_layout.addLayout(form_layout)
        main_layout.addWidget(button_box)

        dialog.setLayout(main_layout)

        if dialog.exec() == QDialog.Accepted:
            product_name =
name_input.currentText()
            product =
self.db.get_product_by_name(product_name)
            if product:
                quantity = quantity_input.value()
                self.db.add_product_to_stock(product,
quantity)
            else:
                # Створення нового продукту в базі
даних
                unit = self.unit_input.currentText()
                barcode = self.barcode_input.text()
                has_excise =
self.has_excise_checkbox.isChecked()
                price = self.price_input.value()
                new_product = Product(None,
product_name, unit, barcode, has_excise, price)
                self.db.add_product(new_product)

self.db.add_product_to_stock(new_product,
quantity_input.value())

            self.load_products()

    def edit_product(self, item):
        # Редагування інформації про товар
        if item:
            product = item.data(Qt.UserRole)
            dialog = QDialog(self)
            dialog.setWindowTitle(f"Редагувати
товар: {product.name}")
            form_layout = QFormLayout()
            name_input = QLineEdit(product.name)
            unit_input = QLineEdit(product.unit)
            quantity_input = QSpinBox()

quantity_input.setValue(product.quantity)
            barcode_input =
QLineEdit(product.barcode)
            has_excise_checkbox =
QCheckBox("Акциз")

has_excise_checkbox.setChecked(product.has_excise)
            price_input = QDoubleSpinBox()
            price_input.setValue(product.price)
            form_layout.addRow("Назва товару:",
name_input)
            form_layout.addRow("Одиниця
виміру:", unit_input)
            form_layout.addRow("Кількість:",
quantity_input)
            form_layout.addRow("Штрих-код:",
barcode_input)
            form_layout.addRow("Акциз:",
has_excise_checkbox)
            form_layout.addRow("Ціна:",
price_input)

            button_box =
QDialogButtonBox(QDialogButtonBox.Ok |
QDialogButtonBox.Cancel)

button_box.accepted.connect(dialog.accept)

button_box.rejected.connect(dialog.reject)
```

```
                delete_button =
QPushButton("Видалити товар")
                delete_button.clicked.connect(lambda:
self.remove_product_from_storage(product.id))

                main_layout = QVBoxLayout()
                main_layout.addLayout(form_layout)
                main_layout.addWidget(button_box)
                main_layout.addWidget(delete_button)
                dialog.setLayout(main_layout)
                if dialog.exec() == QDialog.Accepted:
                    product.name = name_input.text()
                    product.unit = unit_input.text()
                    product.barcode =
barcode_input.text()
                    product.has_excise =
has_excise_checkbox.isChecked()
                    product.price = price_input.value()
                    product.quantity =
quantity_input.value()
                    self.db.update_product(product)  #
Оновлення інформації про продукт
                    self.load_products()

            def fill_product_details(self, index):
                if index >= 0:
                    product = self.db.get_products()[index]
                    if self.unit_input:

self.unit_input.setCurrentText(product.unit)
                    if self.barcode_input:

self.barcode_input.setText(product.barcode)
                    if self.has_excise_checkbox:

self.has_excise_checkbox.setChecked(product.has_excis
e)
                    if self.price_input:

self.price_input.setValue(product.price)
                else:
                    if self.unit_input:
                        self.unit_input.setCurrentText("")
                    if self.barcode_input:
                        self.barcode_input.clear()
                    if self.has_excise_checkbox:

self.has_excise_checkbox.setChecked(False)
                    if self.price_input:
                        self.price_input.setValue(0.0)

            def remove_product_from_storage(self,
product_id):
                conn = self.db.connect()
                cursor = conn.cursor()

                try:
                    # Перевірити, чи є продукт у сховищі
                    cursor.execute("SELECT quantity
FROM storage WHERE product_id = ?", (product_id,))
                    result = cursor.fetchone()

                    if result:
                        quantity = result[0]
                        if quantity > 0:
                            # Оновити кількість товару в
сховищі

cursor.execute(Storage.update_quantity_query(), (0,
product_id))
                            conn.commit()
                        else:
                            print(f"Товару з ідентифікатором
{product_id} немає на складі.")
                    else:
                        print(f"Товару з ідентифікатором
{product_id} немає на складі.")

                except Exception as e:
                    print(f"Помилка під час видалення
товару зі сховища: {str(e)}")
                    conn.rollback()

                finally:
```

```
        conn.close()
```

3. debtors_widget.py

```
from PyQt5.QtWidgets import QWidget, QVBoxLayout, QHBoxLayout, QLineEdit, QListWidget, QPushButton, QLabel, QDoubleSpinBox, QListWidgetItem, QDialog, QDialogButtonBox, QFormLayout, QVBoxLayout
from PyQt5.QtCore import Qt

from db.models import Debtor
from bot.telegram_bot import send_message_to_debtor

class DebtorsWidget(QWidget):
    def __init__(self, database):
        super().__init__()
        self.db = database
        self.init_ui()
        self.load_debtors()

    def init_ui(self):
        main_layout = QVBoxLayout()

        # Рядок пошуку боржника
        search_layout = QHBoxLayout()
        self.search_input = QLineEdit()
self.search_input.setPlaceholderText("Пошук боржника...")
self.search_input.textChanged.connect(self.filter_debtors)
search_layout.addWidget(self.search_input)

        add_button = QPushButton("Додати")
add_button.clicked.connect(self.add_debtor)
        search_layout.addWidget(add_button)

        main_layout.addLayout(search_layout)

