Міністерство освіти і науки України

Національний технічний університет
«Дніпровська політехніка»

Навчально-науковий
інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

# ПОЯСНЮВАЛЬНА ЗАПИСКА
## кваліфікаційної роботи ступеня бакалавра

**здобувача** Михайличенко Родіон Анатолійович
(ПІБ)

**академічної групи** 123-21-2
(шифр)

**спеціальності** 123 Комп'ютерна інженерія
(код і назва спеціальності)

**за освітньо-професійною програмою** Комп'ютерна інженерія
(офіційна назва)

**на тему** "Програмний модуль для касової системи самообслуговування в супермаркеті з реалізацією протоколу обміну даними в реальному часі "

(назва за наказом ректора)

| Керівники | Прізвище, ініціали | Оцінка за шкалою | | Підпис |
|---|---|---|---|---|
| | | рейтинговою | інституційною | |
| кваліфікаційної роботи | доц. Каштан В.Ю. | | | |
| загального розділу | доц. Каштан В.Ю. | | | |
| спеціального розділу | доц. Каштан В.Ю. | | | |
| **Рецензент** | | | | |
| **Нормоконтролер** | проф. Цвіркун Л.І. | | | |

**Дніпро**
**2025**

**ЗАТВЕРДЖЕНО:**
завідувач кафедри
інформаційних технологій
та комп'ютерної інженерії
(повна назва)
_____ Гнатушенко В.В.
(підпис) (прізвище, ініціали)

"____" _____ 2025 року

# ЗАВДАННЯ
## на кваліфікаційну роботу
## ступеня бакалавр

**здобувача** Михайличенко Р.А. **академічної групи** 123-21-2
(прізвище та ініціали) (шифр)

**спеціальності** 123 Комп'ютерна інженерія

**за освітньо-професійною програмою** Комп'ютерна інженерія
(офіційна назва)

**на тему** "Програмний модуль для касової системи самообслуговування в супермаркеті з реалізацією протоколу обміну даними в реальному часі "

затверджену наказом ректора НТУ «Дніпровська політехніка» від 05.05.2025 № 336-с

| Розділ | Зміст | Термін виконання |
|---|---|---|
| Загальний розділ | На основі матеріалів виробничих практик, інших науково-технічних джерел показати актуальність завдання, сформулювати мету та задачі виконання кваліфікаційної роботи | 10.02.2025 |
| Спеціальний розділ | Сформулювати найменування й призначення програмного модуля, висунути технічні вимоги до нього | 15.03.2025 |
| | Розробити архітектуру програмного модуля, включаючи вибір оптимального протоколу обміну даними та визначення інтерфейсів взаємодії з іншими компонентами системи. | 20.04.2025 |
| | Реалізувати програмний модуль, використовуючи відповідні інструменти програмування та враховуючи принципи розробки надійних та ефективних програмних систем, що є частиною комп'ютерної інженерії | 07.05.2025 |
| | Протестувати розроблений програмний модуль на предмет його функціональності, продуктивності обміну даними в реальному часі та стійкості до можливих збоїв | 31.05.2025 |

**Завдання видано** _____ доц. Каштан В.Ю.
(підпис керівника) (прізвище, ініціали)

**Дата видачі** 25.02.2025

**Дата подання до екзаменаційної комісії** _____

**Прийнято до виконання** _____ Михайличенко Р.А.

# РЕФЕРАТ

Пояснювальна записка: 77 с., 19 рис., 1 табл., 1 додаток, 11 джерел.

КАСОВА СИСТЕМА, WEBSOCKET, ОБМІН ДАНИМИ В РЕАЛЬНОМУ ЧАСІ.

Об'єктом дослідження є процес обміну даними в касовій системі самообслуговування супермаркету як складна програмно-апаратна система.

Предметом дослідження є програмний модуль для касової системи самообслуговування з реалізацією протоколу обміну даними в реальному часі як елемент комп'ютерної інфраструктури торгового підприємства.

Метою даної бакалаврської роботи, що виконується в рамках спеціальності "Комп'ютерна інженерія", є розробка програмного модуля для касової системи самообслуговування в супермаркеті з реалізацією протоколу обміну даними в реальному часі.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз існуючих касових систем самообслуговування та протоколів обміну даними, що використовуються в них, з точки зору їхньої архітектури та принципів функціонування;

– визначити вимоги до програмного модуля касової системи самообслуговування з підтримкою обміну даними в реальному часі для супермаркету, враховуючи аспекти продуктивності, надійності та безпеки;

– розробити архітектуру програмного модуля, включаючи вибір оптимального протоколу обміну даними та визначення інтерфейсів взаємодії з іншими компонентами системи;

– реалізувати програмний модуль, використовуючи відповідні інструменти програмування та враховуючи принципи розробки надійних та ефективних програмних систем, що є частиною комп'ютерної інженерії;

– протестувати розроблений програмний модуль на предмет його функціональності, продуктивності обміну даними в реальному часі та стійкості до можливих збоїв.

# ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАК, ОДИНИЦЬ ТА ТЕРМІНІВ

| | | |
|---|---|---|
| КССО | – | касова система самообслуговування; |
| SMART | – | Self-Monitoring, Analysis, and Reporting Technology; |
| RFID | – | технологію радіочастотної ідентифікації; |
| IDE | – | Integrated Development Environment; |
| GUI | – | інтерфейс користувача; |
| КТЗ | – | комплекс технічних засобів. |

# ВСТУП

Супермаркети, прагнучи оптимізувати обслуговування клієнтів, підвищити ефективність та зменшити черги на касах, все частіше впроваджують складні програмно-апаратні комплекси, відомі як системи самообслуговування. Ці системи, що є результатом інтеграції апаратних засобів (сканерів, терміналів оплати, сенсорних екранів) та спеціалізованого програмного забезпечення, надають покупцям можливість самостійно сканувати товари, підраховувати вартість покупки та здійснювати оплату, що значно прискорює процес придбання товарів.

Одним із ключових завдань комп'ютерної інженерії при розробці таких систем є забезпечення надійного та швидкого обміну даними між терміналом самообслуговування та центральною системою супермаркету. Обмін даними в реальному часі є критично важливим для підтримки актуальності інформації про товари (ціни, наявність), обробки фінансових транзакцій та формування звітних документів. Ефективна реалізація цього обміну вимагає глибоких знань в області мережевих технологій, протоколів передачі даних та розробки розподілених систем, що є ключовими компетенціями фахівців з комп'ютерної інженерії.

У зв'язку з цим, розробка програмного модуля для касової системи самообслуговування, що забезпечує обмін даними в реальному часі, є актуальною задачею, яка лежить на перетині галузей розробки програмного забезпечення та комп'ютерної інженерії, та має практичне значення для підвищення якості обслуговування та ефективності роботи супермаркетів.

Метою даної бакалаврської роботи, що виконується в рамках спеціальності "Комп'ютерна інженерія", є розробка програмного модуля для касової системи самообслуговування в супермаркеті з реалізацією протоколу обміну даними в реальному часі.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз існуючих касових систем самообслуговування та

протоколів обміну даними, що використовуються в них, з точки зору їхньої архітектури та принципів функціонування;

– визначити вимоги до програмного модуля касової системи самообслуговування з підтримкою обміну даними в реальному часі для супермаркету, враховуючи аспекти продуктивності, надійності та безпеки, що є важливими в комп'ютерній інженерії;

– розробити архітектуру програмного модуля, включаючи вибір оптимального протоколу обміну даними та визначення інтерфейсів взаємодії з іншими компонентами системи;

– реалізувати програмний модуль, використовуючи відповідні інструменти програмування та враховуючи принципи розробки надійних та ефективних програмних систем, що є частиною комп'ютерної інженерії;

– протестувати розроблений програмний модуль на предмет його функціональності, продуктивності обміну даними в реальному часі та стійкості до можливих збоїв.

Об'єктом дослідження є процес обміну даними в касовій системі самообслуговування супермаркету як складна програмно-апаратна система.

Предметом дослідження є програмний модуль для касової системи самообслуговування з реалізацією протоколу обміну даними в реальному часі як елемент комп'ютерної інфраструктури торгового підприємства.

Практичне значення отриманих результатів полягає у можливості використання розробленого програмного модуля для модернізації існуючих або створення нових касових систем самообслуговування в супермаркетах, що дозволить підвищити швидкість обслуговування клієнтів та оптимізувати роботу торгового підприємства за рахунок впровадження сучасних рішень в області комп'ютерної інженерії..

# 1 ЗАГАЛЬНИЙ РОЗДІЛ

## 1.1 Призначення модуля для касової системи самообслуговування в супермаркеті

Впровадження касових систем самообслуговування (КССО) є стратегічним кроком для сучасних супермаркетів, спрямованим на підвищення операційної ефективності, покращення досвіду клієнтів та оптимізацію використання ресурсів. Програмний модуль, що є серцем такої системи, відіграє ключову роль у забезпеченні її безперебійної та продуктивної роботи.

Однією з головних переваг КССО є можливість одночасного обслуговування більшої кількості покупців порівняно з традиційними касами з касирами. Ефективний програмний модуль забезпечує швидку обробку товарів, мінімізує час очікування в черзі та підвищує пропускну здатність торгової точки, особливо в години пік.

Впровадження КССО дозволяє супермаркету перерозподілити персонал з виконання рутинних касових операцій на інші завдання, що вимагають більшої кваліфікації та людської взаємодії, наприклад, консультації покупців, викладку товарів або контроль за залом. Це сприяє підвищенню загальної продуктивності праці та оптимізації витрат на оплату праці.

Багато покупців цінують можливість самостійно контролювати процес сканування та оплати товарів, що надає їм відчуття більшої автономії та зручності. Інтуїтивно зрозумілий інтерфейс програмного модуля, швидка робота та підтримка різних способів оплати сприяють позитивному досвіду покупок та підвищують лояльність клієнтів.

Людський фактор може призводити до помилок при введенні цін або скануванні товарів на традиційних касах. Програмний модуль КССО, при правильній розробці та інтеграції з базами даних, значно мінімізує ризик таких помилок, забезпечуючи точність розрахунків та обліку товарів [1].

Програмний модуль КССО є важливим джерелом даних про продажі, поведінку покупців (наприклад, середній розмір чека, популярність певних товарів у певний час). Ці дані можуть бути використані для аналізу ефективності маркетингових кампаній, оптимізації асортименту товарів, планування закупівель та прийняття обґрунтованих управлінських рішень.

Сучасні програмні модулі для КССО розробляються з урахуванням можливості їхньої інтеграції з іншими системами супермаркету (наприклад, системами управління складом, платіжними системами, системами лояльності). Гнучка архітектура дозволяє легко додавати нові функції та масштабувати систему при розширенні торговельної мережі.

Програмний модуль повинен забезпечувати підтримку різноманітних способів оплати, включаючи готівкові розрахунки, банківські картки (контактні та безконтактні), мобільні платіжні системи та інші сучасні технології, що відповідає очікуванням сучасних покупців.

## 1.2 Вплив технологічного розвитку та зміна споживчих очікувань на роздрібну торгівлю

Прагнення компаній оптимізувати процес здійснення покупок та оплати клієнтами, забезпечуючи його більшу цілісність та інтегрованість, є прямим наслідком еволюції споживчих очікувань, зумовленої стрімким технологічним прогресом. З кожним днем нові технології все активніше впроваджуються в різноманітних галузях, а автоматизація набуває широкого поширення з метою мінімізації людського втручання. Сфера роздрібної торгівлі та шопінгу не є винятком. Цей сектор, особливо протягом останніх років, демонструє значну динаміку змін, де всі зацікавлені сторони галузі надають першочергове значення адаптації для збереження конкурентоздатності. Однією з помітних тенденцій, що виникає внаслідок цих трансформацій, є концепція безкасових магазинів.

Швидкий перехід від традиційних фізичних магазинів до електронної комерції, який став одним із найважливіших трендів останніх років, сприяв

значному прогресу галузі в багатьох аспектах, зумовлюючи необхідність впровадження інноваційних рішень. Однак, попри всі ці досягнення, дослідження, проведене в Сполучених Штатах, виявило, що споживачі не повністю готові відмовитися від фізичних магазинів. Незважаючи на цифрові зручності, клієнти все ще віддають перевагу фізичним торговим точкам через такі переваги, як миттєвий доступ до товарів та відсутність витрат на доставку [1], що підкреслює неминущу важливість фізичних магазинів. Аналогічно, дослідження еволюції продуктового ринку в Канаді (рис. 1.1) показало, що канадці планують здійснювати більшість своїх продуктових покупок у фізичних магазинах протягом наступних шести місяців. Це свідчить про те, що споживачі віддають перевагу багатоканальним підходам, таким як купівля онлайн з самовивозом у магазині, а не виключно онлайн-замовлення своїх продуктових потреб.

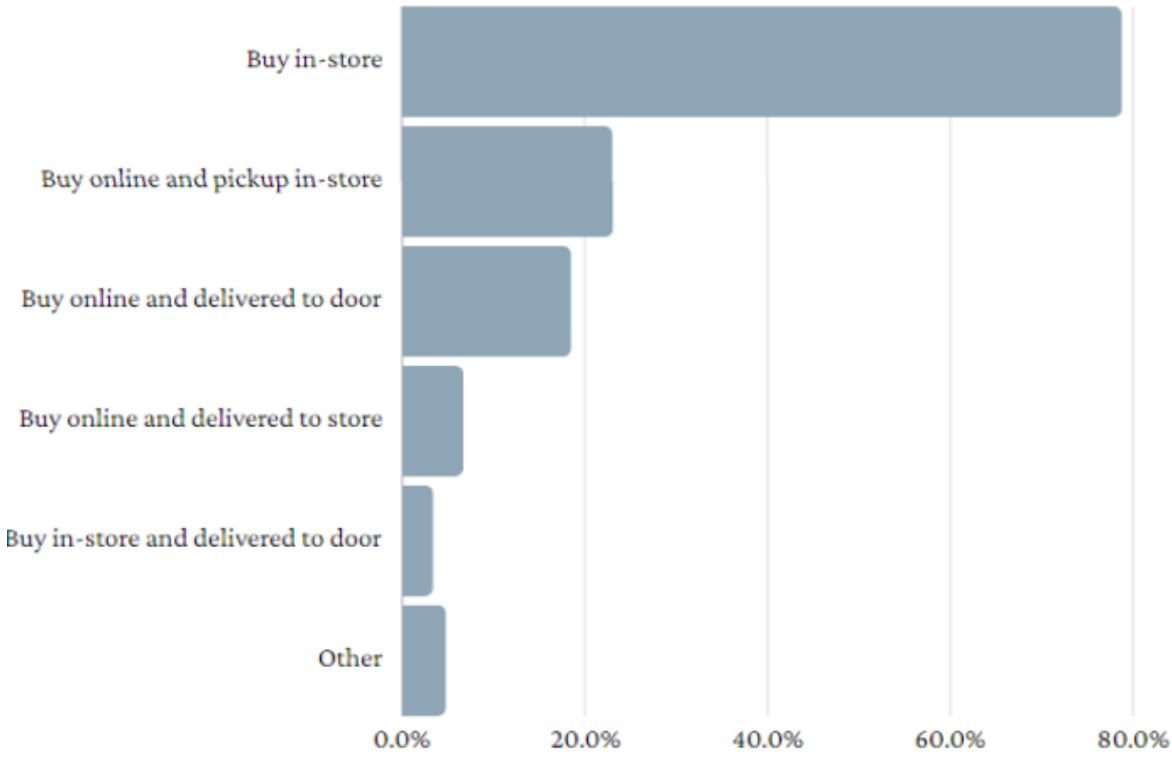


Рисунок 1.1 – Продуктовий ринок в Канаді [1]

В українському контексті впровадження касових систем самообслуговування може мати додаткове значення, зокрема, в умовах

потенційного дефіциту кваліфікованого персоналу та необхідності оптимізації витрат. Крім того, технологічні рішення, що мінімізують фізичний контакт при оплаті, можуть набувати особливої актуальності в умовах пандемій чи інших кризових ситуацій. Таким чином, розуміння глобальних тенденцій та їхньої адаптації до специфічних умов України є ключовим для успішного розвитку роздрібної торгівлі в країні.

Дослідження також засвідчили зростання зацікавленості користувачів у системах самообслуговування. Зокрема, було відмічено потенційне збільшення інтересу канадців до подібних інновацій, особливо після пандемії COVID-19. Згідно з результатами дослідження (рис. 1.2), понад 57% клієнтів заявили про намір використовувати мобільні касові системи, такі як Self-Checkout або Amazon Go, протягом наступних шести місяців [2]. У цьому контексті концепція безкасових магазинів, яка мінімізує необхідність безпосередньої взаємодії з персоналом, одночасно задовольняючи потребу клієнтів у фізичних торгових точках, набуває особливої актуальності [3].

