

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА
ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«POS-система для автоматизації продажів на
платформі Android»**

Виконав студент групи ЗПЗ-22-1
ДЕРЖАНІВСЬКИЙ Артем Олегович

Керівник роботи:
ПІОНТКІВСЬКИЙ Владислав Ігорович

Рецензент:
КОНЦИДАЙЛО Андрій Михайлович

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА
ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк
«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ДЕРЖАНІВСЬКИЙ Артем Олегович**

Керівник роботи: **ПІОНТКІВСЬКИЙ Владислав Ігорович**

Тема роботи: **«POS-система для автоматизації продажів на платформі Android»,**

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №93/с**

Вихідні дані для роботи: об'єктом дослідження є процес автоматизації продажів та обліку товарів у торговельних закладах, а предметом – мобільна POS-система для автоматизації цих процесів

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Піонтківський В.І.	02.03.2026р.	02.03.2026р.
2	Піонтківський В.І.	01.04.2026р	01.04.2026р
3	Піонтківський В.І	01.05.2026р	01.05.2026р

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з мобільного додатку та пояснювальної записки.

Пояснювальна записка до випускної кваліфікаційної роботи містить 78 сторінок друкованого тексту, 26 рисунки, 23 таблиці, 2 додатки та список використаних джерел із 30 найменувань.

Метою роботи є розробка мобільної POS-системи для автоматизації процесів продажу, обліку товарів та формування звітності у торговельних закладах на платформі Android.

У роботі проведено аналіз предметної області автоматизації продажів, досліджено існуючі POS-рішення та визначено їх основні функціональні можливості. Для реалізації системи використано сучасні методи та технології розробки мобільних застосунків, зокрема архітектурний підхід MVVM, мову програмування Kotlin, хмарну платформу Supabase, Hilt, Jetpack Navigation і Material Design Components.

У результаті виконання роботи розроблено Android-застосунок, що забезпечує ведення бази даних товарів і продажів, автоматизацію оформлення замовлень, управління складськими залишками, формування звітності. Реалізовано зручний користувацький інтерфейс.

Отримані результати можуть бути використані для автоматизації діяльності невеликих магазинів та інших торговельних закладів. Розроблена система може бути впроваджена як основа для подальшого розвитку мобільних POS-рішень на платформі Android.

Ключові слова: ANDROID, KOTLIN, MVVM, POS-СИСТЕМА, АВТОМАТИЗАЦІЯ ПРОДАЖІВ.

					ІПЗ.КР.Б – 121 – 26 – ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата							
Розроб.		Держанівський			POS-система для автоматизації продажів на платформі Android			Літ.	Арк.	Аркушів	
Керівник		Піонткіський В.								3	100
								Житомирська політехніка, група ЗІПЗ-22-1			
Н. контр.											
Зав. каф.											

ABSTRACT

The bachelor's qualification thesis consists of a mobile application and an explanatory note.

The explanatory note to the bachelor's qualification thesis contains 78 pages of printed text, 26 figures, 23 tables, 2 appendices, and a list of references with 30 sources.

The aim of the work is to develop a mobile POS system for automating sales processes, inventory management, and reporting in retail establishments on the Android platform.

The study includes an analysis of the domain of sales automation, an examination of existing POS solutions, and identification of their main functional capabilities. Modern methods and technologies for mobile application development were used in the implementation of the system, including the MVVM architectural approach, the Kotlin programming language, the Supabase cloud platform, and libraries such as Hilt, Jetpack Navigation, and Material Design Components.

As a result of the work, an Android application was developed that provides a database for products and sales, automates order processing, manages stock levels, and generates reports. A user-friendly interface was implemented in accordance with Material Design principles.

The obtained results can be used to automate the operations of small shops, cafés, and other retail establishments. The developed system can serve as a foundation for further development of mobile POS solutions on the Android platform.

Keywords: ANDROID, KOTLIN, MVVM, POS SYSTEM, SALES AUTOMATION.

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

РЕФЕРАТ	3
ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ	10
1.1. Постановка задачі.....	10
1.2. Аналіз аналогів програмного продукту.....	12
1.3. Архітектура та укрупнена структура мобільного додатку.....	21
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	25
Висновки до першого розділу.....	31
РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ	33
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи	33
2.2. Розробка бази даних системи на базі хмарного сервісу Supabase.....	42
2.3. Проєктування та реалізація алгоритмів роботи системи	48
2.4. Реалізація мобільного додатку системи.....	53
Висновки до другого розділу	59
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З POS-СИСТЕМОЮ АВТОМАТИЗАЦІЇ ПРОДАЖІВ	61
3.1. Порядок встановлення та налаштування параметрів системи	61
3.2. Структура інтерфейсу мобільного додатку	64
3.3. Тестування та перевірка роботи додатку	72
Висновки до третього розділу.....	75
ВИСНОВКИ.....	77
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	79
ДОДАТКИ	82

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних.

MVVM – архітектурний шаблон побудови застосунків Model-View-ViewModel.

POS – Point of Sale, система автоматизації продажів.

SDK – Software Development Kit, набір засобів для розробки програмного забезпечення.

Android SDK – набір інструментів для розробки Android-застосунків.

UI – User Interface, користувацький інтерфейс.

XML – eXtensible Markup Language, мова розмітки даних.

Supabase – Backend-as-a-Service платформа для роботи з базою даних, авторизацією та API.

Kotlin – мова програмування, що використовується для розробки Android-застосунків.

Coroutines – механізм асинхронного програмування в Kotlin.

Flow – асинхронний потік даних у Kotlin.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		6

ВСТУП

У сучасному цифровому середовищі автоматизація бізнес-процесів є ключовим фактором ефективного функціонування підприємств роздрібної торгівлі. Торговельні заклади щоденно здійснюють велику кількість операцій, пов'язаних із продажем товарів, обліком складських залишків, контролем фінансових надходжень та формуванням звітності. Використання застарілих або розрізнених інструментів ускладнює ведення обліку, підвищує ймовірність помилок та знижує продуктивність роботи персоналу.

Стрімкий розвиток мобільних технологій та операційної системи Android відкриває нові можливості для створення сучасних POS-рішень. Смартфони та планшети можуть використовуватися як повноцінні касові термінали, засоби обліку товарів і аналітики продажів. Завдяки використанню сучасних інструментів розробки, зокрема мови програмування Kotlin, архітектури MVVM та хмарної платформи Supabase, стає можливим створення надійної, масштабованої та зручної у використанні POS-системи.

Актуальність теми зумовлена необхідністю розробки мобільної POS-системи для автоматизації процесів продажів у торговельних закладах, що забезпечить комплексне управління операціями з товарами, замовленнями та фінансовими даними. Така система повинна підтримувати функції оформлення продажів, обліку товарних залишків, формування звітності та аналізу ключових показників діяльності. Важливим аспектом є забезпечення інтуїтивно зрозумілого інтерфейсу, який дозволить швидко освоїти систему та ефективно використовувати її в щоденній роботі.

Метою роботи є розробка мобільної POS-системи для автоматизації продажів на платформі Android із використанням сучасних технологій мобільної розробки.

Для досягнення мети потрібно виконати завдання:

- Провести аналіз предметної області та огляд існуючих рішень конкурентів
- Визначити функціонал, та потенційні напрямки для розширення

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

- Спроекувати архітектуру системи
- Спроекувати базу даних
- Реалізація інтерфейсу користувача
- Проведення тестування системи

Об’єктом дослідження є процес автоматизації продажів та обліку товарів у торговельних закладах.

Предметом дослідження є методи та засоби розробки мобільних POS-систем для Android із використанням мови програмування Kotlin, архітектури MVVM та хмарної бази даних Supabase.

Методи дослідження включають аналіз існуючих POS-систем, порівняльний аналіз функціональних можливостей мобільних рішень, методи проєктування програмного забезпечення, а також методи розробки мобільних додатків на платформі Android. У процесі реалізації використано архітектурний підхід MVVM та сучасні інструменти розробки, зокрема Kotlin і Supabase. Інформаційною базою дослідження є наукові джерела, документація Android SDK, офіційна документація Supabase та аналіз існуючих програмних продуктів у сфері POS-систем.

Результатом виконання є функціональна мобільна POS-система, яка забезпечує ефективний облік продажів, управління товарними залишками, формування звітності та підтримку прийняття управлінських рішень. Реалізація даного проєкту сприятиме підвищенню ефективності роботи торговельних закладів, зменшенню кількості помилок та оптимізації бізнес-процесів.

Таким чином, дана робота спрямована на вирішення актуальної задачі створення сучасного, надійного та зручного мобільного інструменту для автоматизації продажів у сфері роздрібної торгівлі.

За темою випускної кваліфікаційної роботи бакалавра були опубліковані тези:

1. Держанівський А.О., Піонтківський В.І. Аналіз існуючих POS-систем управління // Тези XVI Міжнародної науково-технічної конференції «Інформаційно-комп’ютерні технології», м. Житомир, 2–3 квітня 2026 р. Житомир: Державний університет «Житомирська політехніка», 2026. С. 47–48 [1].

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

2. Держанівський А.О., Піонтківський В.І. Опис застосування продукту Supabase для використання в мобільній розробці // Тези доповідей науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки, 11-15 травня 2026 р. Житомир: Державний університет «Житомирська політехніка», 2026. С. 122 [\[2\]](#).

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. АНАЛІЗ ПРОБЛЕМАТИКИ, МЕТОДІВ ТА ЗАСОБІВ ВИРІШЕННЯ ЗАДАЧІ

1.1. Постановка задачі

У сучасних умовах розвитку інформаційних технологій та автоматизації бізнес-процесів особливої актуальності набуває використання мобільних систем для обліку та управління продажами. Значна кількість торговельних підприємств використовує застарілі або частково автоматизовані засоби ведення обліку, що ускладнює контроль товарообігу, процес формування звітності та управління персоналом. Відсутність централізованої системи обліку може призводити до помилок під час виконання операцій продажу, втрати даних, некоректного ведення складського обліку та зниження ефективності роботи підприємства.

Для вирішення зазначених проблем необхідною є розробка мобільної POS-системи для платформи Android, яка забезпечить автоматизацію процесів продажу, контроль товарних залишків, формування звітності та управління користувачами системи. Використання мобільного застосунку дозволяє забезпечити швидкий доступ до функціоналу системи, зручність використання та можливість роботи на різних типах Android-пристроїв.

Для досягнення поставленої мети та систематизації процесу розробки мобільної системи, було визначено ряд послідовних кроків, які охоплюють усі стадії створення програмного продукту – від дослідження ідеї до перевірки готового рішення.

Першочерговим завданням визначено проведення детального аналізу предметної області та огляду існуючих рішень конкурентів. Цей етап передбачає глибоке вивчення специфіки роботи торговельних закладів, процесів обслуговування клієнтів та особливостей ведення обліку. Планується вивчити досвід наявних на ринку програмних продуктів, щоб виявити їхні сильні та слабкі сторони. Це дозволить сформулювати концепцію власної системи, яка б поєднувала простоту використання з високою ефективністю вирішення повсякденних задач малого бізнесу.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

Наступним логічним етапом є визначення функціональних можливостей системи та опис потенційних напрямків для її подальшого розширення. На цій стадії планується чітко окреслити межі проєкту: які саме операції з товарами, продажами та персоналом має підтримувати додаток. Важливо не лише задовольнити поточні потреби користувача, а й передбачити гнучкість системи для впровадження нових інструментів у майбутньому, таких як розширена аналітика чи інтеграція з додатковими зовнішніми сервісами, що забезпечить життєздатність продукту в довгостроковій перспективі.

Фундаментальним кроком у підготовці до розробки є проєктування архітектури системи. Це завдання спрямоване на створення логічної структури майбутнього застосунку, де кожен компонент має свою чітко визначену роль. Планується розробити таку модель взаємодії між частинами програми, яка б забезпечила стабільність роботи, легкість у внесенні змін та високу швидкість обробки інформації. Основна мета тут – побудувати надійний каркас, на якому триматиметься весь програмний функціонал.

Окремим завданням стоїть проєктування бази даних, що є основою інформаційного середовища системи. Планується розробити схему зберігання даних, яка б гарантувала цілісність та безпеку інформації про товари, історію фінансових операцій та профілі користувачів. Процес проєктування має враховувати необхідність швидкого доступу до даних та можливість їх надійної синхронізації з хмарним сховищем.

Безпосередня взаємодія користувача з програмою забезпечується через реалізацію інтерфейсу користувача. Завданням цього етапу є створення графічних оболонок, які будуть максимально зручними та зрозумілими для працівників закладу. Планується розробити візуальний стиль та логіку переходів між екранами таким чином, щоб оформлення продажу чи пошук потрібного товару займали мінімум часу та не вимагали спеціального навчання. Головний пріоритет — ергономічність та швидкість роботи касового термінала.

Завершальним, але не менш важливим завданням є проведення комплексного тестування системи. На цьому етапі планується ретельна перевірка всіх

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

розроблених модулів на предмет відповідності початковим вимогам. Передбачається виявлення та усунення можливих помилок у логіці програми, перевірка коректності обчислень та стійкості додатка до навантажень. Фінальне тестування має підтвердити готовність системи до повноцінної експлуатації та її здатність надійно виконувати поставлені бізнес-задачі.

Результатом виконання випускної роботи має стати мобільна POS-система для автоматизації продажів, яка забезпечуватиме зручне ведення обліку товарів, швидке виконання операцій продажу, формування звітності та централізоване зберігання даних.

1.2. Аналіз аналогів програмного продукту.

Процес розробки мобільної POS-системи для автоматизації продажів на платформі Android вимагає детального та глибокого аналізу існуючих програмних продуктів у сфері автоматизації торгівлі та обліку складських операцій. Дослідження ринку аналогів дозволяє чітко визначити сучасні архітектурні та функціональні підходи до організації касових операцій, ведення складського обліку, моніторингу товарних потоків та формування управлінської аналітики, а також виявити критичні переваги й недоліки популярних рішень. Такий аналіз є обов'язковим етапом проєктування, оскільки він закладає підґрунтя для формування вимог до власного програмного продукту та допомагає визначити вектори його функціонального вдосконалення. На сьогоднішній день ринок пропонує значну кількість POS-систем для малого та середнього бізнесу, більшість з яких базується на хмарній архітектурі, підтримує синхронізацію між різними типами клієнтських пристроїв та інтеграцію з державними фіскальними сервісами. Перед початком безпосереднього проєктування власного додатка доцільно детально вивчити функціональну структуру аналогів, їхні інтерфейсні рішення та моделі взаємодії з користувачем, щоб виокремити базовий набір можливостей і зменшити ризики повторення чужих помилок.

Одним із найбільш поширених та технологічно якісних рішень на ринку є хмарна система Poster POS (рис. 1.1) [3]. Функціонально цей програмний продукт

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

розділений на два великих блоки, де першим є безпосередньо касовий додаток для планшетів та смартфонів, а другим – веб-панель для адміністратора. Касовий модуль Poster POS забезпечує швидке формування чеків за допомогою інтерактивного графічного меню з категоріями товарів, підтримує роботу з модифікаторами (наприклад, вибір об’єму порції кави чи додавання сиропу) та дозволяє розділяти чеки між гостями або відкладати їх для подальшого розрахунку. Система автоматично вираховує решту, підтримує комбіновану оплату готівкою та карткою, а також містить вбудований інструмент лояльності для ідентифікації постійних клієнтів за номерами телефонів чи штрих-кодами.

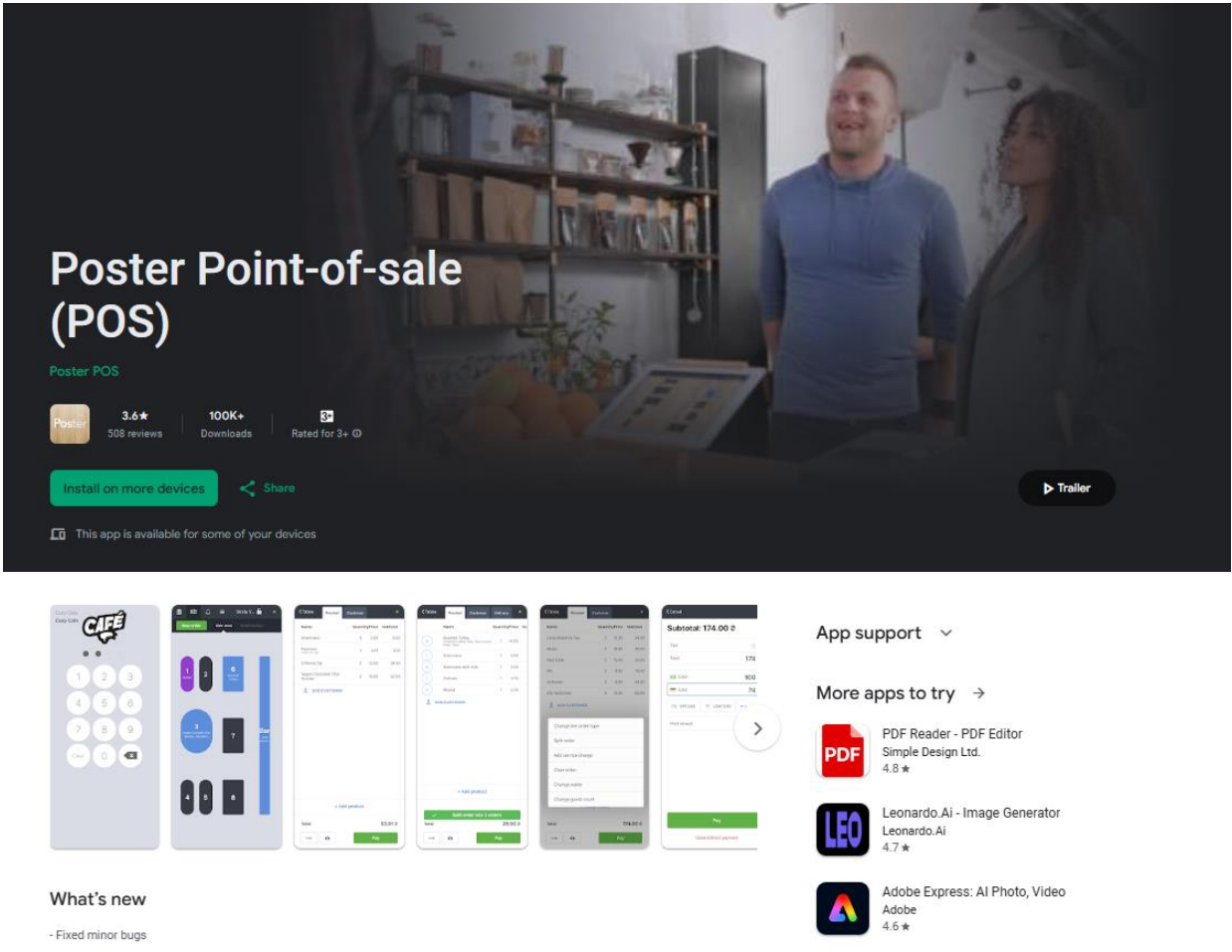


Рис. 1.1. Додаток Poster POS

З боку складського обліку функціонал Poster POS реалізує повний цикл руху інгредієнтів та готової продукції, включаючи створення технологічних карт для складних страв і напоїв, де при продажу одного кубка кави зі складу автоматично

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

списується чітко визначена кількість грамів кавового зерна, мілілітрів молока та цукру. Окрім автоматичного списання, складський модуль дозволяє проводити ручні операції оприбуткування нових партій від постачальників, фіксувати акти списання псування чи браку, проводити покрокову інвентаризацію окремих категорій товарів та контролювати критичні залишки, надсилаючи повідомлення адміністратору, коли певний продукт добігає кінця. Аналітичний блок системи у реальному часі акумулює дані з касових терміналів та формує детальні звіти про виручку, чистий прибуток, середній чек, а також визначає найбільш маржинальні та ходові позиції за допомогою вбудованого аналізу, надаючи можливість експортувати статистику у формі таблиць для подальшої обробки.

Серед головних переваг Poster POS варто виділити високу швидкість обробки замовлень завдяки ергономічному дизайну інтерфейсу, повноцінну підтримку хмарної синхронізації та наявність кросплатформеного віддаленого доступу до панелі адміністратора з будь-якої точки світу. Також система має гнучкі інструменти інтеграції з фіскальними реєстраторами та програмними РРО, що дозволяє бізнесу працювати у правовому полі. Проте, детальний розбір виявляє і низку суттєвих недоліків, де ключовим є висока вартість використання, оскільки повноцінний функціонал складського обліку та аналітики доступний виключно за щомісячною платною підпискою, що створює постійне фінансове навантаження на маленькі підприємства. Іншим вагомим мінусом є жорстка залежність від стабільного інтернет-з'єднання, адже хоча локальна каса і може деякий час працювати в офлайн-режимі, повноцінна синхронізація залишків та оновлення даних зупиняються до відновлення мережі. Крім того, незважаючи на продуманість, початкове налаштування технологічних карт, податкових ставок та прав доступу користувачів є занадто складним для початківців у малому бізнесі, що потребує додаткового часу на навчання персоналу.

Іншим сучасним та вузькоспеціалізованим рішенням, розробленим переважно з орієнтацією на кав'ярні та невеликі заклади громадського харчування, є система автоматизації KavApp (рис. 1.2) [4]. На відміну від універсальних комерційних систем, цей програмний продукт спроектований з урахуванням

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

специфіки швидких продажів, де ключову роль відіграє швидкість обслуговування клієнта та точний облік специфічних інгредієнтів. Функціонал касового термінала KavApp Seller дозволяє касиру оформлювати замовлення буквально в кілька натискань за рахунок адаптованого під роботу на смартфонах інтерфейсу. Касовий модуль повністю закриває потреби у виборі об'ємів напоїв, обліку додатків типу сиропів чи молока, а також підтримує автоматичний розрахунок здачі та роботу з різними типами оплат, включаючи швидкі QR-платежі. Крім того, в інтерфейс закладено систему швидкої інвентаризації стаканчиків на початку та наприкінці зміни, що є класичним методом контролю персоналу в кавовому бізнесі.

Kavapp Seller

SoftWars

4,4★

Відгуків: 29

1 тис.+

Завантаження

Установити

Надсилати

Додати в список бажань

Цей додаток доступний для всіх ваших пристроїв



3+

Від 3 років

Взаємодія користувачів

[Докладніше](#)

Підтримка додатка ▾

Про цей додаток →

Kavapp Seller — застосунок для продавця магазину, кав'ярні чи закладу харчування. У робочому просторі можна швидко оформити продаж, видати чек, прийняти оплату, створити заготівлі та вести облік товарів. Застосунок є частиною системи автоматизації Kavarr й повністю синхронізований з кабінетом адміністратора. Завдяки цьому він може контролювати усі дії касира, облік запасів та керувати налаштуваннями.

Рис. 1.2. Додаток KavApp Seller

Складський облік у KavApp побудований на принципах динамічного списання сировини через калькуляційні карти напоїв та страв. Система дозволяє автоматично списувати зі складу точну вагу кави на кожну порцію еспресо, об'єми молока, вершків та навіть цукру-стіку. Окрім стандартних операцій оприбуткування продукції від постачальників та формування актів списання, у системі реалізовано механізм переміщення товарів між різними торговими точками

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

однієї мережі, що дозволяє власнику бачити актуальні залишки по всій компанії. Аналітичний блок програми автоматично збирає інформацію про кожну транзакцію та генерує щоденні звіти про виручку, прибутковість закладу, витрати на закупівлі та чистий дохід, а також дозволяє відстежувати ефективність роботи окремих співробітників за кількістю витягнутих ними чеків та середнім чеком за зміну.

Аналіз переваг KavApp показує, що система має чудову мобільну адаптацію, оскільки її інтерфейс спочатку створювався під екрани звичайних Android-смартфонів, а не лише великих планшетів, що суттєво знижує витрати підприємця на закупівлю обладнання. Також до плюсів можна віднести стабільну роботу хмарної синхронізації.