        # Список боржників
        self.debtor_list = QListWidget()
self.debtor_list.itemDoubleClicked.connect(lambda: self.edit_debtor(self.debtor_list.currentItem().data(Qt.UserRole)))
        main_layout.addWidget(self.debtor_list)

        # Управління боржником
        debtor_actions_layout = QHBoxLayout()

        pay_debt_button = QPushButton("Сплатити борг")
pay_debt_button.clicked.connect(self.pay_debt)
debtor_actions_layout.addWidget(pay_debt_button)

        remind_button = QPushButton("Нагадати за борг")
remind_button.clicked.connect(self.remind_debt)
debtor_actions_layout.addWidget(remind_button)

        remove_debtor_button = QPushButton("Видалити боржника")
remove_debtor_button.clicked.connect(self.remove_debtor)
debtor_actions_layout.addWidget(remove_debtor_button)

main_layout.addLayout(debtor_actions_layout)

        self.setLayout(main_layout)

    def load_debtors(self):
        debtors = self.db.get_debtors()
```

```
        for debtor in debtors:
            item = QListWidgetItem(f"{debtor.name} - {debtor.debt:.2f} грн")
            item.setData(Qt.UserRole, debtor)
            self.debtor_list.addItem(item)

    def filter_debtors(self, text):
        for i in range(self.debtor_list.count()):
            item = self.debtor_list.item(i)
            debtor = item.data(Qt.UserRole)
            if text.lower() in debtor.name.lower():
                item.setHidden(False)
            else:
                item.setHidden(True)

    def add_debtor(self):
        # Відкриття діалогового вікна для додавання нового боржника
        dialog = QDialog(self)
        dialog.setWindowTitle("Додати боржника")

        form_layout = QFormLayout()
        name_input = QLineEdit()
        debt_input = QDoubleSpinBox()
        phone_input = QLineEdit()

        form_layout.addRow("Ім'я та Прізвище:", name_input)
        form_layout.addRow("Сума боргу:", debt_input)
        form_layout.addRow("Номер телефону:", phone_input)

        button_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

        button_box.accepted.connect(dialog.accept)
        button_box.rejected.connect(dialog.reject)

        main_layout = QVBoxLayout()
        main_layout.addLayout(form_layout)
        main_layout.addWidget(button_box)

        dialog.setLayout(main_layout)

        if dialog.exec() == QDialog.Accepted:
            name = name_input.text()
            debt = debt_input.value()
            phone = phone_input.text()

            code = self.db.generate_unique_code()
            new_debtor = Debtor(None, name, debt, phone, None, code)
            self.db.add_debtor(new_debtor)
            self.debtor_list.clear()
            self.load_debtors()

    def pay_debt(self):
        selected_items = self.debtor_list.selectedItems()
        if selected_items:
            debtor = selected_items[0].data(Qt.UserRole)
            dialog = QDialog(self)
            dialog.setWindowTitle(f"Сплата боргу для {debtor.name}")
            form_layout = QFormLayout()

            debt_label = QLabel(f"Сума боргу: {debtor.debt:.2f} грн")
            amount_input = QDoubleSpinBox()
            amount_input.setMaximum(debtor.debt)
            amount_input.setValue(debtor.debt)
            form_layout.addRow(debt_label)
            form_layout.addRow("Сума для сплати:", amount_input)

            button_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
```

```
button_box.accepted.connect(dialog.accept)

button_box.rejected.connect(dialog.reject)

                main_layout = QVBoxLayout()
                main_layout.addLayout(form_layout)
                main_layout.addWidget(button_box)
                dialog.setLayout(main_layout)

                if dialog.exec() == QDialog.Accepted:
                    amount_paid = amount_input.value()
                    if amount_paid >= debtor.debt:
                        debtor.debt = 0
                    else:
                        debtor.debt -= amount_paid
                    self.db.update_debtor(debtor)
                    self.debtor_list.clear()
                    self.load_debtors()
                    if debtor.telegram_id:
                        message = f"Ви сплатили {amount_paid:.2f} грн зі свого боргу. Залишок: {debtor.debt:.2f} грн"

send_message_to_debtor(debtor.telegram_id, message)

            def remind_debt(self):
                selected_items = self.debtor_list.selectedItems()

                if selected_items:
                    debtor = selected_items[0].data(Qt.UserRole)

                    if debtor.telegram_id:
                        message = f"Нагадування про борг!\n\nВаш баланс боргу: {debtor.debt:.2f} грн"

send_message_to_debtor(debtor.telegram_id, message)
                    else:
                        print(f"Боржник {debtor.name} не зареєстрований у Telegram-боті.")

            def edit_debtor(self, debtor):
                dialog = QDialog(self)
                dialog.setWindowTitle(f"Редагувати боржника: {debtor.name}")
                form_layout = QFormLayout()

                name_input = QLineEdit(debtor.name)
                form_layout.addRow("Ім'я та Прізвище:", name_input)

                debt_input = QDoubleSpinBox()
                debt_input.setValue(debtor.debt)
                form_layout.addRow("Сума боргу:", debt_input)

                phone_input = QLineEdit(debtor.phone)
                form_layout.addRow("Номер телефону:", phone_input)

                code_label = QLabel(debtor.code)
                form_layout.addRow("Унікальний код:", code_label)

                add_debt_input = QDoubleSpinBox()
                add_debt_input.setMinimum(0.0)
                form_layout.addRow("Додати борг:", add_debt_input)

                button_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)

button_box.accepted.connect(dialog.accept)
                button_box.rejected.connect(dialog.reject)

                main_layout = QVBoxLayout()
                main_layout.addLayout(form_layout)
                main_layout.addWidget(button_box)
                dialog.setLayout(main_layout)

                if dialog.exec() == QDialog.Accepted:
```

```
                debtor.name = name_input.text()
                debtor.phone = phone_input.text()
                debtor.debt += add_debt_input.value()
                self.db.update_debtor(debtor)
                self.debtor_list.clear()
                self.load_debtors()

        def remove_debtor(self):
            selected_items = self.debtor_list.selectedItems()
            if selected_items:
                debtor = selected_items[0].data(Qt.UserRole)
                self.db.delete_debtor(debtor)
                self.debtor_list.clear()
                self.load_debtors()
```

4. sale_widget.py