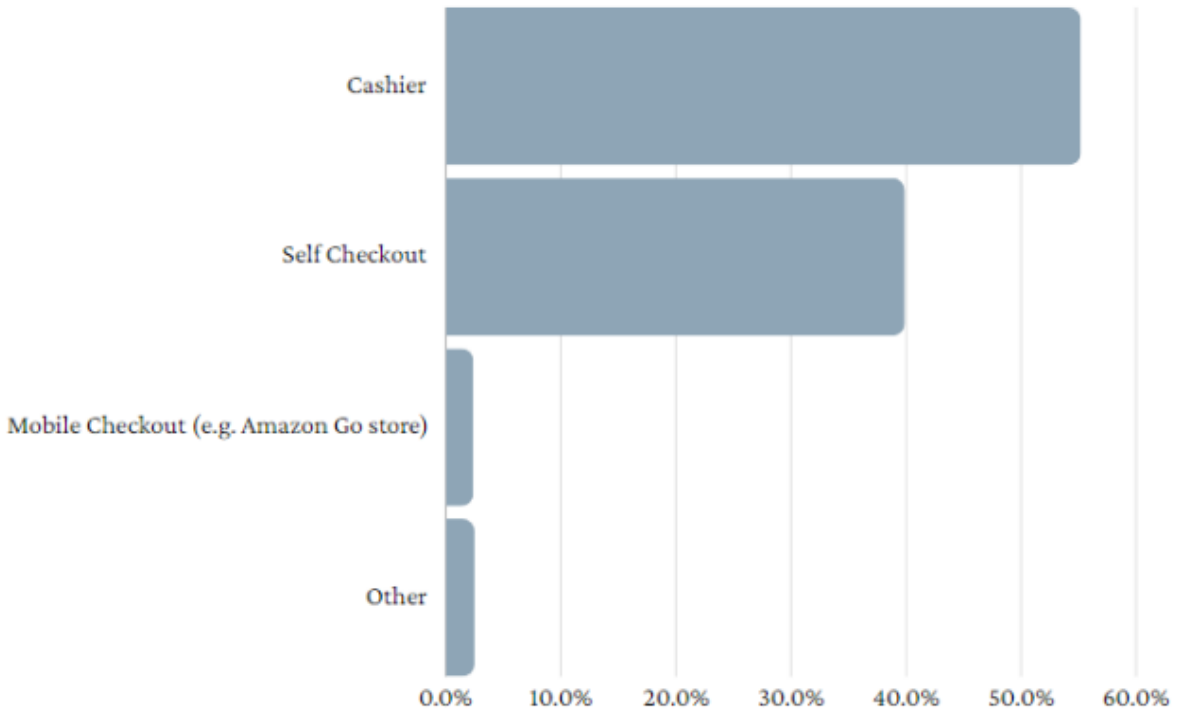


Рисунок 1.2 – Результати соціологічного опитування

Окрім розвитку багатоканальної торгівлі та інтелектуальних платіжних

систем, сучасні клієнти також очікують, що процеси оплати будуть максимально простими, швидкими та безпечними. Багато дослідницьких компаній, зокрема JP. Morgan, проводять відповідні дослідження та готують звіти для аналізу цієї тенденції. Отримані дані свідчать про значну зміну в перевагах клієнтів щодо методів оплати, і ця тенденція, як очікується, збережеться в найближчі роки. Одним із таких авторитетних досліджень є "The Global Payments Report", який аналізує глобальні методи оплати в торгових точках та вартість відповідних транзакцій.

Згідно з даними на рис. 1.3, оплата за допомогою цифрових гаманців у 2023 році вже займала лідируючу позицію за вартістю транзакцій і, за прогнозами, збереже своє домінування до 2027 року, навіть збільшивши відрив від інших способів оплати. Очікується, що цифрові гаманці стануть найбільш динамічно зростаючим методом оплати, збільшивши свою частку ринку з 30% у 2023 році до 46% до 2027 року. Частка кредитних і дебетових карток у 2023 році становила 50%, проте прогнозується її зниження до 40% до 2027 року. Аналогічна тенденція до зменшення очікується і для готівкових платежів, частка яких, за прогнозами, скоротиться з 16% у 2023 році до приблизно 11% до 2027 року [4].

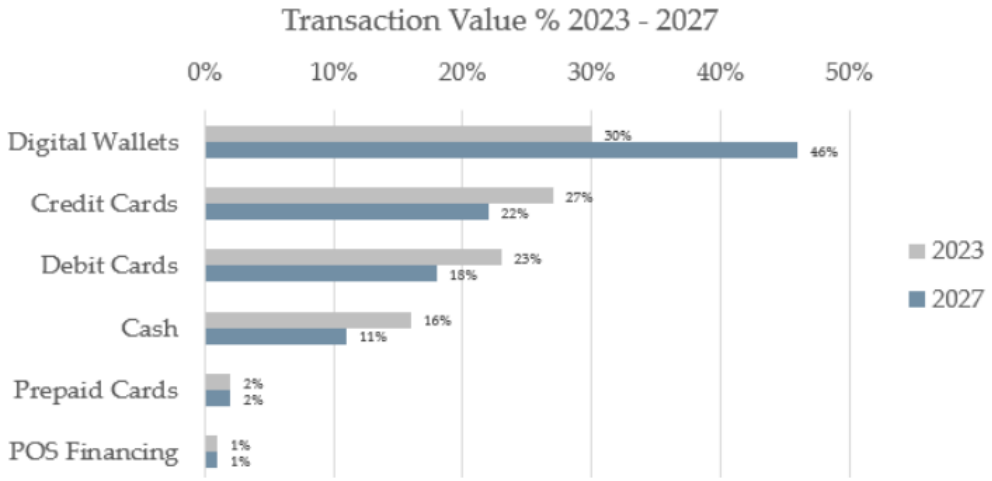


Рисунок 1.3 – Обсяги транзакцій за допомогою POS-платежів, 2023-2027 роки

Як підкреслюється у звіті, цифрові гаманці – технологія, що дозволяє

клієнтам здійснювати фінансові операції в електронному вигляді, – відіграватимуть ключову роль як сьогодні, так і в майбутньому [5]. Ця технологія включає три основні компоненти: пристрій, що підтримує функціональність, програмне забезпечення для здійснення транзакцій та відповідну електронну інфраструктуру [6]. Після активації цифрові гаманці дозволяють зберігати платіжні інструменти, такі як кредитні та дебетові картки, безпосередньо на пристрої та здійснювати платежі без використання фізичних карток.

Історія розвитку цифрових гаманців розпочалася з торгового автомата Coca-Cola, який у 1997 році дозволив оплату через SMS-повідомлення, і продовжилася зі створенням таких платіжних систем, як PayPal та AliPay, а також з появою мобільних телефонів з підтримкою технології NFC. Подальший розвиток цієї технології був стимульований інвестиціями інших великих компаній, таких як Google (Google Wallet), Tencent (WeChat Pay), Apple (Apple Pay) та Samsung (Samsung Pay), що зробило цифрові гаманці одним із найпопулярніших способів оплати на сьогоднішній день. Переваги цієї технології, які виходять далеко за межі простого способу оплати, пояснюють її стрімке поширення.

Фінтех-компанія Wise визначає цифрові гаманці як "гаманець на мобільному телефоні" та підкреслює їхню здатність зберігати не лише дані кредитних і дебетових карток, але й цифрові квитки, картки лояльності, а також інформацію про фізичні та цифрові картки [7]. Для багатьох користувачів ці технології фактично замінюють традиційні банки в контексті надання фінансових послуг та спрощують грошові перекази. Використання цифрових гаманців у маркетингових стратегіях відкриває нові можливості для компаній у сфері підвищення лояльності клієнтів. Прикладами є програми лояльності, інтегровані в платіжні додатки, що дозволяють клієнтам використовувати свої цифрові гаманці для отримання персоналізованих купонів та пропозицій. Ще однією сферою застосування є зберігання особистих документів, таких як посвідчення особи, паспорти та

водійські права. Крім того, цифрові гаманці забезпечують можливість здійснення операцій з криптовалютами, популярність яких стрімко зростає [8].

### 1.3 Концепція "розумних" систем та інтелектуальні касові рішення в роздрібній торгівлі

Термін SMART, що є абревіатурою від "самоконтроль, аналіз і технологія звітування" (Self-Monitoring, Analysis, and Reporting Technology), описує системи, здатні здійснювати самостійний контроль та отримувати значущі результати шляхом аналізу зібраної інформації [4]. У сфері роздрібної торгівлі широко застосовується різноманіття інтелектуальних систем, і однією з них є інтелектуальна каса.

Інтелектуальні касові системи можуть бути реалізовані в різних формах, включаючи смарт-кошики, системи самообслуговування (self-checkout), мобільні каси та системи автоматичного виїзду (just walk out). Магазини, що впроваджують ці технології, замінюють традиційний касовий процес новими рішеннями та поступово відмовляються від традиційної моделі оплати з безпосередньою взаємодією між касиром та покупцем. Прикладами успішного застосування такого підходу є магазини Amazon Go та Habitat від HonestBee.

Традиційний процес здійснення покупок у звичайному магазині розпочинається з моменту, коли клієнт заходить до торговельного залу. Покупці переміщуються між рядами, обирають потрібні товари або повертають ті, які їм більше не потрібні. Після цього вони прямують до каси, де змушені стояти в черзі. Коли настає їхня черга, вони очікують, поки касир відскануює штрих-коди товарів один за одним або, за необхідності, вручну введе відповідні номери. Завершивши оплату, клієнти покидають магазин. Ця система є досить часомісткою, оскільки покупцям часто доводиться проводити значний час у довгих чергах, особливо в періоди пікового навантаження. Крім того, оскільки така система не дозволяє одночасно

відстежувати кожного клієнта, зростає ризик крадіжок.

На противагу традиційним покупкам, інтелектуальні касові системи (ІКС) усувають проблему довгих черг та зайвого часу очікування. Існують різні способи реалізації таких систем. У дослідженні Шогеля та Ліенхарда процес купівлі в системі інтелектуальної каси поділяється на чотири основні етапи: перед покупкою, реєстрація, вибір товару та оплата (checkout), як показано на рисунку 1.4. Для початку процесу купівлі деякі системи можуть вимагати від клієнта встановлення мобільного застосунку магазину. Наприклад, для використання Amazon Go та Habitat від HonestBee попереднє встановлення програми є обов'язковим, тоді як для доступу до Apple Store це не потрібно.

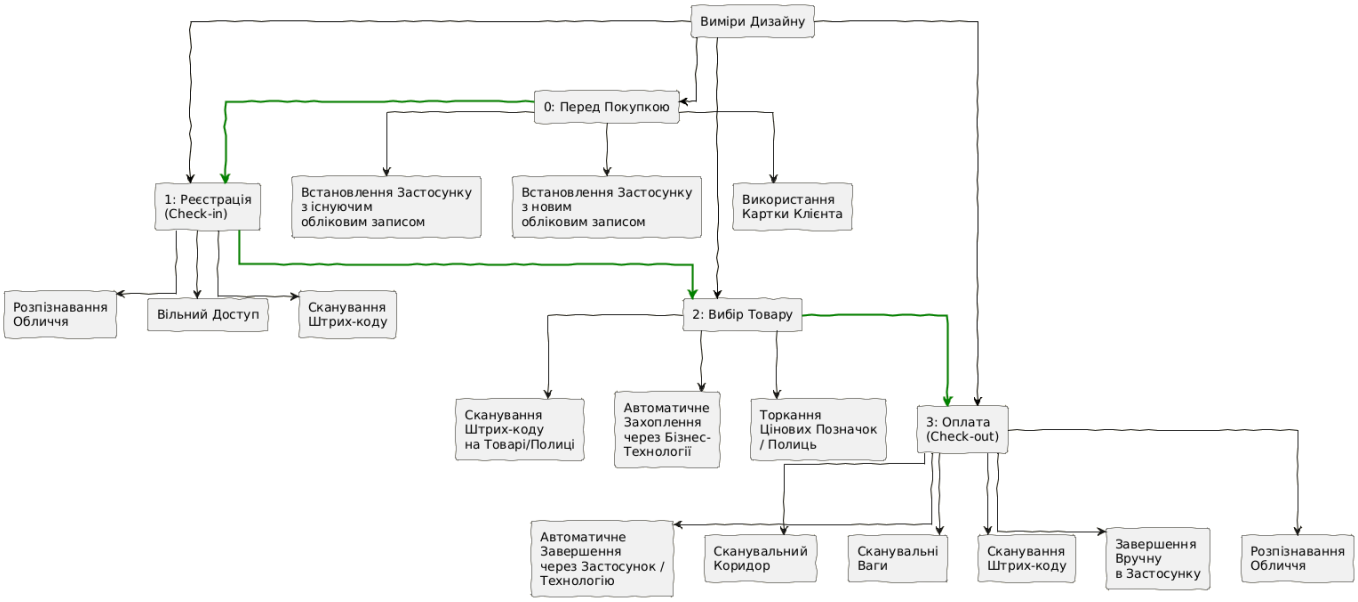


Рисунок 1.4 – Етапи процесу купівлі в системі інтелектуальної каси

Аналогічно, існують різні методи ідентифікації клієнта при вході до магазину. Деякі торговельні заклади використовують біометричну ідентифікацію для розпізнавання покупців, тоді як інші вимагають від клієнтів сканувати штрих-код. Також є магазини, які надають клієнтам вільний вхід. Наступним етапом є ідентифікація товарів, які клієнт бере з полиць. У деяких магазинах цей процес автоматизовано завдяки

використанню датчиків, комп'ютерного зору та технологій машинного навчання. У системах, де автоматичне виявлення товарів не передбачено, клієнтам необхідно самостійно сканувати штрих-код або торкатися карткою (схожою на картку постійного клієнта, отриману раніше) до екрана цін біля відповідного товару. Завершальний етап – оплата та вихід з магазину – пропонує найбільше різноманітних варіантів. Після завершення покупок клієнти можуть скористатися такими способами оплати, як сканування штрих-коду, завершення покупки через мобільний застосунок або здійснення оплати за допомогою біометричної аутентифікації [9].

Одним із прикладів системи, спрямованої на забезпечення зручного процесу покупок та оплати для користувачів, є RFGo. У цій системі до моменту розрахунку клієнт проходить через традиційний досвід купівлі. Він заходить до магазину, обирає необхідні товари та доставляє їх до платіжної зони у кошику, візку, руках або власній сумці. Підійшовши до каси, покупець очікує завершення транзакції попереднього клієнта, якщо така є, а потім переходить до зони оплати. Пасивні RFID-мітки, розміщені на всіх товарах, які приносить клієнт, автоматично виявляються на касі, що усуває необхідність викладати товари з сумки чи кошика на цьому етапі. Ідентифікація товарів відбувається приблизно за 2 секунди, після чого загальна сума покупки та QR-код для оплати відображаються на екрані. Після того, як клієнт завершує оплату та покидає касову зону, інфрачервоні датчики фіксують це, дозволяючи новому клієнту розпочати свою транзакцію [6].

Система, описана в дослідженні "Система кас самообслуговування для продуктів харчування", являє собою апарат, за допомогою якого клієнти можуть самостійно сканувати свої товари та здійснювати оплату. Покупці, завершивши свій вибір, підходять до пристрою та сканують придбані продукти. Для здійснення оплати в цьому дослідженні пропонується новий метод, спрямований на усунення недоліків традиційних способів оплати, що використовуються в системах самообслуговування. Запропонований підхід

передбачає використання попередньо поповнюваних платіжних карток, аналогічних карткам лояльності, для кожного магазину, що виключає необхідність підключення системи до банківської чи платіжної інфраструктури. Для функціонування цієї системи достатньо доступу до локальної бази даних. Клієнти здійснюють оплату, скануючи свою RFID-картку на POS-терміналі, а з метою безпеки розміщують усі відскановані товари у спеціально обладнаній зоні з датчиками ваги. Зафіксована вага відсканованих клієнтом товарів порівнюється з вагою, виміряною датчиками. У разі відсутності розбіжностей у вазі клієнту дозволяється покинути зону оплати [25].

Інше дослідження в цій галузі, на відміну від попередніх, також розглядає можливість обробки неупакованих товарів та продуктів без штрих-кодів, таких як фрукти та овочі. У цьому дослідженні підкреслюється необхідність здатності таких систем ідентифікувати подібні товари для безперебійної роботи, використовуючи алгоритм псевдомаркування. За допомогою цього алгоритму система навчається розпізнавати товари на зображеннях зверху, використовуючи різні набори даних, включаючи як природні зображення, так і зображення, створені за допомогою об'єктного розширення, які використовувалися як дані для класифікації фруктів. Завдяки цим наборам даних було досягнуто точності не менше 95% [7].

## 1.4Огляд існуючих аналогів систем

У липні 2020 року компанія Amazon анонсувала впровадження системи Dash Cart у своїх продуктових магазинах Amazon. Для використання переваг цієї технології клієнтам необхідно завантажити фірмовий застосунок Amazon на свої смартфони. Після прибуття до магазину вони можуть взяти один із "розумних" візків Dash Cart та відсканувати QR-код, згенерований у застосунку, за допомогою вбудованого у візок зчитувача. Ця дія авторизує клієнта в системі та дозволяє розпочати процес покупок.

Розроблена на основі алгоритмів розпізнавання зображень та

комплексу датчиків, система Dash Cart здатна автоматично ідентифікувати товари, які клієнти додають до свого кошика. Вміст кошика, ціни на окремі товари та загальна сума покупки відображаються в режимі реального часу на екрані, розташованому на візку. Після завершення покупок клієнти можуть просто покинути магазин, минаючи традиційні касові стійки. Важливо зазначити, що клієнти, які використовують систему Amazon Dash Cart, повинні виходити через спеціально призначені для цього виходи. Після виходу з магазину вони отримують рахунок-фактуру на свою електронну пошту, а платіж автоматично списується з кредитної картки, прив'язаної до їхнього облікового запису Amazon.

