Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Автоматизація процесу постачання та продажу продукції в магазині»

Виконав: студент ІІ курсу, групи ЗАКІТР-24м, спеціальності 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Вадим ВОЙТЕНКО
Керівник: д.т.н., професор кафедри КСУ
Марія ЮХИМЧУК
« 12 » грудня 2025 р.
Опонент: доц. каф. АІІТ
Володимир ГАРМАШ
« 12 » грудня 2025 р.

Допущено до захисту
Завідувач кафедри АІІТ
д.т.н.,проф.
Олег БІСІКАЛО
« 15 » 12 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти другий (магістерський)
Галузь знань 17 – Електроніка, автоматизація та електронні комунікації
Спеціальність 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітньо-професійна програма Інформаційні системи і Інтернет речей

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф Олег БІСІКАЛО
«26» вересня 2025 року

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Войтенку Вадиму Мирославовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Автоматизація процесу постачання та продажу продукції в магазині».
Керівник роботи: д.т.н. Юхимчук Марія Сергіївна, професор кафедри КСУ.
Затверджені наказом ВНТУ від 24 вересня 2025 року № 313.
2. Строк подання роботи студентом: до 12 грудня 2025 року.
3. Вихідні дані до роботи: Журнали активних замовлень та архіву; середній вихідний час оформлення одного замовлення в ручному режимі 6–7 хвилин та час оновлення інформації про статус замовлення 10–12 хвилин; скорочення середнього часу оформлення замовлення до 3–4 хвилин, часу оновлення статусу до 1–2 хвилин.
4. Зміст текстової частини: Вступ; Аналіз предметної галузі; Проектування автоматизованої системи управління; Аналіз інструментів для виконання задач; Розробка автоматизованої системи; Висновки; Список використаних джерел.
5. Перелік ілюстративного (графічного) матеріалу: Use Case UML-діаграма варіантів використання; UML-діаграма діяльності; UML-діаграма класів; UML-діаграма компонентів; Скріншоти основних екранів PWA-додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Юхимчук М. С., проф. каф. КСУ	200925	200925

7. Дата видачі завдання: 25 вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Термін виконання	Примітка
1	Аналіз предметної області. Попередній огляд літературних та нормативних джерел.	25.09.2025 – 05.10.2025	
2	Детальний аналіз предметної області та бізнес-процесів.	06.10.2025 – 18.10.2025	
3	Розробка інформаційної моделі системи	19.10.2025 – 31.10.2025	
4	Обґрунтування вибору інструментів реалізації	01.11.2025 – 08.11.2025	
5	Розробка та вдосконалення автоматизованої системи	09.11.2025 - 23.11.2025	
6	Проведення експериментальних досліджень, аналіз результатів, оформлення пояснювальної записки, графічного матеріалу та презентації	24.11.2025 – 01.12.2025	
7	Підготовка та проведення попереднього захисту	до 02.12.2025	
8	Захист роботи	до 19.12.2025	

Студент Вадим ВОЙТЕНКО

Керівник роботи Марія ЮХИМЧУК

АНОТАЦЯ

УДК 004:658

Войтенко В. М. Автоматизація процесу постачання та продажу продукції в магазині. Магістерська кваліфікаційна робота зі спеціальності 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка, освітньо-професійна програма – Інформаційні системи і Інтернет речей. Вінниця: ВНТУ, 2025. 127 с.

Українською мовою Бібліогр.: 61 назв; рис.: 32; табл.: 4.

Магістерська кваліфікаційна робота присвячена розробці та дослідженню автоматизованої системи підтримки процесів постачання та продажу продукції в роздрібному магазині на основі сучасних веб-технологій. На основі аналізу предметної області та існуючих систем автоматизації роздрібної торгівлі сформовано вимоги до функціоналу системи й побудовано інформаційну модель даних основних сутностей. Спроєктовано клієнт–серверну архітектуру на базі фреймворку Angular з використанням технології Progressive Web Application (PWA) та хмарної платформи Firebase / Cloud Firestore для зберігання й обробки даних, реалізовано програмний прототип, що забезпечує авторизацію користувачів із різними ролями, ведення довідника продукції, облік постачань та оформлення замовлень. Експериментальні дослідження засвідчили зменшення часу обробки операцій і підвищення прозорості обліку, що підтверджує доцільність упровадження розробленої системи в діяльність роздрібних магазинів.

Ключові слова: автоматизація постачання, автоматизація продажу, роздрібний магазин, управління товарними запасами, Angular, Firebase, Firestore, Progressive Web Application (PWA).