```
    from PyQt5.QtWidgets import QWidget, QVBoxLayout, QHBoxLayout, QLineEdit, QSpinBox, QGridLayout, QCompleter, QListWidget, QLabel, QComboBox, QPushButton, QInputDialog, QListWidgetItem, QDoubleSpinBox, QDialog, QDialogButtonBox
    from PyQt5.QtCore import Qt, QDate, QStringListModel
    from PyQt5.QtGui import QCursor
    from PyQt5.QtMultimedia import QCameraInfo, QCamera
    from PyQt5.QtMultimediaWidgets import QCameraViewfinder

    from db.models import Debtor, Sale

    from bot.telegram_bot import send_message_to_debtor

    class SaleWidget(QWidget):
        def __init__(self, database):
            super().__init__()
            self.db = database
            self.cart = []
            self.total_price = 0
            self.init_ui()

            self.camera = QCamera()
            self.viewfinder = QCameraViewfinder()
            available_cameras = QCameraInfo.availableCameras()
            if available_cameras:
                self.camera = QCamera(available_cameras[0])

            self.camera.setCaptureMode(QCamera.CaptureViewfinder)

            self.camera.setViewfinder(self.viewfinder)
            self.viewfinder.show()
            self.camera.start()

            self.viewfinder.setCursor(QCursor(Qt.CrossCursor))

        def init_ui(self):
            main_layout = QHBoxLayout()

            # Left side: search and cart
            left_layout = QVBoxLayout()

            search_layout = QHBoxLayout()
            self.search_input = QLineEdit()

            self.search_input.setPlaceholderText("Пошук товару...")

            self.search_input.returnPressed.connect(self.add_to_cart)

            search_layout.addWidget(self.search_input)

            self.product_completer = QCompleter(self.get_product_names(), self)

            self.product_completer.setCaseSensitivity(Qt.CaseInsensitive)

            self.search_input.setCompleter(self.product_completer)
```

```
        left_layout.addLayout(search_layout)

        self.cart_list = QListWidget()
self.cart_list.itemDoubleClicked.connect(self.edit_cart_item)
        left_layout.addWidget(self.cart_list, stretch=1)

        main_layout.addLayout(left_layout, stretch=1)

        # Right side: payment details
        right_widget = QWidget()
        right_layout = QVBoxLayout()

        order_details_layout = QGridLayout()
        payment_layout = QVBoxLayout()
        payment_label = QLabel("Спосіб оплати:")
payment_layout.addWidget(payment_label)

        self.cash_input = QDoubleSpinBox()
        self.cash_input.setDecimals(2)
        self.cash_input.setRange(0.0, 999999999.99)
        self.cash_input.setValue(0.0)
        cash_label = QLabel("Готівка:")
        payment_layout.addWidget(cash_label)
payment_layout.addWidget(self.cash_input)

        self.card_input = QDoubleSpinBox()
        self.card_input.setDecimals(2)
        self.card_input.setRange(0.0, 999999999.99)
        self.card_input.setValue(0.0)
        card_label = QLabel("Картка:")
        payment_layout.addWidget(card_label)
payment_layout.addWidget(self.card_input)

        self.debt_input = QDoubleSpinBox()
        self.debt_input.setDecimals(2)
        self.debt_input.setRange(0.0, 999999999.99)
        self.debt_input.setValue(0.0)
        debt_label = QLabel("Борг:")
        payment_layout.addWidget(debt_label)
payment_layout.addWidget(self.debt_input)

        calculate_button = QPushButton("Розрахувати")
calculate_button.clicked.connect(self.calculate_change)
payment_layout.addWidget(calculate_button)

order_details_layout.addLayout(payment_layout, 0, 0)

        total_layout = QVBoxLayout()
        self.total_label = QLabel(f"Сума замовлення: {self.total_price:.2f} грн")
        self.total_label.setStyleSheet("font-weight: bold; font-size: 16px;")
        total_layout.addWidget(self.total_label)

        self.change_label = QLabel("Решта: 0.00 грн")
        self.change_label.setStyleSheet("font-weight: bold; font-size: 16px;")
        total_layout.addWidget(self.change_label)

        pay_button = QPushButton("Оплатити")
pay_button.clicked.connect(self.process_payment)
        total_layout.addWidget(pay_button)
```

```
order_details_layout.addLayout(total_layout, 0, 1)


right_layout.addLayout(order_details_layout)
            right_layout.addStretch(1)  # Add a stretchable space at the bottom to push everything up
            right_widget.setLayout(right_layout)

            main_layout.addWidget(right_widget)

            self.setLayout(main_layout)

        def scan_barcode(self):
            barcode = self.barcode_scanner.scan()
            if barcode:
                product = self.db.get_product_by_barcode(barcode)
                if product:
                    self.add_to_cart(product, 1)
                else:
                    pass

        def get_product_names(self):
            products = self.db.get_products_in_stock()
            if products:
                return [product.name for product in products]
            return []

        def filter_products(self, text):

self.product_completer.setModel(QStringListModel([product.name for product in self.db.get_products_in_stock() if text.lower() in product.name.lower()]))


        def process_payment(self):
            payment_methods = {
                "Готівка": self.cash_input.value(),
                "Картка": self.card_input.value(),
                "Борг": self.debt_input.value()
            }