Система Dash Cart розроблена насамперед для невеликих та середніх за обсягом покупок. Крім основної функціональності, вона має й додаткові можливості. Завдяки інтеграції з голосовим помічником Alexa, клієнти можуть переглядати свій список покупок Alexa безпосередньо на екрані візка. Крім того, клієнти, які бажають скористатися купонами на знижку, можуть легко застосувати їх за допомогою спеціального зчитувача, вбудованого у візок [30].

У Японії компанія 7-Eleven представила клієнтам пілотну систему самообслуговування. За цією системою клієнти здійснюють традиційний процес покупок до моменту підходу до касової зони. Вони заходять до магазину, обирають необхідні товари та прямують до каси. Однак у цих магазинах касові апарати мають форму кіосків. Клієнти самостійно сканують обрані товари за допомогою вбудованого в кіоск сканера штрих-кодів.

Інструкції, перелік відсканованих товарів та інша необхідна інформація відображаються клієнтам на голографічному екрані, який є видимим лише під певними кутами. Клієнти взаємодіють із цим екраном та виконують необхідні дії, не торкаючись фізичного дисплея, а через тривимірне зображення, що проектується в повітрі.

Після сканування всіх своїх товарів клієнти повинні обрати спосіб оплати, наприклад, кредитну картку, картку електронних грошей або QR-код,

використовуючи голографічний екран. Після успішного завершення оплати клієнти можуть покинути магазин [8].

Ще один приклад каси самообслуговування у форматі кіоску можна знайти на ринку Yahoo Japan у Токіо. Ця система надає клієнтам можливість здійснювати покупки, не використовуючи гаманець чи смартфон, забезпечуючи таким чином швидкий та зручний досвід. На відміну від системи, представленої в 7-Eleven, для використання цієї технології покупцям необхідно попередньо зареєструвати своє обличчя у фірмовому застосунку магазину та прив'язати свій обліковий запис PayPay як спосіб оплати.

Клієнти, які пройшли процедуру реєстрації, можуть заходити до магазину та обирати товари так само, як у звичайному магазині. Після завершення покупок їм потрібно самостійно відсканувати всі обрані товари на спеціальному кіоску. Для здійснення оплати їм достатньо лише відсканувати своє обличчя на камеру, вбудовану в кіоск. Сума покупки буде автоматично списана з їхнього попередньо зареєстрованого рахунку PayPay [7].

Одним із великих представників роздрібної торгівлі, який впроваджує технологію Just Walk Out, є мережа Aldi. Ця система, яку Aldi розпочала тестувати в Нідерландах, функціонує за допомогою мобільного застосунку. Клієнти, які бажають увійти до магазину, повинні завантажити застосунок Shop&Go та відсканувати QR-код, згенерований у ньому, на зчитувачі турнікета біля входу.

Опинившись у магазині, клієнти можуть здійснювати покупки у звичайному режимі. Вони можуть брати товари з полиць та повертати їх, якщо передумають. Відмінність від традиційних магазинів полягає у відсутності необхідності оплати на касі. Придбані клієнтом продукти автоматично розпізнаються за допомогою камер, датчиків та штучного інтелекту. Коли клієнт сканує QR-код зі свого застосунку на вихідному турнікеті, платіж автоматично списується з його зареєстрованої кредитної

картки [3].

### 1.5 Переваги та недоліки впровадження інтелектуальних касових систем у роздрібній торгівлі

Здійснення покупок у магазинах, особливо в супермаркетах та продуктових крамницях, є щоденною рутиною для багатьох клієнтів. Однак, у певні години та в особливих випадках, магазини стають надто завантаженими, що призводить до утворення довгих черг та збільшення часу очікування для покупців. Останні дослідження показують, що непродуктивно витрачений час на очікування є вагомою причиною, через яку клієнти уникають відвідування фізичних магазинів [2]. Саме тому швидший процес оформлення покупок, який забезпечують інтелектуальні касові системи, з мінімізацією або повною відсутністю часу очікування, є суттєвою перевагою цих систем.

Ще однією важливою перевагою є можливість для клієнтів бачити вміст свого віртуального кошика та підсумкову суму в режимі реального часу на екрані або іншому пристрої. Це допомагає їм контролювати свій бюджет під час покупок та уникати придбання зайвих товарів [3]. Для власників магазинів однією з ключових переваг є усунення необхідності у великій кількості касирів, що призводить до зменшення витрат на оплату праці [4]. Зниження цих витрат, які є значною частиною операційних витрат, може дозволити магазинам пропонувати клієнтам більш конкурентоспроможні ціни, підвищуючи таким чином їхню загальну конкурентоздатність. Крім того, мінімізація взаємодії з касиром під час процесу покупки сприяє автоматизації процесів виставлення рахунків та зменшує ймовірність людських помилок при розрахунках.

Додатковою перевагою є відмова від традиційних касових зон. Вивільнення простору, який раніше займали каси, або їх заміна компактними кіосками, дозволяє перетворити ці площі на більш прибуткові торгові зони. Ефективне використання торгового простору може призвести до збільшення

товарообігу на квадратний метр магазину.

Одним із найбільш значущих недоліків впровадження інтелектуальних касових систем є потенційне зростання безробіття, зумовлене зменшенням потреби в робочій силі. Автоматизація процесів призводить до того, що завдання, які раніше виконувалися людьми, тепер покладаються на машини. Наприклад, заміна традиційних касових зон на технологію Just Walk Out неминуче призводить до скорочення кількості необхідних касирів. Проте, з іншого боку, впровадження цих систем також сприяє виникненню нових сфер діяльності та зростанню потреби в кваліфікованих фахівцях для розробки, обслуговування та встановлення відповідного обладнання [3]. Крім того, людський персонал залишається необхідним для виконання таких важливих функцій, як надання допомоги клієнтам, розміщення товарів на полицях та управління запасами, подібно до традиційних систем.

Ще однією проблемою, пов'язаною зі зменшенням рівня людської взаємодії в процесі покупок, є скорочення соціальних контактів. Згідно з результатами досліджень, значна частина споживачів (наприклад, 77% американських клієнтів) все ще віддає перевагу спілкуванню з живою людиною, а не з автоматизованою системою.

Іншим суттєвим недоліком впровадження інтелектуальних касових систем є їхня висока вартість. Закупівля та встановлення необхідних технологій та обладнання, а також подальше технічне обслуговування та ремонт цих систем, потребують значних фінансових інвестицій.

Крім того, для ефективної роботи інтелектуальних касових систем може виникнути необхідність адаптації товарів. Ця обставина може становити певні труднощі, особливо для виробників. Багато таких систем покладаються на технологію радіочастотної ідентифікації (RFID). Додавання RFID-міток до всіх товарів може бути не лише дорогим, але й знижувати ефективність процесу через його трудомісткість [6].

### 1.5 Мета і задачі роботи

Метою кваліфікаційної роботи є розробка програмного модуля для касової системи самообслуговування в супермаркеті, що забезпечує базову функціональність обробки транзакцій та підготовку до реалізації протоколу обміну даними в реальному часі.

Для досягнення поставленої мети необхідно вирішити задачі. Провести аналіз існуючих підходів до розробки програмних модулів для касових систем самообслуговування та принципів їхньої взаємодії з центральними системами супермаркетів. Визначити основні функціональні вимоги до програмного модуля касової системи самообслуговування, включаючи додавання, редагування, видалення товарів з кошика, перегляд замовлення та розрахунок загальної вартості. Розробити структуру програмного модуля, використовуючи об'єктно-орієнтований підхід, для забезпечення гнучкості та можливості подальшого розширення функціональності. Реалізувати базову функціональність програмного модуля для касової системи самообслуговування, включаючи: додавання товарів до кошика з зазначенням назви, кількості та ціни; можливість оновлення назви, кількості та ціни обраного товару в кошику; видалення окремих товарів з кошика; повне очищення кошика; перегляд вмісту кошика з відображенням назви, кількості, ціни та загальної вартості кожного товару; розрахунок загальної вартості покупки з урахуванням можливих знижок. Підготувати програмний модуль до інтеграції з протоколом обміну даними в реальному часі шляхом визначення необхідних інтерфейсів та структур даних для передачі інформації. Описати потенційні підходи до реалізації протоколу обміну даними в реальному часі для оновлення інформації про товари, ціни та обробки платежів.

# 2 СПЕЦІАЛЬНИЙ РОЗДІЛ

## 2.1 Технічні вимоги до програмного модуля касової системи самообслуговування в супермаркеті

### 2.1.1 Найменування і призначення програмного модуля

Програмний модуль для касової системи самообслуговування в супермаркеті з реалізацією протоколу обміну даними в реальному часі призначений для забезпечення функціональності самостійного оформлення покупок клієнтами супермаркету та обміну даними про товари, ціни та транзакції з центральною системою в режимі реального часу.

### 2.1.2 Вимоги до структури і функціонування об’єкту професійної діяльності

Об'єктом професійної діяльності в даній роботі є програмний модуль касової системи самообслуговування з графічним інтерфейсом.

Програмний модуль повинен бути чітко розділений на окремі функціональні блоки, кожен з яких відповідає за певну частину процесу самообслуговування. Основними структурними елементами є [3]:

– відображення кошика покупок – елемент інтерфейсу для наочного представлення обраних товарів (назва, кількість, ціна);

– додавання товару – елементи інтерфейсу (поля введення, кнопка) для введення інформації про товар та його додавання до кошика;

– оновлення товару – елементи інтерфейсу (випадаючий список, поля введення, кнопки) для вибору товару в кошику та зміни його характеристик (назва, кількість, ціна);

– видалення та скидання транзакції – елементи інтерфейсу (випадаючий список, кнопки) для видалення окремих товарів або повного очищення кошика;

– відображення та розрахунок загальної вартості – елементи

інтерфейсу (мітка, кнопка) для відображення підсумкової суми покупки та ініціації її розрахунку;

– кнопка закриття програми – елемент інтерфейсу для завершення роботи з програмою;

– внутрішня логіка обробки транзакцій – набір програмних компонентів (класів, функцій) для зберігання даних про кошик, виконання операцій додавання, редагування, видалення, розрахунку вартості та підготовки даних для можливого обміну з іншими системами.

Кожен структурний елемент повинен виконувати свою функцію чітко та передбачувано, забезпечуючи наступну логіку роботи:

– користувач має можливість інтуїтивно зрозуміло взаємодіяти з графічним інтерфейсом для додавання товарів до кошика, вказуючи їхню назву, кількість та ціну;

– користувач повинен мати можливість легко переглядати вміст свого кошика з детальною інформацією про кожен товар;

– система повинна надавати користувачеві засоби для редагування кількості та ціни вже доданих до кошика товарів, а також можливість змінити назву товару;

– користувач повинен мати можливість видаляти окремі товари з кошика або повністю очистити його за потреби;

– система повинна забезпечувати точний розрахунок загальної вартості покупки на основі цін та кількості товарів, а також враховувати можливі знижки;

– інтерфейс повинен чітко відображати загальну суму до оплати;

– програмний модуль повинен бути підготовлений до майбутньої інтеграції з іншими системами (наприклад, обліку товарів, платіжними системами) шляхом наявності структурованих даних про транзакцію.

Таким чином, об'єкт професійної діяльності повинен мати модульну структуру, що забезпечує чітке розділення відповідальності між елементами інтерфейсу та внутрішньою логікою, а його функціонування має бути

спрямоване на забезпечення зручного, зрозумілого та ефективного процесу самостійного оформлення покупок для користувача.

### 2.1.3 Вимоги до показників призначення

Програмний модуль КСС та його функціональні складові повинні забезпечувати наступні показники призначення [9]:

– відображення кошика покупок: відображення назви товару у текстовому форматі; відображення кількості товару у числовому форматі (ціле число); відображення ціни товару у числовому форматі (десяткове число з двома знаками після коми); відображення загальної вартості позиції (кількість * ціна) у числовому форматі (десяткове число з двома знаками після коми);

– додавання товару: можливість введення назви товару довжиною до 255 символів; можливість введення кількості товару в діапазоні від 1 до 999; можливість введення ціни товару в діапазоні від 0.01 до 9999.99; відображення повідомлення про успішне додавання товару; валідація введених даних (кількість та ціна повинні бути числовими значеннями);

– оновлення товару: випадаючий список повинен містити назви всіх товарів, присутніх у кошику; можливість введення нової назви товару довжиною до 255 символів; можливість введення нової кількості товару в діапазоні від 1 до 999; можливість введення нової ціни товару в діапазоні від 0.01 до 9999.99; відображення повідомлення про успішне оновлення відповідного параметра товару; валідація введених даних (нова кількість та нова ціна повинні бути числовими значеннями);

– видалення товару: випадаючий список повинен містити назви всіх товарів, присутніх у кошику; відображення повідомлення про успішне видалення товару;

– скидання транзакції: очищення таблиці кошика покупок; скидання відображення загальної ціни до 0.00; відображення повідомлення про успішне очищення кошика;

– відображення загальної вартості: відображення загальної суми до оплати у числовому форматі (десяткове число з двома знаками після коми); автоматичне оновлення загальної вартості при додаванні, редагуванні або видаленні товарів; відображення інформації про застосовані знижки (якщо є);

– швидкість реакції: відображення змін у кошику та загальної вартості повинно відбуватися не пізніше ніж за 0.5 секунди після дій користувача;

– зручність використання: інтерфейс повинен бути інтуїтивно зрозумілим та легким у навігації для користувача; елементи керування повинні бути чітко позначені українською мовою.

Підготовка даних для обміну: внутрішня структура даних (self.trnsct.shopping_cart) повинна забезпечувати зберігання інформації про кожен товар у JSON-форматі, придатному для передачі до центральної системи.

### 2.1.4 Вимоги до програмного забезпечення

Для забезпечення розробки та функціонування програмного модуля КСС висуваються наступні вимоги до програмного забезпечення:

– мова програмування – Python версії 3.x;

– бібліотека для графічного інтерфейсу – Tkinter (вбудована бібліотека Python) з використанням модуля tkinter.ttk для створення сучасних віджетів;

– бібліотека для табличного представлення даних – бібліотека tabulate (версія 0.8.x або новіша) для форматованого виведення даних у консоль під час розробки та тестування (хоча в остаточному варіанті використовується ttk.Treeview);

– середовище розробки – IDE (Integrated Development Environment), що підтримує розробку на Python (наприклад, PyCharm, VS Code, Sublime Text);

– операційна система розробки – кросплатформність (Windows,

Linux, macOS);

– система контролю версій – Git для управління кодом;

– модуль tkinter.messagebox для відображення діалогових вікон (повідомлень, попереджень, підтверджень);

– можливість інтеграції з бібліотеками для роботи з форматами даних обміну (json для JSON, xml.etree.ElementTree або lxml для XML);

– можливість інтеграції з бібліотеками для здійснення мережевих запитів (наприклад, requests для HTTP, websockets для WebSocket).

## 2.2 Розробка апаратної частини

### 2.2.1 Розробка функціональної схеми

Функціональна схема програмного модуля касової системи самообслуговування відображає основні функціональні блоки та потоки даних між ними, забезпечуючи візуалізацію архітектури розробленого програмного забезпечення. Модульна структура дозволяє чітко розділити відповідальність між різними частинами системи, спрощуючи розробку, тестування та подальше розширення функціональності, зокрема інтеграцію з системами обміну даними в реальному часі.

Структура комплексу технічних засобів IoT-системи на рівні пристроїв включає різноманітні фізичні та віртуальні компоненти, що безпосередньо взаємодіють з офісним середовищем та виконують функції контролю доступу, забезпечення безпеки та автоматизації окремих процесів (рис.2.1).

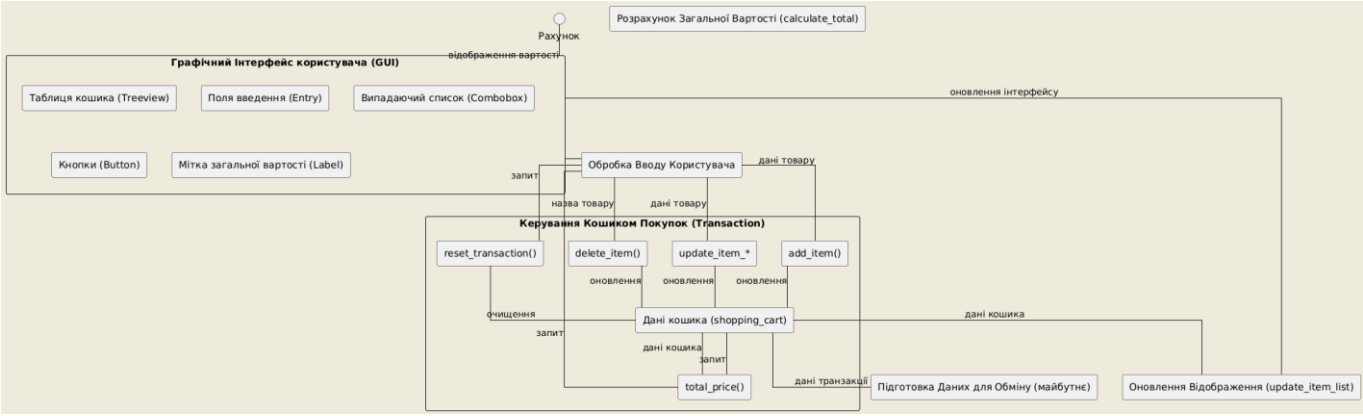


Рисунок 2.1 – Функціональна схема

Основним елементом взаємодії з користувачем є графічний інтерфейс користувача (GUI), реалізований за допомогою бібліотеки tkinter. Цей блок відповідає за відображення інформації для користувача та отримання його дій. Він включає такі елементи інтерфейсу, як таблиця кошика покупок (Treeview), поля введення (Entry) для введення назви, кількості та ціни товару, випадаючий список (Combobox) для вибору товарів при редагуванні та видаленні, кнопки (Button) для ініціації дій (додавання, оновлення, видалення, скидання, розрахунок), а також мітка (Label) для відображення загальної вартості. Дії користувача, такі як введення даних у поля або натискання кнопок, ініціюють події, які обробляються відповідними функціями в програмному модулі.