Водночас система має і суттєві мінуси, серед яких виділяється тарифна політика, де найбільш корисні інструменти аналітики та складського контролю заблоковані у безкоштовній версії та вимагають щомісячної плати за кожну підключену торгову точку. Також присутня жорстка залежність від наявності мережі Інтернет для передачі даних на сервер та оновлення залишків у кабінеті адміністратора. Останнім важливим недоліком є надмірна спеціалізація системи під кавову індустрію, що робить її складною для адаптації під інші формати бізнесу, наприклад, роздрібну торгівлю супутніми штучними товарами чи кондитерськими виробами, які не мають чітких калькуляційних карт.

Ще одним популярним і максимально універсальним рішенням, яке широко використовується на ринку України для автоматизації роздрібної торгівлі та сфери послуг, є кросплатформена хмарна система SkyService POS (рис. 1.3) [5]. Функціональні можливості цього програмного продукту охоплюють повний цикл автоматизації бізнес-процесів торгових точок, починаючи від швидкої фіксації продажів на касовому терміналі й закінчуючи глибоким управлінським та фінансовим аудитом у кабінеті керівника. Касовий модуль програми дозволяє касирам оперативно формувати чеки за допомогою зручного меню товарів, підтримує поділ замовлень, застосування знижкових купонів, роботу зі штрих-

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

кодами через камеру смартфона, а також інтегрований із програмним РРО для миттєвої реєстрації чеків у податковій службі.

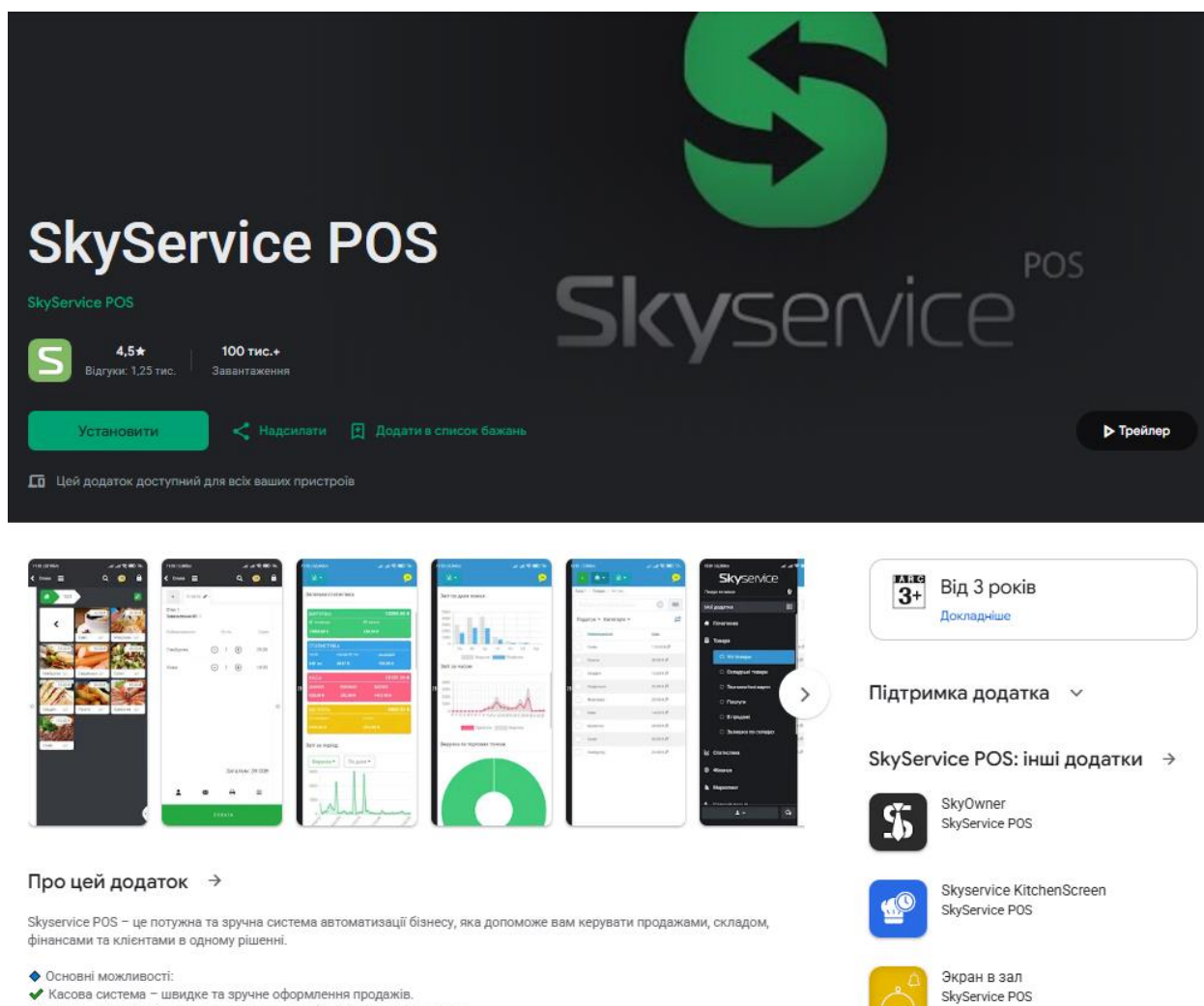


Рис. 1.3. Додаток SkyService POS

Складський облік у SkyService POS надає користувачам розширені інструменти для повного контролю залишків продукції в реальному часі. Система підтримує створення складних технологічних карт, проведення інвентаризацій без зупинки роботи торгової точки, оформлення замовлень постачальникам безпосередньо через інтерфейс програми, а також відстеження переміщень товарів між різними складами чи філіями мережі. Аналітичний блок додатка автоматично структурує дані про витрати та доходи закладу, формує звіти про рух грошових коштів, розраховує заробітну плату персоналу на основі відпрацьованих змін або

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		17

відсотка від продажів і надає власнику графіки динаміки виручки з можливістю фільтрації за годинами, днями чи категоріями товарів.

Серед ключових переваг SkyService POS варто відзначити повну кросплатформеність, оскільки додаток однаково стабільно працює на смартфонах, планшетах, комп'ютерах і навіть на спеціалізованих POS-терміналах. Важливим плюсом є наявність продуманого автономного режиму, який дозволяє касі безперебійно вибивати чеки та проводити продажі навіть при повному зникненні інтернету, зберігаючи всі транзакції в локальній пам'яті пристрою.

Основними недоліками системи є комерційна модель розповсюдження, де безкоштовна версія має жорсткі ліміти на кількість товарів та чеків, а розширений аналітичний і складський функціонал потребує щомісячної оплати, яка зростає залежно від масштабів бізнесу. Крім того, хоча локальна каса і працює без мережі, повноцінна синхронізація та відображення актуальних залишків на інших пристроях все одно критично залежать від інтернет-з'єднання. Останнім вагомим мінусом є перевантаженість інтерфейсу налаштувань через надмірну універсальність системи, що змушує користувача витратити багато часу на первинну адаптацію та конфігурацію програми під специфіку конкретної кав'ярні чи магазину.

Для українського ринку також максимально актуальною та затребуваною є система Checkbox, яка першочергово спеціалізується на програмному забезпеченні реєстраторів розрахункових операцій та миттєвій фіскалізації роздрібних продажів (рис. 1.4) [6]. Функціональне призначення цього продукту суттєво відрізняється від класичних облікових систем, оскільки його головна задача – забезпечити повну легалізацію комерційної діяльності підприємства згідно з чинним законодавством України. Функціонал системи дозволяє автоматично відкривати та закривати робочі зміни, миттєво реєструвати кожну платіжну транзакцію на серверах Державної податкової служби, формувати електронні чеки з необхідними податковими ставками, акцизами чи знижками, а також відправляти ці чеки покупцям через месенджери, SMS або у вигляді динамічних QR-кодів на екрані пристрою.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

Checkbox - програмний РРО

Checkbox LLC

4,9★
Відгуки: 4,07 тис.

50 тис.+
Завантаження

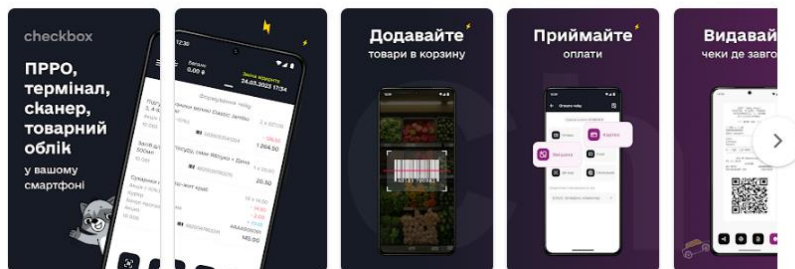
3+
Від 3 років

Установити

Надсилати

Додати в список бажань

Цей додаток доступний для деяких ваших пристроїв



Підтримка додатка

Схожі додатки

âbank24 — Банк з акцентом
ABank24
4,9★

ROZGOLOS: адвокат у кишені
SMEDIAWEB LTD
4,4★

Нова пошта
ТОВ «Нова пошта»
4,7★

Київ Цифровий
Municipal Main Information and Computing
4,6★

Про цей додаток

Checkbox – програмний РРО, створений для бізнесу будь-якого масштабу. Це сучасний сервіс, який здешевить реєстрацію розрахункових операцій, а також спростить і впорядкує ведення звітності та контроль платежів.

Checkbox – це РРО для тих, хто звик цінувати свій час, гроші і націлений на зростання прибутку бізнесу.

Застосунок Checkbox стане в пригоді, якщо вам потрібно створювати чеки далеко від комп'ютера. Це повноцінний РРО...

Рис. 1.4. Додаток CheckBox

З точки зору архітектури, Checkbox працює як через власні касові веб-інтерфейси та мобільні додатки, так і у вигляді гнучкого API, що дозволяє легко інтегрувати його інструменти фіскалізації у сторонні POS-системи, CRM чи інтернет-магазини. Що стосується складського обліку та звітності, то фінансовий блок програми обмежується генерацією обов'язкових за законом Z-звітів, X-звітів, веденням журналів продажів та базовим підрахунком виручки за зміну, що дає власнику загальне уявлення про поточний обіг коштів, але не надає інструментів для глибокого маркетингового чи операційного аналізу.

Серед ключових переваг Checkbox виділяється повна відповідність усім актуальним вимогам податкового законодавства України, регулярне оновлення системи під нові державні стандарти та надзвичайно низька вартість використання порівняно з повноцінними хмарними обліковими гігантами. Також вагомим плюсом є зручність у роботі з електронними чеками, що дозволяє бізнесу повністю відмовитися від закупівлі дорогих паперових термопринтерів і класичних залізних касових апаратів.

		Держанівський			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		19

Головним недоліком системи є її вузька орієнтованість виключно на процеси фіскалізації та реєстрації продажів, а не на комплексне управління підприємством. Програма практично не містить інструментів для повноцінного ведення складського обліку, тут неможливо створити технологічні чи калькуляційні карти для списання інгредієнтів кави, відсутні функції оприбуткування партій товарів від постачальників, проведення інвентаризацій або контролю критичних залишків на складі. Крім того, функціонал управління персоналом та глибокої бізнес-аналітики тут зведений до мінімуму, що змушує підприємців використовувати Checkbox або як додатковий плагін для інших систем обліку, або самостійно добудовувати необхідні складські модулі у власних індивідуальних розробках.

На основі проведеного аналізу можна сформулювати порівняльну таблицю (табл. 1.1).

Таблиця 1.1

Порівняльна таблиця існуючих рішень

Характеристика	Poster POS	KavApp	SkyService POS	Checkbox
1.	2.	3.	4.	5.
Простий інтерфейс	-	+	-	+
Хмарне збереження даних	+	+	+	+
Облік товарів і складу	+	+	+	-
Фіскалізація (Україна)	+	-	+	+
Аналітика продажів	+	+	+	-
Управління персоналом	+	+	+	-
Робота на мобільних пристроях	+	+	+	+

Проаналізувавши наведені програмні продукти, можна зробити висновок, що більшість сучасних POS-систем мають широкий функціонал, проте значна частина можливостей доступна лише за платною підпискою або потребує складного налаштування. Частина рішень орієнтована переважно на фіскалізацію, без реалізації повноцінного управління продажами та складським обліком.

Враховуючи результати дослідження, доцільно розробити Android-застосунок для автоматизації продажів, який поєднуватиме основний необхідний функціонал: облік товарів, оформлення продажів, контроль залишків, формування звітності та управління користувачами. Важливою особливістю застосунку має стати простота використання, сучасний інтерфейс та адаптація до потреб малого бізнесу.

Таким чином, майбутній програмний продукт повинен забезпечувати швидке оформлення продажів, зручний облік товарів, базову фінансову аналітику та стабільну роботу на мобільних Android-пристроях. Реалізація таких вимог дозволить створити ефективний інструмент автоматизації продажів для малого та середнього бізнесу.

1.3. Архітектура та укрупнена структура мобільного додатку

На етапі проєктування мобільного додатку ключовим завданням є побудова такої архітектури, яка б дозволила створити стабільний, надійний та максимально гнучкий програмний продукт. Оскільки майбутній додаток має працювати з великими обсягами товарних позицій, складськими замовленнями, фінансовими чеками та персональними обліковими даними користувачів у реальному часі, архітектурне рішення повинно гарантувати не лише високу швидкість роботи, а й надійність збереження інформації, хорошу масштабованість та зручність подальшого супроводу.

Правильно обрана архітектура дозволяє значно спростити подальший розвиток системи, підвищити загальну якість програмного коду, полегшити командну розробку та суттєво зменшити часові й фінансові витрати на внесення будь-яких змін чи додавання нових функцій у майбутньому. Без чітко спланованого каркасу процес написання коду касової системи може призвести до створення заплутаної структури, де зміна логіки в одному місці викликатиме випадкові помилки в інших частинах програми.

Для реалізації цих завдань та забезпечення чистоти кодової бази було прийнято рішення проєктувати додаток на основі відомих принципів Clean

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

Architecture у тісному поєднанні з архітектурним патерном MVVM (Model-View-ViewModel). Такий комплексний підхід дозволяє повністю розділити глобальну логіку програми на окремі незалежні шари, де кожен елемент виконує виключно свою роль і не знає про технічні деталі реалізації сусідніх компонентів [7, 8].

Основою взаємодії на рівні користувацького інтерфейсу стає шаблон MVVM (рис. 1.5), який допомагає відокремити візуальне відображення від бізнес-правил торгової точки. Компонент View проєктується абсолютно пасивним: його єдине завдання полягає в тому, щоб відповідно намалювати графічні елементи на екрані пристрою та слухати дії користувача, передаючи їх далі.

Компонент ViewModel виступає в ролі головного керуючого центра для екранів [9]. Вона акумулює поточний стан інтерфейсу, обробляє запити від View, звертається до внутрішньої логіки та проєктується таким чином, щоб бути повністю захищеною від особливостей життєвого циклу мобільної платформи [10]. Це означає, що під час повороту екрана смартфона чи планшета всі незавершені операції або введені касиром дані не будуть втрачені чи скинуті.

Зв'язок між View та ViewModel планується зробити реактивним на основі спеціальних фонових потоків даних, завдяки чому інтерфейс автоматично оновлюється в ту ж секунду, коли ViewModel отримує нову інформацію з нижчих рівнів програми. Роль третього компонента (Model), повністю перебирають на себе інші рівні архітектури, що містять у собі безпосередні бази даних та алгоритми обробки інформації.

Вся внутрішня структура проєкту концептуально ділиться на три основні архітектурні шари, кожен з яких ізольований від інших. Перший шар – представлення (Presentation Layer) – безпосередньо об'єднує в собі майбутні елементи візуального дизайну та відповідні класи ViewModel [11]. На цьому рівні на етапі проєктування закладаються інструменти для створення гнучкої та адаптивної розмітки, яка буде однаково коректно відображатися як на компактних екранах телефонів, так і на великих дисплеях планшетів. Також у шарі представлення плануються спеціальні механізми безпечного зв'язування графічних шаблонів із логічним кодом, щоб уникнути помилок через звернення до

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		22

неіснуючих елементів, та компоненти для централізованого керування переходами між екранами з можливості безпечної передачі параметрів. Весь цей шар заздалегідь розбивається на окремі бізнес-контексти торгової точки, включаючи модулі авторизації працівників, перегляду каталогу товарів, обліку продажів та проведення операцій списання чи закупівлі.

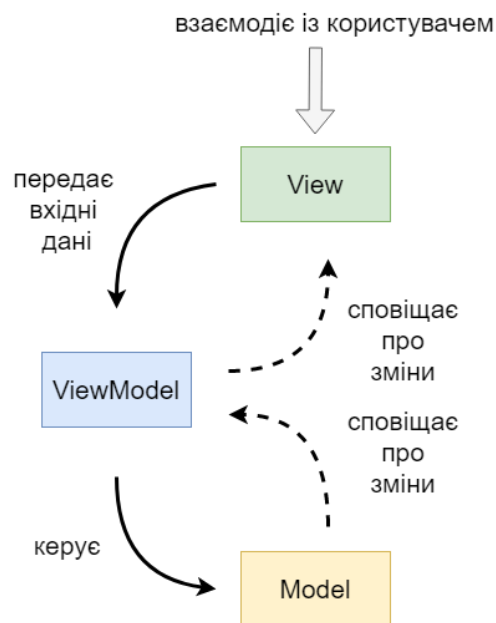


Рис. 1.5. Діаграма взаємодії між компонентами шаблону MVVM

Другий шар (Domain Layer) – є головним ядром додатка [12]. Він проектується абсолютно чистим і повністю незалежним від майбутніх баз даних, хмарних серверів чи будь-яких особливостей самої операційної системи. На цьому рівні описуються сценарії використання, тобто чіткі покрокові алгоритми та правила життєдіяльності торгової точки: як саме має проходити реєстрація нового співробітника, яким чином перевіряються залишки продукції на складі під час її актування та за якими фінансовими формулами підраховується підсумкова виручка. Тут же створюються чисті моделі сутностей та загальні інтерфейси для роботи з інформацією у вигляді паттерну Repository. Ці абстрактні інтерфейси репозиторіїв виступають у ролі суворих контрактів, які лише описують правила доступу до даних, але нічого не знають про спосіб їхнього збереження, що робить цей рівень повністю захищеним від будь-яких технічних змін.

Третій шар (Data Layer) – відповідає за збір, збереження та передачі всієї необхідної інформації додатка [13]. На цьому рівні проєктуються реальні механізми взаємодії з джерелами даних за допомогою паттерну Data Source та конкретних реалізацій репозиторіїв. Паттерн Data Source розділяється на два незалежні потоки, де перший (Local Data Source) відповідає за локальне збереження та кешування інформації через вбудовану базу даних пристрою, а другий (Remote Data Source) забезпечує зв'язок із головним віддаленим сервером через сучасну хмарну платформу, яка відповідатиме за перевірку паролів, надійне збереження медіафайлів магазину та синхронізацію даних про продажі в реальному часі через мережеві запити. Конкретна ж реалізація репозиторіїв на цьому шарі збирає ці джерела даних до купи, приховуючи від решти програми логіку вибору між локальним кешем та хмарою.

Щоб усі ці три архітектурні шари та їхні численні компоненти могли злагоджено працювати як цілісна система і при цьому не залежали жорстко один від одного, на етапі проєктування обов'язково закладається механізм автоматичного впровадження залежностей (Dependency Injection) [14]. Це архітектурне рішення дозволить розробнику централізовано керувати створенням, налаштуванням та часом життя всіх важливих об'єктів у системі, таких як модулі підключення до локальної бази даних чи клієнти для виконання мережевих запитів до хмарної бази даних.

Автоматичне впровадження залежностей виступає в ролі надійного містка, який зв'язує ізольовані шари додатка між собою на основі заздалегідь описаних абстрактних інтерфейсів. Такий підхід значно спростить майбутній процес написання коду, допоможе уникнути хаосу в налаштуваннях програми та дозволить легко проводити тестування бізнес-логіки додатка окремо від інтерфейсу чи бази даних, використовуючи тимчасову підміну реальних джерел інформації на тестові.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

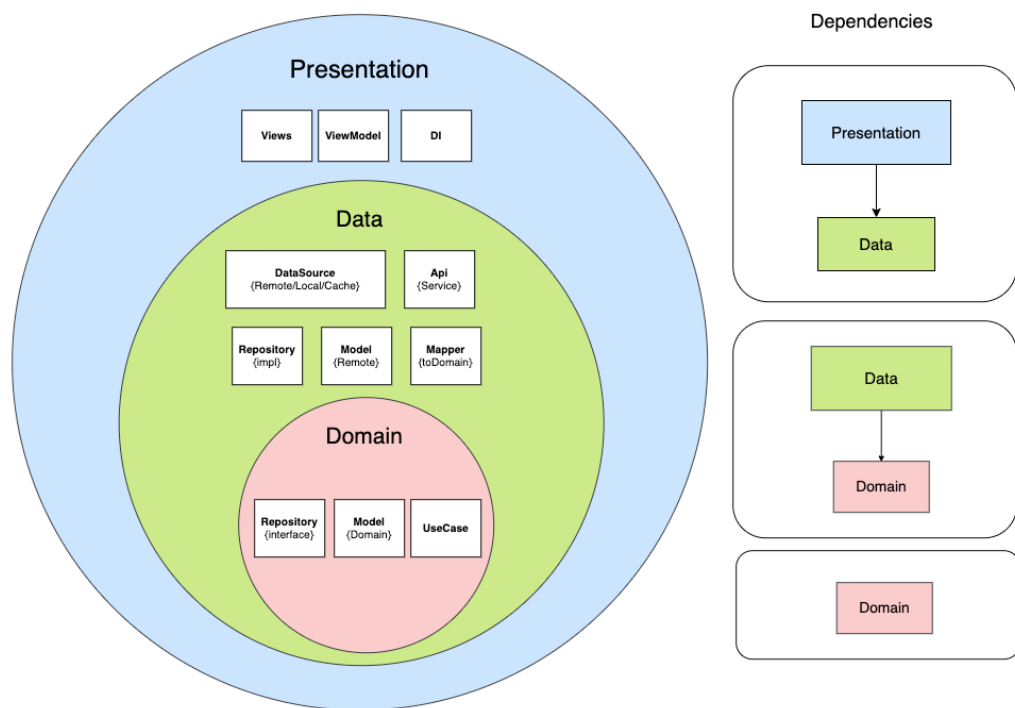


Рис. 1.6. Структура організації програми за принципами Clean Architecture

Завдяки такому комплексному проектуванню архітектури (рис 1.6), майбутній мобільний додаток буде максимально адаптивним до будь-яких зовнішніх змін: якщо в процесі роботи знадобиться повністю переробити дизайн інтерфейсу, додати нові функції або навіть перейти на зовсім інший тип серверної інфраструктури, внутрішнє ядро додатка та його головні бізнес-правила залишаться неущодженими.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

На етапі проектування будь-якого мобільного продукту, окрім проектування самої архітектури, найважливішим кроком є правильний підбір технологічного стеку. Для мобільного додатку, який має функціонувати як внутрішня система автоматизації та облік, вибір операційної системи, мови програмування та середовища розробки напряму визначає швидкість написання коду, загальну вартість запуску всього проєкту, а також стабільність і надійність роботи програми на реальних пристроях у реальних умовах щоденного використання.

Для створення системи автоматизації було обрано мобільну платформу Android. Головною причиною цього вибору є фінансова доступність та величезний

асортимент різноманітного обладнання на сучасному ринку техніки. Для комерційного бізнесу, особливо на етапі його старту та відкриття перших точок, критично важливо максимально мінімізувати початкові витрати на закупівлю дорогого обладнання. Пристрої на базі Android, такі як планшети чи смартфони, коштують значно дешевше за аналогічні продукти конкурентів, що дозволяє власнику торгової точки підібрати необхідний пристрій під будь-який наявний бюджет без втрати продуктивності системи.