            # If debt is involved, handle debtor information
            debtor = None
            if self.debt_input.value() > 0:
                debtor_name, ok = QInputDialog.getText(self, "Ім'я боржника", "Введіть ім'я та прізвище боржника:")
                if ok and debtor_name:
                    matched_debtors = self.db.find_debtor_by_name(debtor_name)
                    if matched_debtors:
                        debtor_list = [f"{debtor.name} ({debtor.phone})" for debtor in matched_debtors]
                        debtor_choice, ok = QInputDialog.getItem(self, "Вибір боржника", "Виберіть боржника з списку:", debtor_list, 0, False)
                        if ok:
                            selected_debtor = matched_debtors[debtor_list.index(debtor_choice)]
                            debtor = selected_debtor

self.add_debt_to_existing_debtor(debtor)
                    else:
                        debtor = self.create_new_debtor(debtor_name)

            # Create sale object
            sale = Sale(None, QDate.currentDate().toString("yyyy-MM-dd"), self.total_price, payment_methods)

            # Insert sale into the database and get the generated sale id
            cursor = self.db.connection.cursor()
            cursor.execute(Sale.insert_query(), sale.to_tuple())
            self.db.connection.commit()
            sale.id = cursor.lastrowid

            # Add sale items to the database
```

```
            for item in self.cart:
                sale.add_sale_item(self.db, item["product"], item["quantity"])

            self.db.update_product_quantity(item["product"], -item["quantity"])  # Update product quantity in stock

            # Update shift information
            current_date = QDate.currentDate().toString("yyyy-MM-dd")
            open_shift = self.db.get_open_shift(current_date)
            if open_shift:
                open_shift.total += self.total_price
                open_shift.cash += self.cash_input.value()
                open_shift.card += self.card_input.value()
                open_shift.debts += self.debt_input.value()
                self.db.update_shift(open_shift)

            self.clear_cart()

        def add_to_cart(self):
            product_name = self.search_input.text()
            self.search_input.clear()
            product = self.db.get_product_by_name(product_name)

            if product:
                item = {
                    "product": product,
                    "quantity": 1
                }
                self.cart.append(item)
                self.update_cart_list()
                self.update_total_price()

        def remove_from_cart(self, cart_item, item_index):
            if cart_item in self.cart:
                self.cart.remove(cart_item)
                self.cart_list.takeItem(item_index)
                self.update_total_price()

        def update_cart_list(self):
            self.cart_list.clear()
            for item in self.cart:
                product_name = item["product"].name
                quantity = item["quantity"]
                price = item["product"].price * quantity
                cart_item = QListWidgetItem(f"{product_name} x {quantity} - {price:.2f} грн")
                cart_item.setData(Qt.UserRole, item)
                self.cart_list.addItem(cart_item)

        def update_total_price(self):
            self.total_price = sum(item["product"].price * item["quantity"] for item in self.cart)
            self.total_label.setText(f"Сума замовлення: {self.total_price:.2f} грн")

        def update_cart_quantity(self, cart_item, new_quantity, item_index):
            if new_quantity > 0:
                self.cart[item_index]["quantity"] = new_quantity
                self.update_cart_list()
                self.update_total_price()
            else:
                self.remove_from_cart(cart_item, item_index)

        def calculate_change(self):
            total_paid = self.cash_input.value() + self.card_input.value() + self.debt_input.value()
            change = total_paid - self.total_price
            self.change_label.setText(f"Решта: {change:.2f} грн")

        def remind_debt(self):
```

```
            selected_items = self.debtor_list.selectedItems()

            if selected_items:
                debtor = selected_items[0].data(Qt.UserRole)

                if debtor.telegram_id:
                    message = f"Нагадування про борг!\n\nВаш баланс боргу: {debtor.debt:.2f} грн"

send_message_to_debtor(debtor.telegram_id, message)
                else:
                    print(f"Боржник {debtor.name} не зареєстрований у Telegram-боті.")

        def handle_debt_payment(self):
            name, ok = QInputDialog.getText(self, "Ім'я боржника", "Введіть ім'я та прізвище боржника:")

            if ok and name:
                debtor = self.find_debtor_by_name(name)

                if debtor:

self.add_debt_to_existing_debtor(debtor)

                    if debtor.telegram_id:
                        message = f"Ви взяли товар на суму {self.total_price:.2f} грн у борг в магазині 'Даринка'.\nВаш баланс боргу: {debtor.debt:.2f} грн"

send_message_to_debtor(debtor.telegram_id, message)
                    elif debtor.phone:
                        print(f"Надіслано повідомлення 'Ви взяли товар на суму {self.total_price:.2f} грн у борг в магазині 'Даринка'.' на номер {debtor.phone}")
                    else:
                        print(f"Не вдалося надіслати повідомлення для {debtor.name}")
                else:
                    self.create_new_debtor(name)
                    message = f"Ви взяли товар на суму {self.total_price:.2f} грн у борг в магазині 'Даринка'.\n\nВаш унікальний код боржника: {debtor.code}"

send_message_to_debtor(debtor.telegram_id, message)