Центральним елементом обробки даних є блок Керування Кошиком Покупок, представлений класом Transaction. Цей клас містить внутрішню структуру даних (self.shopping_cart) для зберігання інформації про товари, додані до кошика (назва, кількість, ціна), та надає методи для виконання основних операцій з кошиком: add_item (додавання товару), update_item_name, update_item_quantity, update_item_price (оновлення характеристик товару), delete_item (видалення товару) та reset_transaction (очищення кошика). Дані про стан кошика є ключовими для всіх інших операцій, включаючи відображення та розрахунок вартості.

Для забезпечення актуального відображення інформації в GUI

використовується блок Оновлення Відображення, представлений функцією update_item_list класу SelfServiceCashierGUI. Ця функція відповідає за синхронізацію даних між внутрішнім станом кошика (у класі Transaction) та елементами графічного інтерфейсу, зокрема оновлення вмісту таблиці ttk.Treeview та випадаючого списку ttk.Combobox при кожній зміні кошика.

Введення даних користувачем обробляється блоком обробка вводу користувача, який включає функції обробників подій, пов'язані з елементами GUI (наприклад, функції, що викликаються при натисканні кнопок "Додати товар", "Оновити", "Видалити" тощо). Ці функції отримують дані, введені користувачем через відповідні поля, та передають їх до методів класу Transaction для виконання необхідних дій з кошиком.

Розрахунок підсумкової вартості покупки здійснюється блоком «Розрахунок загальної вартості», представленим функцією calculate_total класу SelfServiceCashierGUI, яка викликає метод total_price об'єкта Transaction. Цей блок отримує дані про товари в кошику, обчислює їхню загальну вартість з урахуванням можливих знижок та оновлює відповідну мітку в GUI.

На етапі підготовки до майбутньої інтеграції з іншими системами передбачається наявність блоку «Підготовка Даних для Обміну». Хоча на даний момент цей блок явно не реалізований у графічному інтерфейсі, структура класу Transaction вже містить необхідні дані про транзакцію (self.shopping_cart). У майбутньому буде розроблено функціональність для форматування цих даних (наприклад, у форматі JSON) та їхньої передачі до центральної системи обліку або платіжних шлюзів.

### 2.2.2 Розробка принципової схеми

Центральним елементом схеми є мікроконтролер (позначений як U2, ATmega328P), який виконує роль "мозку" системи, керуючи обміном даними між різними компонентами. В аналогії з нашим програмним модулем, клас Transaction та основні функції обробки транзакцій виконують подібну

керуючу роль, координуючи роботу різних частин програмного забезпечення (рис.2.2).

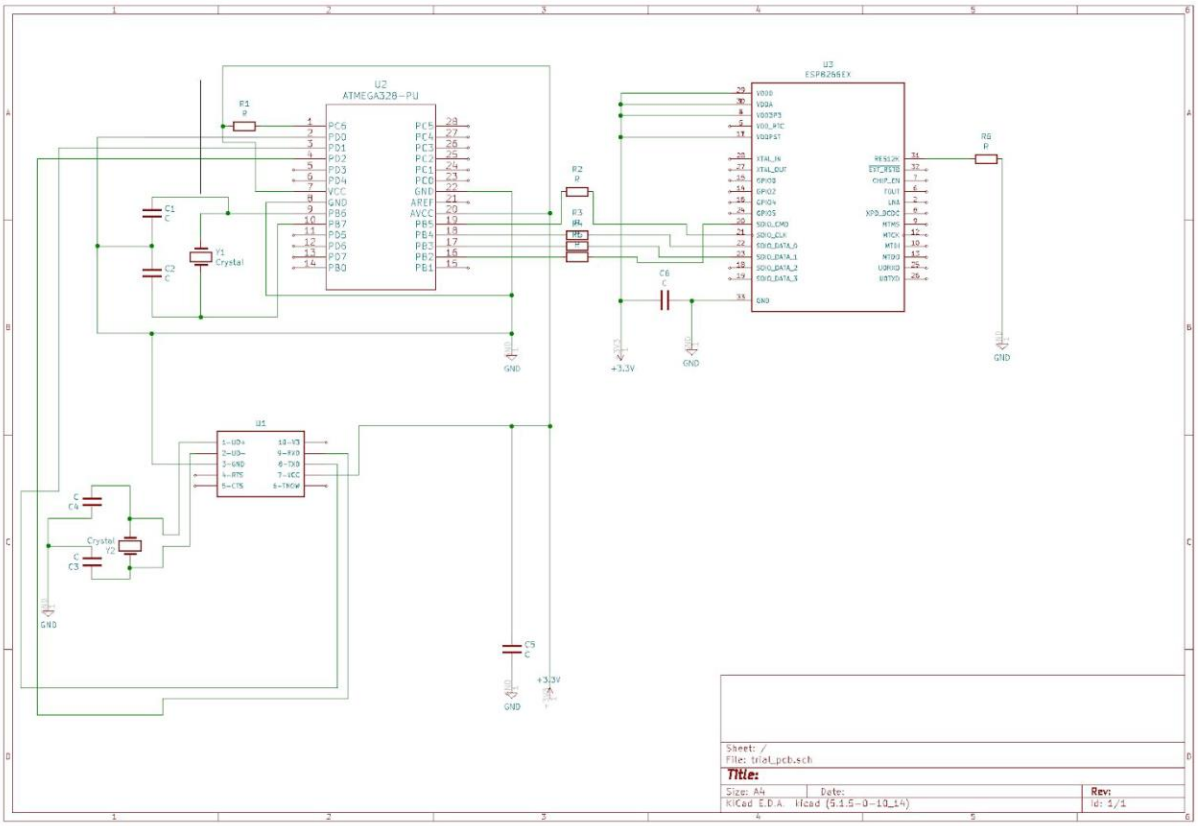


Рисунок 2.2 – Принципова схема

До мікроконтролера підключено підсилювач тензодатчиків (U4, INA125), який отримує сигнали від датчиків навантаження (не показані явно, але згадуються в описі як підключені за конфігурацією моста Уітстона). В контексті нашого програмного модуля, цей підсилювач можна зіставити з блоком обробки даних про товари (наприклад, отримання назви, кількості, ціни), хоча ці дані вводяться користувачем через графічний інтерфейс, а не з фізичних датчиків. Кольорове кодування проводів тензодатчиків (ЧЕРВОНИЙ - живлення+, ЧОРНИЙ - живлення-, БІЛИЙ - вихід+, ЗЕЛЕНИЙ - вихід-, ЖОВТИЙ - захист) показує їхнє призначення для правильного підключення до підсилювача. В контексті нашого програмного модуля, цей підсилювач можна зіставити з блоком обробки даних про товари (наприклад, отримання назви, кількості, ціни), хоча ці дані вводяться

користувачем через графічний інтерфейс, а не з фізичних датчиків.

РК-екран (LCD Connector Module 08_pin) підключений до мікроконтролера через паралельний 4-бітний інтерфейс і використовується для відображення інформації про продукт, що зважується. В нашому програмному модулі цю функцію виконує Графічний Інтерфейс користувача (GUI), який відображає вміст кошика покупок (назву, кількість, ціну) та загальну вартість для клієнта. Інформація про товари, введена користувачем або отримана з внутрішніх структур даних, відображається на екрані GUI.

Чіп WIFI (U5, ESP8266-ESP-01S), підключений до мікроконтролера за протоколом SPI, що використовується для синхронізації з онлайн-сервером. В контексті нашого програмного модуля, цей чіп можна зіставити з блоком Підготовки Даних для Обміну, який відповідатиме за форматування даних про транзакції та їхню передачу до центральної системи супермаркету для оновлення запасів, обробки платежів тощо.

### 2.2.3 Вибір елементної бази системи

У таблиці 2.1 наведено перелік основних реальних апаратних компонентів, які зазвичай використовуються в касових системах самообслуговування. Для кожного компонента вказано його призначення, наведено приклади конкретних виробників та моделей (які є загальновідомими та використовуються в індустрії), а також надано короткий опис його функціональності в контексті системи самообслуговування.

Таблиця 2.1 – Апаратні компоненти

| Компонент | Призначення | Приклади реалізації | Опис |
|---|---|---|---|
| Процесорний блок | Виконання програмного коду, керування периферією, обмін даними. | Raspberry Pi 4B/5, Intel NUC, Gigabyte BRIX, промислові міні-ПК (Advantech, Axiomtek) | "Серце" системи, забезпечує обчислювальну потужність для роботи програмного модуля КСС, керує підключеними пристроями та відповідає за комунікацію з центральною системою. |

| | | | |
|---|---|---|---|
| Пристрій введення | Отримання даних від користувача. | Сенсорні екрани (Elo Touch Solutions, 3M), сканери штрих-кодів (Honeywell, Zebra), зчитувачі карток (Ingenico, Verifone) | Забезпечує інтеракцію користувача з системою. Сенсорний екран є основним засобом управління, сканер штрих-кодів прискорює ідентифікацію товарів, а зчитувач платіжних карток забезпечує проведення безготівкових платежів. |
| Пристрій відображення | Відображення інформації для користувача. | Сенсорні екрани (див. вище), додаткові дисплеї (Samsung, LG) | Відображає інтерфейс програмного модуля, вміст кошика, підсумкову суму, підказки та іншу важливу інформацію для клієнта. Додаткові дисплеї можуть використовуватись для реклами або відображення іншої корисної інформації. |
| Пристрій зв'язку | Забезпечення обміну даними з мережею. | Ethernet-контролери, Wi-Fi модулі (вбудовані або зовнішні), 4G/5G модеми (Sierra Wireless, Telit) | Забезпечує підключення до локальної мережі магазину та інтернету для синхронізації транзакцій, оновлення цін та іншої інформації з центральною системою. Бездротові модулі дають гнучкість у розміщенні пристроїв. |
| Периферійні пристрої | Додаткова функціональність системи. | Ваги DIGI SM-5300, принтери чеків Epson TM-T88VII, камери Hikvision, пристрої для видачі готівки CashCode | Розширюють функціональність КСС. Ваги – для зважування товарів, принтери – для друку чеків, камери – для моніторингу безпеки, пристрої видачі готівки – для реалізації готівкових розрахунків. |
| Корпус та кріплення | Захист компонентів та зручність використання. | Корпуси (VeriFone, KIOSK Information Systems), кронштейни, стійки | Забезпечує фізичний захист внутрішніх компонентів від механічних пошкоджень, вандалізму, впливу середовища. Крім того, ергономіка конструкції забезпечує зручність доступу для користувача. |
| Блок живлення | Забезпечення енергією всіх компонентів. | Промислові блоки живлення (Mean Well, Delta Electronics) | Забезпечує стабільне електроживлення для всіх компонентів. Надійність та відповідність технічним вимогам живлення є критичною умовою безперервної роботи системи. |

# 3 РОЗРОБКА ПРОГРАМНОГО МОДУЛЯ

## 3.1 Архітектура програмного модуля КСС

Архітектура системи самообслуговування (КСС) складається з кількох основних компонентів, кожен з яких виконує свою функцію, щоб забезпечити ефективну і зручну взаємодію користувача з системою (рис.3.1).

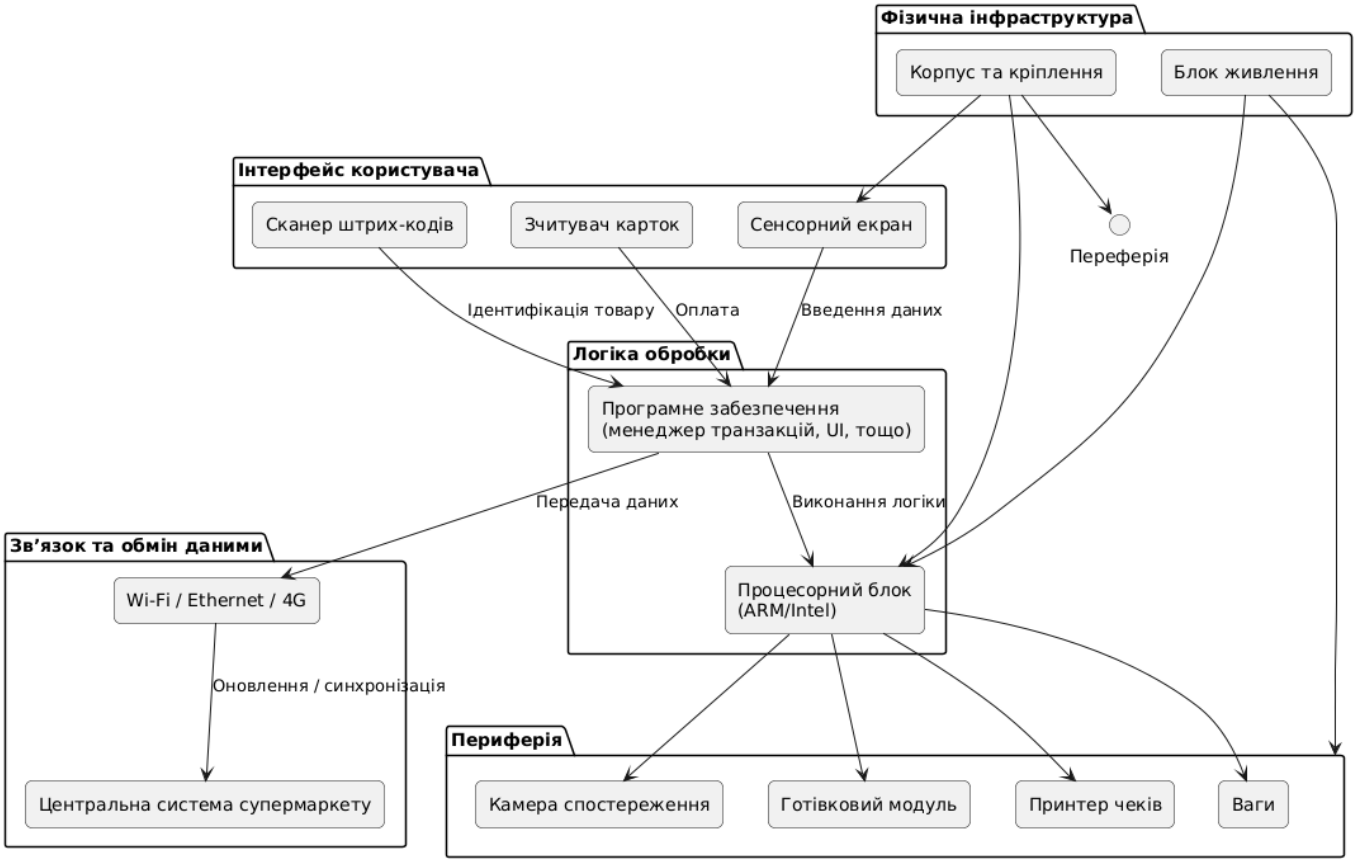


Рисунок 3.1 – Багаторівнева архітектура КСС

Інтерфейс користувача є першим рівнем системи, де відбувається взаємодія з кінцевим користувачем. Він включає сенсорний екран для введення даних користувачем, сканер штрих-кодів для автоматичної ідентифікації товарів та зчитувач карток для безготівкових платежів. Ці компоненти дозволяють швидко і зручно здійснювати покупки та взаємодіяти з системою.

Логіка обробки даних відповідає за виконання обчислювальних завдань і управління транзакціями. Центральним компонентом цього рівня є

процесорний блок, який може бути побудований на основі ARM чи Intel процесорів. Він керує усіма іншими компонентами і виконує програмне забезпечення для управління транзакціями, включаючи менеджер транзакцій, інтерфейс користувача та інші функції системи. Програмне забезпечення забезпечує обробку даних, отриманих від користувача, і управління всіма процесами, пов’язаними з покупкою та оплатою.

Зв'язок та обмін даними відповідає за передачу даних між КСС та центральною системою супермаркету. Для цього використовується мережевий модуль, який підтримує з’єднання через Wi-Fi, Ethernet або 4G. Цей компонент дозволяє синхронізувати дані про товари, ціни, наявність та виконувати оновлення в реальному часі, забезпечуючи актуальність інформації в КСС. Обмін даними між КСС і центральною системою супермаркету є критичним для підтримки правильності обчислень та правильного відображення цін [9].

Периферія забезпечує додаткову функціональність для здійснення транзакцій. Це включає принтер чеків для друку підтвердження оплати, ваги для товарів на вагу, камери для моніторингу та забезпечення безпеки, а також модуль для видачі готівки, якщо система підтримує оплату готівкою. Периферійні пристрої доповнюють основні функції системи, створюючи повноцінну екосистему для обробки транзакцій.

Фізична інфраструктура складається з корпусу та кріплення, які забезпечують фізичний захист компонентів системи від пошкоджень, а також блок живлення, що стабільно постачає енергію всім елементам КСС. Корпус також грає важливу роль у зручності використання системи, адже зручний та ергономічний дизайн забезпечує комфортне використання користувачем.

Усі ці компоненти працюють разом, щоб забезпечити безперебійну роботу системи самообслуговування, де користувач може без проблем здійснити покупку, а система буде ефективно обробляти всі дані, зберігати їх актуальність і підтримувати зв'язок із центральною системою супермаркету для оновлення інформації.