Крім економічної вигоди, операційна система Android відкриває надзвичайно широкі можливості для кастомізації інтерфейсу, налаштування системи та підключення найрізноманітнішого зовнішнього периферійного обладнання. Оскільки мобільний додаток у майбутньому має активно працювати в єдиній зв'язці з касовими апаратами, принтерами для друку чеків, грошовими скриньками та дротовими чи бездротовими сканерами штрих-кодів, відкритість платформи Android та наявність великої кількості готових інструментів суттєво спрощують проєктування цих складних взаємодій.

Також під час проєктування варто враховувати простоту тестування, відлагодження програмних помилок та подальшого розгортання готової програми. Для встановлення розробленого додатку на робочі пристрої безпосередньо всередині торгової точки розробнику не обов'язково проходити складні, дорогі й тривалі перевірки у фірмових закритих магазинах додатків. Android дозволяє легко інстальювати готову систему напрямку через завантаження файлу збірки безпосередньо на цільовий планшет чи телефон, що є ідеальним і максимально зручним варіантом для локального корпоративного використання.

Офіційним, найбільш логічним та оптимізованим середовищем розробки для реалізації цього проєкту було обрано середовище Android Studio. Це спеціалізована програма, створена компанією Google спеціально для написання, тестування та проєктування Android-додатків. Вона поширюється на повністю безкоштовній основі і містить у собі повний комплекс необхідних інструментів.

Середовище розробки включає в себе все: від інтелектуального та зручного редактора коду з підказками до потужного вбудованого емулятора пристроїв. Цей

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

емулятор дозволяє детально перевірити логіку роботи програми на різних віртуальних смартфонах чи планшетах із різними роздільними здатностями екранів прямо під час проєктування, не купуючи для цього десятки реальних пристроїв. Android Studio має чудову інтеграцію з усіма сучасними інструментами аналізу коду, модульного тестування та автоматично допомагає розробнику знаходити логічні помилки чи заплутані місця у структурі проєкту ще до моменту його фінального запуску.

В якості мови програмування для додатку було обрано мову Kotlin. На сьогоднішній день вона офіційно визнана пріоритетною та базовою мовою для розробки під Android за прямими рекомендаціями компанії Google. Вибір Kotlin замість класичної, але вже застарілої мови Java обґрунтований кількома очевидними перевагами, які суттєво полегшують роботу над проєктом [15, 16].

По-перше, Kotlin має дуже лаконічний, сучасний та інтуїтивно зрозумілий синтаксис. Це дозволяє писати значно меншу кількість рядків коду для виконання тих самих технічних завдань, що автоматично підвищує читабельність проєкту, полегшує його структуру та сильно зменшує ймовірність зробити випадкову механічну помилку під час проєктування логіки програми.

По-друге, архітектура мови повністю типобезпечна завдяки суворій типізації даних та вбудованому механізму Null-Safety. Ця технологія захищає програму від найпопулярнішої помилки в програмуванні – звернення до порожніх змінних, яке зазвичай призводить до раптового аварійного завершення додатку. Оскільки всі типи даних жорстко контролюються, більшість потенційних збоїв виявляються та виправляються ще на етапі компіляції проєкту, тобто до того, як код потрапить на реальний пристрій касира. Для системи управління та обліку в магазині це має вирішальне значення, адже обробка фінансових операцій, підрахунок виручки та списання інгредієнтів повинні бути максимально точними.

По-третє, мова Kotlin ідеально пристосована для швидкої *асинхронної роботи* завдяки вбудованому та оптимізованому механізму Coroutines [17, 18]. Це є критично важливим фактором для додатку автоматизації, адже будь-які важкі внутрішні процеси – наприклад, запити до локальної бази даних, збереження

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		27

великих звітів чи тривале очікування відповіді від віддаленого сервера – повинні відбуватися непомітно у фоновому режимі. Вони не мають гальмувати роботу інтерфейсу користувача, створювати зависання екрана чи змушувати касира чекати під час обслуговування черги клієнтів.

Окрім вибору базової платформи та мови програмування, найважливішим етапом проєктування мобільного додатку є підбір спеціалізованих бібліотек та готових архітектурних компонентів. Використання перевірених технічних рішень дозволяє не писати базові механізми з нуля, а зосередитися на логіці роботи самого закладу. Для забезпечення швидкої взаємодії між шарами Clean Architecture та патерном MVVM було підібрано оптимальний набір інструментів, які гарантують стабільність системи, безпеку даних та сучасний вигляд користувацького інтерфейсу.

Для реалізації зв'язку між різними рівнями програми та автоматичного керування створенням усіх об'єктів на етапі проєктування закладається використання фреймворку Hilt [19]. Цей інструмент відповідає за впровадження залежностей (Dependency Injection) і дозволяє уникнути жорсткої прив'язки компонентів один до одного. Головною перевагою впровадження Hilt є автоматизація життєвого циклу об'єктів, завдяки чому програма самостійно вирішує, коли саме потрібно створити підключення до бази даних чи мережевого клієнта, і вчасно очищає пам'ять після завершення їхньої роботи. Це значно спрощує структуру коду, робить його чистішим і дозволяє легко замінювати технічні модулі додатка або проводити їхнє ізольоване тестування без ризику зламати інші частини системи. Окрім цього, використання Hilt зменшує кількість шаблонного коду, який розробнику довелося б писати вручну для зв'язування шарів архітектури, що суттєво пришвидшує процес створення продукту.

Оскільки додаток є багатоекранною системою з різними функціональними блоками, для організації зручних та зрозумілих переходів планується впровадження компонента Navigation [20]. Цей компонент дозволяє спроектувати логіку переміщення користувача між розділами в єдиному графічному графі. Головна перевага використання Navigation Component полягає в тому, що він

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		28

повністю бере на себе складну логіку управління стеком екранів, гарантує правильну роботу системної кнопки «Назад» та захищає додаток від поширених помилок, наприклад, від подвійного натискання на кнопку переходу, яке в класичному Android часто призводить до збоїв. Крім того, цей компонент підтримує технологію безпечної передачі аргументів, що дозволяє безпомилково передавати інформацію про вибраний товар чи користувача з одного модуля програми в інший ще на етапі компіляції коду.

Візуальна частина додатку проєктується на основі сучасних дизайн-стандартів Material Design [21, 22]. Головною перевагою використання цієї бібліотеки компонентів є наявність величезної бази вже готових, адаптованих та інтуїтивно зрозумілих елементів інтерфейсу, таких як зручні кнопки, поля введення інформації з валідацією тексту, картки товарів, списки та діалогові вікна підтвердження операцій. Це дозволяє заздалегідь закласти єдиний візуальний стиль для всієї програми, забезпечує її охайний зовнішній вигляд на будь-яких пристроях і дозволяє працівникам миттєво орієнтуватися в робочому просторі без тривалого попереднього навчання. Також Material Design має чудову вбудовану підтримку адаптивності, що спрощує налаштування відображення елементів як на маленьких смартфонах, так і на великих касових планшетах.

Для надійної роботи з інформацією безпосередньо на самому мобільному пристрої проєктується локальне сховище на базі бібліотеки Room [23]. У контексті архітектури додатка вона виконує надзвичайно важливу функцію – збереження та ведення поточного чека під час обслуговування клієнта. Головна перевага Room полягає в тому, що вона є офіційною та безпечною надбудовою над базою даних SQLite, яка автоматично перевіряє правильність SQL-запитів ще під час збірки проєкту, що повністю виключає помилки при роботі з таблицями. Бібліотека забезпечує миттєвий запис кожного доданого в кошик товару, зміну його кількості чи застосування знижки без найменших затримок в інтерфейсі. Це гарантує максимальну надійність: якщо в процесі формування замовлення телефон або планшет раптово вимкнеться чи розрядиться, поточний сформований чек не зникне

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

з пам'яті, і працівник зможе спокійно продовжити роботу з того самого місця відразу після перезапуску програми.

Для реалізації серверної частини системи використовується хмарна платформа Supabase [24]. Вона забезпечує централізоване зберігання даних, роботу з базою даних PostgreSQL. Використання Supabase дозволяє організувати роботу системи у багатокористувацькому режимі, забезпечуючи доступ до актуальної інформації з різних пристроїв одночасно. Supabase виступає центральним компонентом інфраструктури системи, забезпечуючи надійне збереження детальної інформації про товари, ведення обліку продажів та архівацію повної історії транзакцій. Через можливості платформи здійснюється оперативне управління товарними залишками, реалізуються механізми авторизації користувачів, а також накопичуються дані, необхідні для подальшого формування фінансової звітності.

Вибір цієї технології зумовлений низкою суттєвих переваг, серед яких ключове місце посідає використання PostgreSQL як перевіреної та надійної реляційної бази даних із повною підтримкою SQL-запитів. Supabase гарантує не лише централізоване зберігання всієї робочої інформації, а й забезпечує безперебійну синхронізацію даних між різними пристроями, що є критично важливим для оперативності роботи торговельного закладу. Додатковою перевагою є вбудована підтримка інструментів авторизації та гнучкого розмежування прав доступу користувачів, а загальна масштабованість платформи створює надійну базу для майбутнього розвитку та розширення функціональних можливостей розробленого програмного продукту.

Використання хмарної платформи дозволяє забезпечити збереження даних навіть у випадку втрати або пошкодження окремого пристрою, що є важливим для систем автоматизації продажів.

Для стабільної роботи системи визначено мінімальні вимоги до пристрою опис яких наведено у табл. 1.2.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		30

Таблиця 1.2

Мінімальні вимоги для пристрою

Операційна система	Android 7.0 або новіша
Зв'язок	Стабільне підключення до мережі Інтернет (Wi-Fi, 3G/4G/5G)
Оперативна пам'ять	не менше 2 ГБ
Внутрішня пам'ять	не менше 50 МБ вільного місця у пам'яті пристрою

Таким чином, використання Android, Kotlin, Supabase, Hilt, Jetpack Navigation, Room та Material Design Components дозволяє створити сучасну, надійну та масштабовану POS-систему для автоматизації продажів, яка забезпечує ефективну роботу з даними, підтримку багатокористувацького режиму та зручний інтерфейс користувача.

Висновки до першого розділу

У процесі дослідження та проєктування мобільної POS-системи для автоматизації продажів було обґрунтовано вибір технологій та інструментальних засобів розробки. Проведений аналіз предметної області засвідчив доцільність використання мобільного Android-застосунку як основного клієнтського компонента системи, що забезпечує зручність використання, мобільність та швидкий доступ до функціоналу автоматизації продажів.

Для розробки програмного комплексу було обрано мову програмування Kotlin, яка забезпечує сучасний підхід до створення Android-застосунків завдяки високій продуктивності, строгій типізації, підтримці асинхронного програмування та зручності підтримки програмного коду. Використання Android SDK дозволяє реалізувати необхідний функціонал мобільного застосунку та забезпечити його сумісність із сучасними версіями операційної системи Android.

Серверна частина системи реалізована на базі платформи Supabase, яка забезпечує централізоване зберігання даних, авторизацію користувачів та синхронізацію інформації між клієнтським застосунком і базою даних. Для

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		31

зберігання інформації використовується база даних PostgreSQL, що забезпечує надійність, цілісність та ефективну обробку даних системи.

Додатково використання Hilt спрощує управління залежностями між компонентами застосунку, Jetpack Navigation забезпечує організацію навігації між екранами, а Material Design Components дозволяють створити сучасний, адаптивний та інтуїтивно зрозумілий інтерфейс користувача. Важливу роль на рівні збереження даних відіграє бібліотека Room, яка проєктується для надійної роботи з локальною базою даних та забезпечує миттєве ведення й захист інформації поточного чека.

Таким чином, результати виконаної роботи підтверджують обґрунтованість обраного технологічного стеку для розробки мобільної POS-системи. Запропоноване рішення забезпечує високу продуктивність, надійність зберігання даних, зручність використання та можливість подальшого розвитку функціональності системи.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		32

РОЗДІЛ 2. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Процес проєктування системи автоматизації починається з детального вивчення процесів торгівлі та опису всіх технічних завдань. Для мобільного застосунку, цей етап є найважливішим кроком, адже саме тут вирішується, як саме працівник здійснюватиме продажі, додаватиме позиції до чека, а власник контролюватиме залишки продукції. Якщо чітко розписати всі можливості програми та її обмеження ще до початку написання коду, це дозволить уникнути переписування архітектури в майбутньому і допоможе створити зручний інструмент, який пришвидшить обслуговування покупців. Такий аналіз допомагає об'єднати всі задуми у чіткий план розробки, перетворити їх на технічні правила та визначити, які функції програми потрібно зробити в першу чергу.

Враховуючи специфіку роботи системи, було виділено три ролі користувачів системи :

- 1) Власник – користувач, який створює майбутню торгову точку та має повний доступ до системи, включаючи управління товарами, користувачами, складом та звітністю;
- 2) Адміністратор – користувач, який має повний доступ до системи, включаючи управління товарами, користувачами, складом та звітністю;
- 3) Співробітник – користувач, відповідальний за оформлення продажів та роботу з чеками.

Зафіксуємо функціональні та нефункціональні вимоги до програмного комплексу (табл. 2.1 та 2.2).

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		33

Функціональні вимоги

Модуль	Вимоги
1.	2.
Модуль управління товарами	– Додавання нових товарів (назва, ціна, одиниці вимірювання, опис). – Редагування та архівування товарних позицій. – Відображення списку товарів
Модуль продажів (POS-каса)	– Створення нового чека продажу. – Додавання товарів до чека. – Автоматичний розрахунок загальної вартості покупки. – Закриття чека з фіксацією продажу.
Модуль складського обліку	– Збереження інформації про залишки товарів. – Автоматичне зменшення залишків після продажу. – Можливість додавання товарів після закупівлі. – Списання бракованого або пошкодженого товару. – Перегляд історії закупок та списання.
Модуль фінансової звітності	– Формування звітів за день. – Відображення статистики продажів та загальної виручки. – Експорт звітів у файл (XLSX).
Модуль управління користувачами	– Додавання та редагування користувачів системи. – Розмежування ролей (адміністратор, касир).
Модуль збереження даних	– Збереження та синхронізація даних у хмарній базі даних Supabase. – Автоматичне оновлення інформації у реальному часі при внесенні змін. – Забезпечення доступу до даних з різних пристроїв користувача. – Забезпечення цілісності даних та коректної синхронізації після повторного підключення до системи.

Нефункціональні вимоги

Модуль	Вимоги
1.	2.
Інтерфейс користувача	– Інтуїтивно зрозумілий та сучасний інтерфейс користувача. – Реалізація інтерфейсу за допомогою XML розмітки. – Швидка та зручна навігація між екранами.
Продуктивність	– Швидка обробка операцій продажу. – Стабільна робота при великій кількості товарів та транзакцій
Сумісність	– Підтримка Android 7.0 та новіших версій. – Коректна робота на смартфонах і планшетах різних розмірів
Масштабованість	– Можливість розширення функціональних можливостей застосунку без зміни основної архітектури системи. – Інтеграція додаткових модулів і сервісів, зокрема платіжних систем та інструментів аналітики. – Підтримка подальшого розвитку системи відповідно до потреб підприємства.
Надійність та безпека	– Обмеження доступу до адміністративних функцій. – Захист даних користувачів та транзакцій.

Для точного проєктування логіки взаємодії користувачів із POS-системою на основі сформованих вимог розробляються специфікації основних варіантів використання. Кожен такий сценарій описує послідовність дій, яку виконує працівник для досягнення конкретного результату в додатку.

Діаграма варіантів використання представляє візуальну взаємодію між системою та користувачами (рис. 2.1)

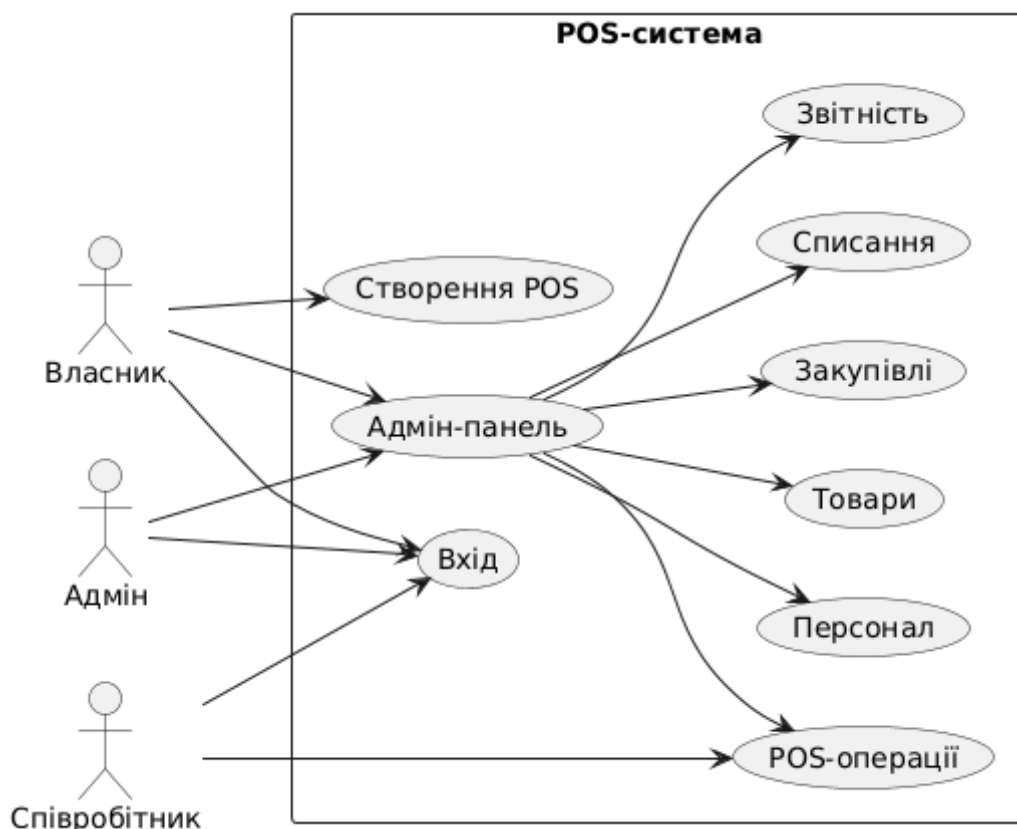


Рис. 2.1 – Діаграма варіантів використання

Для детального проєктування системи необхідно чітко розписати кожен прецедент взаємодії працівників із програмою. Першочерговим етапом взаємодії будь-якого користувача з мобільною POS-системою є ідентифікація його особи та визначення рівня прав доступу. Це дозволяє захистити комерційні дані підприємства від несанкціонованого втручання. Специфікація прецеденту «Авторизація» наведена у табл. 2.3.

Таблиця 2.3

Прецедент «Авторизація»

Актори	Всі користувачі
1.	2.
Мета	Отримання безпечного доступу до даних конкретного закладу
Передумови	Наявність облікового запису в системі
Опис сценарію	Користувач вводить email та пароль; система перевіряє дані
Результат	Відкриття екрана з доступом до функцій згідно з роллю

Після успішного входу в систему з правами максимального рівня доступу, керівник закладу отримує можливість формувати штат співробітників торгових точок. Для організації кадрового обліку та фіксації відповідальності за касові зміни розроблено відповідний функціонал, опис якого представлено у табл. 2.4.

Таблиця 2.4

Прецедент «Створення облікового запису працівника»

Актори	Адміністратор
Мета	Реєстрація нового співробітника в системі.
Передумови	Адміністратор авторизований.
Опис сценарію	Адміністратор вводить дані нового користувача в формі, вказує роль та автоматично прив'язує його до поточної торгової точки.
Результат	Створено новий запис у таблиці користувачів; працівник може увійти в систему

Функціонування будь-якого торгового майданчика неможливе без наповнення номенклатури товарів та послуг. Процес формування цифрового меню закладу, який включає заповнення карток позицій, встановлення цін та завантаження медіафайлів, деталізовано в табл. 2.5.

Таблиця 2.5

Прецедент «Додавання/редагування товару»

Актори	Адміністратор
Мета	Введення нових позицій у меню або зміна характеристик існуючих
Передумови	Наявність даних про товар (назва, ціна, одиниці виміру).
Опис сценарію	Користувач заповнює картку товару. Якщо додається фото, воно стискається. Дані синхронізуються з хмарою.
Результат	Товар з'являється в загальному списку та стає доступним для продажу

У процесі комерційної діяльності певні позиції меню можуть ставати неактуальними або виводитися з асортименту. Щоб не перевантажувати інтерфейс касира, але водночас зберегти цілісність історичних звітів, реалізовано механізм деактивації, описаний у табл. 2.6.

Таблиця 2.6

Прецедент «Архівування товару»

Актори	Адміністратор
Мета	Приховати товар, який більше не продається, зберігаючи історію операцій з ним
Передумови	Товар існує в базі даних
Опис сценарію	Адміністратор встановлює відповідний прапорець у налаштуваннях товару
Результат	Товар зникає з вибору для продажу, але залишається в звітах та історії

Однією з ключових вимог до сучасних хмарних систем є забезпечення актуальності даних на всіх підключених пристроях одночасно. Завдяки використанню технології WebSocket-прослуховування у Supabase, користувачі завжди володіють оперативною інформацією, як це показано в специфікації сценарію у табл. 2.7.

Таблиця 2.7

Прецедент «Моніторинг залишків у реальному часі»

Актори	Всі користувачі
Мета	Оперативний контроль кількості інгредієнтів/товарів
Передумови	Активне інтернет-з'єднання
Опис сценарію	Завдяки прослуховуванням змін Supabase, будь-які зміни залишків на інших пристроях миттєво відображаються в списку
Результат	Користувач бачить актуальну кількість без перезавантаження сторінки.

Головним операційним процесом, який генерує прибуток закладу та виконується безпосередньо на касовому терміналі Android, є обслуговування покупця. Логіка формування чека, списання позицій та фіксації транзакції відображена у табл. 2.8.

Таблиця 2.8

Прецедент «Продаж товарів»

Актори	Всі користувачі
Мета	Оформлення замовлення клієнта та фіксація доходу
Передумови	Наявність товарів у кошику
Опис сценарію	Працівник додає позиції до чеку, обирає тип оплати. Система створює об'єкт Receipt та пов'язані з ним записи ProductSold.
Результат	Створено чек; залишки проданих товарів автоматично списані; дані додані до фінансового звіту

Окрім прямих продажів, на складі закладу постійно виникають нетоварні витрати, пов'язані з сировинними втратами, псуванням чи дегустаціями. Для точного відображення реального стану складу адміністратор використовує інструмент, специфікацію якого наведено у табл. 2.9.

Таблиця 2.9

Прецедент «Списання товарів»

Актори	Адміністратор
Мета	Облік нетоварних витрат (брак, дегустація, псування).
Передумови	Виявлення факту необхідності списання
Опис сценарію	Користувач створює списання, додає деталі (що і скільки списано)
Результат	Кількість товару в системі зменшена; операція зафіксована в журналі списань

Для підтримання безперебійної роботи торгової точки необхідне своєчасне поповнення товарних запасів та реєстрація поставок від контрагентів. Сценарій

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		39

взаємодії користувача із системою під час оприбуткування нових партій продукції описано в табл. 2.10.

Таблиця 2.10

Прецедент «Закупівля товарів»

Актори	Адміністратор
Мета	Облік нових надходжень товарів
Передумови	Виявлення факту необхідності закупівлі
Опис сценарію	Користувач створює закупівлю, додає деталі (що і скільки закуплено)
Результат	Кількість товару в системі збільшена; операція зафіксована в журналі закупівель

Управлінський контроль вимагає регулярного аудиту дій персоналу та перевірки коректності проведених розрахунків протягом робочого дня. Функціонал, який дозволяє адміністратору детально вивчити кожну касову транзакцію, представлено в табл. 2.11.