                self.clear_cart()

        def find_debtor_by_name(self, name):
            matched_debtors = self.db.find_debtor_by_name(name)
            if matched_debtors:
                debtor_list = [f"{debtor.name} ({debtor.phone})" for debtor in matched_debtors]
                debtor_choice, ok = QInputDialog.getItem(self, "Вибір боржника", "Виберіть боржника з списку:", debtor_list, 0, False)
                if ok:
                    selected_debtor = matched_debtors[debtor_list.index(debtor_choice)]
                    return selected_debtor
            return None

        def create_new_debtor(self, name):
            code = self.db.generate_unique_code()
            new_debtor = Debtor(None, name, self.debt_input.value(), None, None, code)
            self.db.add_debtor(new_debtor)
            message = f"Ви взяли товар на суму {self.debt_input.value():.2f} грн у борг в магазині 'Даринка'.\n\nВаш унікальний код боржника: {code}"

send_message_to_debtor(new_debtor.telegram_id, message)
            return new_debtor

        def add_debt_to_existing_debtor(self, debtor):
            debtor.debt += self.debt_input.value()
```

```
            self.db.update_debtor(debtor)
            # Send message about new debt through Telegram bot
            if debtor.telegram_id:
                message = f"Ви взяли товар на суму {self.debt_input.value():.2f} грн у борг в магазині 'Даринка'.\n\nВаш баланс боргу: {debtor.debt:.2f} грн"

send_message_to_debtor(debtor.telegram_id, message)

        def edit_cart_item(self, item):
            cart_item = item.data(Qt.UserRole)
            item_index = self.cart_list.indexFromItem(item).row()
            dialog = QDialog(self)
            dialog.setWindowTitle("Редагувати кількість")
            layout = QVBoxLayout()

            quantity_input = QSpinBox()

quantity_input.setValue(cart_item["quantity"])
            layout.addWidget(quantity_input)

            button_box = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
            button_box.accepted.connect(lambda: self.update_cart_quantity(cart_item, quantity_input.value(), item_index))

button_box.accepted.connect(dialog.accept)
            button_box.rejected.connect(dialog.reject)
            layout.addWidget(button_box)

            remove_button = QPushButton("Видалити")
            remove_button.clicked.connect(lambda: self.remove_from_cart(cart_item, item_index))

remove_button.clicked.connect(dialog.accept)
            layout.addWidget(remove_button)
            dialog.setLayout(layout)
            dialog.exec_()

        def clear_cart(self):
            self.cart.clear()
            self.cart_list.clear()
            self.total_price = 0
            self.total_label.setText(f"Сума замовлення: {self.total_price:.2f} грн")
            self.change_label.setText("Решта: 0.00 грн")
```

5. models.py

```
class Product:
    def __init__(self, id, name, unit, barcode, has_excise, price):
        self.id = id
        self.name = name
        self.unit = unit
        self.barcode = barcode
        self.has_excise = has_excise
        self.price = price

    @staticmethod
    def create_table_query():
        return """
            CREATE TABLE IF NOT EXISTS products (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                unit TEXT NOT NULL,
                barcode TEXT UNIQUE,
                has_excise INTEGER NOT NULL,
                price REAL NOT NULL
            )
        """

    @staticmethod
    def select_all_query():
        return """
```

```
            SELECT id, name, unit, barcode, has_excise, price
            FROM products
        """

    @staticmethod
    def update_quantity_query():
        return """
            UPDATE products
            SET quantity = ?
            WHERE id = ?
        """

    @staticmethod
    def insert_query():
        return """
            INSERT INTO products (name, unit, barcode, has_excise, price)
            VALUES (?, ?, ?, ?, ?)
        """

    def to_tuple(self):
        return (self.name, self.unit, self.barcode, self.has_excise, self.price)

    @staticmethod
    def from_tuple(row):
        id, name, unit, barcode, has_excise, price = row
        return Product(id, name, unit, barcode, has_excise, price)

class Debtor:
    def __init__(self, id, name, debt, phone, telegram_id, code):
        self.id = id
        self.name = name
        self.debt = debt
        self.phone = phone
        self.telegram_id = telegram_id
        self.code = code

    def to_tuple(self):
        return (self.name, self.debt, self.phone, self.telegram_id, self.code)

    def from_tuple(row):
        return Debtor(id=row[0], name=row[1], debt=row[2], phone=row[3], telegram_id=row[4], code=row[5])

    @staticmethod
    def create_table_query():
        return """
        CREATE TABLE IF NOT EXISTS debtors (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            debt REAL,
            phone TEXT,
            telegram_id INTEGER,
            code TEXT UNIQUE
        )
        """

    @staticmethod
    def insert_query():
        return """
        INSERT INTO debtors (name, debt, phone, telegram_id, code) VALUES (?, ?, ?, ?, ?)
        """

    @staticmethod
    def select_by_code_query():
        return "SELECT * FROM debtors WHERE code = ?"

    @staticmethod
    def select_by_telegram_id_query():
        return "SELECT * FROM debtors WHERE telegram_id = ?"

    @staticmethod
```

```
        def insert_query():
            return "INSERT INTO debtors (name, debt, phone, telegram_id, code) VALUES (?, ?, ?, ?, ?)"