### 3.2 Кошик покупок

В Python реалізовано клас Transaction, призначений для моделювання кошика покупок у системі самообслуговування. Клас надає функціональність для додавання, оновлення, видалення товарів, перегляду замовлення, скидання транзакції та розрахунку загальної вартості з урахуванням системи знижок.

Першим кроком є імпорт необхідних бібліотек для роботи з мережею та асинхронністю. Для встановлення WebSocket-з'єднання, яке є зручним для обміну даними в реальному часі, можна використовувати бібліотеку websockets. Для керування асинхронними операціями знадобиться бібліотека asyncio. Також може бути корисною бібліотека json для серіалізації та десеріалізації даних, що передаються між клієнтом та сервером.

Клас Transaction ініціалізується методом __init__(self), який створює порожній словник self.shopping_cart. Цей словник слугує основним сховищем даних про товари в кошику, де ключами є назви товарів, а значеннями - списки, що містять кількість та ціну за одиницю кожного товару.

Метод add_item(self) дозволяє користувачеві додавати товари до кошика. Він використовує нескінченний цикл while True, що триває до тих пір, поки користувач не вирішить припинити додавання, ввівши 'n' (рис.3.2). У циклі відбувається обробка введення назви товару, кількості та ціни. Блок try...except ValueError забезпечує обробку ситуацій, коли користувач вводить нечислові значення для кількості або ціни, виводячи відповідне повідомлення про помилку. Після успішного введення даних, інформація про товар додається до словника self.shopping_cart, і користувачеві пропонується додати ще один товар.

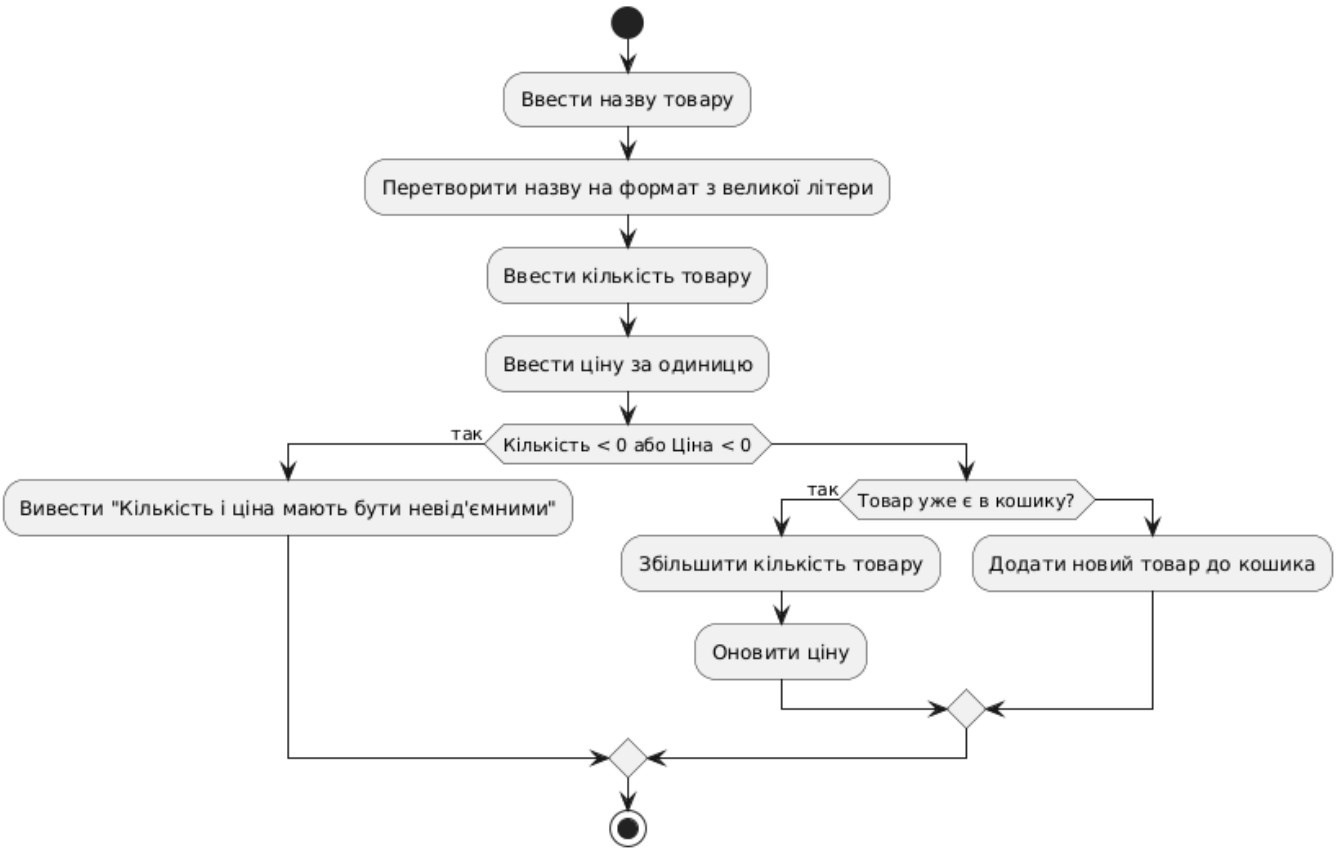


Рисунок 3.2 – Структурна схема алгоритму додавання товару

Методи update_item_name(self), update_item_quantity(self) та update_item_price(self) надають можливість редагувати існуючі записи в кошику. Кожен з цих методів спочатку перевіряє, чи не є кошик порожнім (рис.3.3). Потім запитує назву товару, який потрібно змінити, та перевіряє його наявність у кошику. У разі успішної перевірки, метод запитує нове значення (назву, кількість або ціну відповідно) та оновлює словник self.shopping_cart. Якщо товар не знайдено, користувачеві виводиться повідомлення про помилку.


Рисунок 3.3 – Структурна схема алгоритму оновлення назви товару

Метод delete_item(self) призначений для видалення товару з кошика (рис.3.4). Він також спочатку перевіряє, чи не порожній кошик, потім запитує назву товару для видалення та, якщо товар знайдено, видаляє відповідний запис зі словника self.shopping_cart за допомогою методу pop().

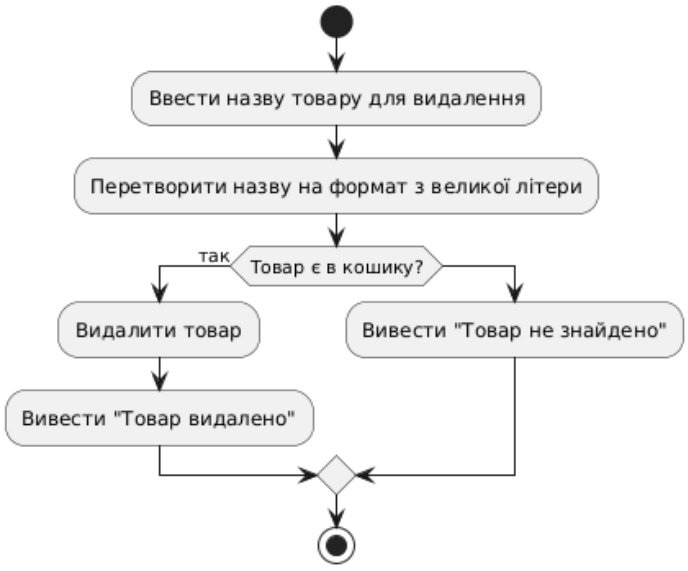


Рисунок 3.4 – Структурна схема алгоритму видалення товару

Метод reset_transaction(self) виконує повне очищення кошика покупок. Він викликає метод clear() для словника self.shopping_cart, видаляючи всі збережені товари.

Метод check_order(self) відповідає за відображення вмісту кошика користувачеві у форматованому вигляді (рис.3.5). Він створює список self.data, який включає заголовок таблиці та дані про кожен товар з кошика (назва, кількість, ціна, загальна вартість). Для форматування виводу використовується бібліотека tabulate. Крім того, метод перевіряє, чи не є кількість або ціна якогось товару від'ємними, і в разі виявлення виводить попередження.

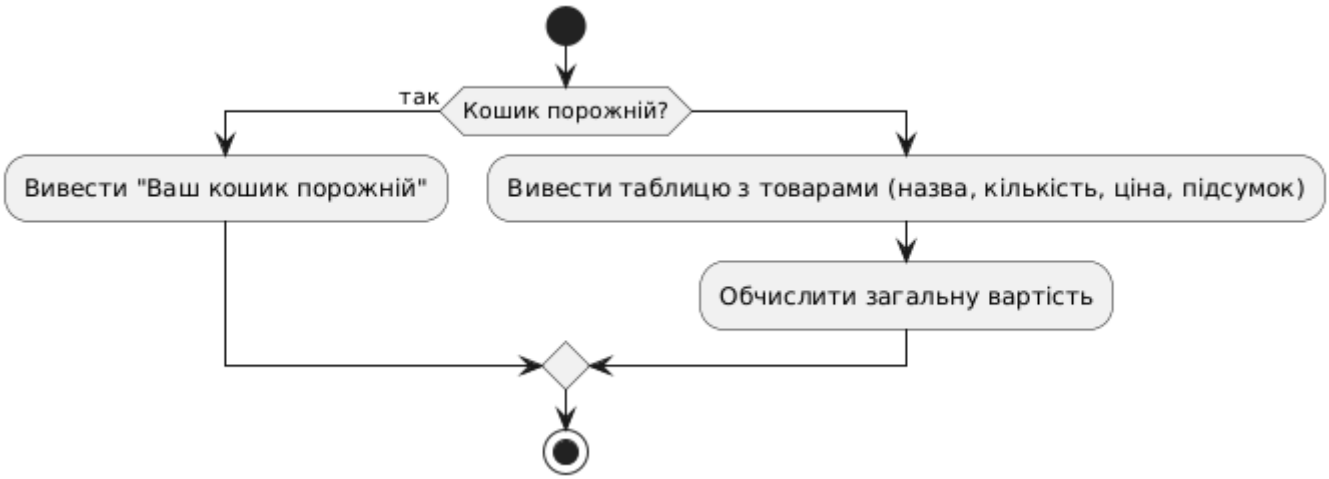


Рисунок 3.5 – Структурна схема алгоритму відображення вмісту товару

Метод total_price(self) здійснює розрахунок загальної вартості всіх товарів у кошику (рис.3.6). Він ітерується по товарах у кошику, обчислюючи загальну вартість кожного з них та підсумовуючи їх. Після отримання загальної суми, метод застосовує систему знижок, яка залежить від цієї суми: 5% знижка при сумі понад 200 000, 8% при сумі понад 300 000 та 10% при сумі понад 500 000. Якщо знижка не застосовується, користувачеві виводиться інформація про доступні рівні знижок. Наприкінці метод виводить остаточну суму до оплати з урахуванням застосованої знижки, форматуючи число для кращого сприйняття.


Рисунок 3.6 – Структурна схема алгоритму розрахунку загальної вартості всіх товарів

У своїй сукупності, клас Transaction надає базову функціональність для управління процесом купівлі в системі самообслуговування через текстовий інтерфейс. Він дозволяє додавати, редагувати, видаляти товари, переглядати кошик та розраховувати підсумкову вартість з урахуванням знижок. Для інтеграції з графічним інтерфейсом або реалізації обміну даними в реальному часі знадобиться розширення цього класу або створення додаткових модулів.

### 3.3 Графічний інтерфейс

Реалізовано графічний інтерфейс користувача (GUI) для системи самообслуговування, використовуючи бібліотеку tkinter (імпортовану як tk) та її розширений модуль tkinter.ttk для створення стилізованих віджетів [10]. Основна логіка управління кошиком покупок інкапсульована в окремому класі Transaction, який імпортується з файлу Transaction.py, що забезпечує розділення відповідальності між представленням даних та їхньою обробкою.

Клас SelfServiceCashierGUI (рис.3.7) відповідає за створення та управління всіма візуальними елементами інтерфейсу. У методі __init__(self, master), який є конструктором класу, відбувається ініціалізація головного вікна (master), встановлення його заголовка та створення об'єкта класу Transaction (self.trnsct). Далі створюється віджет ttk.Treeview (self.item_list)

для відображення товарів у табличному вигляді з заголовками "Назва товару", "Кількість" та "Ціна".

Для додавання нових товарів до кошика створюється фрейм ttk.LabelFrame (add_frame), який містить мітки (ttk.Label) та поля введення (ttk.Entry) для назви, кількості та ціни товару, а також кнопку "Додати товар" (add_button). При натисканні цієї кнопки викликається метод self.add_item(), який обробляє введені дані та додає товар до кошика через об'єкт self.trnsct.

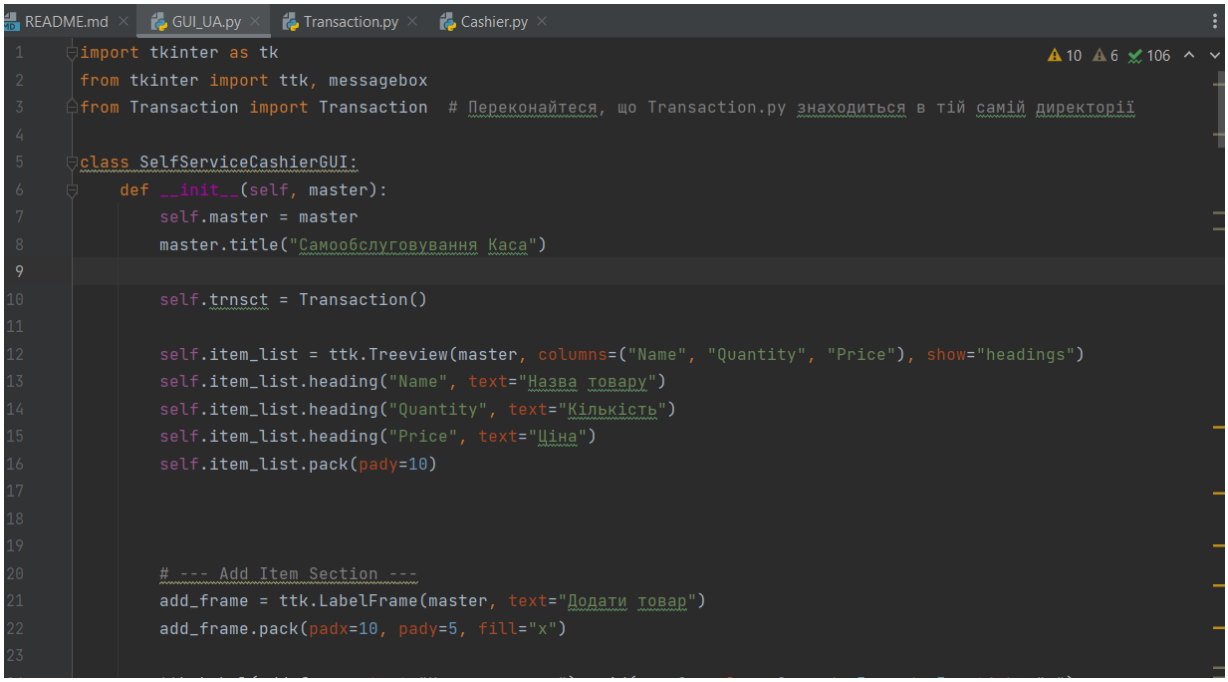


Рисунок 3.7 – Фрагмент програмного коду класу SelfServiceCashierGUI

На рис.3.8 наведено результат графічного інтерфейсу.

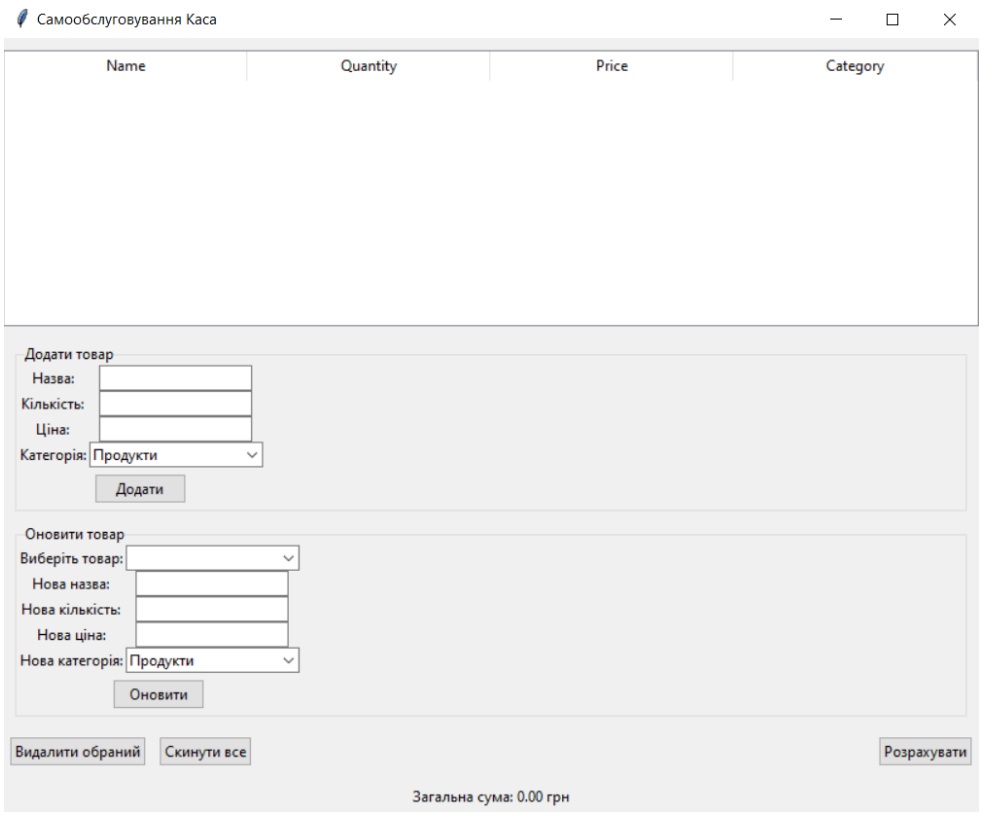


Рисунок 3.8 – Графічний інтерфейс КСС

Для оновлення інформації про існуючі товари створюється фрейм ttk.LabelFrame (update_frame). Він включає випадаючий список (ttk.Combobox, self.update_item_combo), що відображає назви товарів з кошика, поля введення для нової назви, кількості та ціни, а також окремі кнопки для оновлення кожного з цих параметрів (update_name_button, update_quantity_button, update_price_button). Вибір товару у випадаючому списку та введення нових значень з подальшим натисканням відповідної кнопки призводить до виклику методів self.update_item_name(), self.update_item_quantity() або self.update_item_price(), які оновлюють інформацію про товар у кошику.