Таблиця 2.11

Прецедент «Перегляд історії операцій та чеків»

Актори	Адміністратор
Мета	Контроль здійснених транзакцій за день
Передумови	Наявність проведених операцій у системі
Опис сценарію	Користувач відкриває екран звітності. Система підтягує дані за обраний користувачем день
Результат	Відображення списку всіх дій із можливістю перегляду деталізації кожного чека

Завершальним етапом аналітичного циклу в межах POS-системи є підготовка даних для подальшої обробки у зовнішніх програмах або передачі бухгалтеру. Процес конвертації накопиченої фінансової статистики у загальноприйнятий формат таблиць наведено в табл. 2.12.

Прецедент «Експорт звітів у Excel»

Актори	Адміністратор
Мета	Підготовка даних для зовнішньої бухгалтерії або глибокого аналізу
Передумови	Наявність проведених операцій у системі
Опис сценарію	Адміністратор ініціює експорт. Система формує таблицю .xlsx
Результат	Файл звіту готовий до відправки або збереження на пристрої

Після детального аналізу функціональних вимог та опису основних сценаріїв використання програми, необхідно спроектувати внутрішню структуру даних застосунку. Для цього створюється об'єктно-орієнтована модель, яка відображає реальні сутності торгової точки у вигляді програмних об'єктів, їхніх внутрішніх параметрів та зв'язків між ними.

Головним центром усієї структури виступає торгова точка, яка безпосередньо пов'язана з користувачами системи, сформованими чеками та наявними товарами. Програма чітко розмежовує права доступу працівників завдяки фіксації їхніх ролей, а також веде точний облік кожної фінансової та складської операції. Оскільки ціни на товари або їхні назви з часом можуть змінюватися, архітектура додатку заздалегідь проектується так, щоб у момент продажу або списання створювався окремий незалежний зріз даних. Це дозволяє зберігати історію торгової точки в первісному вигляді, навіть якщо картку самого товару пізніше буде відредаговано або видалено в архів.

Для детального відображення того, як саме влаштовані ці сутності, які типи даних вони використовують і як взаємодіють між собою в коді програми, було розроблено загальну UML-діаграму класів. Повна схема цієї структури з усіма полями, переліками та логічними зв'язками наведена на рисунку 2.2.

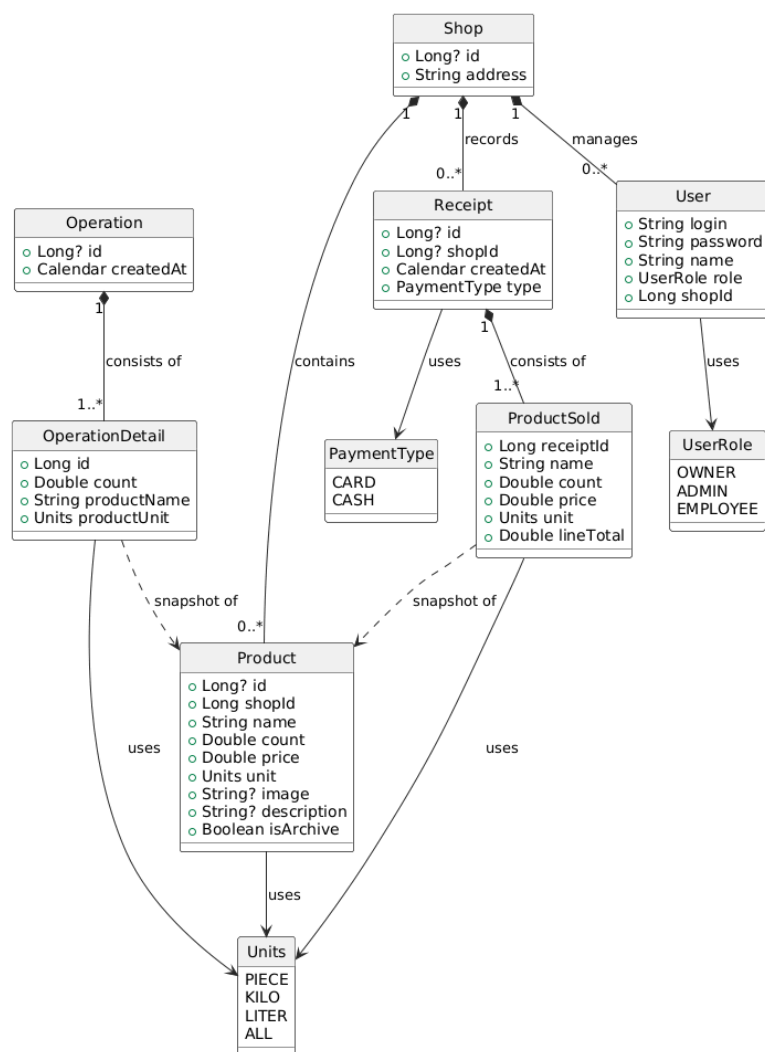


Рис. 2.2 – Діаграма сутностей додатку

2.2. Розробка бази даних системи на базі хмарного сервісу Supabase

Для забезпечення стабільної роботи POS-системи, централізованого збереження інформації та підтримки багатокористувацького режиму було розроблено структуру бази даних на основі реляційної СУБД PostgreSQL, що використовується у складі хмарної платформи Supabase. Використання реляційної моделі даних дозволяє забезпечити цілісність інформації, підтримку зв'язків між сутностями системи та ефективне виконання SQL-запитів.

База даних системи призначена для комплексного зберігання відомостей про торгові точки, обліку операцій із закупівлі та списання продукції, а також детальної фіксації кожного факту продажу. Крім того, вона забезпечує ефективне управління профілями користувачів і їхніми правами доступу та виступає основним джерелом

даних для генерації аналітичної та фінансової звітності закладу. Структура бази даних побудована за принципом нормалізації даних, що дозволяє уникнути дублювання інформації та забезпечує зручність подальшого масштабування системи.

Основою інформаційної моделі системи є сукупність взаємопов'язаних сутностей, що забезпечують повний цикл обліку діяльності закладу. До базових компонентів належать торгові точки (shops), які містять дані про місця реалізації, користувачі (users) для керування персоналом та безпосередньо товари (products), що формують каталог продукції. Складські операції відображені через сутності закупівель (procurements) та списань (write_offs), кожна з яких має пов'язані таблиці для деталізації конкретних позицій – в закуплені продукти (procurement_products) та списані продукти (writeoff_products) відповідно. Фіксація фінансових результатів здійснюється через сутність продажів (receipt), що зберігає загальну інформацію про видані чеки, та продані продукти (products_sold), яка дозволяє відстежувати детальний перелік та кількість реалізованого асортименту в межах кожної транзакції.

Центральною сутністю розробленої бази даних є таблиця shops, яка призначена для збереження інформації про торгові точки системи (табл. 2.13). Кожен запис у таблиці відповідає окремій торговій точці та містить його адресу. Саме таблиця shops виступає основою для організації зв'язків між іншими сутностями системи, оскільки всі операційні та довідкові дані прив'язуються до конкретної торгової точки через зовнішній ключ shop_id. З таблицею shops реалізовано зв'язки типу «один до багатьох» (1:N), оскільки одна торгова точка може містити декілька користувачів, товарів, продажів, документів закупівлі та списання.

Таблиця 2.13

Структура полів таблиці shops

Назва поля	Тип	Дані
id	Числовий	id торгової точки
address	Текстовий	Адреса торгової точки

Таблиця users призначена для зберігання персональних даних працівників закладу, їхніх облікових параметрів для входу в систему та визначення рівня прав доступу (табл. 2.14). Вона дозволяє ідентифікувати кожного користувача та закріпити його за конкретним підрозділом для коректного ведення персональної звітності та гарантування безпеки даних.

Таблиця users пов'язана з таблицею shops через поле shop_id. Даний зв'язок забезпечує належність користувача до конкретної торгової точки та дозволяє реалізувати механізм розмежування доступу в межах магазину.

Таблиця 2.14

Структура полів таблиці users

Назва поля	Тип	Дані
1	2	3.
login	Текстовий	Логін користувача
password	Текстовий	Пароль користувача
name	Текстовий	ПІБ користувача
shop_id	Числовий	id торгової точки
role	Текстовий	Роль користувача

Таблиця products призначена для систематизації та зберігання відомостей про весь асортимент товарів, що представлені в торговельній точці (табл. 2.15). Вона виступає базовим джерелом інформації для формування каталогу товарів у робочій зоні касира та забезпечує можливість детального відстеження стану, вартості та характеристик кожної номенклатурної одиниці.

Таблиця products пов'язана з таблицею shops зв'язком типу «один до багатьох». Кожен товар належить певному магазину та містить інформацію про назву, кількість, ціну, одиницю виміру, опис та інші характеристики.

Таблиця 2.15

Структура полів таблиці products

Назва поля	Тип	Дані
id	Числовий	id товару
shop_id	Числовий	id торгової точки

name	Текстовий	Найменування товару
count	Числовий	Наявна кількість товару
image	Текстовий	Посилання на фото товару
price	Числовий	Ціна товару
description	Текстовий	Опис товару
unit	Текстовий	Одиниця вимірювання товару
is_archive	Логічний	Недоступність товару

Таблиця procurements слугує журналом реєстрації всіх надходжень товарів до магазину (табл. 2.16). Вона призначена для створення документів закупівлі, що дозволяє фіксувати час оприбуткування продукції на склад. Завдяки цій таблиці в системі зберігається історія поповнення запасів, що є необхідним для контролю товарного залишку та подальшого звітування про роботу торгової точки. Також дана таблиця пов'язана з таблицею shops за допомогою поля shop_id, що забезпечує чітку прив'язку кожної операції закупівлі до конкретної торгової точки.

Таблиця 2.16

Структура полів таблиці procurements

Назва поля	Тип	Дані
id	Числовий	id закупівлі
created_at	Дата і час	Дата створення закупівлі
shop_id	Числовий	id торгової точки

Оскільки одна закупівля може включати декілька товарів, а один товар може бути присутнім у різних закупівлях, між таблицями procurements та products реалізовано зв'язок типу «багато до багатьох» (M:N). Для цього використовується проміжна таблиця procurement_products, яка містить зовнішні ключі procurement_id та product_id, а також кількість товару (табл. 2.17).

Таблиця 2.17

Структура полів таблиці procurement_products

Назва поля	Тип	Дані
product_id	Числовий	id товару
procurement_id	Числовий	id закупівлі
count	Числовий	Кількість доданого товару

Таблиця write_offs виступає журналом реєстрації операцій зі списання товарів зі складу закладу (табл. 2.18). Вона призначена для фіксації випадків псування, закінчення терміну придатності або іншого вибуття продукції, що не пов'язане з процесом продажу.

Сутність write_offs пов'язана з таблицею shops через ідентифікатор shop_id, для відстеження операцій списання в розрізі кожної окремої торгової точки.

Таблиця 2.18

Структура полів таблиці write_offs

Назва поля	Тип	Дані
id	Числовий	id списання
created_at	Дата і час	Дата створення списання
shop_id	Числовий	id торгової точки

Оскільки одне списання може включати декілька товарів, а один товар може бути присутнім у різних списаннях, між таблицями write_offs та products реалізовано зв'язок типу «багато до багатьох» (M:N). Для цього використовується проміжна таблиця writeoff_products, яка містить зовнішні ключі procurement_id та product_id, а також кількість товару (табл. 2.19).

Таблиця 2.19

Структура полів таблиці writeoff_products

Назва поля	Тип	Дані
product_id	Числовий	id товару
write_off_id	Числовий	id закупівлі
count	Числовий	кількість прибраного товару

Для реалізації операцій продажу використовується таблиця `receipt`, яка містить інформацію про чеки продажів (табл. 2.20). Кожен чек належить конкретній торговій точці та містить дату створення і тип оплати. Між таблицями `receipt` та `products_sold` реалізовано зв'язок типу «один до багатьох», оскільки один чек може містити декілька товарних позицій.

Таблиця 2.20

Структура полів таблиці `receipt`

Назва поля	Тип	Дані
<code>id</code>	Числовий	id продажі
<code>created_at</code>	Дата і час	Дата створення продажі
<code>shop_id</code>	Числовий	id торгової точки
<code>payment_type</code>	Текстовий	тип оплати продажі

Таблиця `products_sold` використовується для збереження інформації про товари, що були продані в межах певного чека (табл. 2.21). У таблиці зберігаються назва товару, кількість, ціна та одиниця виміру на момент продажу. Такий підхід дозволяє зберігати історичні дані про продажі навіть у випадку зміни інформації про товар у таблиці `products`.

Таблиця 2.21

Структура полів таблиці `products_sold`

Назва поля	Тип	Дані
<code>receipt_id</code>	Числовий	id продажі
<code>name</code>	Текстовий	Найменування проданого товару
<code>count</code>	Числовий	Кількість проданого товару
<code>price</code>	Числовий	Ціна проданого товару
<code>unit</code>	Текстовий	Одиниця вимірювання проданого товару

Використання зовнішніх ключів забезпечує цілісність даних та коректність взаємозв'язків між таблицями. Розроблена структура бази даних дозволяє ефективно організувати зберігання інформації, забезпечити підтримку багатокористувацького режиму роботи та реалізувати основні функції POS-

системи, пов'язані з продажами, складським обліком і формуванням звітності. На рис. 2.3 наведено ER-діаграму бази даних системи.

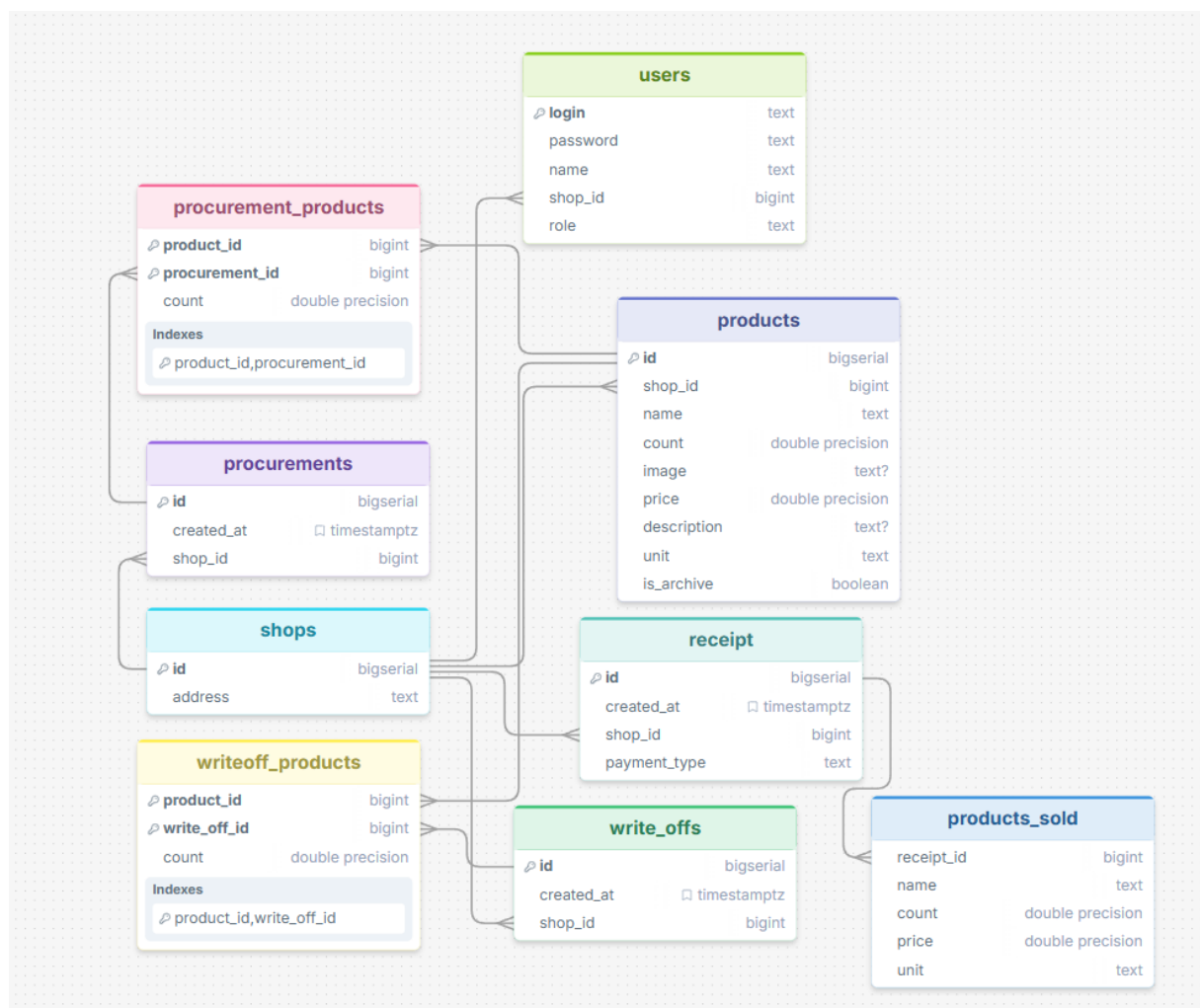


Рис. 2.3 – ER-діаграми бази даних системи

2.3. Проектування та реалізація алгоритмів роботи системи

В основі розробки мобільної POS-системи лежить проектування алгоритмів, що забезпечують автоматизацію ключових процесів торгівлі та складського обліку. Цей етап дозволяє перейти від статичного опису структури даних до моделювання динамічної поведінки системи, визначаючи чітку послідовність дій програмних компонентів у відповідь на запити користувача.

Для формалізації логіки функціонування застосовано графічне моделювання, зокрема UML-діаграми послідовності. Це дає змогу детально описати процеси обміну даними між інтерфейсом користувача, бізнес-логікою та зовнішніми

сервісами, що є критично важливим для забезпечення стабільної роботи системи в умовах реального закладу. Такий підхід гарантує інженерну керованість розробки, дозволяє мінімізувати помилки при проведенні транзакцій та забезпечити цілісність інформації. Особлива увага приділена алгоритмам реалізації продажів, складських рухів та синхронізації даних у реальному часі, що є фундаментальними для ефективного управління сучасним підприємством сфери торгівлі чи послуг.

Основним функціональним алгоритмом POS-системи є процес оформлення продажу, що забезпечує перетворення вибраних товарів у фіскальну операцію. Цей процес вимагає координації між локальним станом активного кошика та віддаленою базою даних для фіксації доходу й актуалізації складських залишків (рис. 2.4).

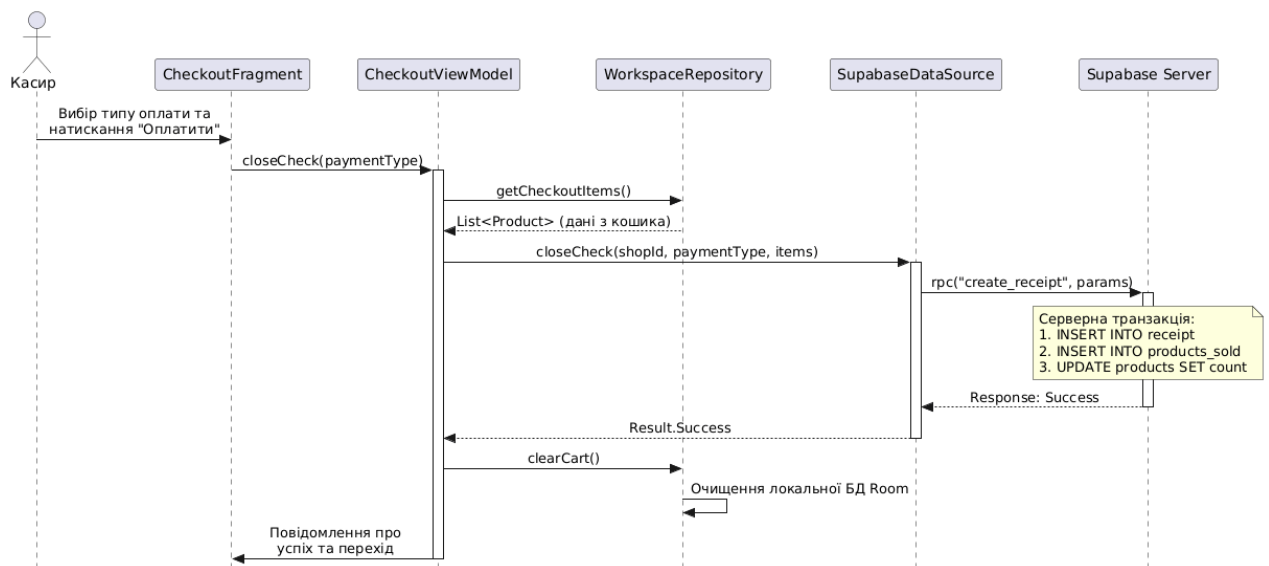


Рис. 2.4 – Діаграма алгоритму продажів

Алгоритм оформлення замовлення починається з моменту ініціалізації транзакції, коли користувач у вікні оформлення обирає бажаний тип оплати та підтверджує завершення чека. На етапі збору та підготовки даних CheckoutViewModel звертається до репозиторію для отримання повного переліку об'єктів у кошику, формуючи структуру з ідентифікаторами товарів, їхньою кількістю та актуальними цінами на момент продажу. Далі через рівень абстракції SupabaseDataSource здійснюється мережевий запит із викликом віддаленої процедури create_receipt, куди передаються параметри закладу, тип платежу та масив проданих товарів. Детальний код методу наведено в додатку А.1.

Серверна обробка цього запиту виконується в хмарному сховищі як атомарна транзакція, що забезпечує одночасне створення запису в таблиці чеків, деталізацію позицій у таблиці проданих товарів та автоматичне коригування залишків у головній таблиці продукції. Після отримання підтвердження про успішне збереження даних додаток виконує фіналізацію локального стану, яка полягає в очищенні кошика в базі даних Room та поверненні користувача до головного екрана для роботи з наступним замовленням.

Реалізація цього алгоритму через серверні функції (RPC) дозволяє мінімізувати кількість мережових запитів та гарантує, що дані не будуть пошкоджені у разі раптового зриву з'єднання під час проведення оплати. Використання реактивних потоків даних забезпечує миттєве оновлення інтерфейсу на всіх пов'язаних пристроях закладу після завершення транзакції.

Процес складського обліку в POS-системі реалізований через алгоритм реєстрації операцій приходу (закупівлі) та видатку (списання). Ключовою вимогою до цього алгоритму є забезпечення суворой відповідності між зафіксованою подією та фактичною кількістю товарів на складі. Логіка побудована за транзакційним принципом, що запобігає виникненню розбіжностей у даних при коригуванні залишків (рис 2.5).