        @staticmethod
        def update_query():
            return "UPDATE debtors SET name = ?, debt = ?, phone = ?, telegram_id = ? WHERE id = ?"


    class Sale:
        def __init__(self, id, date, total_amount, payment_methods, closed=False):
            self.id = id
            self.date = date
            self.total_amount = total_amount
            self.payment_methods = payment_methods or {}
            self.closed = closed

        @staticmethod
        def create_table_query():
            return """
                CREATE TABLE IF NOT EXISTS sales (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    date TEXT NOT NULL,
                    total_amount REAL NOT NULL,
                    cash_amount REAL NOT NULL,
                    card_amount REAL NOT NULL,
                    debt_amount REAL NOT NULL
                )
            """

        @staticmethod
        def insert_query():
            return """
                INSERT INTO sales (date, total_amount, cash_amount, card_amount, debt_amount)
                VALUES (?, ?, ?, ?, ?)
            """

        @staticmethod
        def select_by_date_query():
            return """
                SELECT id, date, total_amount, cash_amount, card_amount, debt_amount
                FROM sales
                WHERE date = ?
            """

        @staticmethod
        def update_closed_query():
            return """
                UPDATE sales
                SET closed = 1
                WHERE date = ?
            """

        def to_tuple(self):
            return (self.date, self.total_amount, self.payment_methods.get('Готівка', 0), self.payment_methods.get('Картка', 0), self.payment_methods.get('Борг', 0))

        @staticmethod
        def from_tuple(row):
            id, date, total_amount, cash_amount, card_amount, debt_amount = row
            payment_methods = {
                'Готівка': cash_amount,
                'Картка': card_amount,
                'Борг': debt_amount
            }
            return Sale(id, date, total_amount, payment_methods)

    class SaleItem:
        def __init__(self, id, sale_id, product_id, product_name, quantity, price):
            self.id = id
            self.sale_id = sale_id
            self.product_id = product_id
```

```
        self.product_name = product_name
        self.quantity = quantity
        self.price = price

    @staticmethod
    def create_table_query():
        return """
            CREATE TABLE IF NOT EXISTS sale_items (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                sale_id INTEGER NOT NULL,
                product_id INTEGER NOT NULL,
                product_name TEXT NOT NULL,
                quantity INTEGER NOT NULL,
                price REAL NOT NULL,
                FOREIGN KEY (sale_id) REFERENCES sales(id),
                FOREIGN KEY (product_id) REFERENCES products(id)
            )
        """

    @staticmethod
    def insert_query():
        return """
            INSERT INTO sale_items (sale_id, product_id, product_name, quantity, price)
            VALUES (?, ?, ?, ?, ?)
        """

    @staticmethod
    def select_by_sale_id_query():
        return """
            SELECT id, sale_id, product_id, product_name, quantity, price
            FROM sale_items
            WHERE sale_id = ?
        """

    def to_tuple(self):
        return (self.sale_id, self.product_id, self.product_name, self.quantity, self.price)

    @staticmethod
    def from_tuple(row):
        return SaleItem(*row)

    @staticmethod
    def add_sale_item_query():
        return """
            INSERT INTO sale_items (sale_id, product_id, product_name, quantity, price)
            VALUES (?, ?, ?, ?, ?)
        """

    def add_sale_item(self, db, product, quantity):
        sale_item = SaleItem(None, self.id, product.id, product.name, quantity, product.price * quantity)

db.connection.execute(Sale.add_sale_item_query(), sale_item.to_tuple())

class Shift:
    def __init__(self, id, date, total, cash, card, debts, report=None):
        self.id = id
        self.date = date
        self.total = total
        self.cash = cash
        self.card = card
        self.debts = debts
        self.report = report

    @staticmethod
    def create_table_query():
        return """
            CREATE TABLE IF NOT EXISTS shifts (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                date TEXT NOT NULL,
```

```
                total REAL NOT NULL,
                cash REAL NOT NULL,
                card REAL NOT NULL,
                debts REAL NOT NULL,
                report TEXT
            )
        """

    @staticmethod
    def insert_query():
        return """
            INSERT INTO shifts (date, total, cash, card, debts)
            VALUES (?, ?, ?, ?, ?)
        """

    @staticmethod
    def select_by_date_query():
        return """
            SELECT id, date, total, cash, card, debts
            FROM shifts
            WHERE date = ?
        """

    def to_tuple(self):
        return (self.date, self.total, self.cash, self.card, self.debts)

    @staticmethod
    def from_tuple(row):
        return Shift(*row)

class Storage:
    def __init__(self, id, product_id, quantity):
        self.id = id
        self.product_id = product_id
        self.quantity = quantity

    @staticmethod
    def create_table_query():
        return """
            CREATE TABLE IF NOT EXISTS storage (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                product_id INTEGER NOT NULL,
                quantity INTEGER NOT NULL,
                FOREIGN KEY (product_id) REFERENCES products(id)
            )
        """

    @staticmethod
    def update_quantity_query():
        return """
            UPDATE storage
            SET quantity = ?
            WHERE product_id = ?
        """
```

6. database.py

```
import sqlite3
from contextlib import contextmanager
from db.models import Product, Storage, Debtor, Sale, SaleItem, Shift

class Database:
    def __init__(self, db_path):
        self.connection = sqlite3.connect(db_path)
        self.create_tables()

    def create_tables(self):
        with self.conn_context():
            self.connection.execute(Debtor.create_table_query())
            self.connection.execute(Storage.create_table_query())
            self.connection.execute(Sale.create_table_query())
            self.connection.execute(SaleItem.create_table_query())
            self.connection.execute(Product.create_table_query())
```

```
self.connection.execute(Shift.create_table_query())