Для видалення товарів та скидання всієї транзакції створюється фрейм ttk.Frame (delete_reset_frame) з кнопками "Видалити товар" (delete_button, викликає self.delete_item()) та "Скинути транзакцію" (reset_button, викликає self.reset_transaction()). Кнопка "Видалити товар" використовує вибраний товар з випадаючого списку для видалення його з кошика, а кнопка "Скинути

транзакцію" повністю очищає кошик після підтвердження користувача.

Для відображення загальної вартості покупок створюється фрейм ttk.LabelFrame (total_frame), який містить мітку (ttk.Label, self.total_label) для виведення суми та кнопку "Розрахувати загальну ціну" (calculate_button, викликає self.calculate_total()). Метод self.calculate_total() звертається до об'єкта self.trnsct для отримання та відображення загальної вартості товарів у кошику з урахуванням можливих знижок.

Метод self.update_item_list() відповідає за оновлення відображення таблиці товарів (self.item_list) на основі поточного стану кошика (self.trnsct.shopping_cart) та оновлення значень у випадаючому списку для оновлення товарів. Інші методи класу обробляють дії користувача, отримують дані з віджетів інтерфейсу та взаємодіють з об'єктом self.trnsct для виконання відповідних операцій над кошиком покупок. У випадку некоректного введення даних (наприклад, нечислових значень для кількості або ціни) використовуються вікна повідомлень (messagebox) для інформування користувача про помилку. Блок if __name__ == "__main__": забезпечує запуск графічного інтерфейсу при безпосередньому виконанні файлу зі скриптом.

Після успішного встановлення з'єднання, GUI повинен мати можливість як відправляти дані на сервер, так і отримувати від нього оновлення. Відправка даних може відбуватися при кожній значній зміні в кошику покупок (додавання, оновлення, видалення товару) або при завершенні певного етапу транзакції (наприклад, розрахунок загальної вартості). Дані про кошик покупок, включаючи перелік товарів, їх кількість та ціни, повинні бути серіалізовані у формат JSON та відправлені через WebSocket-з'єднання.

Одночасно з відправкою даних, GUI повинен постійно прослуховувати вхідні повідомлення від сервера. Сервер може надсилати різноманітні оновлення, такі як зміни цін на товари, інформацію про їхню наявність, дані про акції або інші важливі повідомлення. При отриманні такого

повідомлення, GUI повинен десеріалізувати його та відповідним чином оновлювати свій інтерфейс, відображаючи актуальну інформацію користувачеві.

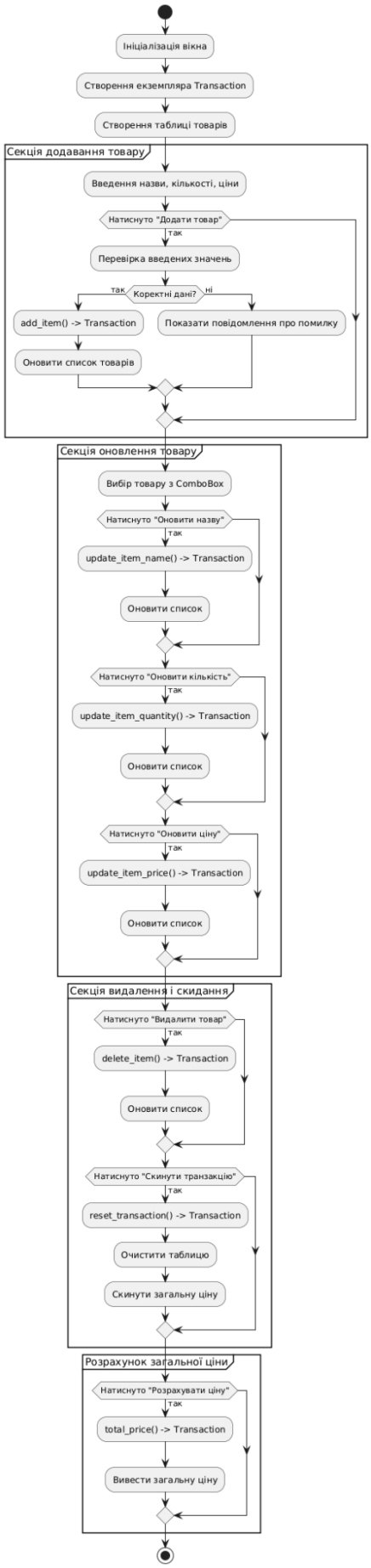


Рисунок 3.9 – Графічний інтерфейс КСС

Оскільки мережеві операції можуть займати певний час і не повинні блокувати роботу GUI, необхідно використовувати асинхронне програмування. Це означає, що відправка та отримання даних повинні виконуватися у фоновому режимі, не перериваючи основний цикл обробки подій tkinter. Для цього можна використовувати ключові слова async та await при визначенні та виклику функцій, що працюють з мережею.

Для забезпечення кращого розуміння користувачем стану системи, до GUI слід додати елементи відображення статусу з'єднання з сервером. Це може бути простий текстовий індикатор або графічний елемент, який показує, чи підключена каса до сервера в даний момент.

### 3.4 Взаємодія користувача з програмним модулем

Основна функціональність модуля включає додавання товарів до віртуального кошика шляхом введення їх назви, кількості та ціни через відповідні поля введення, після чого натискання кнопки "Додати товар" фіксує позицію в кошику та відображає її у верхній таблиці. Користувач має змогу переглядати вміст свого кошика в реальному часі, контролюючи обрані товари та їхні параметри.

Для внесення змін до вже доданих товарів передбачена секція "Оновити товар", де користувач може вибрати потрібну позицію зі спадного списку, що містить назви всіх товарів у кошику. Після вибору стають активними поля для введення нової назви, кількості або ціни. Натискання відповідних кнопок "Оновити назву", "Оновити кількість" або "Оновити ціну" застосовує внесені зміни до вибраного товару, що негайно відображається в таблиці кошика. У випадку помилково доданого товару або зміни рішення, користувач може скористатися кнопкою "Видалити товар", попередньо вибравши його зі списку оновлення.

Для повного очищення кошика та початку нової транзакції передбачена кнопка "Скинути транзакцію", яка після підтвердження користувачем видаляє всі товари з кошика, готуючи систему до нових покупок. Після того,

як усі необхідні товари додані до кошика, користувач може натиснути кнопку "Розрахувати загальну ціну", що ініціює обчислення сумарної вартості всіх товарів з урахуванням їхньої кількості та ціни. Результат розрахунку відображається в секції "Загалом". Завершення роботи з програмним модулем здійснюється натисканням кнопки "Закрити програму", що призводить до закриття вікна GUI та завершення сеансу самообслуговування.

Діаграма взаємодії представлена на рис.3.10 та візуалізує послідовність дій між користувачем та графічним інтерфейсом (GUI) програмного модуля "Самообслуговування Каса", а також його зв'язок з внутрішньою логікою програми, що відповідає за обробку транзакцій. Користувач, представлений актором, є ініціатором усіх операцій у системі, взаємодіючи безпосередньо з різноманітними елементами керування, згрупованими у фреймі "Графічний інтерфейс (GUI)".

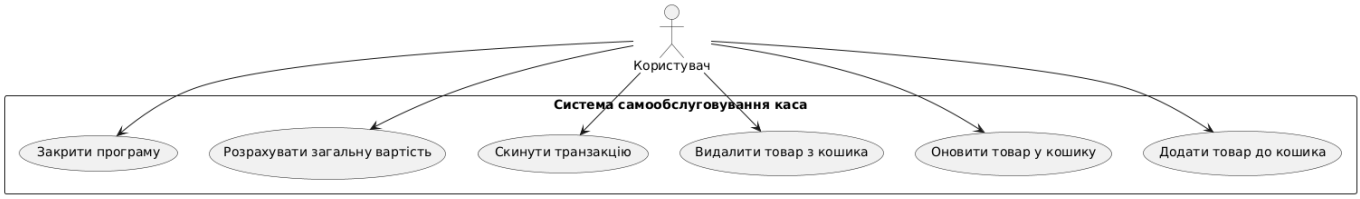


Рисунок 3.10 – Діаграма взаємодії програмного модуля "Самообслуговування Каса"

У межах GUI розташоване основне "Вікно 'Самообслуговування Каса'", яке містить функціональні секції для виконання конкретних дій. Секція "Додати товар" надає користувачеві поля введення для зазначення назви, кількості та ціни товару. Після заповнення цих полів та натискання кнопки "Додати товар", введені дані передаються до внутрішньої логіки програми. Внутрішня логіка, представлена як "Внутрішня логіка програми (Transaction)", оновлює віртуальний кошик покупок, а зміни відображаються у таблиці "Кошик покупок", яка є частиною GUI.

Для редагування вже доданих товарів передбачена секція "Оновити

товар". Користувач спочатку обирає товар зі спадного списку "Виберіть товар", після чого стають доступними поля для введення нових значень назви, кількості або ціни. Введення нових даних та натискання відповідних кнопок ("Оновити назву", "Оновити кількість", "Оновити ціну") призводить до передачі оновленої інформації до внутрішньої логіки, яка модифікує кошик, а зміни знову ж таки відображаються у таблиці кошика. У випадку необхідності видалення товару, користувач, обравши його у секції оновлення, натискає кнопку "Видалити товар", що ініціює відповідну операцію у внутрішній логіці з подальшим оновленням відображення кошика.

Для початку нової транзакції користувач може скористатися кнопкою "Скинути транзакцію". Натискання цієї кнопки надсилає запит до внутрішньої логіки на очищення кошика, після чого таблиця кошика в GUI стає порожньою, а мітка "Загальна ціна" повертається до значення "0.00". Після додавання всіх необхідних товарів, користувач натискає кнопку "Розрахувати загальну ціну". Ця дія передає запит до внутрішньої логіки на обчислення підсумкової вартості, яка потім повертається до GUI та відображається у секції "Загалом" через мітку "Загальна ціна". Завершення роботи з програмним модулем здійснюється натисканням кнопки "Закрити програму", що призводить до закриття головного вікна GUI та завершення сеансу користувача.

Після запуску програми "Самообслуговування Каса" на екрані з'являється головне вікно з таблицею у верхній частині, яка на початку є порожньою та призначена для відображення кошика покупок. У секції "Додати товар" користувач вводить назву бажаного товару, вказує його кількість, ціну за одиницю та обирає відповідну категорію з випадаючого списку. Після заповнення всіх необхідних полів, натискання кнопки "Додати" призводить до того, що в таблиці кошика покупок з'являється новий рядок з інформацією про доданий товар, включаючи його назву, кількість, ціну та категорію. Одночасно на екрані виникає невелике спливаюче вікно з

заголовком "Успіх" та повідомленням "Додано [назва товару]", підтверджуючи успішне виконання операції. Після ознайомлення з повідомленням, користувач закриває вікно натисканням кнопки "OK". При цьому поля введення в секції "Додати товар" залишаються заповненими даними останнього доданого товару, що може бути зручним, якщо потрібно додати кілька одиниць одного й того ж товару або товари зі схожими параметрами (рис.3.11).

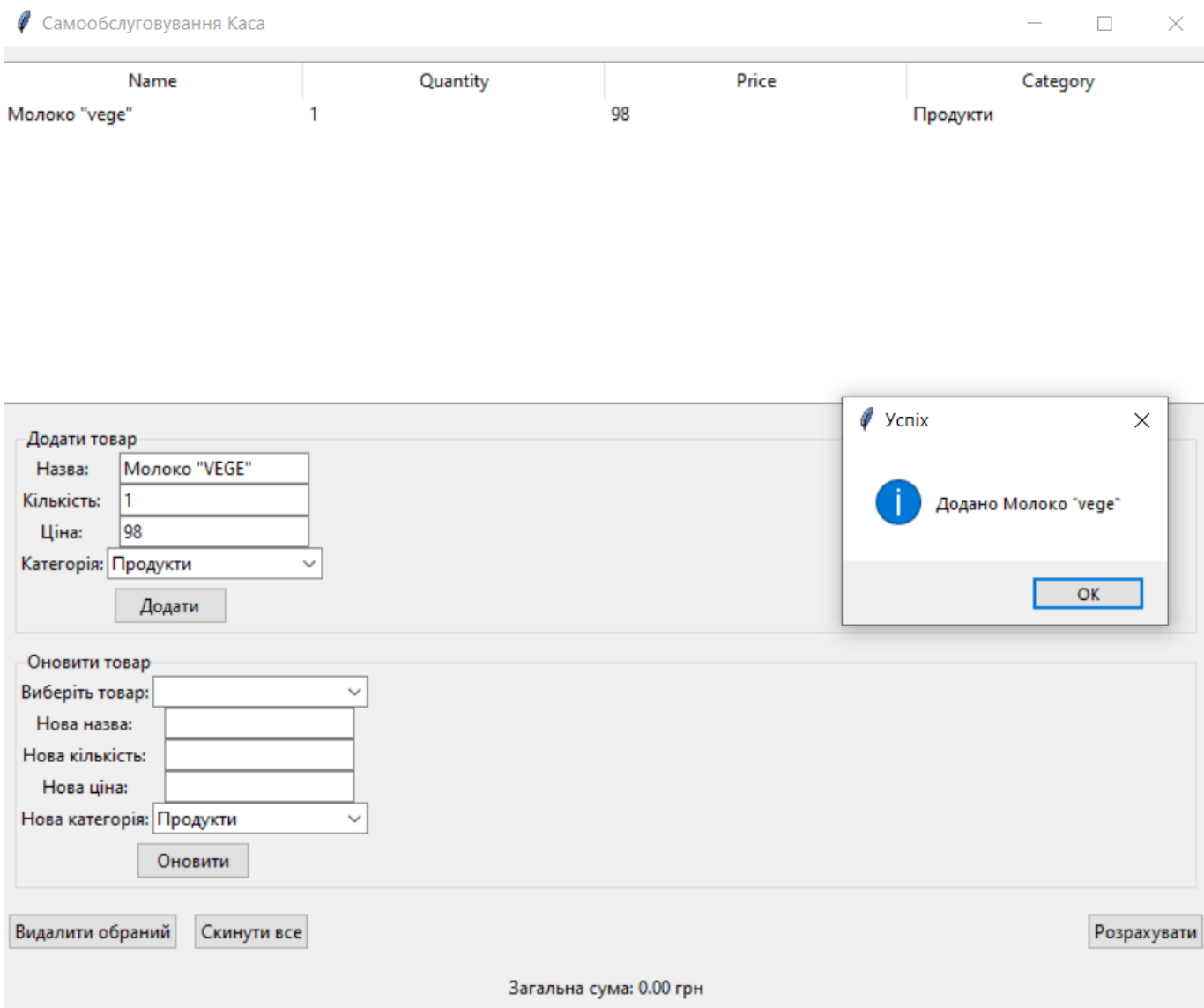


Рисунок 3.11 – Введення даних

Для додавання наступного товару користувачеві необхідно вручну очистити ці поля або ввести нові значення. Загальна сума покупки,

відображена в нижній частині вікна, залишається незмінною до моменту натискання кнопки "Розрахувати", що свідчить про те, що підрахунок відбувається на вимогу користувача, а не автоматично після кожного додавання товару. Таким чином, користувач отримує миттєве візуальне підтвердження успішного додавання товару до свого віртуального кошика та може продовжувати додавати інші товари або переходити до наступних етапів, таких як редагування кількості, видалення позицій чи розрахунок загальної вартості.

Для оновлення інформації про вже доданий товар, такого як "Молоко 'vege'", він спочатку звертає увагу на секцію "Оновити товар" (рис.3.12). У випадаючому списку "Виберіть товар:" користувач обирає назву товару, який потребує змін, в даному випадку "Молоко 'vege'". Після вибору товару, у полях "Нова назва:", "Нова кількість:", "Нова ціна:", та "Нова категорія:" відображаються поточні дані вибраного товару.