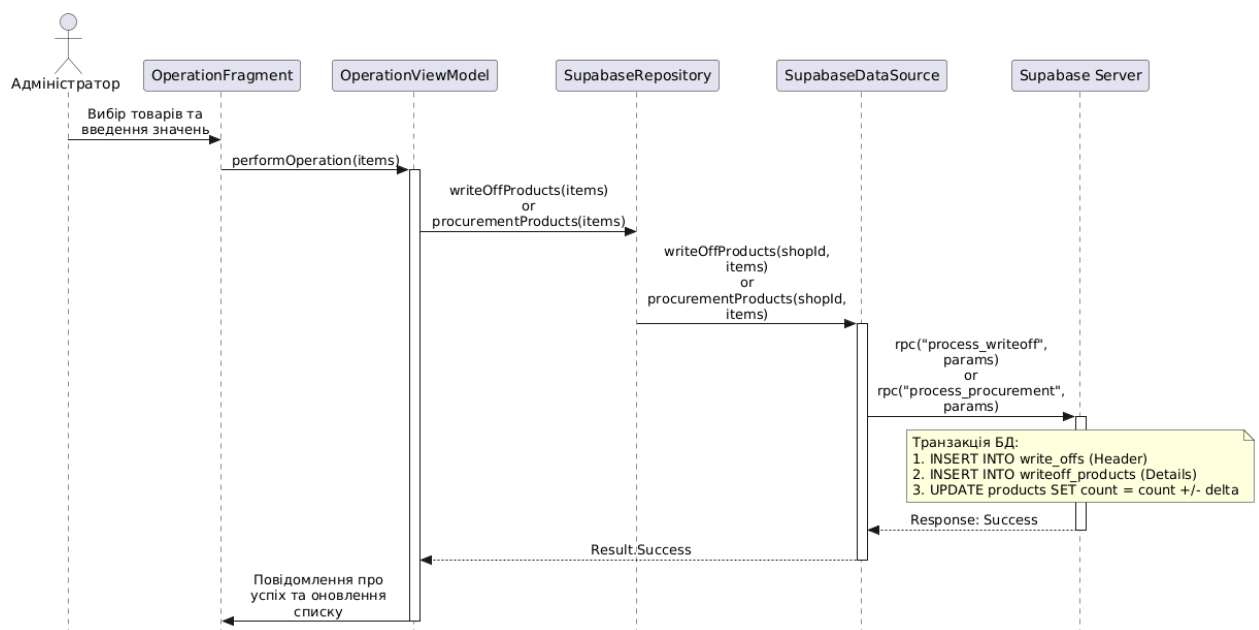


Рис. 2.5 – Діаграма алгоритму управління складськими операціями

Процес виконання алгоритму складських операцій розпочинається з етапу формування списку змін, під час якого користувач у розділі OperationFragment обирає тип операції та перелік товарів, а система створює масив об'єктів ProductCount із відповідними ідентифікаторами та обсягами коригувань. На етапі обробки бізнес-логіки OperationViewModel ініціює виклик через доменний шар, де здійснюється перевірка коректності введених даних, зокрема контроль неможливості списання від'ємної кількості, після чого запит передається до репозиторію.

Далі виконується транзакція у хмарі, де репозиторій через механізм RPC звертається до серверних функцій process_procurement або process_writeoff, що забезпечує виконання всієї логіки оновлення таблиць на стороні PostgreSQL як однієї неподільної дії (код методів наведено в додатках А.2 та А.3). Під час оновлення бази даних на сервері одночасно створюється заголовок операції в журналі, додаються записи з деталізацією по кожному товару та вносяться зміни в поточну кількість продукції в основному каталозі. Завершується алгоритм синхронізацією результатів, коли після отримання відповіді додаток сповіщає користувача про успіх, а нові значення залишків автоматично оновлюються в інтерфейсі через механізм реактивних підписок.

Такий алгоритм гарантує надійність обліку: навіть у разі критичної помилки або втрати зв'язку під час запиту, база даних залишиться у цілісному стані – або всі зміни будуть застосовані разом, або жодна з них не вплине на залишки. Це унеможливорює ситуації «загублених» товарів або некоректних звітів.

Для забезпечення актуальності інформації в умовах одночасної роботи декількох користувачів (наприклад, декількох касових зон або терміналів адміністратора), у системі реалізовано алгоритм реактивної синхронізації. Його особливістю є стратегія вибіркового оновлення, яка дозволяє відстежувати зміни лише для тих об'єктів, що є критичними для поточної сесії користувача, мінімізуючи навантаження на мережу та акумулятор пристрою (рис 2.6).

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

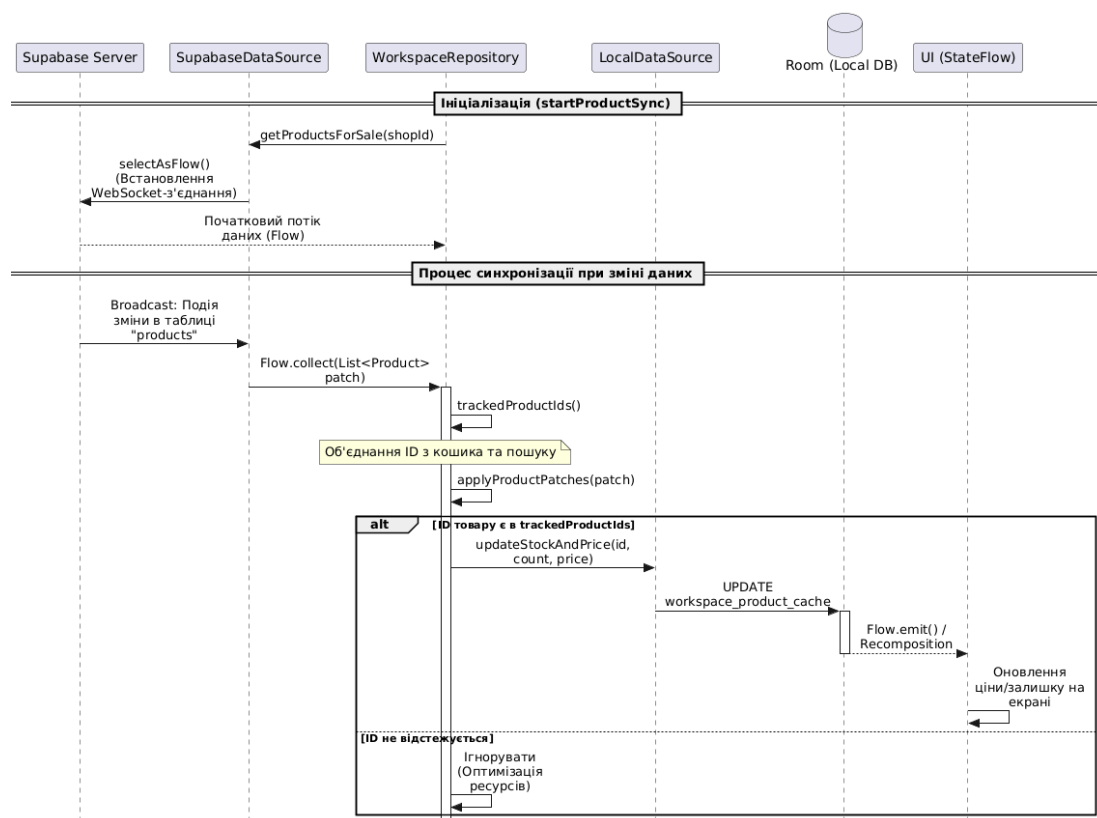


Рис. 2.6 – Діаграма алгоритму вибіркової синхронізації даних у реальному часі

Процес реактивного оновлення даних розпочинається з ініціалізації реактивного каналу під час запуску робочого простору, коли репозиторій через SupabaseDataSource активує підписку на таблицю товарів за допомогою методу selectAsFlow, встановлюючи постійне WebSocket-з'єднання з сервером [25]. У процесі моніторингу сервер Supabase автоматично відстежує будь-які зміни в базі даних (INSERT, UPDATE чи DELETE) та при зміні залишку чи ціни товару іншим користувачем миттєво надсилає відповідне повідомлення підключеним клієнтам.

Отриманий потік даних обробляється в методі applyProductPatches, де додаток виконує фільтрацію «патчів», перевіряючи, чи належить оновлений товар до списку активних позицій, що наразі знаходяться у кошику або відображені в поточних результатах пошуку. У разі підтвердження актуальності товару система здійснює точкове оновлення в локальній базі даних Room, що гарантує збереження цілісності інформації та забезпечує миттєвий доступ до неї. Завершальним етапом є реактивне оновлення інтерфейсу: завдяки використанню Kotlin Flow та StateFlow зміна даних у локальному сховищі автоматично ініціює нову емісію стану. Це дозволяє компонентам інтерфейсу через ViewModel отримувати актуальні дані та

автоматично оновлювати ціни чи залишки товарів без додаткових дій з боку користувача, при цьому StateFlow забезпечує надійне збереження та поширення поточного стану між усіма підписниками [26, 27, 28].

Такий підхід поєднує переваги хмарних Realtime-технологій та локального кешування. Користувач завжди бачить актуальну інформацію про ті товари, які він збирається продати, при цьому додаток залишається швидким, оскільки не намагається синхронізувати весь каталог товарів закладу одночасно.

2.4. Реалізація мобільного додатку системи

Реалізацію додатку найкраще розглядати не просто як набір файлів, а як живий механізм, де кожна деталь має своє чітке місце. В основі всього лежить ідея чистої архітектури, яка на перший погляд здається складною, але на практиці вона просто дозволяє запобігти хаотичному нагромадженню логіки в кодовій базі. Щоб зрозуміти, як це все працює в реальності, можна простежити повний шлях однієї дії – наприклад, коли касир шукає товар та додає її в чек у робочій зоні, яку названо Workspace та розглянемо надалі.

Реалізація взаємодії між інтерфейсом користувача та логікою представлення у модулі робочої зони касира побудована на чіткому розподілі функцій між класом WorkspaceFragment та відповідною моделлю представлення WorkspaceViewModel. Цей процес можна описати як складну систему обміну сигналами та станами, де кожен компонент суворо дотримується свого життєвого циклу.

Робота починається у WorkspaceFragment, головним завданням якого є ініціалізація візуальних компонентів та встановлення каналів зв'язку з логікою. На відміну від застарілих підходів, де фрагмент міг самостійно керувати даними, у даній реалізації він виступає лише в ролі спостерігача. Для забезпечення стабільної роботи інтерфейсу під час отримання оновлень використовується механізм viewLifecycleOwner. Це технічне рішення дозволяє фрагменту підписуватися на потоки даних від ViewModel лише тоді, коли його графічна частина дійсно існує та готова до відображення. Використання саме життєвого циклу представлення (viewLifecycleOwner) замість життєвого циклу самого фрагмента є критично

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		53

важливим, оскільки це запобігає витокам пам'яті та спробам оновити елементи екрана, які вже були знищені або ще не створені. Приклад реалізації класу наведено у додатку Б.1.

Центральним вузлом обробки подій є `WorkspaceViewModel`. Будь-яка активність касира, чи то введення назви товару в пошуковий рядок, чи натискання кнопки зміни кількості одиниць, миттєво транслюється у `ViewModel`. Для виконання асинхронних операцій, таких як звернення до бази даних або очікування результатів пошуку, використовується спеціалізована область видимості `viewModelScope`. Це дозволяє запускати корутини, які автоматично припиняють свою роботу, як тільки користувач залишає екран робочої зони. Такий підхід гарантує, що ресурси смартфона не будуть витрачатися на виконання фонових завдань, результати яких вже нікому не потрібні. Наприклад, якщо співробітник ініціював пошук великого списку товарів і одразу закрив додаток, `viewModelScope` миттєво скасує цей запит, зберігаючи заряд акумулятора та оперативну пам'ять. Приклад реалізації класу наведено у додатку Б.2.

Процес оновлення даних на екрані реалізовано через реактивну модель. `WorkspaceViewModel` постійно формує та транслює об'єкт стану `WorkspaceUiState`. Коли у `viewModelScope` завершується обробка нових даних (наприклад, розрахунок загальної суми чека після додавання нової позиції), оновлений стан відправляється в потік `StateFlow`. У свою чергу, `WorkspaceFragment` використовує корутинну область коду `viewLifecycleOwner.lifecycleScope`, яка є невіддільною складовою життєвого циклу фрагмента, щоб безпечно підхоплювати цей стан та оновлювати відповідні текстові поля чи списки товарів, не ризикуючи втратити або заблокувати дані при закритті екрана. Це створює ефект безперервної роботи: співробітник лише натискає на кнопку, а всі складні процеси валідації залишків на складі та перерахунку цін відбуваються на рівні внутрішньої логіки моделі представлення, не блокуючи основний потік інтерфейсу.

Така структура взаємодії дозволяє досягти високої відмовостійкості системи. Оскільки фрагмент повністю ізольований від механізмів отримання даних, будь-які помилки в мережі або збої в базі даних не призводять до аварійного завершення

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

роботи інтерфейсу. `WorkspaceViewModel` просто формує відповідний стан помилки, а фрагмент відображає його користувачеві у зручному вигляді. Завдяки використанню `viewModelScope` та `viewLifecycleOwner`, дозволило отримати систему, яка не лише ефективно виконує бізнес-задачі, а й коректно поводить себе в межах архітектури Android, забезпечуючи плавність роботи та раціональне використання системних ресурсів пристрою.

Наступним етапом реалізації програмного комплексу є організація взаємодії між моделлю представлення та шаром даних. У проєкті цей процес реалізовано через абстракцію репозиторіїв, що дозволяє відокремити бізнес-логіку від конкретних механізмів отримання інформації з мережі чи локальної бази даних.

Центральною ланкою в цьому ланцюзі виступає `WorkspaceRepository`. У моделі представлення `WorkspaceViewModel` цей репозиторій з'являється завдяки механізму впровадження залежностей Hilt. Оскільки сам репозиторій описаний як інтерфейс у доменному шарі, `ViewModel` не знає нічого про внутрішню роботу бази даних чи серверні запити, що відповідає принципу інверсії залежностей. Реальна ж робота зосереджена в класі `WorkspaceRepositoryImpl`, який знаходиться в шарі даних. Технічно ця реалізація визначена як `Singleton`, що гарантує існування лише одного екземпляра об'єкта протягом усього сеансу роботи користувача [29]. Це є критично важливим для підтримання цілісності даних у кошику та уникнення конфліктів при роботі фонових процесів синхронізації, оскільки всі компоненти додатка звертаються до одного і того ж джерела істини. Код реалізації класу наведено у додатку Б.3 та Б.4. Технічна реалізація репозиторію базується на поєднанні двох джерел даних: локального (`LocalDataSource`) та віддаленого (`SupabaseDataSource`). Така структура дозволяє реалізувати стратегію «Offline-first». Коли програма запускається, репозиторій ініціює фоновий процес синхронізації через метод `startProductSync`. У середині цього методу створюється спеціальний робочий процес `syncJob`, який у реальному часі «слухає» потік даних від сервера Supabase. Як тільки на сервері змінюється ціна або залишок товару, репозиторій отримує ці оновлення через корутини та негайно оновлює локальну

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

базу даних Room. Це гарантує, що співробітник завжди бачить актуальну інформацію навіть при нестабільному зв'язку.

Окремої уваги заслуговує реалізація пошуку товарів. Коли ViewModel викликає метод searchProducts, репозиторій спочатку звертається до віддаленого джерела SupabaseDataSource. Отримані результати не просто повертаються на екран, а спочатку проходять через процес мапінгу та зберігаються у локальному сховищі. Для цього використовується метод upsertProducts, який оновлює існуючі записи або додає нові. Такий підхід дозволяє зберігати історію пошуку локально та забезпечує миттєву швидкість роботи при повторних запитах. Для ідентифікації поточних результатів пошуку в пам'яті репозиторію використовується атомарне сховище AtomicReference, що запобігає конфліктам даних при одночасному виконанні запису з кількох потоків.

Робота з кошиком покупця реалізована через реактивні потоки Flow. Репозиторій надає метод observeCart, який повертає потік даних із локальної бази. Це означає, що як тільки в іншому місці програми (наприклад, через метод adjustSaleCount) змінюється кількість товару в кошику, Room автоматично генерує новий список, який через репозиторій потрапляє у ViewModel і миттєво відображається на екрані. Весь цей ланцюг працює асинхронно, не навантажуючи основний потік інтерфейсу.

Для забезпечення цілісності даних при закритті чека реалізовано метод getCheckoutItems. Він збирає всі позиції з локальної таблиці кошика, перетворює їх у доменні моделі продуктів і готує до фінальної відправки на сервер. Така багаторівнева взаємодія – від інтерфейсу репозиторію до конкретних джерел даних – робить систему гнучкою та надійною, де кожен компонент відповідає за свою частину роботи: сервер надає дані, локальна база їх кешує, а репозиторій керує цими потоками для забезпечення безперебійної роботи касира.

Окремим важливим елементом архітектури нашого проєкту є використання так званих Use Cases або сценаріїв використання. Якщо репозиторії відповідають за те, звідки беруться дані, то юзкейси – це вузькоспеціалізовані класи, які виконують одну конкретну, часто складну технічну задачу, яку не варто змішувати

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

з основною бізнес-логікою. У нашому випадку одним із найбільш наочних прикладів є сценарій стиснення зображень, реалізований в інтерфейсі ImageCompressUseCase. Реалізація інтерфейсу описана в додатку Б.5.

Проблема, яку вирішує цей компонент, знайома кожному розробнику мобільних додатків: сучасні смартфони роблять фотографії величезного розміру, які важать по 10-15 мегабайт. Якщо спробувати завантажити таку картинку на сервер без обробки, це займе багато часу, призведе до перевитрати трафіку користувача, а в найгіршому випадку – додаток просто завершиться аварійно через брак оперативної пам'яті під час спроби обробити такий об'єм даних. Тому всю логіку підготовки зображення винесено в окремий сценарій, який працює незалежно від того, на якому саме екрані додається фото..

Технічна реалізація цього сценарію в класі ImageCompressUseCaseImpl працює досить витончено. Замість того, щоб просто завантажити картинку в пам'ять, спочатку розглядається її реальний розмір. Це дозволяє розрахувати коефіцієнт стиснення так, щоб велике фото зменшилося до розумних меж ще до того, як воно повністю потрапить в оперативну пам'ять смартфона. Ще одна цікава технічна деталь – це робота з метаданими EXIF. Буває так, що камера смартфона записує фото боком, і лише спеціальний тег каже системі, як його правильно повернути. Наш юзкейс зчитує цей тег і автоматично повертає картинку в правильне положення, щоб у касира в каталозі товарів не було перевернутих зображень. Реалізація класу описана в додатку Б.6.

Ми використовуємо цей сценарій у моделі представлення додавання продуктів – ProductAddViewModel. Коли користувач обирає фото з галереї або робить знімок на камеру, ViewModel отримує посилання на цей файл і просто передає його в наш юзкейс. Важливо те, що сама ViewModel поняття не має про складні формули розрахунку розмірів чи роботу з матрицями для повороту картинки. Вона звертається до юзкейсу надаючи великий файл, замість якого отримує вже легкий масив байтів готовий до відправки на сервер до сховища Supabase Storage [30]. Реалізація класу описана в додатку Б.7.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

Другим важливим сценарієм використання у нашому програмному комплексі є ExportUseCase, який відповідає за формування та вивантаження звітів про роботу закладу. У реальному бізнесі недостатньо просто бачити цифри на екрані смартфона – власнику часто потрібно отримати ці дані у вигляді окремого файлу, який можна відкрити в Excel, надіслати бухгалтеру або зберегти для подальшого аналізу. Саме цю технічну задачу винесено в окремий юзкейс, щоб не перевантажувати логіку екранів звітності зайвим кодом для роботи з файловою системою. Реалізація інтерфейсу описана в додатку Б.8.

Основна складність реалізації цього сценарію полягає в тому, що дані в додатку зберігаються у вигляді складних об'єктів (наприклад, списки чеків або складських операцій), а для експорту їх потрібно перетворити у структурований текстовий формат, зрозумілий для сторонніх програм. Клас ExportUseCaseImpl бере на себе всю рутинну роботу: він проходить по кожному рядку звіту, правильно форматує дати, ціни та назви товарів, збираючи їх у єдиний текстовий блок. Це вимагає уважності до деталей, адже навіть зайва кома у файлі може призвести до того, що таблиця буде помилковою при відкритті на комп'ютері. Реалізація класу описана в додатку Б.9.

Технічно цей процес реалізований через асинхронні потоки, оскільки формування звіту за довгий період може містити тисячі записів і займати певний час. Щоб додаток не «завис» у цей момент, юзкейс виконує всю обробку у фоновому потоці. Окрім самого формування тексту, цей сценарій взаємодіє з операційною системою Android для того, щоб коректно створити файл у пам'яті пристрою та надати користувачеві посилання на нього. Це вимагає роботи з такими системними компонентами, як ContentResolver та Uri, що є досить специфічною задачею, яку ізольовано від решти бізнес-логіки.

Ми використовуємо цей сценарій, наприклад, на екрані звітності, в історії операцій. Коли користувач натискає кнопку «Завантажити звіт», відповідна ViewModel просто звертається до ExportUseCase, передаючи йому список даних, які він бачить на екрані. Результатом роботи юзкейсу є готовий файл, який користувач може одразу поширити через месенджери або зберегти на диск.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

Такий підхід підкреслює переваги обраної архітектури. Експорт даних – це складна технічна операція, яка включає роботу з форматуванням тексту та файловою системою. Винісши її в окремий ExportUseCase, код зроблено модульним: якщо в майбутньому буде необхідно змінити формат експорту (наприклад, замість простого тексту формувати PDF-документи), достатньо буде змінити лише цей один клас, а логіка самих екранів звітності залишиться незмінною. По суті, використання юзкейсу сховати дозволяє приховати деталі технічної реалізації за простим і зрозумілим інтерфейсом, роблячи розробку складних функцій простішою та надійнішою. Це робить систему стабільною та легкою у підтримці.

Висновки до другого розділу

У ході виконання другого розділу було проведено повний цикл розробки та реалізації системи, починаючи від проєктування структури і закінчуючи створенням функціонального мобільного додатка.

На основі аналізу варіантів використання було вибудовано об'єктно-орієнтовану структуру системи за принципами Clean Architecture. Це дозволило розділити відповідальність між шарами додатка, зробивши бізнес-логіку незалежною від технічних особливостей платформи. Впровадження паттерну MVVM забезпечило стабільний зв'язок між інтерфейсом та логікою обробки даних, а використання сценаріїв використання (Use Cases) дозволило ізолювати складні операції, такі як стиснення зображень та експорт звітів, у окремі модулі, що значно підвищило читабельність та надійність коду.

Важливим етапом стала розробка бази даних на базі хмарного сервісу Supabase. Обраний підхід Backend-as-a-Service дозволив реалізувати надійну систему зберігання інформації, авторизації користувачів та хмарного розміщення медіафайлів без необхідності розробки окремої серверної частини. Проєктування структури таблиць та зв'язків між ними забезпечило цілісність даних про товари, складські операції та продажі.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		59

Особлива увага була приділена проектуванню та реалізації алгоритмів роботи системи. Було впроваджено стратегію Offline-first, яка завдяки локальному кешуванню через бібліотеку Room гарантує безперебійну роботу касира навіть за відсутності стабільного інтернет-з'єднання. Реалізовані алгоритми обробки зображень (динамічне масштабування та корекція орієнтації) та реактивної синхронізації даних у реальному часі дозволили оптимізувати використання системних ресурсів смартфона.

Підсумком розділу стала повна реалізація мобільного додатка на мові Kotlin з використанням сучасного стеку технологій, включаючи Coroutines для асинхронних обчислень, Flow для реактивних потоків даних та Hilt для впровадження залежностей. Створений програмний продукт має модульну структуру, легко піддається масштабуванню та відповідає всім поставленим вимогам щодо функціональності, швидкодії та зручності користувацького інтерфейсу. Таким чином, розроблена система є повноцінним інструментом для автоматизації процесів обліку та продажу.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		60

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З POS-СИСТЕМОЮ АВТОМАТИЗАЦІЇ ПРОДАЖІВ

3.1. Порядок встановлення та налаштування параметрів системи

Процес розгортання та повного налаштування системи являє собою послідовний ланцюг технічних дій, спрямованих на встановлення стабільного та захищеного з'єднання між мобільним додатком та хмарною інфраструктурою. Оскільки архітектура проєкту передбачає розподілене зберігання даних та активну синхронізацію в реальному часі, процедура встановлення не обмежується лише інсталяцією файлу на пристрій, а вимагає детальної попередньої підготовки серверної частини та специфічної конфігурації середовища розробки. Весь шлях впровадження системи можна розділити на кілька фундаментальних етапів, кожен з яких має суворі вимоги до технічної послідовності виконання операцій.