    @contextmanager
    def conn_context(self):
        try:
            yield self.connection
        except sqlite3.Error as e:
            print(f"Error: {e}")
        finally:
            self.connection.commit()

    def add_product(self, product):
        with self.conn_context():

self.connection.execute(Product.insert_query(),
product.to_tuple())

    def update_product(self, product):
        with self.conn_context():
            self.connection.execute("""
                UPDATE products
                SET name = ?, unit = ?, barcode = ?,
has_excise = ?, price = ?
                WHERE id = ?
            """, (product.name, product.unit,
product.barcode, product.has_excise, product.price,
product.id))

    def get_products(self):
        with self.conn_context():
            try:
                cursor =
self.connection.execute(Product.select_all_query())
                return [Product.from_tuple(row) for
row in cursor.fetchall()]
            except sqlite3.Error as e:
                print(f"Error while retrieving products:
{e}")
                return []

    def update_product_quantity(self, product,
quantity_change):
        with self.conn_context():

self.connection.execute(Storage.update_quantity_query()
, (quantity_change, product.id))



    def add_debtor(self, debtor):
        with self.conn_context():

self.connection.execute(Debtor.insert_query(),
debtor.to_tuple())

    def get_debtors(self):
        with self.conn_context():
            try:
                cursor =
self.connection.execute("SELECT * FROM debtors")
                rows = cursor.fetchall()
                return [Debtor.from_tuple(row) for
row in rows]
            except sqlite3.Error as e:
                print(f"Error while retrieving debtors:
{e}")
                return []

    def get_debtor_by_code(self, code):
        with self.conn_context():
            cursor =
self.connection.execute(Debtor.select_by_code_query(),
(code,))
            row = cursor.fetchone()
            if row:
                return Debtor.from_tuple(row)
            return None

    def update_debtor(self, debtor):
        with self.conn_context():
            self.connection.execute(
                Debtor.update_query(),
```

```
                (debtor.name, debtor.debt, debtor.phone, debtor.telegram_id, debtor.id)
            )

    def delete_debtor(self, debtor):
        with self.conn_context():
            self.connection.execute("DELETE FROM debtors WHERE id = ?", (debtor.id,))

    def get_debtor_by_telegram_id(self, telegram_id):
        with self.conn_context():
            cursor = self.connection.execute(Debtor.select_by_telegram_id_query(), (telegram_id,))
            row = cursor.fetchone()
            if row:
                return Debtor.from_tuple(row)
            return None

    def get_sales_by_date(self, date):
        with self.conn_context():
            cursor = self.connection.execute(Sale.select_by_date_query(), (date,))
            rows = cursor.fetchall()
            return [Sale.from_tuple(row) for row in rows]

    def add_product_to_stock(self, product, quantity):
        with self.conn_context():
            cursor = self.connection.execute("""
                SELECT id FROM storage WHERE product_id = ?
            """, (product.id,))
            row = cursor.fetchone()
            if row:
                storage_id = row[0]
                self.connection.execute("""
                    UPDATE storage SET quantity = quantity + ? WHERE id = ?
                """, (quantity, storage_id))
            else:
                self.connection.execute("""
                    INSERT INTO storage (product_id, quantity) VALUES (?, ?)
                """, (product.id, quantity))

    def remove_product_from_stock(self, product):
        with self.conn_context():
            self.connection.execute("""
                DELETE FROM products WHERE id = ?
            """, (product.id,))

    def get_products_in_stock(self):
        with self.conn_context():
            try:
                cursor = self.connection.execute("""
                    SELECT products.id, products.name, products.unit, products.barcode, products.has_excise, products.price, storage.quantity
                    FROM products
                    JOIN storage ON products.id = storage.product_id
                    WHERE storage.quantity > 0
                """)
                rows = cursor.fetchall()
                products = []
                for row in rows:
                    product = Product(*row[:6])
                    storage_item = Storage(None, product.id, row[6])
                    setattr(product, 'quantity', storage_item.quantity)
                    products.append(product)
                return products
            except sqlite3.Error as e:
                print(f"Error while retrieving products from stock: {e}")
                return None
```

```
        def get_product_by_name(self, name):
            with self.conn_context():
                cursor = self.connection.execute("SELECT * FROM products WHERE name = ?", (name,))
                row = cursor.fetchone()
                if row:
                    return Product.from_tuple(row)
                return None

        def add_sale(self, sale):
            with self.conn_context():

self.connection.execute(Sale.insert_query(), sale.to_tuple())
                sale.id = self.connection.execute("SELECT last_insert_rowid()").fetchone()[0]
                for item in sale.items:
                    sale.add_sale_item(self, item["product"], item["quantity"])

        def get_sale_items_by_sale_id(self, sale_id):
            with self.conn_context():
                cursor = self.connection.execute(SaleItem.select_by_sale_id_query(), (sale_id,))
                rows = cursor.fetchall()
                return [SaleItem.from_tuple(row) for row in rows]

        def find_debtor_by_name(self, name):
            with self.conn_context():
                cursor = self.connection.execute("SELECT * FROM debtors WHERE name LIKE ?", (f"%{name}%",))
                rows = cursor.fetchall()
                if rows:
                    return [Debtor.from_tuple(row) for row in rows]
                return []

        def generate_unique_code(self):
            import random
            import string

            def generate_code():
                return ''.join(random.choices(string.digits, k=6))

            code = generate_code()
            while self.get_debtor_by_code(code):
                code = generate_code()
            return code

        def create_shift(self, date, total, cash, card, debts, sales_id):
            with self.conn_context():
                shift = Shift(None, date, total, cash, card, debts, sales_id)

self.connection.execute(Shift.insert_query(), shift.to_tuple())