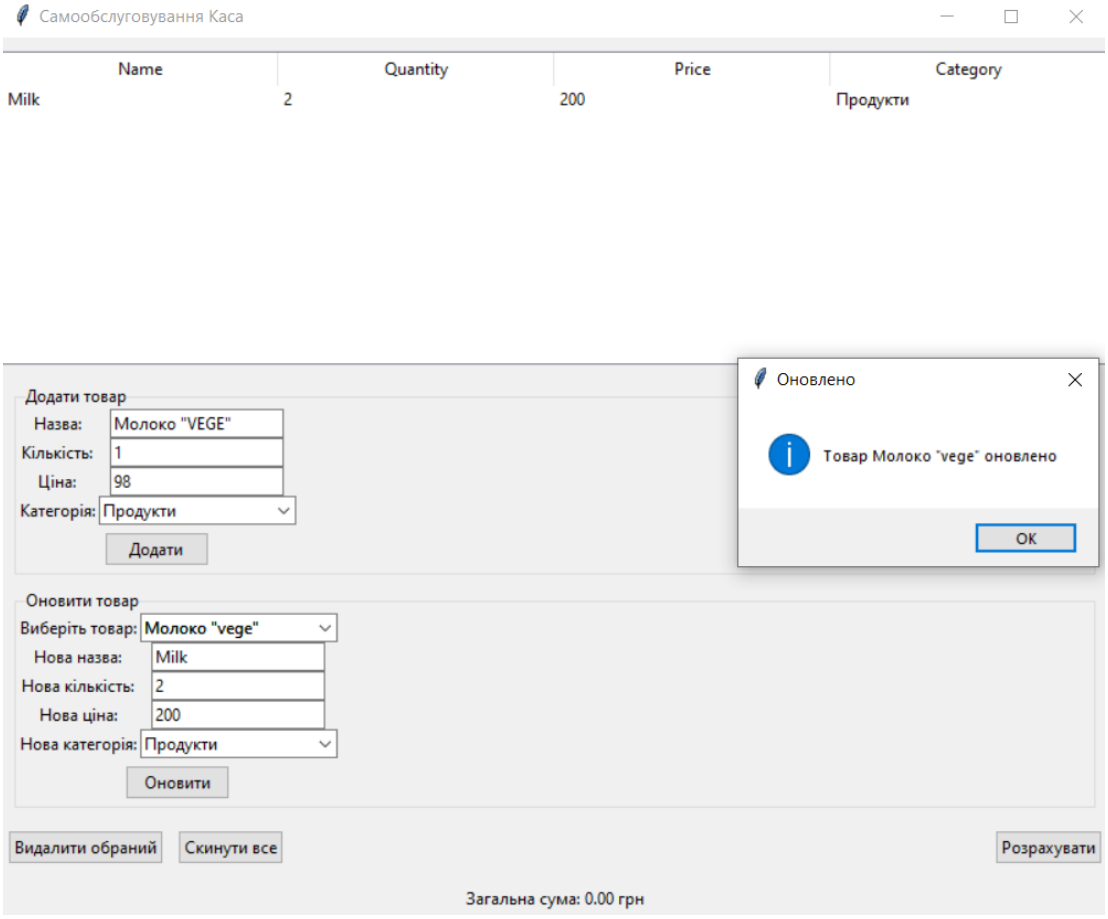


Рисунок 3.12 – Оновлення даних

Щоб внести зміни, користувач вводить нові значення у відповідні поля. На скріншоті видно, що користувач ввів "Milk" у поле "Нова назва:", "2" у поле "Нова кількість:", та "200" у поле "Нова ціна:". Категорія залишена без змін ("Продукти").

Після введення нових даних, користувач натискає кнопку "Оновити", розташовану під секцією оновлення. Внаслідок цієї дії, інформація про вибраний товар у таблиці кошика покупок у верхній частині вікна повинна оновитися. На скріншоті, хоча спливаюче вікно про успішне оновлення відсутнє, можна припустити, що після натискання кнопки "Оновити", рядок з "Молоко 'vege'" в таблиці зміниться на "Milk" з кількістю "2" та ціною "200", а категорія залишиться "Продукти".

Після внесення всіх бажаних товарів до кошика та їхнього редагування, наступним кроком є розрахунок загальної суми покупки. Для цього користувач переводить погляд у нижню праву частину вікна та натискає кнопку "Розрахувати" (рис.3.13).

Після натискання цієї кнопки, в нижній частині вікна, під кнопками "Видалити обраний" та "Скинути все", з'являється інформація про загальну суму покупки. На скріншоті відображається текст "Загальна сума: 400.00 грн (знижка 0%)".

Це означає, що програма успішно просумувала вартість усіх товарів у кошику з урахуванням їхньої кількості. У даному випадку, в кошику знаходиться один товар "Milk" у кількості 2 одиниць за ціною 200 грн за одиницю, що дає загальну суму 2 * 200 = 400.00 грн.

Додатково відображається інформація про застосовану знижку. У цьому випадку "знижка 0%", що свідчить про те, що для цієї покупки жодна знижка не була застосована (можливо, загальна сума ще не досягла порогу для отримання знижки, або в програмі немає активних знижок на придбані товари).

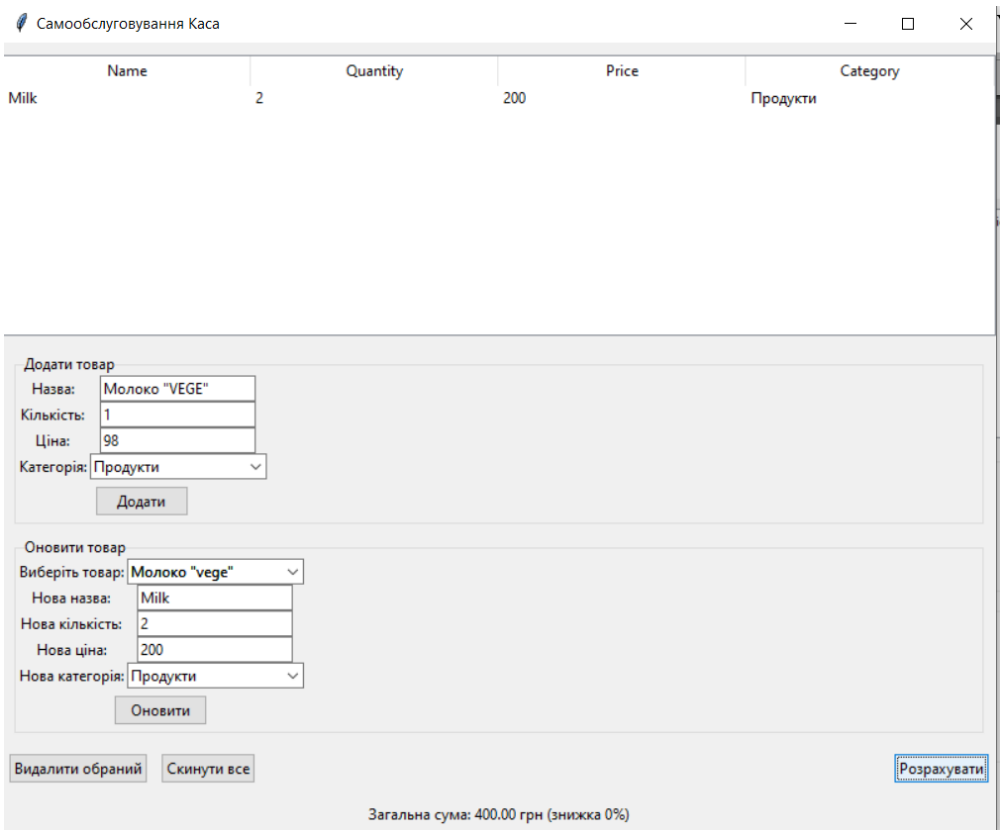


Рисунок 3.13 – Результат розрахунку кошика покупок

### 3.5 Налаштування сервера

WebSocket-сервера, розробленого з використанням фреймворку FastAPI. Файл server.py визначає вебсокетний ендпоїнт /ws, який забезпечує двосторонній обмін даними в реальному часі між сервером та підключеними клієнтами (рис.3.14) [11].

При запуску сервера FastAPI створюється екземпляр класу FastAPI, позначений змінною app. У коді також ініціалізується порожній список clients, який буде використовуватися для зберігання об'єктів WebSocket-з'єднань усіх підключених клієнтів.

Основна функціональність сервера реалізована в асинхронній функції websocket_endpoint, яка декорована @app.websocket("/ws"). Це означає, що коли клієнт встановлює WebSocket-з'єднання за адресою /ws на сервері, буде викликана саме ця функція. Функція приймає один аргумент - об'єкт

WebSocket, який представляє з'єднання з конкретним клієнтом.

```
# server.py
from fastapi import FastAPI, WebSocket
from fastapi.responses import HTMLResponse

app = FastAPI()

clients = []

@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    clients.append(websocket)
    try:
        while True:
            data = await websocket.receive_text()
            print(f"Отримано: {data}")
            # Переслати всім клієнтам (опціонально)
            for client in clients:
                await client.send_text(f"Підтверджено: {data}")
    except:
        clients.remove(websocket)
```

Рисунок 3.14 – Фрагмент реалізації сервера

Першим кроком у функції websocket_endpoint є прийняття вхідного з'єднання за допомогою await websocket.accept(). Після успішного прийняття, об'єкт WebSocket поточного клієнта додається до списку clients. Далі запускається нескінченний цикл while True, який відповідає за постійне прослуховування вхідних повідомлень від клієнта.

Усередині циклу за допомогою data = await websocket.receive_text() сервер очікує отримання текстових даних від підключеного клієнта. Після отримання повідомлення, воно виводиться на консоль сервера за допомогою print(f"Отримано: {data}"). Наступним кроком (який є опціональним, але присутній у коді) є пересилання отриманого повідомлення всім підключеним клієнтам. Це здійснюється шляхом ітерації по списку clients і відправки кожному клієнту тексту f"Підтверджено: {data}" за допомогою await client.send_text(). Таким чином, сервер діє як простий exo-сервер,

підтверджуючи отримання повідомлення та розсилаючи його назад (або іншим клієнтам).

Блок try...except обертає основний цикл обробки повідомлень. У випадку виникнення будь-якої неочікуваної помилки під час обміну даними з клієнтом (наприклад, розриву з'єднання), спрацьовує блок except, який видаляє об'єкт WebSocket цього клієнта зі списку clients за допомогою clients.remove(websocket). Це необхідно для підтримки актуального списку активних підключень.

Діаграма взаємодії (рис.3.15) ілюструє процес обміну даними в реальному часі між користувачем, клієнтським програмним модулем касового обслуговування та сервером центральної системи супермаркету, використовуючи WebSocket-з'єднання. Користувач ініціює всі дії через інтуїтивно зрозумілий графічний інтерфейс (GUI) клієнтського модуля. Взаємодія з GUI призводить до оновлення локальної логіки транзакцій, яка відповідає за керування віртуальним кошиком покупок. Після кожної значущої дії користувача, такої як додавання, оновлення або видалення товару, логіка транзакцій формує відповідні повідомлення у структурованому форматі (наприклад, JSON) та передає їх до WebSocket-клієнта.

WebSocket-клієнт, у свою чергу, встановлює та підтримує постійне двостороннє з'єднання з WebSocket-сервером, розташованим на центральній системі. Всі сформовані повідомлення від клієнта надсилаються на сервер через це з'єднання. Сервер, побудований на основі фреймворку FastAPI, приймає вхідні WebSocket-з'єднання та передає отримані дані до модуля обробки транзакцій. Цей модуль відповідає за виконання бізнес-логіки, включаючи перевірку наявності товарів, застосування знижок та взаємодію з базою даних, де зберігається актуальна інформація про товари та транзакції.

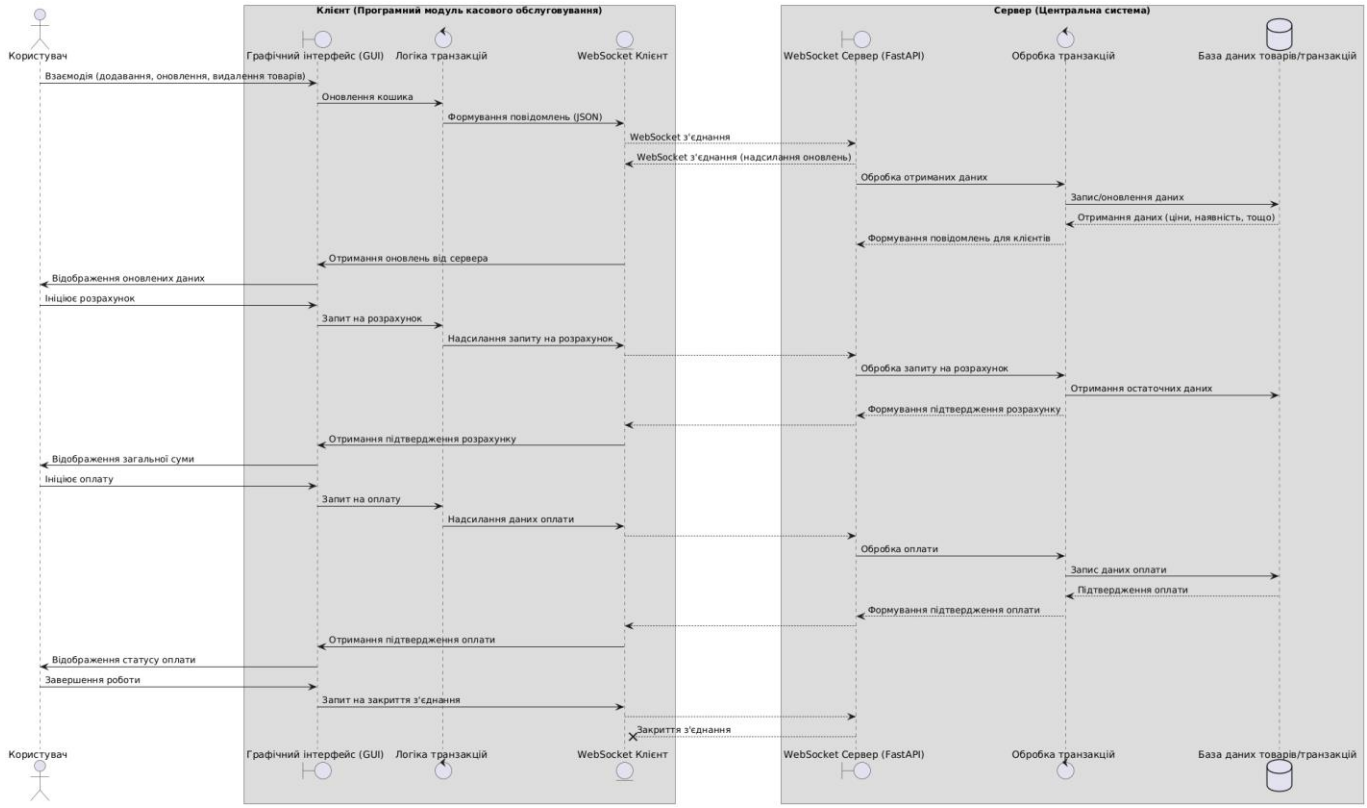


Рисунок 3.15 – Діаграма взаємодії клієнтським програмним модулем касового обслуговування та сервером центральної системи

У процесі обробки отриманих від клієнта даних, сервер може звертатися до бази даних для отримання необхідної інформації, наприклад, актуальних цін або даних про наявність товарів. На основі результатів обробки, сервер може формувати оновлення для клієнтів, які надсилаються назад через WebSocket-з'єднання. WebSocket-клієнт на стороні касового модуля отримує ці оновлення та передає їх до GUI для відображення користувачеві, забезпечуючи таким чином відображення актуальної інформації в реальному часі.

Аналогічний механізм використовується при ініціації користувачем процесу розрахунку та оплати. Клієнтський модуль надсилає відповідні запити на сервер, який їх обробляє, взаємодіє з базою даних для отримання остаточних даних та надсилає підтвердження розрахунку та статусу оплати назад клієнту для відображення користувачеві. При завершенні роботи користувач ініціює закриття з'єднання, про що клієнтський WebSocket

повідомляє сервер, після чого WebSocket-з'єднання закривається. Таким чином, WebSocket виступає ключовою технологією для забезпечення безперервного та миттєвого обміну даними між клієнтською касовою програмою та центральною системою, що є критично важливим для функціонування системи самообслуговування в реальному часі.

# ВИСНОВКИ

Проведено комплексне дослідження та розроблено функціональний програмний модуль, призначений для інтеграції в касові системи самообслуговування сучасних супермаркетів. Ключовим аспектом дослідження стала не лише розробка інтуїтивно зрозумілого та зручного інтерфейсу для кінцевого користувача, але й ефективна реалізація протоколу обміну даними в реальному часі з централізованою системою управління супермаркету.

Створений програмний модуль забезпечує користувачам повний цикл самостійного здійснення покупок. Це включає можливість зручного додавання товарів до віртуального кошика, їх редагування шляхом зміни кількості або видалення, що відображається в реальному часі. Розроблений інтерфейс націлений на максимальну простоту та ефективність взаємодії, задовольняючи основні потреби покупців при самостійному обслуговуванні.

Центральним елементом даної роботи є проектування та успішна інтеграція протоколу обміну даними в реальному часі. Застосування технології WebSocket, як надійного засобу встановлення постійного двостороннього зв'язку, у поєднанні зі стандартизованим форматом обміну даними JSON, дозволило досягти високої швидкості та ефективності передачі інформації між клієнтським касовим модулем та сервером центральної системи. Завдяки цьому забезпечується миттєве відображення будь-яких змін у кошику користувача, актуалізація цін на товари та даних про їхню наявність, а також оперативна обробка фінансових транзакцій та оновлення інформації про стан оплати.