Початковий етап реалізації зосереджений на платформі Supabase, яка обрана як основне серверне рішення через високу продуктивність та вбудовані інструменти обробки даних у реальному часі. На цій платформі розробник або системний адміністратор здійснює створення нового проєкту, обираючи оптимальний географічний регіон для розміщення сервера з метою мінімізації затримок при передачі пакетів даних. Після створення проєкту необхідно ініціалізувати реляційну базу даних PostgreSQL, що є основою архітектури. На цьому етапі створюються таблиці для обліку товарів, профілів користувачів, об'єктів торгових точок, складських операцій та фінансових транзакцій. Важливим аспектом є правильне налаштування типів даних, визначення первинних та зовнішніх ключів, а також встановлення обмежень цілісності на рівні бази даних для забезпечення логічної валідації бізнес-процесів. Варто зазначити, що на етапі конфігурації особлива увага приділяється налаштуванню політик безпеки на рівні рядків, що гарантує повну ізоляцію даних між різними торговими точками та запобігає несанкціонованому доступу до конфіденційної інформації. Крім реляційної бази, окремо проводиться конфігурація об'єктного сховища для мультимедійних файлів, де створюється розділ для збереження зображень

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

номенклатури товарів. Для забезпечення взаємодії мобільного додатка з хмарним середовищем адміністратор отримує унікальні ідентифікатори доступу, а саме адресу проєкту та секретний ключ авторизації, які використовуються для встановлення захищеного з'єднання.

Після завершення підготовки серверного середовища процес переходить до етапу конфігурації клієнтського програмного забезпечення в інтегрованому середовищі розробки Android Studio. Це відповідальний момент, оскільки на етапі компіляції в код інтегруються параметри зв'язку з сервером. Отримані ключі доступу вносяться у конфігураційні файли проєкту, після чого запускається процедура синхронізації з системою автоматизації збирання Gradle. Під час цієї операції середовище розробки проводить завантаження та перевірку необхідних бібліотек, які забезпечують функціонування всіх шарів системи: від інструментів впровадження залежностей Hilt до засобів роботи з локальною базою даних Room та засобів асинхронного програмування. Після успішної синхронізації та валідації коду виконується збірка проєкту для створення інсталяційного файлу формату APK. Під час формування цього файлу середовище розробки проводить автоматичну оптимізацію ресурсів та перевірку коду на відповідність стандартам якості, що забезпечує стабільну роботу додатка на широкому спектрі мобільних пристроїв. Цей файл є повноцінним дистрибутивом, готовим до розгортання на пристроях під управлінням операційної системи Android версії 7.0 або новішої. Після встановлення файлу на пристрій необхідно надати дозволи на доступ до мережевих ресурсів та внутрішньої пам'яті, що є технічною необхідністю для забезпечення обміну даними та завантаження графічної інформації.

Наступна частина налаштування параметрів системи проводиться безпосередньо в графічному інтерфейсі додатка після його успішного запуску. Система спроектована за принципом автоматичного призначення ролей, де перший зареєстрований користувач отримує статус власника з повним набором адміністративних прав. Таке рішення дозволяє проводити щоденне управління системою безпосередньо з мобільного пристрою без залучення зовнішніх інструментів адміністрування. Власник створює профіль торгового об'єкта,

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

вказуючи його найменування та адресу, що призводить до генерації та закріплення унікального ідентифікатора магазину в хмарному сховищі. Це забезпечує логічну ізоляцію даних однієї торгової точки від іншої. Після створення торгової точки власник переходить до формування структури персоналу. У відповідному розділі адміністративної панелі проводиться реєстрація співробітників із присвоєнням їм індивідуальних облікових даних. На цьому етапі активується модель контролю доступу на основі ролей, яка розмежовує доступ до функціональних модулів системи: власник отримує доступ до аналітичних звітів та фінансових показників, тоді як для касирів доступною залишається лише функціональність проведення торгових операцій.

Завершальним етапом розгортання системи є первинне наповнення бази даних номенклатурою товарів. Це відповідальний процес, де відповідальна особа через форму додавання вносить характеристики кожної позиції. Окрім назви та ціни, на цьому етапі проводиться визначення одиниць вимірювання, що безпосередньо впливає на логіку обробки даних у робочій зоні каси. Для кількісних товарів система передбачає зміну кількості в чеку, тоді як для вагових товарів автоматично активується модуль введення точних числових значень. Під час реєстрації товарів також здійснюється обробка зображень, які автоматично піддаються стисканню перед відправкою на сервер для оптимізації об'єму переданих даних та раціонального використання дискового простору сервера. Застосування сучасних алгоритмів обробки графічної інформації дозволяє не лише економити простір у хмарному сховищі, а й значно прискорює процес відображення каталогів на мобільних пристроях з низькою пропускнуою здатністю каналу зв'язку.

Усі внесені дані синхронізуються та стають доступними на всіх пристроях у межах однієї торгової точки. Використання спеціалізованого класу для збереження локальних налаштувань дозволяє системі зберігати ідентифікатор магазину та дані авторизації в енергонезалежній пам'яті пристрою. Локальна база даних у цьому ланцюзі виконує роль надійного буфера, який гарантує цілісність кожної транзакції навіть у моменти перемикання між різними типами мережевих з'єднань або при їх

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

повній відсутності. Таким чином, встановлена послідовність дій гарантує стабільне функціонування системи, забезпечуючи точність фінансового обліку та цілісність інформації протягом усього періоду експлуатації. Така детальна конфігурація дозволяє мінімізувати ймовірність виникнення технічних збоїв та робить систему стійкою до змін умов мережевого середовища.

3.2. Структура інтерфейсу мобільного додатку

Робота з додатком розпочинається з екрана авторизації, де користувач повинен ввести свою електронну адресу та пароль для входу в систему. Цей інтерфейс виглядає максимально лаконічно, зосереджуючи увагу лише на полях введення та кнопці підтвердження (рис. 3.1).

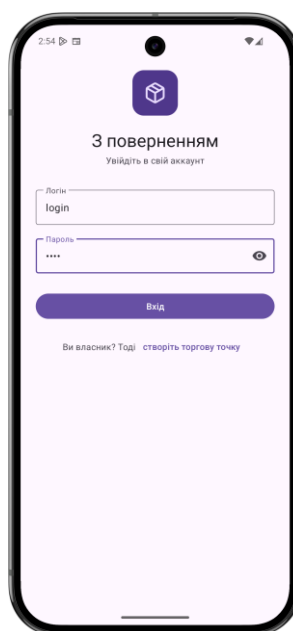


Рис. 3.1. Екран авторизації користувача

Після натискання кнопки входу програма автоматично перевіряє права доступу та, залежно від ролі користувача, направляє його на відповідний головний екран. У випадку, коли новий користувач ще не має прив'язаного закладу, можна перейти на екран створення магазину. На цьому екрані потрібно вказати логін та пароль власника для подальшої авторизації, ПІБ власника, адресу торгової точки, що дозволяє системі створити простір для ведення обліку саме цього закладу (рис. 3.2).

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		64

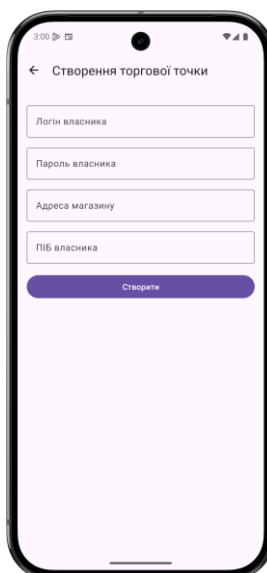


Рис. 3.2. Екран створення торгової точки

Для власників та адміністраторів головним навігаційним вузлом є панель управління. Цей екран реалізований у вигляді набору плиток, кожна з яких веде до конкретного розділу: продажі, звіти, управління товарами або складські операції (рис. 3.3).

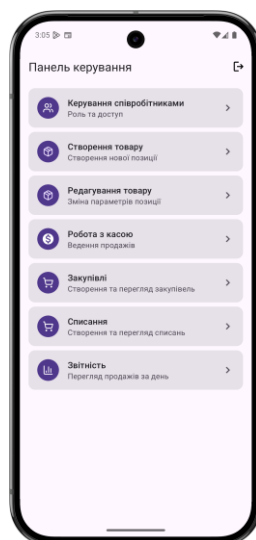


Рис. 3.3. Панель управління адміністратора

Користуватися цією панеллю дуже зручно, адже замість дрібних пунктів меню касир бачить великі кнопки, по яких легко влучити пальцем навіть під час швидкої роботи. Натискаючи на кнопку «Робота з касою», виконується перехід в основну робочу зону, де відбувається весь процес обслуговування клієнтів (рис. 3.4).

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		65

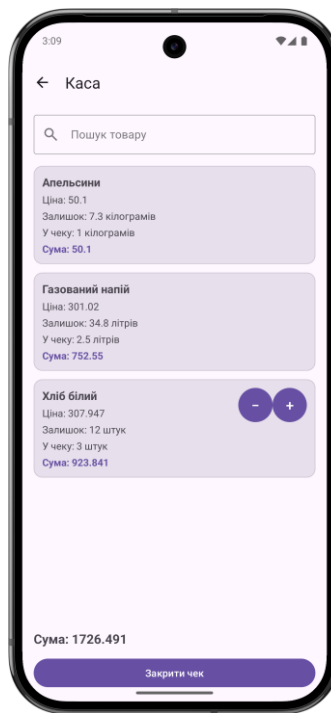


Рис. 3.4. Екран роботи з касою

Робоча зона касира є найбільш відвідуваним екраном програми. У верхній частині користувач бачить рядок пошуку, куди можна вводити назву товару. Список товарів миттєво реагує на введення, показуючи відповідні картки з актуальними залишками. Користувач додає товари до замовлення простим натисканням на них. Для вагових товарів передбачено автоматичну появу діалогового вікна, де касир за допомогою цифрової клавіатури вводить точну вагу в кілограмах або літрах.

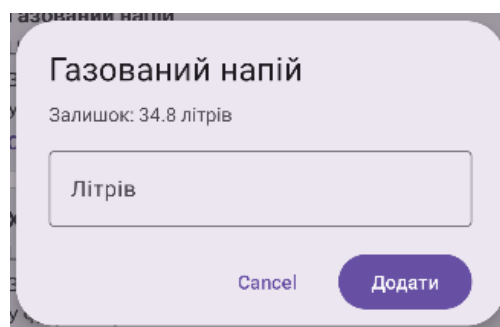


Рис. 3.4. Екран роботи з касою

Після формування списку покупок у нижній частині екрана з'являється поле з підсумковою сумою та кнопка закриття чеку. Натиснувши на неї, касир переходить на екран фіналізації чека, де обирає метод оплати (готівка або картка)

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

та підтверджує продаж, після чого робоча зона очищається для наступного клієнта (рис. 3.5).

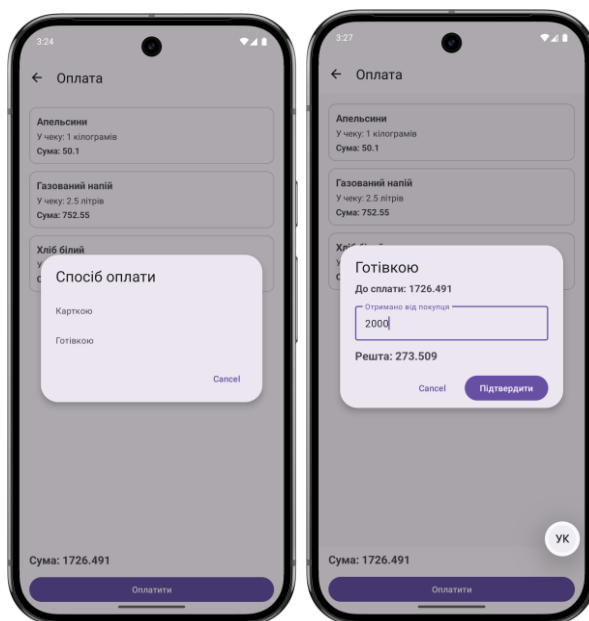


Рис. 3.5. Екран розрахунку та вибору методу оплати

Для управління асортиментом використовується екран додавання та редагування товарів. Користувачеві пропонується заповнити стандартну форму: вказати назву продукту, вибрати одиниці вимірювання (кілограми чи штуки), встановити ціну та додати опис. Особливу увагу варто приділити роботі з фотографіями – користувач може натиснути на іконку зображення, після чого додаток запропонує зробити новий знімок на камеру або вибрати готове фото з галереї телефона. Після заповнення всіх полів потрібно натиснути кнопку збереження, і товар миттєво з'явиться в загальному списку (рис. 3.6).

Окремий великий блок системи присвячений складському обліку, який є фундаментом для коректної роботи всього закладу. Складські операції в додатку розділені на два основні напрямки: прийом товару (закупівля) та його списання. Взаємодія з цим модулем починається з екрана історії операцій, де користувач бачить хронологічний список усіх дій, що впливали на залишки продуктів. Кожен елемент цього списку є інформативною картою, на якій відображається дата проведення операції (рис. 3.7). Це дозволяє адміністратору швидко переглянути останні зміни на складі у хронологічному порядку.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		67

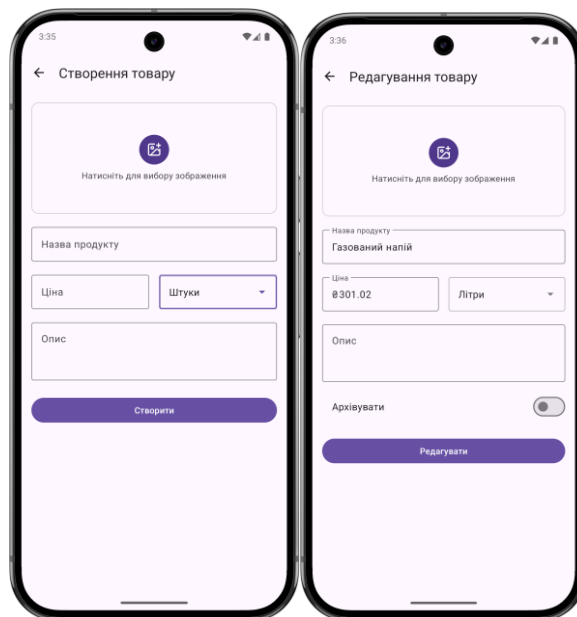


Рис. 3.6. Екран створення/редагування товару.

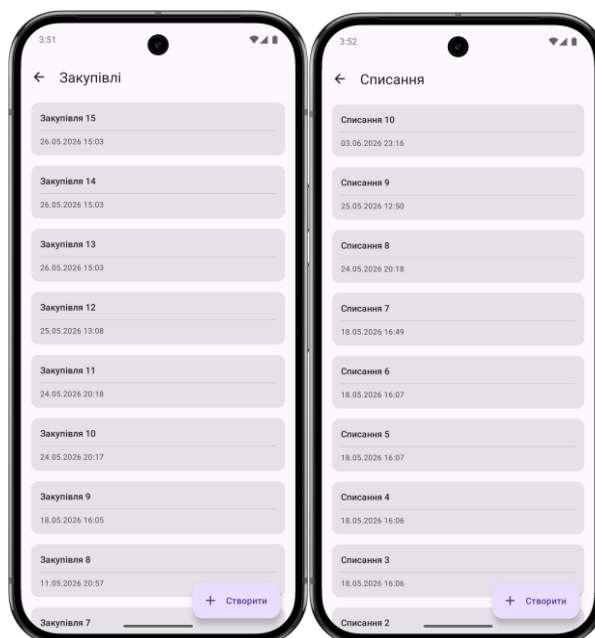


Рис. 3.7. Список складських операцій з прийому та списання.

Для проведення нової операції, наприклад, при надходженні нової партії товару, користувач натискає кнопку створення закупівлі, що відкриває відповідний інтерфейс. Спочатку касир обирає потрібний товар із загального каталогу, після чого відкривається вікно для введення кількості. Після додавання всіх позицій до списку операції, користувач може ще раз перевірити весь перелік і натиснути кнопку підтвердження. У цей момент алгоритми системи автоматично перераховують залишки в базі даних, і нові цифри миттєво стають доступними для продажу в робочій зоні касира. Аналогічно працює і модуль списання, який

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

використовується у випадках псування продуктів або використання їх для внутрішніх потреб закладу (рис. 3.8).

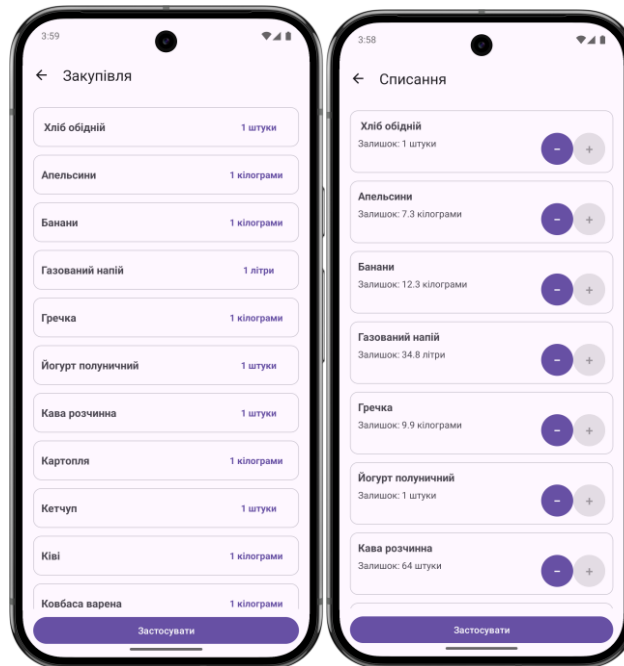


Рис. 3.9. Екран формування нової складської операції

Якщо ж виникає потреба провести аудит або перевірити стару накладну, користувач може просто натиснути на будь-яку операцію в історії. Це відкриє екран детального перегляду, де розписано, яка саме кількість кожного товару була зафіксована, та точний час збереження змін (рис. 3.10).

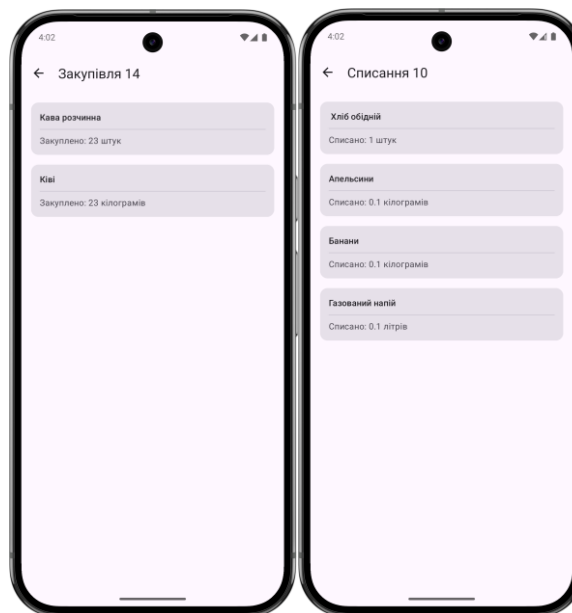


Рис. 3.10. Детальний огляд проведеної складської операції.

Не менш важливим для власника є блок звітності, який перетворює сухі цифри продажів у зрозумілу аналітику. При переході до цього розділу користувач потрапляє на головний екран звітності, де система автоматично підсумовує результати роботи за поточну зміну. Тут у верхній частині відображаються ключові показники: загальна виручка закладу та кількість виданих чеків. Нижче розташований детальний список усіх транзакцій.

Користувач може легко прокручувати цю стрічку, щоб побачити динаміку продажів протягом дня (рис. 3.11). Така прозорість дозволяє власнику в будь-який момент дізнатися, на яку суму було здійснено продажів, навіть не знаходячись фізично в закладі.

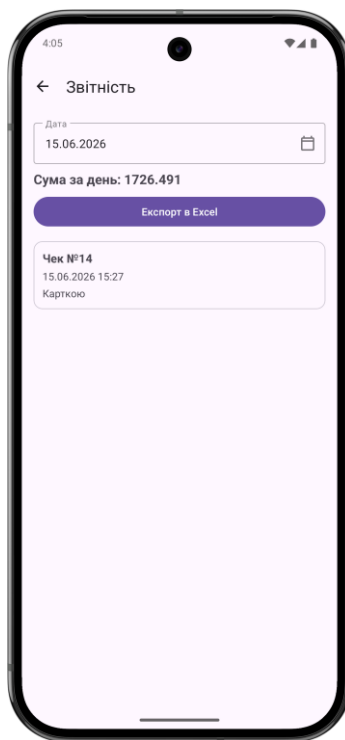


Рис. 3.11. Екран фінансової звітності та списку чеків.

Кожен чек у списку є інтерактивним. Якщо натиснути на нього, відкриється екран деталізації чека, який є цифровою копією виданого клієнту рахунку. Тут можна побачити повний перелік проданих позицій, їхню ціну на момент продажу, кількість та застосований метод оплати (готівковий чи карткою). Цей функціонал є критично важливим для вирішення суперечливих ситуацій або проведення повернень, оскільки власник може точно відстежити, що саме і о котрій годині було

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

реалізовано (рис. 3.12). Система дозволяє не просто бачити загальну суму, а й розуміти структуру кожного продажу.

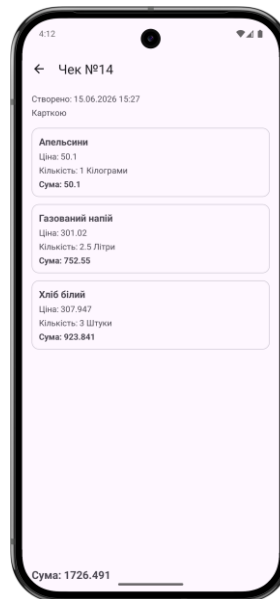


Рис. 3.12. Детальний перегляд інформації про конкретний чек.

Заключним етапом у налаштуванні робочого процесу є управління штатом закладу. У цьому модулі власник може бачити всіх активних працівників, які мають доступ до системи. Екран управління персоналом відображає список користувачів та їхні ролі, що дозволяє контролювати, хто саме має доступ до адмін-панелі, а хто може лише проводити продажі (рис. 3.13). Процес розширення штату реалізовано через максимально спрощену форму реєстрації нового працівника.

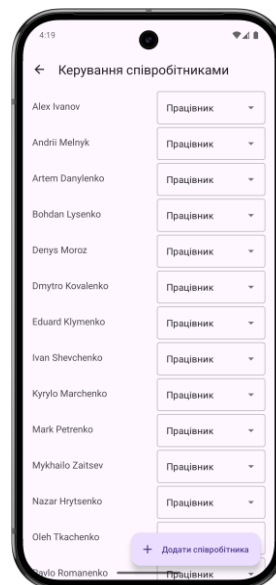


Рис. 3.13. Екран управління персоналом та списком працівників.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		71

Власнику достатньо натиснути кнопку додавання, ввести ім'я нового співробітника та придумати для нього пароль. Після натискання кнопки збереження новий обліковий запис миттєво стає активним, і працівник може одразу зайти в додаток під своїми даними на робочому планшеті чи смартфоні (рис. 3.14).

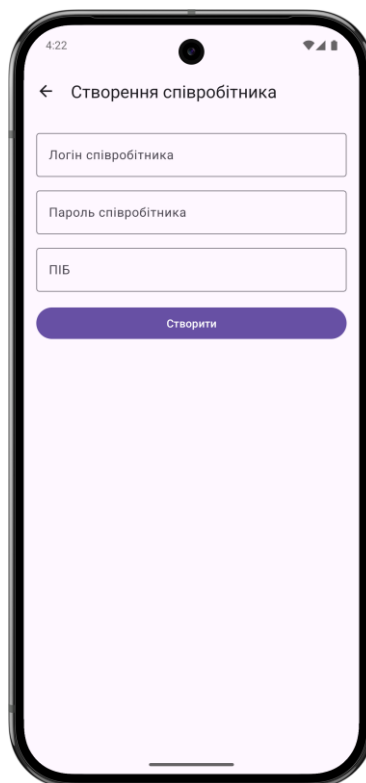


Рис. 3.14. Екран реєстрації нового працівника закладу.

Такий комплексний підхід до побудови інтерфейсу робить систему не просто програмою для каси, а повноцінним мобільним офісом, де в одному місці зібрані всі інструменти для управління складом, фінансами та персоналом, що значно спрощує щоденне ведення бізнесу для власника.