        def get_shift_by_date(self, date):
            with self.conn_context():
                cursor = self.connection.execute(Shift.select_by_date_query(), (date,))
                row = cursor.fetchone()
                if row:
                    return Shift.from_tuple(row)
                return None

        def close_shift(self, date, report):
            with self.conn_context():
                self.connection.execute("""
                    UPDATE shifts SET report = ? WHERE date = ?
                """, (report, date))

        def get_shifts(self):
            with self.conn_context():
```

```
            cursor = self.connection.execute("SELECT * FROM shifts")
            return cursor.fetchall()

        def get_open_shift(self, date):
            with self.conn_context():
                cursor = self.connection.execute("SELECT * FROM shifts WHERE date = ?", (date,))
                row = cursor.fetchone()
                if row:
                    return Shift.from_tuple(row)
                return None

        def create_new_shift(self, date):
            with self.conn_context():
                self.connection.execute("INSERT INTO shifts (date, closed) VALUES (?, 0)", (date,))
```

7. telegram_bot.py

```
import sys
import os
import schedule
import time
import asyncio

from telegram import Bot, Update, KeyboardButton, ReplyKeyboardMarkup, ReplyKeyboardRemove
from telegram.ext import Application, CommandHandler, MessageHandler, filters, CallbackContext

# Додавання кореневого каталогу проекту в sys.path
sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), '..')))

from db.database import Database

# Ініціалізація бази даних
db = Database('store.db')
bot_token = '7084012548:AAGmf4y9vFUBWKApGBWZqqfc2_DQYXq-Zp0'
bot = Bot(token=bot_token)

async def send_message_to_debtor(telegram_id, message):
    try:
        await bot.send_message(chat_id=telegram_id, text=message)
    except Exception as e:
        print(f"Помилка при надсиланні повідомлення боржнику: {e}")

def remind_debtors():
    debtors = db.get_debtors()
    for debtor in debtors:
        if debtor.debt > 0 and debtor.telegram_id:
            message = f"Нагадування! Ваш баланс боргу: {debtor.debt:.2f} грн"

send_message_to_debtor(debtor.telegram_id, message)

schedule.every().sunday.do(remind_debtors)

def get_debtor_by_code(code):
    return db.get_debtor_by_code(code)

def get_debtor_by_telegram_id(telegram_id):
    return db.get_debtor_by_telegram_id(telegram_id)

async def start(update: Update, context: CallbackContext):
    keyboard = [
        [KeyboardButton("Поділитися номером телефону", request_contact=True)],
        [KeyboardButton("Пропустити")]
    ]
```

```
        reply_markup = ReplyKeyboardMarkup(keyboard, resize_keyboard=True, one_time_keyboard=True)
        await update.message.reply_text("Вітаю! Це бот магазину 'Даринка'. Будь ласка, поділіться своїм номером телефону або натисніть 'Пропустити'.", reply_markup=reply_markup)

    async def phone_number(update: Update, context: CallbackContext):
        contact = update.message.contact
        if contact is not None:
            await update.message.reply_text("Дякую! Тепер введіть свій унікальний код для підключення.", reply_markup=ReplyKeyboardRemove())
        else:
            await update.message.reply_text("Будь ласка, скористайтеся кнопкою для поділу номером телефону.")

    async def skip_phone_number(update: Update, context: CallbackContext):
        await update.message.reply_text("Добре, пропускаємо введення номера телефону. Тепер введіть свій унікальний код для підключення.", reply_markup=ReplyKeyboardRemove())

    async def register_code(update: Update, context: CallbackContext):
        code = update.message.text
        print(f"Реєстрація коду: {code}")
        debtor = get_debtor_by_code(code)
        if debtor:
            context.user_data['code'] = code
            debtor.telegram_id = update.effective_user.id
            db.update_debtor(debtor)
            await context.bot.send_message(chat_id=update.effective_chat.id, text="Ви успішно зареєстровані як боржник.", reply_markup=ReplyKeyboardMarkup([["Баланс"]], resize_keyboard=True))
        else:
            await context.bot.send_message(chat_id=update.effective_chat.id, text="Невірний код. Спробуйте ще раз.")

    async def get_balance(update: Update, context: CallbackContext):
        code = context.user_data.get('code')
        print(f"Перевірка балансу для коду: {code}")
        if code:
            debtor = get_debtor_by_code(code)
            print(f"Знайдено боржника: {debtor}")
            if debtor:
                await context.bot.send_message(chat_id=update.effective_chat.id, text=f"Ваш баланс боргу: {debtor.debt:.2f} грн")
            else:
                await context.bot.send_message(chat_id=update.effective_chat.id, text="Боржника з вашим кодом не знайдено.")
        else:
            await context.bot.send_message(chat_id=update.effective_chat.id, text="Ви не зареєстровані як боржник або не ввели код.")

    # Запуск бота
    def main():
        application = Application.builder().token(bot_token).build()

        application.add_handler(CommandHandler('start', start))
```

```
application.add_handler(MessageHandler(filters.CONT
ACT, phone_number))

application.add_handler(MessageHandler(filters.TEXT
& ~filters.COMMAND & filters.Regex("Пропустити"),
skip_phone_number))

application.add_handler(MessageHandler(filters.TEXT
& ~filters.COMMAND & ~filters.Regex("Баланс"),
register_code))

application.add_handler(MessageHandler(filters.TEXT
& filters.Regex('Баланс'), get_balance))

        application.run_polling()

        while True:
            schedule.run_pending()
            time.sleep(1)

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