# ПЕРЕЛІК ПОСИЛАНЬ

1. T. Clawson, “Online Shopping Is Killing Physical Stores - Can A Digital Platform Come To The Rescue?,” Forbes, 28 July 2019. [Електронний ресурс] – Режим доступу до ресурсу: https://www.forbes.com/sites/trevorclawson/2019/07/28/online-shopping-iskilling-physical-stores-can-a-digital-platform-come-to-the-rescue

2. P. Gazzola, D. Grechi, I. Martinelli and R. Pezzetti, “The Innovation of the Cashierless Store: A Preliminary Analysis in Italy,” Sustainability, vol. 14, no. 4, pp. 1-7, 2022.

3. Y.-W. Chang, P.-Y. Hsu, J. Chen, W.-L. Shiau and N. Xu, “Utilitarian and/or hedonic shopping – consumer motivation to purchase in smart stores,” Industrial Management & Data Systems, vol. 123, no. 3, pp. 821-842, 2023.

4. M. Nallamothu, M. Galipelli, N. Jangam and M. B. Myneni, “SMART SHOPPING TROLLEY WITH AUTOMATED BILLING USING ARDUINO,” European Chemical Bulletin, vol. 12, no. 5, pp. 3084-3089, 2023.

5. S. Nirmalraj, C. Pranavan, M. Prem and S. Saravanan, “Smart Trolley With IoT Based Billing System,” International Journal of New Innovations in Engineering and Technology, vol. 22, no. 3, pp. 111-115, 2023.

6. R. Rahul, S. N. P. Saastha, P. S. Sai, M. S. Yashwanth and R. R, “Automated Smart Trolley System using RFID Technology,” in 2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA), Coimbatore, 2023.

7. S. R. Subudhi and R. N. Ponnalagu, “An Intelligent Shopping Cart with Automatic Product Detection and Secure Payment System,” in IEEE 16th India Council International Conference (INDICON), Rajkot, 2019.

8. C. Ezhilazhagan, R. Adithya, Y. L. Burhanuddin and F. Charles, “AUTOMATIC PRODUCT DETECTION AND SMART BILLING for SHOPPING USING Li-Fi,” in IEEE International Conference On Recent Trends In Electronics Information Communication Technology, 2016.

9. V. P. Chaudhari, R. V. Chaudhari, A. S. Burgul and S. A. Jain, “Smart SelfCheckout System for Supermarkets,” in IEEE 3rd International Conference on Technology, Engineering, Management for Societal impact using Marketing, Entrepreneurship and Talent (TEMSMET), Mysuru, 2023

10. Tabulate [Електронний ресурс] – Режим доступу до ресурсу: https://pypi.org/project/tabulate/

11. Fastapi [Електронний ресурс] – Режим доступу до ресурсу: https://fastapi.tiangolo.com/

# ДОДАТОК А

Текст програми програмного модуля для касової системи самообслуговування в супермаркеті з реалізацією протоколу обміну даними в реальному часі

**Міністерство освіти і науки України**
**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**
**“ДНІПРОВСЬКА ПОЛІТЕХНІКА”**

**ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ**

# ПРОГРАМНОГО МОДУЛЯ ДЛЯ КАСОВОЇ СИСТЕМИ САМООБСЛУГОВУВАННЯ В СУПЕРМАРКЕТІ З РЕАЛІЗАЦІЄЮ ПРОТОКОЛУ ОБМІНУ ДАНИМИ В РЕАЛЬНОМУ ЧАСІ

Текст програми

804.02070743.25018-01 12 01

Листів 18

2025

# АНОТАЦІЯ

Дана програма містить у собі програмний код модуля для касової системи самообслуговування, розробленого для інтеграції в супермаркетах з метою оптимізації процесу покупок.

Програма призначена для забезпечення користувачам можливості самостійного управління віртуальним кошиком покупок, включаючи додавання, редагування (зміна назви, кількості, ціни, категорії) та видалення товарів, а також для розрахунку загальної суми транзакції. Особливістю програми є реалізація протоколу обміну даними в реальному часі за допомогою WebSocket-з'єднання між клієнтським модулем та центральною системою супермаркету, що забезпечує миттєву актуалізацію інформації про товари та транзакції.

Програма написана мовою Python, відлагоджена із застосуванням бібліотек tkinter для графічного інтерфейсу та FastAPI для реалізації серверної частини WebSocket-з'єднання, і призначена для застосування в розподілених касових системах самообслуговування.

# ЗМІСТ

Server.py

```
# server.py
from fastapi import FastAPI, WebSocket
from fastapi.responses import HTMLResponse

app = FastAPI()

clients = []

@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    clients.append(websocket)
    try:
        while True:
            data = await websocket.receive_text()
            print(f"Отримано: {data}")
            # Переслати всім клієнтам (опціонально)
            for client in clients:
                await client.send_text(f"Підтверджено: {data}")
    except:
        clients.remove(websocket)
```

Transaction.py

```
# Importing tabulate
from tabulate import tabulate

class Transaction:
    '''A class for transaction purposes.'''

    def __init__(self):
        # Initializing empty shopping cart dictionary
        self.shopping_cart = {}

    def add_item(self):
        '''
        Method to add one or more items to shopping cart

        Parameters:
            item_name (str): Name of the item
            quantity (int): Quantity of the item
            price (int): Unit price of the item
            continue_loop (str): looping break/continue indicator
        '''
        #while loop to add items continously if desired
        while True:
            try: #initialize exception handling
                #Input item name, quantity, and price
                self.item_name = input('Input item name: ').capitalize()
                self.quantity = int(input('Input quantity: '))
                self.price = int(input('Input item price: '))

            except ValueError:
```

```
                # Shows Error alert and instruction then continue looping
                print('Quantity or price must be numeric')
                print('Please enter valid input')
                print('-----------------------------------------------')
                continue

            # Shows added item
            print(f'{self.item_name} with quantity {self.quantity} '
                f'and unit price {self.price} is added to shopping cart')
            print('-----------------------------------------------')

            # Update shopping_cart dictionary with added items
            self.shopping_cart.update({self.item_name: [self.quantity,
self.price]})

            # While loop to store looping break indicator
            while True:
                #Input looping break indicator
                self.continue_loop = input('Add more? (y/n): ').lower()
                #If the input is correct break the loop
                if self.continue_loop == 'n':
                    break
                elif self.continue_loop == 'y':
                    break

                # Shows instruction for wrong input
                print('Please enter "y" or "n"')
                print('-----------------------------------------------')

            # Conditional to break looping if the indicator is "n"
```

```
        # or continue the looping if the indicator is "y"
        if self.continue_loop == 'n':
            break
        else:
            continue

    def update_item_name(self):
        '''
        Method to update item name in shopping cart

        Parameters:
            item_name (str): Item name which needs to be changed
            new_item_name (str): New name for the item
        '''
        # While loop to let users try again if the input is wrong
        while True:
            # Check if the shopping cart is empty
            if len(self.shopping_cart) == 0:
                # Shows alert and instruction
                print('The shopping cart is empty!')
                print('Please add some items')
                break

            # Input item name
            self.item_name = input('Item name: ').capitalize()
            # Indicator to check whether the item name is in
            # shopping cart or not
            is_in_cart = self.item_name in self.shopping_cart.keys()

            # Conditional to change item name then break the loop
```

```
            # if the indicator is True
            if is_in_cart == True:
                # Input new item name
                self.new_item_name = input('New item name: ').capitalize()
                self.shopping_cart[self.new_item_name] =
self.shopping_cart[self.item_name]
                del self.shopping_cart[self.item_name]

                # Shows success message
                print('item name updated successfully')
                print('--------------------------------------------')
                break

            # Shows alert and instruction if the input is wrong
            print('There is no such item in shopping cart!')
            print('Please try again')
            print('--------------------------------------------')

    def update_item_quantity(self):
        '''
        Method to update the quantity of an item in shopping cart

        Parameters:
            item_name (str): Item name which quantity needs to be changed
            new_quantity (str): New quantity of the item
        '''
        # While loop to let users try again if the input is wrong
        while True:
            # Check if the shopping cart is empty
            if len(self.shopping_cart) == 0:
```

```
            # Shows alert and instruction
            print('The shopping cart is empty!')
            print('Please add some items')
            break

        # Input item name
        self.item_name = input('Item name: ').capitalize()
        # Indicator to check whether the item name is in
        # shopping cart or not
        is_in_cart = self.item_name in self.shopping_cart.keys()

        # Conditional to change item quantity then break the loop
        # if the indicator is True
        if is_in_cart == True:
            # Input new item quantity
            self.new_quantity = int(input('New item quantity: '))
            self.shopping_cart[self.item_name][0] = self.new_quantity

            # Shows success message
            print('item quantity updated successfully')
            print('-----------------------------------------------')
            break

        # Shows alert and instruction if the input is wrong
        print('There is no such item in shopping cart!')
        print('Please try again')
        print('-----------------------------------------------')

def update_item_price(self):
    '''
```

```
    Method to update the price of an item in shopping cart

    Parameters:
        item_name (str): Item name which quantity needs to be changed
        new_price (int): New price of the item
    '''
    # While loop to let users try again if the input is wrong
    while True:
        # Check if the shopping cart is empty
        if len(self.shopping_cart) == 0:
            # Shows alert and instruction
            print('The shopping cart is empty!')
            print('Please add some items')
            break

        # Input item name
        self.item_name = input('Item name: ').capitalize()
        # Indicator to check whether the item name is in
        # shopping cart or not
        is_in_cart = self.item_name in self.shopping_cart.keys()

        # Conditional to change item price then break the loop
        # if the indicator is True
        if is_in_cart == True:
            self.new_price = int(input('New item price: '))
            self.shopping_cart[self.item_name][1] = self.new_price

            # Shows success message
            print('item price updated successfully')
            print('-----------------------------------------------')
```

```
            break

        # Shows alert and instruction if the input is wrong
        print('There is no such item in shopping cart!')
        print('Please try again')
        print('--------------------------------------------')

def delete_item(self):
    '''
    Method to remove an item from shopping cart

    Parameters:
        item_name (str): Item name which wants to be removed
    '''
    # While loop to let users try again if the input is wrong
    while True:
        # Check if the shopping cart is empty
        if len(self.shopping_cart) == 0:
            # Shows alert and instruction
            print('The shopping cart is empty!')
            print('Please add some items')
            break

        # Input item name
        self.item_name = input('Item name: ').capitalize()
        # Indicator to check whether the item name is in
        # shopping cart or not
        is_in_cart = self.item_name in self.shopping_cart.keys()

        # Conditional to remove the item from shopping cart
```

```
        # if the indicator is True then break the loop
        if is_in_cart == True:
            self.shopping_cart.pop(self.item_name)

            # Shows success message
            print('item deleted successfully')
            print('-----------------------------------------------')
            break

        # Shows alert and instruction if the input is wrong
        print('There is no such item in shopping cart!')
        print('Please try again')
        print('-----------------------------------------------')

    def reset_transaction(self):
        '''Method to delete all items from shopping cart'''
        # Clear all keys and values in shopping_cart dictionary
        self.shopping_cart.clear()

        # Shows success message
        print('Shopping cart is now empty!')
        print('-----------------------------------------------')

    def check_order(self):
        '''Method to check and show the items in shopping cart'''
        # Data for shopping list table
        self.data = [
            ['Item Name', 'Quantity', 'Price', 'Total Price'] # Headers
        ]
```

```
    # For loop to add data for each headers
    for key, value in self.shopping_cart.items():
        self.data.append([key, value[0], value[1], value[0]*value[1]])

    # Print table of item lists in shopping cart
    print(tabulate(self.data, headers="firstrow"))

    # For loop to check the quantity and price
    # of items in shopping list
    for key, value in self.shopping_cart.items():
        if value[0] < 0 or value[1] < 0:
            # Shows alert and instruction for wrong input
            print(f'The quantity or price of {key} is wrong')
            print('Please update the item quantity or price')


def total_price(self):
    '''Function to calculate total price'''
    # Check if the shopping cart is empty
    if len(self.shopping_cart) == 0:
        # Shows alert and instruction
        print('The shopping cart is empty!')
        print('Please add some items')
    else: # Calculate total price if not empty
        # Initialize total_price variable as 0
        total_price = 0

        # Calculate the sum of each item's total price
        for index in range(1, len(self.data)):
            total_price += self.data[index][3]
```

```
# Conditional for discount
if total_price > 500_000:
    discount = 0.1 # 10% discount
    total_price -= discount * total_price
    print('You get 10% discount!')
elif total_price > 300_000:
    discount = 0.08 # 8% discount
    total_price -= discount * total_price
    print('you get 8% discount!')
elif total_price > 200_000:
    discount = 0.05 #5% discount
    total_price -= discount * total_price
    print('you get 5% discount!')
else:
    # Shows discount list if the user
    # does not get any discount
    print('You get no discount')
    print('''
    Discount list if you are interested:
    5%: if total price is greater than 200000
    8%: if total price is greater than 300000
    10%: if total price is greater than 500000
    ''')

# Shows the total price after discount (if any)
print(f'Total price: Rp.{total_price:,.2f}')
```

Cashier.py

```
import tkinter as tk
from tkinter import messagebox, ttk
from tabulate import tabulate

class Transaction:
    def __init__(self):
        self.shopping_cart = {}

    def add_item(self, item_name, quantity, price, category):
        self.shopping_cart[item_name] = [quantity, price, category]

    def update_item(self, item_name, quantity=None, price=None,
category=None):
        if item_name in self.shopping_cart:
            if quantity is not None:
                self.shopping_cart[item_name][0] = quantity
            if price is not None:
                self.shopping_cart[item_name][1] = price
            if category is not None:
                self.shopping_cart[item_name][2] = category

    def reset_transaction(self):
        self.shopping_cart.clear()

    def get_cart_data(self):
        data = [["Item", "Quantity", "Price", "Category", "Total"]]
        for k, v in self.shopping_cart.items():
            data.append([k, v[0], v[1], v[2], v[0]*v[1]])
        return tabulate(data, headers="firstrow")
```

```
    def get_total_price(self):
        total = sum(q * p for q, p, _ in self.shopping_cart.values())
        discount = 0
        if total > 500_000:
            discount = 0.10
        elif total > 300_000:
            discount = 0.08
        elif total > 200_000:
            discount = 0.05
        discounted_total = total * (1 - discount)
        return discounted_total, discount

# --- GUI ---
class App:
    def __init__(self, root):
        self.root = root
        self.root.title("Shopping Cart with Categories")
        self.tx = Transaction()

        self.name_var = tk.StringVar()
        self.qty_var = tk.StringVar()
        self.price_var = tk.StringVar()
        self.cat_var = tk.StringVar()

        # Entry fields
        tk.Label(root, text="Item Name").grid(row=0, column=0)
        tk.Entry(root, textvariable=self.name_var).grid(row=0, column=1)

        tk.Label(root, text="Quantity").grid(row=1, column=0)
```

```
        tk.Entry(root, textvariable=self.qty_var).grid(row=1, column=1)

        tk.Label(root, text="Price").grid(row=2, column=0)
        tk.Entry(root, textvariable=self.price_var).grid(row=2, column=1)

        tk.Label(root, text="Category").grid(row=3, column=0)
        categories = ["Продукти", "Електроніка", "Одяг", "Інше"]
        self.cat_combo = ttk.Combobox(root, textvariable=self.cat_var,
values=categories)
        self.cat_combo.grid(row=3, column=1)
        self.cat_combo.set(categories[0])

        # Buttons
        tk.Button(root, text="Add Item", command=self.add_item).grid(row=4,
column=0, columnspan=2, pady=5)
        tk.Button(root, text="Update Item",
command=self.update_item).grid(row=5, column=0, columnspan=2,
pady=5)
        tk.Button(root, text="Show Cart",
command=self.show_cart).grid(row=6, column=0, columnspan=2, pady=5)
        tk.Button(root, text="Reset Cart",
command=self.reset_cart).grid(row=7, column=0, columnspan=2, pady=5)
        tk.Button(root, text="Total Price",
command=self.show_total).grid(row=8, column=0, columnspan=2, pady=5)

    def add_item(self):
        try:
            name = self.name_var.get().capitalize()
            qty = int(self.qty_var.get())
            price = int(self.price_var.get())
```

```
            cat = self.cat_var.get()
            self.tx.add_item(name, qty, price, cat)
            messagebox.showinfo("Success", f"Додано товар: {name}")
        except ValueError:
            messagebox.showerror("Error", "Кількість і ціна мають бути числами")

    def update_item(self):
        try:
            name = self.name_var.get().capitalize()
            qty = int(self.qty_var.get())
            price = int(self.price_var.get())
            cat = self.cat_var.get()
            self.tx.update_item(name, quantity=qty, price=price, category=cat)
            messagebox.showinfo("Updated", f"Оновлено товар: {name}")
        except ValueError:
            messagebox.showerror("Error", "Кількість і ціна мають бути числами")

    def show_cart(self):
        cart_data = self.tx.get_cart_data()
        messagebox.showinfo("Кошик", cart_data)

    def reset_cart(self):
        self.tx.reset_transaction()
        messagebox.showinfo("Очищено", "Кошик очищено")

    def show_total(self):
        total, discount = self.tx.get_total_price()
        percent = int(discount * 100)
```

```
        messagebox.showinfo("Разом", f"Сума зі знижкою {percent}%:
{total:,.2f} грн")

# Run app
if __name__ == "__main__":
    root = tk.Tk()
    app = App(root)
    root.mainloop()
```