3.3. Тестування та перевірка роботи додатку

Заключним етапом розробки мобільної POS-системи стала серія комплексних випробувань та перевірка всіх компонентів програмного продукту, що дозволило підтвердити стабільність обраної архітектури та коректність роботи закладених алгоритмів. Тестування проводилося систематично та поетапно, починаючи від перевірки базових функцій ідентифікації користувачів і закінчуючи складними інтеграційними тестами системи синхронізації даних у реальному часі. Головною метою цього етапу було не лише підтвердження виконання всіх пунктів

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

технічного завдання, а й перевірка здатності додатка адекватно реагувати на нестандартні ситуації, помилкові дії персоналу або технічні проблеми зі зв'язком, що є критично важливим для стабільного функціонування торговельного закладу.

Першочергово було проведено детальне тестування ролівої моделі доступу та системи безпеки. Для цього в базі даних було змодельовано різні типи облікових записів, включаючи власника закладу, адміністратора та звичайного працівника. Під час серії випробувань було зафіксовано, що система розмежування прав працює без збоїв: при авторизації під акаунтом лінійного персоналу додаток автоматично обмежує доступ до конфіденційних розділів фінансової аналітики, звітів про прибутки та модулів управління кадровим складом. Навігаційний граф системи динамічно перебудовується, залишаючи доступною лише робочу зону касира та базові функції перегляду товарів. Це довело правильність реалізації логіки розподілу прав на рівні доменного шару та коректність перевірки статусу сесії при кожному запуску програми.

Особливу увагу в процесі верифікації було приділено перевірці механізмів синхронізації даних між хмарною платформою та локальним сховищем пристрою. Методологія тестування передбачала створення ситуації, коли дані змінюються одночасно з декількох джерел. Наприклад, проводився експеримент, у якому адміністратор змінював ціну та найменування товару через серверний інтерфейс у той час, коли співробітник працював із відкритим списком продукції на мобільному пристрої. Результати показали, що завдяки використанню реактивних потоків даних та постійного з'єднання, інформація на екрані смартфона оновлювалася миттєво без необхідності ручного перезавантаження сторінки або повторного входу в розділ. Це підтвердило надійність фонових процесів синхронізації та ефективність спроектованої моделі взаємодії з віддаленою базою даних.

Наступним важливим блоком випробувань стала перевірка алгоритмів обробки мультимедійного контенту, зокрема сценаріїв підготовки зображень товарів перед їх завантаженням у хмарне сховище. Оскільки мобільні пристрої часто створюють фотографії великого об'єму, що може сповільнювати роботу

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

системи та швидко заповнювати серверний простір, було протестовано модуль стиснення зображень. Для випробування використовувалися файли високої роздільної здатності об'ємом понад десять мегабайт. Алгоритм успішно проводив автоматичне масштабування та оптимізацію, зменшуючи розмір файлу до ста шістдесяти – двохсот кілобайт без втрати візуальної чіткості, необхідної для ідентифікації товару працівником. Додатково була перевірена точність обробки метаданих фотографій, що гарантувало правильну орієнтацію знімків незалежно від положення смартфона під час зйомки.

Найбільш критичним аспектом для забезпечення фінансової стабільності бізнесу було тестування логіки роботи кошика та механізмів валідації складських залишків. Процес тестування включав спроби проведення операцій, що суперечать встановленим бізнес-правилам. Зокрема, імітувалася ситуація продажу кількості товару, що перевищує його фактичну наявність на складі. Система продемонструвала високу точність обробки, негайно заблокувавши можливість закриття чека. Це підтвердило цілісність бізнес-логіки, закладеної на рівні репозиторіїв, та неможливість виникнення від'ємних значень на складі при звичайних операціях продажу.

Додатково проводилося тестування стійкості інтерфейсу до некоректного введення даних та перевірка ергономіки системи. При спробі створення нових карток товарів із незаповненими обов'язковими полями, такими як назва або ціна, система візуально підсвічувала помилки та деактивувала елементи управління, що запобігало забрудненню бази даних неповними записами. Окремо перевірялася логіка роботи з ваговими товарами, де замість цілих одиниць використовуються дробові числа. Випробування підтвердили, що система коректно обробляє операції з плаваючою точкою, правильно розраховує підсумкові суми в чеках та враховує одиниці вимірювання при формуванні звітів.

Окремим етапом випробувань стала детальна перевірка функціональності модуля формування звітності та механізмів експорту накопичених даних у зовнішні формати, що є важливим для ведення бухгалтерського обліку. Для проведення цього тесту в системі було змодельовано значний обсяг торгових

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		74

операцій за різні часові періоди, після чого здійснювалася генерація аналітичних звітів із використанням складних фільтрів за датами, типами оплати та конкретними категоріями товарів. Результати вивантаження даних у формат електронних таблиць Excel були ретельно зіставлені з первинними записами в хмарній базі даних Supabase, що підтвердило повну точність перенесення фінансових показників, розрахунку підсумкових сум та збереження часових міток кожної операції. Додатково було проведено перевірку коректності структури згенерованих файлів при їх відкритті в різних програмних пакетах для роботи з таблицями, що довело універсальність обраного методу формування документів. Тестування також охопило перевірку продуктивності пошукових алгоритмів у журналах операцій, де система продемонструвала високу швидкість фільтрації записів навіть при значному наповненні бази даних. Це підтвердило готовність аналітичного блоку програми до використання в умовах інтенсивного товарообігу, коли оперативний доступ до точних підсумкових показників діяльності закладу є ключовим фактором для прийняття управлінських рішень.

Завершальна фаза тестування включала перевірку адаптивності графічної оболонки на пристроях із різними розмірами екранів та версіями операційної системи Android, починаючи з версії 7.0. Було встановлено, що використання компонентів сучасної бібліотеки проектування інтерфейсів забезпечує коректне відображення всіх елементів керування, текстів та таблиць без накладання об'єктів один на одного. Загалом, результати всіх етапів випробувань показали, що розроблена мобільна POS-система повністю відповідає поставленим вимогам, демонструє високу продуктивність, стабільність роботи та є готовою до впровадження в реальний процес експлуатації торговельними підприємствами.

Висновки до третього розділу

У ході виконання третього розділу було проведено всебічну перевірку та опис практичного застосування реалізованої системи, що дозволило підтвердити її повну готовність до експлуатації. Завершальний етап роботи охопив усі аспекти –

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		75

від технічного розгортання системи до детальної перевірки коректності виконання алгоритмів у критичних ситуаціях.

На основі розробленого регламенту встановлення було доведено, що обрана архітектура на базі хмарного сервісу Supabase та мобільної платформи Android дозволяє швидко та надійно розгорнути систему без необхідності використання складного серверного обладнання. Чітко визначена послідовність дій – від налаштування бази даних у хмарі до первинної конфігурації магазину безпосередньо в додатку – забезпечує високу мобільність рішення та легкість його впровадження у реальний бізнес-процес.

Детальний аналіз структури інтерфейсу продемонстрував, що розроблена графічна оболонка відповідає вимогам ергономіки та професійної спрямованості. Логічне розділення функцій між робочою зоною касира та адміністративною панеллю, а також використання реактивних елементів керування, дозволило створити інтуїтивно зрозуміле середовище для користувача. Опис кожного екрана у вигляді послідовних інструкцій підтвердив, що додаток мінімізує кількість помилок при введенні даних та прискорює процес обслуговування клієнтів за рахунок оптимізованого пошуку та зручної системи роботи з ваговими товарами.

Ключовим результатом розділу стало успішне тестування програмного комплексу. Проведені випробування довели надійність алгоритмів синхронізації даних у реальному часі та ефективність механізмів обробки зображень, що дозволяє раціонально використовувати ресурси пристрою та хмарного сховища.

Таким чином, результати третього розділу повністю підтверджують відповідність розробленої системи поставленим завданням. Система продемонструвала високу стабільність, точність обчислень та зручність у використанні, що дає підстави рекомендувати його як ефективний інструмент для автоматизації обліку та управління в закладах малого та середнього бізнесу. Всі реалізовані рішення пройшли практичну перевірку і є готовими до впровадження у реальну діяльність підприємств.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		76

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було успішно розроблено та реалізовано систему призначену для автоматизації процесів обліку, управління продажами та складськими залишками в закладах малого та середнього бізнесу. Проведений аналіз предметної області підтвердив актуальність створення мобільного та гнучкого рішення, яке б поєднувало в собі потужність хмарних технологій із простотою використання на персональних пристроях працівників.

Головним результатом роботи стала розробка цілісної архітектури додатка, побудованої на принципах Clean Architecture. Використання розподілу на рівні бізнес-логіки, даних та представлення дозволило створити систему, яка є легкою у підтримці та готовій до майбутнього масштабування. Впровадження паттерну MVVM у поєднанні з реактивними потоками даних забезпечило миттєвий відгук інтерфейсу на дії користувача та автоматичну синхронізацію стану додатка з віддаленим сервером. Окрему цінність має реалізація стратегії автономної роботи, де завдяки використанню локальної бази даних Room та механізмів фоновій синхронізації було досягнуто високої відмовостійкості системи.

Технічна реалізація проєкту базується на сучасному стеку технологій Android-розробки. Використання мови програмування Kotlin разом із корутинами та потоками Flow дозволило написати безпечний та продуктивний код для обробки складних операцій. Вибір платформи Supabase як бекенд-рішення виявився повністю виправданим, оскільки це дозволило інтегрувати хмарне зберігання медіафайлів та реляційну базу даних без витрат ресурсів на розробку власного серверного коду. Реалізовані алгоритми оптимізації зображень та валідації складських запасів довели свою ефективність, забезпечуючи раціональне використання пам'яті пристроїв та точність фінансового обліку.

Створений інтерфейс користувача відповідає всім сучасним вимогам до ергономіки та зручності. Робоча зона касира, адміністративна панель та модулі звітності спроектовані таким чином, щоб мінімізувати кількість рутинних дій та запобігти виникненню помилок через людський фактор. Чітке розмежування ролей

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				77
Змн.	Арк.	№ докум.	Підпис	Дата		

користувачів гарантує безпеку даних та конфіденційність комерційної інформації власника закладу. Комплексне тестування розробленого програмного забезпечення підтвердило коректність роботи всіх функцій – від першого входу в систему до формування детальних звітів про продажі за тривалі періоди.

Підсумовуючи результати, можна стверджувати, що всі поставлені перед початком роботи завдання були виконані в повному обсязі. Система автоматизації продажів є завершеним продуктом, який демонструє високу стабільність, швидкість роботи та зручність експлуатації. Практичне значення роботи полягає в можливості безпосереднього впровадження розробленого додатка в діяльність крамниць чи інших торговельних точок, що дозволить власникам оптимізувати свої бізнес-процеси, отримати прозорий контроль над фінансами та підвищити якість обслуговування клієнтів. Перспективи подальшого розвитку проєкту можуть включати інтеграцію з платіжними терміналами, розширення функціоналу звітності та впровадження системи лояльності для постійних покупців.

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Держанівський А.О., Піонтківський В.І. Аналіз існуючих POS-систем управління // Тези XVI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», м. Житомир, 2–3 квітня 2026 р. Житомир: Державний університет «Житомирська політехніка», 2026. С. 47–48. URL: <https://conf.ztu.edu.ua/wp-content/uploads/2026/06/povnyj-tekst.pdf> (дата звернення: 13.06.2026).
2. Держанівський А.О., Піонтківський В.І. Опис застосування продукту Supabase для використання в мобільній розробці // Тези доповідей науково-практичної on-line конференції здобувачів вищої освіти і молодих учених, присвяченої Дню науки, 11-15 травня 2026 р. Житомир: Державний університет «Житомирська політехніка», 2026. С. 122. URL: https://conf.ztu.edu.ua/wp-content/uploads/2026/06/sekcija_4.pdf (дата звернення: 13.06.2026).
3. Poster POS URL: <https://joinposter.com/ua> (дата звернення: 13.03.2026).
4. KavApp URL: <https://kavapp.com/uk> (дата звернення: 13.03.2026).
5. SkyService POS URL: <https://skyservice.pro/uk/main/> (дата звернення: 13.03.2026).
6. CheckBox URL: <https://checkbox.ua> (дата звернення: 13.03.2026).
7. Guide to app architecture. Android Developers. App architecture. URL: <https://developer.android.com/topic/architecture> (дата звернення: 04.06.2026).
8. Ahmed Jawad Aman. MVVM Architecture in Android – A Complete Beginner's Guide By Ahmed Jawad Aman – Sr. Android Developer. Medium. 03.08.2025. URL: <https://medium.com/@ahmedjawadaman/mvvm-architecture-in-android-a-complete-beginners-guide-by-ahmed-jawad-aman-sr-653630189f6f> (дата звернення: 04.06.2026).
9. ViewModel overview. Android Developers. App architecture. URL: <https://developer.android.com/topic/libraries/architecture/viewmodel> (дата звернення: 04.06.2026).

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				79
Змн.	Арк.	№ докум.	Підпис	Дата		

10. The activity lifecycle. Android Developers. App architecture. URL: <https://developer.android.com/guide/components/activities/activity-lifecycle> (дата звернення: 05.06.2026).
11. UI layer. Android Developers. App architecture. URL: <https://developer.android.com/topic/architecture/ui-layer> (дата звернення: 24.03.2026).
12. Domain layer. Android Developers. App architecture. URL: <https://developer.android.com/topic/architecture/domain-layer> (дата звернення: 24.03.2026).
13. Data layer. Android Developers. App architecture. URL: <https://developer.android.com/topic/architecture/data-layer> (дата звернення: 24.03.2026).
14. Dependency injection in Android. Android Developers. App architecture. URL: <https://developer.android.com/training/dependency-injection> (дата звернення: 04.06.2026).
15. Kotlin docs. Kotlin Programming Language. URL: <https://kotlinlang.org/docs/home.html>
16. Develop Android apps with Kotlin. Android Developers. URL: <https://developer.android.com/kotlin> (дата звернення: 04.06.2026).
17. Kotlin coroutines on Android. Android Developers. Guides. URL: <https://developer.android.com/kotlin/coroutines> (дата звернення: 04.06.2026).
18. Coroutines. Kotlin Programming Language. URL: <https://kotlinlang.org/docs/coroutines-overview.html> (дата звернення: 04.06.2026).
19. Dependency injection with Hilt. Android Developers. App architecture. URL: <https://developer.android.com/training/dependency-injection/hilt-android> (дата звернення: 04.06.2026).
20. Navigation component. Android Developers. App architecture. URL: <https://developer.android.com/guide/navigation> (дата звернення: 04.06.2026).

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		80

21. Material Design. URL: <https://m3.material.io> (дата звернення: 04.06.2026).
22. Material Design for Android. Android Developers. Views. URL: <https://developer.android.com/develop/ui/views/theming/look-and-feel> (дата звернення: 04.06.2026).
23. App data and file. Android Developers. Save data in a local database using Room. URL: <https://developer.android.com/training/data-storage/room> (дата звернення: 04.06.2026).
24. Supabase URL: <https://supabase.com> (дата звернення: 04.06.2026).
25. OkHttp – HTTP & HTTP/2 Client for Android and Java Applications. Square, Inc. URL: <https://square.github.io/okhttp/> (дата звернення: 04.06.2026).
26. Kotlin flows on Android. Android Developers. Kotlin. Guides. URL: <https://developer.android.com/kotlin/flow> (дата звернення: 04.06.2026).
27. StateFlow and SharedFlow. Android Developers. Kotlin. Guides. URL: <https://developer.android.com/kotlin/flow/stateflow-and-sharedflow> (дата звернення: 04.06.2026).
28. Jomar Tigcal. Using Kotlin Flows. Simplifying Android Development with Coroutines and Flows. Birmingham : Packt Publishing, 2022. ISBN BN 978-1-80181-624-3.
29. Manish Bannur. Singleton Design Pattern in Android. Medium. 11.06.2024. URL: https://medium.com/@manish_bannur/singleton-design-pattern-in-android-96dedc18619d (дата звернення: 04.06.2026).
30. Storage. Supabase Docs. Overview. URL: <https://supabase.com/docs/guides/storage> (дата звернення: 04.06.2026).

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

ДОДАТКИ

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

А.1 Лістинг методу create_receipt:

```

create or replace function create_receipt(
    shop_id bigint,
    payment_type text,
    products jsonb
)
returns bigint
language plpgsql
as $$
declare
    v_receipt_id bigint;
    v_product jsonb;
begin

    insert into receipt(shop_id, payment_type)
    values (shop_id, payment_type)
    returning id into v_receipt_id;

    for v_product in
        select * from jsonb_array_elements(products)
    loop
        insert into products_sold(
            name,
            count,
            price,
            unit,
            receipt_id
        )
        values (
            v_product->>'name',
            (v_product->>'count')::double precision,
            (v_product->>'price')::double precision,
            v_product->>'unit',
            v_receipt_id
        );
    end loop;

    return v_receipt_id;
end;
$$;

```

А.2 Лістинг методу process_writeoff:

```

create or replace function process_writeoff(
    shop_id bigint,
    items jsonb
)
returns void as $$
declare
    new_writeoff_id bigint;

```

		Держанівський			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

```

begin

insert into write_offs (shop_id, created_at)
values (shop_id, now())
returning id into new_writeoff_id;

for i in 0 .. jsonb_array_length(items) - 1 loop

insert into writeoff_products (
    write_off_id,
    product_id,
    count
)
values (
    new_writeoff_id,
    (items->i->>'id')::bigint,
    (items->i->>'count')::float8
);

update products
set count = greatest(
    count - ((items->i->>'count')::float8),
    0
)
where id = (items->i->>'id')::bigint
and products.shop_id = process_writeoff.shop_id;

end loop;

end;
$$ language plpgsql;

```

A.3 Лістинг методу process_procurement:

```

create or replace function process_procurement(shop_id bigint, items jsonb)
returns void as $$
declare
    new_procurement_id bigint;
begin
insert into procurements (shop_id, created_at)
values (shop_id, now())
returning id into new_procurement_id;

for i in 0 .. jsonb_array_length(items) - 1 loop

insert into procurement_products (procurement_id, product_id, count)
values (
    new_procurement_id,
    (items->i->>'id')::bigint,
    (items->i->>'count')::float8
);

```

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

```

update products
set count = count + ((items->i->>'count')::float8)
where id = (items->i->>'id')::bigint;

end loop;
end;
$$ language plpgsql;

```

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		85

Б.1 Лістинг класу WorkspaceFragment:

```

package com.symphony.cofein.presentation.workspace

import android.os.Bundle
import android.view.Menu
import android.view.MenuInflater
import android.view.MenuItem
import android.view.View
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.MenuProvider
import androidx.core.view.isVisible
import androidx.core.widget.doAfterTextChanged
import androidx.fragment.app.viewModels
import androidx.lifecycle.lifecycleScope
import androidx.navigation.NavOptions
import androidx.navigation.fragment.findNavController
import androidx.navigation.fragment.navArgs
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import com.symphony.cofein.R
import com.symphony.cofein.databinding.DialogWorkspaceQuantityBinding
import com.symphony.cofein.databinding.FragmentWorkspaceBinding
import com.symphony.cofein.presentation.base.fragment.BaseFragment
import com.symphony.cofein.util.ParentInflate
import com.symphony.cofein.util.flowWithLifecycle
import com.symphony.cofein.util.showToast
import com.symphony.cofein.util.string
import com.symphony.cofein.util.view.decimalFormat
import com.symphony.cofein.util.view.doOnClick
import com.symphony.cofein.util.view.doOnEdgeDimenInsets
import com.symphony.cofein.util.view.margins
import com.symphony.cofein.util.view.text
import com.symphony.cofein.util.view.paddings
import dagger.hilt.android.AndroidEntryPoint
import kotlinx.coroutines.flow.distinctUntilChanged
import kotlinx.coroutines.flow.filterIsInstance
import kotlinx.coroutines.flow.launchIn
import kotlinx.coroutines.flow.map
import kotlinx.coroutines.flow.onEach

@AndroidEntryPoint
class WorkspaceFragment : BaseFragment<FragmentWorkspaceBinding>() {

    override val inflate: ParentInflate<FragmentWorkspaceBinding>
        get() = FragmentWorkspaceBinding::inflate

    private val args: WorkspaceFragmentArgs by navArgs()

    private val viewModel by viewModels<WorkspaceViewModel>()
    private val adapter = WorkspaceAdapter(
        onMinusClick = { viewModel.onMinusClick(it) },
        onPlusClick = { viewModel.onPlusClick(it) },
        onCountChange = { viewModel.onCountChange(it) },
    )

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.root.doOnEdgeDimenInsets { paddings.top = it.top }
    }

```

		Держанівський			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

```

        (requireActivity() as
AppCompatActivity).setSupportActionBar(binding.toolbar)

        requireActivity().addMenuProvider(object : MenuProvider {
            override fun onCreateMenu(menu: Menu, menuInflater: MenuInflater) {
                if (!args.isAdmin) menuInflater.inflate(R.menu.menu_sign_out,
menu)
            }

            override fun onMenuItemSelected(menuItem: MenuItem): Boolean {
                return when (menuItem.itemId) {
                    R.id.menu_signOut -> {
                        viewModel.onExitClick()
                        true
                    }
                    else -> false
                }
            }
        }, viewLifecycleOwner)

        binding.toolbar.setNavigationOnClickListener {
findNavController().popBackStack() }

        findNavController().currentBackStackEntry
?.savedStateHandle
?.getStateFlow(CHECKOUT_COMPLETED_KEY, false)
?.onEach { completed ->
    if (completed) {
        viewModel.onCheckoutCompleted()
        binding.searchEt.text = null
        findNavController().currentBackStackEntry
?.savedStateHandle
?.set(CHECKOUT_COMPLETED_KEY, false)
    }
}
?.launchIn(viewLifecycleOwner.lifecycleScope)

        binding.rv.adapter = adapter
        binding.searchEt.doAfterTextChanged {
viewModel.onSearchChanged(it.toString()) }
        binding.closeCheckBtn.doOnClick { viewModel.onCloseCheckClick() }

        viewModel.uiState
            .flowWithLifecycle(viewLifecycleOwner)
            .onEach(::handleUiState)
            .launchIn(viewLifecycleOwner.lifecycleScope)

        viewModel.navState
            .flowWithLifecycle(viewLifecycleOwner)
            .onEach(::handleNavState)
            .launchIn(viewLifecycleOwner.lifecycleScope)

        viewModel.uiState
            .flowWithLifecycle(viewLifecycleOwner)
            .filterIsInstance<WorkspaceUiState.Content>()
            .map { it.products }
            .distinctUntilChanged()
            .onEach { adapter.updateItems(it) }
            .launchIn(viewLifecycleOwner.lifecycleScope)
    }

    private fun handleUiState(state: WorkspaceUiState) = when (state) {

```

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		87

```

WorkspaceUiState.Loading -> {
    binding.progressBar.isVisible = true
    binding.contentScroll.isVisible = false
    binding.bottomBar.isVisible = false
}

is WorkspaceUiState.Content -> {
    binding.progressBar.isVisible = false
    binding.contentScroll.isVisible = true
    binding.bottomBar.isVisible = true
    binding.totalSumTv.text(
        R.string.workspace_total_sum,
        state.totalSum.decimalFormat(),
    )
    val emptyMessage = state.emptyMessageRes
    binding.emptyTv.isVisible = emptyMessage != null
    binding.rv.isVisible = emptyMessage == null
    emptyMessage?.let(binding.emptyTv::setText)
}
}

private fun handleNavState(state: WorkspaceNavState) = when (state) {
    WorkspaceNavState.NavigateToCheckout -> findNavController().navigate(
        WorkspaceFragmentDirections.actionKassaToCheckoutFragment(),
    )
    is WorkspaceNavState.ShowQuantityDialog -> showQuantityDialog(state)
    is WorkspaceNavState.Error -> when (state.error) {
        is WorkspaceErrorType.Message -> showToast(state.error.message)
        WorkspaceErrorType.NoProductsSelected ->
            showToast(R.string.no_products_selected)
        WorkspaceErrorType.InvalidQuantity ->
            showToast(R.string.invalid_count)
        WorkspaceErrorType.ExceedsStock ->
            showToast(R.string.workspace_exceeds_stock)
        WorkspaceErrorType.Unknown -> showToast(R.string.unknown_error)
    }
    WorkspaceNavState.SignOut -> {
        val navController = findNavController()
        val options = NavOptions.Builder()
            .setPopUpTo(navController.graph.startDestinationId, true)
            .setLaunchSingleTop(true)
            .build()
        navController.navigate(navController.graph.startDestinationId, null,
options)
    }
}

private fun showQuantityDialog(state: WorkspaceNavState.ShowQuantityDialog) {
    val dialogBinding = DialogWorkspaceQuantityBinding.inflate(layoutInflater)
    val unit = string(state.product.unit.stringRes).lowercase()
    dialogBinding.stockCountTv.text(
        R.string.workspace_stock_count,
        state.product.count.decimalFormat(),
        unit,
    )
    dialogBinding.countTl.hint = string(state.product.unit.stringRes)
    val dialog = MaterialAlertDialogBuilder(requireContext())
        .setTitle(state.product.name)
        .setView(dialogBinding.root)
        .create()
    dialogBinding.cancelBtn.doOnClick { dialog.dismiss() }
    dialogBinding.addBtn.doOnClick {
        val quantity =

```

		Держанівський			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

```

dialogBinding.countEt.text?.toString()?.toDoubleOrNull()
    viewModel.onQuantityConfirmed(state.product.id, quantity ?: 0.0)
    dialog.dismiss()
}
dialog.show()
}

companion object {
    const val CHECKOUT_COMPLETED_KEY = "checkout_completed"
}
}

```

Б.2 Лістинг класу WorkspaceViewModel:

```

package com.symphony.cofein.presentation.workspace

import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.symphony.cofein.R
import com.symphony.cofein.data.local.Preferences
import com.symphony.cofein.domain.model.enums.Units
import com.symphony.cofein.domain.model.product.ProductCount
import com.symphony.cofein.domain.repository.WorkspaceRepository
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.ExperimentalCoroutinesApi
import kotlinx.coroutines.FlowPreview
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.flow.collectLatest
import kotlinx.coroutines.flow.combine
import kotlinx.coroutines.flow.debounce
import kotlinx.coroutines.flow.flatMapLatest
import kotlinx.coroutines.launch
import javax.inject.Inject

@OptIn(FlowPreview::class, ExperimentalCoroutinesApi::class)
@HiltViewModel
class WorkspaceViewModel @Inject constructor(
    private val workspaceRepository: WorkspaceRepository,
    private val preferences: Preferences
) : ViewModel() {

    private val _uiState =
        MutableStateFlow<WorkspaceUiState>(WorkspaceUiState.Loading)
    val uiState: StateFlow<WorkspaceUiState> get() = _uiState.asStateFlow()

    private val _navState = MutableSharedFlow<WorkspaceNavState>()
    val navState get() = _navState.asSharedFlow()

    private val queryFlow = MutableStateFlow("")
    private val searchIds = MutableStateFlow<List<Long>>(emptyList())

    init {
        viewModelScope.launch {
            workspaceRepository.startProductSync()
        }

        viewModelScope.launch {
            queryFlow
                .debounce(SEARCH_DEBOUNCE_MS)
                .collectLatest { query ->

```

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

```

        if (query.length >= MIN_SEARCH_LENGTH) {
            searchIds.value =
workspaceRepository.searchProducts(query)
        } else {
            searchIds.value = emptyList()
        }
    }

viewModelScope.launch {
    val searchResultsFlow = searchIds.flatMapLatest { ids ->
        workspaceRepository.observeSearchResults(ids)
    }
    combine(
        queryFlow,
        workspaceRepository.observeCart(),
        workspaceRepository.observeCartTotal(),
        searchResultsFlow,
    ) { query, cart, totalSum, searchResults ->
        val displayed = if (query.isBlank()) cart else searchResults
        val emptyMessageRes = resolveEmptyMessage(
            isEmpty = displayed.isEmpty(),
            query = query,
        )
        WorkspaceUiState.Content(
            products = displayed,
            totalSum = totalSum,
            emptyMessageRes = emptyMessageRes,
        )
    }.collect { _uiState.emit(it) }
}

fun onSearchChanged(value: String) = viewModelScope.launch {
    queryFlow.emit(value.trim())
}

fun onMinusClick(id: Long) = viewModelScope.launch {
    val step = workspaceRepository.getWeightProduct(id)?.unit?.step ?:
return@launch
    workspaceRepository.adjustSaleCount(id, -step)
}

fun onPlusClick(id: Long) = viewModelScope.launch {
    val step = workspaceRepository.getWeightProduct(id)?.unit?.step ?:
return@launch
    workspaceRepository.adjustSaleCount(id, step)
}

fun onCountChange(id: Long) = viewModelScope.launch {
    val product = workspaceRepository.getWeightProduct(id) ?: return@launch
    _navState.emit(WorkspaceNavState.ShowQuantityDialog(product.toProductCount()))
}

fun onQuantityConfirmed(productId: Long, quantity: Double) =
viewModelScope.launch {
    if (quantity <= 0) {
        _navState.emit(WorkspaceNavState.Error(WorkspaceErrorType.InvalidQuantity))
    }

    val product = workspaceRepository.getWeightProduct(productId) ?:

```

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

```

return@launch
    val available = product.stockCount - product.saleCount
    if (quantity > available) {

_navState.emit(WorkspaceNavState.Error(WorkspaceErrorType.ExceedsStock))
    }
    workspaceRepository.setSaleCount(productId, quantity)
}

fun onCheckoutCompleted() = viewModelScope.launch {
    queryFlow.emit("")
    searchIds.emit(emptyList())
}

fun onCloseCheckClick() = viewModelScope.launch {
    if (workspaceRepository.getCheckoutItems().isEmpty()) {

_navState.emit(WorkspaceNavState.Error(WorkspaceErrorType.NoProductsSelected))
    } else {
        _navState.emit(WorkspaceNavState.NavigateToCheckout)
    }
}

private fun resolveEmptyMessage(isListEmpty: Boolean, query: String): Int? {
    if (!isListEmpty) return null
    return when {
        query.isBlank() -> R.string.workspace_empty_search_prompt
        query.length < MIN_SEARCH_LENGTH ->
R.string.workspace_empty_search_prompt
        else -> R.string.workspace_empty_no_results
    }
}

private val Units.step: Double
    get() = when (this) {
        Units.PIECE -> 1.0
        Units.KILO, Units.LITER -> 0.1
        Units.ALL -> 1.0
    }

private fun WorkspaceProduct.toProductCount() = ProductCount(
    id = id,
    name = name,
    unit = unit,
    count = stockCount,
)

override fun onCleared() {
    viewModelScope.launch {
        workspaceRepository.stopProductSync()
    }
    super.onCleared()
}

fun onExitClick() = viewModelScope.launch {
    workspaceRepository.clearCart()
    preferences.shopId = -1L
    _navState.emit(WorkspaceNavState.SignOut)
}

companion object {
    private const val MIN_SEARCH_LENGTH = 0
    private const val SEARCH_DEBOUNCE_MS = 300L

```

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

```
}
}
```

Б.3 Лістинг класу WorkspaceRepository:

```
package com.symphony.cofein.domain.repository

import com.symphony.cofein.domain.model.product.Product
import com.symphony.cofein.presentation.workspace.WorkspaceItem
import com.symphony.cofein.presentation.workspace.WorkspaceProduct
import kotlinx.coroutines.flow.Flow

interface WorkspaceRepository {

    suspend fun observeCart(): Flow<List<WorkspaceItem>>

    suspend fun observeSearchResults(productIds: List<Long>):
Flow<List<WorkspaceItem>>

    suspend fun searchProducts(query: String): List<Long>

    suspend fun getWeightProduct(productId: Long): WorkspaceProduct?

    suspend fun setSaleCount(productId: Long, saleCount: Double)

    suspend fun adjustSaleCount(productId: Long, delta: Double)

    suspend fun observeCartTotal(): Flow<Double>

    suspend fun clearCart()

    suspend fun getCheckoutItems(): List<Product>

    suspend fun startProductSync()

    suspend fun stopProductSync()
}
```

Б.4 Лістинг класу WorkspaceRepositoryImpl:

```
package com.symphony.cofein.data.repository

import com.symphony.cofein.data.local.Preferences
import com.symphony.cofein.data.source.SupabaseDataSource
import com.symphony.cofein.data.source.LocalDataSource
import com.symphony.cofein.domain.model.product.Product
import com.symphony.cofein.domain.model.workspace.WorkspaceCartLineEntity
import com.symphony.cofein.domain.repository.WorkspaceRepository
import com.symphony.cofein.presentation.workspace.WorkspaceItem
import com.symphony.cofein.presentation.workspace.WorkspaceProduct
import kotlinx.coroutines.CoroutineScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.Job
import kotlinx.coroutines.SupervisorJob
import kotlinx.coroutines.flow.Flow
import kotlinx.coroutines.flow.flowOf
import kotlinx.coroutines.flow.map
import kotlinx.coroutines.launch
import java.util.concurrent.atomic.AtomicReference
import javax.inject.Inject
import javax.inject.Singleton
```

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		92

```

@Singleton
class WorkspaceRepositoryImpl @Inject constructor(
    private val local: LocalDataSource,
    private val supabase: SupabaseDataSource,
    private val preferences: Preferences,
) : WorkspaceRepository {

    private val shopId get() = preferences.shopId

    private val searchSessionIds = AtomicReference<Set<Long>>(emptySet())
    private var syncJob: Job? = null
    private val scope = CoroutineScope(SupervisorJob() + Dispatchers.IO)

    override suspend fun observeCart(): Flow<List<WorkspaceItem>> =
        local.observeCartProducts().map { rows -> rows.map { it.toWorkspaceItem() } }

    override suspend fun observeSearchResults(productIds: List<Long>):
Flow<List<WorkspaceItem>> =
        if (productIds.isEmpty()) {
            flowOf(emptyList())
        } else {
            local.observeProductsWithSale(productIds).map { rows ->
                rows.map { it.toWorkspaceItem() }
            }
        }

    override suspend fun searchProducts(query: String): List<Long> {
        val products = supabase.searchProductsForSale(
            shopId = shopId,
            query = query,
            limit = SEARCH_LIMIT,
            offset = 0
        )
        if (products.isEmpty()) {
            searchSessionIds.set(emptySet())
            return emptyList()
        }
        local.upsertProducts(products.map { it.toWorkspaceEntity(shopId) })
        val ids = products.mapNotNull { it.id }
        searchSessionIds.set(ids.toSet())
        return ids
    }

    override suspend fun getWeightProduct(productId: Long): WorkspaceProduct? {
        val cached = local.getProduct(productId) ?: return null
        val saleCount = local.getCartLine(productId)?.saleCount ?: 0.0
        return cached.let { entity ->
            WorkspaceProduct(
                id = entity.id,
                name = entity.name,
                price = entity.price,
                stockCount = entity.stockCount,
                saleCount = saleCount,
                unit = entity.unit,
            )
        }
    }

    override suspend fun setSaleCount(productId: Long, saleCount: Double) {
        val product = local.getProduct(productId) ?: return
        val clamped = saleCount.coerceIn(0.0, product.stockCount)
        if (clamped <= 0.0) {

```

		Держанівський			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		93

```

        local.deleteCartLine(productId)
    } else {
        local.upsertCartLine(WorkspaceCartLineEntity(productId, clamped))
    }
}

override suspend fun adjustSaleCount(productId: Long, delta: Double) {
    val product = local.getProduct(productId) ?: return
    val current = local.getCartLine(productId)?.saleCount ?: 0.0
    val newCount = (current + delta).coerceIn(0.0, product.stockCount)
    setSaleCount(productId, newCount)
}

override suspend fun observeCartTotal(): Flow<Double> =
    observeCart().map { items -> items.sumOf { it.product.lineTotal } }

override suspend fun clearCart() {
    local.clearCart()
    searchSessionIds.set(emptySet())
}

override suspend fun getCheckoutItems(): List<Product> =
    local.getCartProducts().map { row ->
        Product(
            id = row.product.id,
            name = row.product.name,
            count = row.saleCount,
            unit = row.product.unit,
            price = row.product.price,
        )
    }

override suspend fun startProductSync() {
    syncJob?.cancel()
    syncJob = scope.launch {
        supabase.getProductsForSale(shopId).collect { patch ->
            applyProductPatches(patch)
        }
    }
}

override suspend fun stopProductSync() {
    syncJob?.cancel()
    syncJob = null
}

private suspend fun applyProductPatches(patch: List<Product>) {
    if (patch.isEmpty()) return
    val trackedIds = trackedProductIds()
    if (trackedIds.isEmpty()) return
    patch.forEach { product ->
        val id = product.id ?: return@forEach
        if (id !in trackedIds) return@forEach
        val cached = local.getProduct(id)
        if (cached == null) {
            local.upsertProduct(product.toWorkspaceEntity(shopId))
        } else {
            local.updateStockAndPrice(
                id = id,
                stockCount = product.count,
                price = product.price,
            )
        }
    }
}

```

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		94

```

    }
}

private suspend fun trackedProductIds(): Set<Long> {
    val cartIds = local.getCartProductIds()
    val searchIds = searchSessionIds.get()
    return cartIds.toSet() + searchIds
}

companion object {
    private const val SEARCH_LIMIT = 50
}
}

```

Б.5 Лістинг класу ImageCompressUseCase:

```

package com.symphony.cofein.domain.use_case

import android.net.Uri

interface ImageCompressUseCase {

    suspend fun prepareFile(uri: Uri, targetWidth: Int = 200, quality: Int = 80):
    ByteArray

}

```

Б.6 Лістинг класу ImageCompressUseCaseImpl:

```

package com.symphony.cofein.data.use_case

import android.content.ContentResolver
import android.content.Context
import android.graphics.Bitmap
import android.graphics.BitmapFactory
import android.graphics.Matrix
import androidx.exifinterface.media.ExifInterface
import android.net.Uri
import androidx.core.graphics.scale
import com.symphony.cofein.domain.use_case.ImageCompressUseCase
import dagger.hilt.android.qualifiers.ApplicationContext
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.withContext
import java.io.ByteArrayOutputStream
import javax.inject.Inject

class ImageCompressUseCaseImpl @Inject constructor(
    @param:ApplicationContext private val context: Context
) : ImageCompressUseCase {

    override suspend fun prepareFile(uri: Uri, targetWidth: Int, quality: Int):
    ByteArray {
        return withContext(Dispatchers.IO) {
            try {
                uri.toCompressedByteArray(targetWidth, quality)
            } catch (e: Exception) {
                e.printStackTrace()
                throw e
            }
        }
    }
}

```

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		95

```

private fun Uri.toCompressedByteArray(targetWidth: Int, quality: Int):
ByteArray {
    val contentResolver = context.contentResolver

    val options = BitmapFactory.Options().apply {
        inJustDecodeBounds = true
    }
    contentResolver.openInputStream(this)?.use {
        BitmapFactory.decodeStream(it, null, options)
    }

    options.inSampleSize = calculateInSampleSize(options, targetWidth)
    options.inJustDecodeBounds = false

    val originalBitmap = contentResolver.openInputStream(this)?.use {
        BitmapFactory.decodeStream(it, null, options)
    } ?: throw Exception("Failed to decode image")

    val aspectRatio = originalBitmap.height.toFloat() /
originalBitmap.width.toFloat()
    val targetHeight = (targetWidth * aspectRatio).toInt()

    val scaledBitmap = originalBitmap.scale(targetWidth, targetHeight)

    val rotatedBitmap = scaledBitmap.rotateBitmapIfNeeded(this,
contentResolver)

    val outputStream = ByteArrayOutputStream()
    return outputStream.use { stream ->
        rotatedBitmap.compress(Bitmap.CompressFormat.JPEG, quality, stream)

        if (originalBitmap != scaledBitmap) {
            originalBitmap.recycle()
        }
        scaledBitmap.recycle()

        stream.toByteArray()
    }
}

private fun calculateInSampleSize(options: BitmapFactory.Options, reqWidth:
Int): Int {
    val width = options.outWidth
    var inSampleSize = 1
    if (width > reqWidth) {
        val halfWidth = width / 2
        while (halfWidth / inSampleSize >= reqWidth) {
            inSampleSize *= 2
        }
    }
    return inSampleSize
}

private fun Bitmap.rotateBitmapIfNeeded(
    uri: Uri,
    contentResolver: ContentResolver
): Bitmap {

    val inputStream = contentResolver.openInputStream(uri)
    val exif = inputStream?.use {
        ExifInterface(it)
    } ?: return this

    val orientation = exif.getAttributeInt(

```

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		96

```

        ExifInterface.TAG_ORIENTATION,
        ExifInterface.ORIENTATION_NORMAL
    )

    val matrix = Matrix()

    when (orientation) {
        ExifInterface.ORIENTATION_ROTATE_90 -> {
            matrix.postRotate(90f)
        }

        ExifInterface.ORIENTATION_ROTATE_180 -> {
            matrix.postRotate(180f)
        }

        ExifInterface.ORIENTATION_ROTATE_270 -> {
            matrix.postRotate(270f)
        }

        else -> return this
    }

    return Bitmap.createBitmap(
        this,
        0,
        0,
        width,
        height,
        matrix,
        true
    )
}
}

```

Б.7 Лістинг класу ProductAddViewModel:

```

package com.symphony.cofein.presentation.product.product_add

import android.net.Uri
import androidx.lifecycle.SavedStateHandle
import androidx.lifecycle.ViewModel
import androidx.lifecycle.viewModelScope
import com.symphony.cofein.domain.model.enums.Units
import com.symphony.cofein.domain.model.product.Product
import com.symphony.cofein.domain.repository.SupabaseRepository
import com.symphony.cofein.domain.use_case.ImageCompressUseCase
import dagger.hilt.android.lifecycle.HiltViewModel
import kotlinx.coroutines.flow.MutableSharedFlow
import kotlinx.coroutines.flow.MutableStateFlow
import kotlinx.coroutines.flow.StateFlow
import kotlinx.coroutines.flow.asSharedFlow
import kotlinx.coroutines.flow.asStateFlow
import kotlinx.coroutines.launch
import javax.inject.Inject

@HiltViewModel
class ProductAddViewModel @Inject constructor(
    savedStateHandle: SavedStateHandle,
    private val supabaseRepository: SupabaseRepository,
    private val fileHelper: ImageCompressUseCase
): ViewModel() {

```

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		97

```

        private val productId =
            ProductAddFragmentArgs.fromSavedStateHandle(savedStateHandle).productId
        private val isAdd get() = productId == 0L

        private val _uiState =
            MutableStateFlow<ProductAddUiState>(ProductAddUiState.Loading)
        val uiState: StateFlow<ProductAddUiState> get() = _uiState.asStateFlow()

        private val _navState = MutableSharedFlow<ProductAddNavState>()
        val navState get() = _navState.asSharedFlow()

        private var product = Product(count = 0.0)

        init {
            viewModelScope.launch {
                product = if (isAdd) Product(count = 0.0)
                    else supabaseRepository.getProductById(productId)
                _uiState.emit(ProductAddUiState.Content(product, isAdd))
            }
        }

        fun onNameChanged(name: String) {
            product = product.copy(name = name)
        }

        fun onUnitsSelected(unit: Units) {
            product = product.copy(unit = unit)
        }

        fun onPriceChanged(price: String) {
            val price = price.toDoubleOrNull() ?: 0.0
            product = product.copy(price = price)
        }

        fun onDescriptionChanged(description: String) {
            product = product.copy(description = description)
        }

        fun onArchiveChanged(isArchived: Boolean) {
            product = product.copy(isArchive = isArchived)
        }

        fun onImagePicked(uri: Uri) = viewModelScope.launch {
            val bytes = try {
                fileHelper.prepareFile(uri)
            } catch (e: Exception) {
                e.printStackTrace()
                return@launch
            }
            sendImage(bytes)
        }

        private fun sendImage(bytes: ByteArray) = viewModelScope.launch {
            val url = try {
                supabaseRepository.sendImage(bytes)
            } catch (e: Exception) {
                e.printStackTrace()
                ""
            }
            product = product.copy(image = url)
        }
    }

```

		Держанівський			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		98

```

fun onCreateClick() = viewModelScope.launch {
    _uiState.emit(ProductAddUiState.Loading)
    val isNameError = product.name.isBlank()
    val isUnitError = product.unit == Units.ALL
    val isPriceError = product.price <= 0.0
    if (isNameError || isUnitError || isPriceError) return@launch _uiState
        .emit(ProductAddUiState.Content(product, isAdd, isNameError,
isUnitError, isPriceError))
    try {
        if (productId == 0L) supabaseRepository.createProduct(product)
        else supabaseRepository.updateProduct(product)
        _navState.emit(ProductAddNavState.Done)
    } catch (e: Exception) {
        e.printStackTrace()
        _navState.emit(ProductAddNavState.Error(e.message.toString()))
        _uiState.emit(
            ProductAddUiState.Content(product, isAdd)
        )
    }
}
}

```

Б.8 Лістинг класу ExportUseCase:

```

package com.symphony.cofein.domain.use_case

import android.icu.util.Calendar
import com.symphony.cofein.domain.model.receipt.ReceiptReport
import java.io.File

interface ExportUseCase {

    fun writeReport(date: Calendar, entries: List<ReceiptReport>): File

}

```

Б.9 Лістинг класу ExportUseCaseImpl:

```

package com.symphony.cofein.data.use_case

import android.content.Context
import android.icu.text.SimpleDateFormat
import android.icu.util.Calendar
import com.symphony.cofein.domain.model.receipt.ReceiptReport
import com.symphony.cofein.domain.use_case.ExportUseCase
import dagger.hilt.android.qualifiers.ApplicationContext
import org.dhatim.fastexcel.Workbook
import java.io.File
import java.io.FileOutputStream
import java.util.Locale
import javax.inject.Inject
import kotlin.collections.forEach

class ExportUseCaseImpl @Inject constructor(
    @param:ApplicationContext private val context: Context
): ExportUseCase {
    private val fileDateFormatter = SimpleDateFormat("dd.MM.yyyy",
Locale.getDefault())

    override fun writeReport(date: Calendar, entries: List<ReceiptReport>): File {

```

		Держанівський			ПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		99

```

val dateFileSuffix = fileDateFormatter.format(date.time)
val file = File(context.cacheDir, "report_${dateFileSuffix}.xlsx")
FileOutputStream(file).use { outputStream ->
    val workbook = Workbook(outputStream, "Cofein", "1.0")
    val sheet = workbook.newWorksheet("Звіт")

    sheet.value(0, 0, "Чек №")
    sheet.value(0, 1, "Позиція")
    sheet.value(0, 2, "Кількість")
    sheet.value(0, 3, "Одиниця")
    sheet.value(0, 4, "Ціна")
    sheet.value(0, 5, "Сума позиції")
    sheet.value(0, 6, "Сума чека")

    var row = 1
    entries.forEach { entry ->
        val receiptNumber = entry.receipt.id?.toString().orEmpty()
        if (entry.items.isEmpty()) {
            sheet.value(row, 0, receiptNumber)
            sheet.value(row, 6, entry.total)
            row++
        } else {
            entry.items.forEachIndexed { index, item ->
                if (index == 0) {
                    sheet.value(row, 0, receiptNumber)
                    sheet.value(row, 6, entry.total)
                }
                sheet.value(row, 1, item.name)
                sheet.value(row, 2, item.count)
                sheet.value(row, 3, item.unit.name)
                sheet.value(row, 4, item.price)
                sheet.value(row, 5, item.lineTotal)
                row++
            }
        }
    }

    workbook.finish()
}
return file
}
}

```

		Держанівський			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Піонтківський В.				
Змн.	Арк.	№ докум.	Підпис	Дата		100