ABSTRACT

Voitenko V. M. Automation of the Process of Supply and Sale of Products in a Retail Store. Master's Qualification Thesis in specialty 174 – Automation, Computer-Integrated Technologies and Robotics, educational and professional programme – Intelligent Computer Systems and Internet of Things. Vinnytsia: VNTU, 2025. 127 p.

In Ukrainian. Bibliogr.: 61 references; fig.: 32; tabl.: 4.

The master's thesis is devoted to the development and study of an automated system for supporting supply and sales processes in a retail store based on modern web technologies. On the basis of an analysis of the subject area and existing retail automation systems, the functional requirements of the system were defined and an information model of the main entities was built. A client–server architecture was designed using the Angular framework, Progressive Web Application (PWA) technology and the Firebase / Cloud Firestore cloud platform for data storage and processing. A software prototype was implemented that provides user authentication with different roles, product catalogue management, supply accounting and order processing. Experimental studies showed a reduction in operation processing time and an increase in transparency of accounting, which confirms the feasibility of introducing the developed system into the activity of retail stores.

Keywords: supply automation, sales automation, retail store, inventory management, Angular, Firebase, Firestore, Progressive Web Application (PWA).

ЗМІСТ

ВСТУП

Актуальність теми. У сучасних умовах розвитку цифрової економіки питання автоматизації торговельних процесів набуває особливого значення. Більшість підприємств роздрібної торгівлі прагнуть оптимізувати управління постачанням, продажами та обліком товарів шляхом впровадження автоматизованих інформаційних систем. Такі системи забезпечують підвищення точності обліку, швидкості обробки даних, зменшення людського фактору та покращення якості управлінських рішень.

Особливу роль відіграють інтелектуальні технології, що дозволяють прогнозувати потреби в постачанні, контролювати товарні залишки та формувати аналітичні звіти в режимі реального часу. Застосування принципів автоматизації дозволяє ефективно керувати бізнес-процесами магазину, забезпечуючи взаємозв’язок між постачанням та продажем.

Одним із сучасних підходів до реалізації таких систем є використання Angular Progressive Web Application (PWA) - технології, яка поєднує можливості вебзастосунку та локальної програми. Це забезпечує адаптивність, автономну роботу, швидкий доступ даних і підтримку користувачів із різних пристроїв.

Системи автоматизації процесів постачання та продажу стають основою побудови інтелектуальних інформаційних систем управління, що входять до сфери спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка». Такі рішення сприяють підвищенню ефективності бізнесу, мінімізації витрат та покращенню взаємодії між учасниками торгового процесу.

Мета роботи полягає у створенні інтелектуальної автоматизованої системи управління процесом постачання та продажу продукції в магазині, яка реалізується у вигляді PWA-додатку та забезпечує зручну взаємодію користувачів із системою та оперативну обробку інформації.

Завдання дослідження можна сформувати наступним чином:

- Провести аналіз рішень у сфері автоматизації торгових процесів.

- Визначити функціональні та технічні вимоги до системи управління постачанням і продажами.
- Розробити архітектуру автоматизованої системи.
- Описати принцип роботи користувацького інтерфейсу.

Об’єкт дослідження – процес автоматизованого управління постачанням і продажем товарів у магазині.

Предмет дослідження – моделі, методи та програмні засоби побудови веборієнтованої автоматизованої системи підтримки процесів постачання та продажу продукції в роздрібному магазині.

Методи дослідження. У роботі використано методи системного аналізу та аналізу предметної області роздрібної торгівлі, методи моделювання бізнес-процесів, показники ефективності функціонування системи), методи проєктування баз даних та клієнт–серверних застосунків.

Науково-технічний результат полягає у застосуванні сучасних технологій PWA та елементів аналітичного моделювання для створення універсальної адаптивної системи автоматизації процесів постачання та продажу, що може бути інтегрована у торговельні або логістичні інформаційні комплекси.

Практичне значення роботи полягає у можливості використання розробленої системи для оптимізації управління запасами, скорочення часу обробки замовлень і підвищення ефективності роботи персоналу магазину.

Апробація. Представлені в роботі результати апробовані в результаті участі в конференції КПІ ім. Ігоря Сікорського: «X міжнародна науково-практична конференція молодих учених, аспірантів і студентів АКІТ – 2024».

Публікації. Войтенко Вадим Мирославович «Автоматизована система обліку складу підприємства», «X міжнародна науково-практична конференція молодих учених, аспірантів і студентів АКІТ – 2024», 2024 [1].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Роль інформаційних технологій у сучасній роздрібній торгівлі

Умови функціонування роздрібної торгівлі в цифровій економіці характеризуються високою конкуренцією, зростанням вимог споживачів до швидкості обслуговування, якості сервісу та прозорості цін. У таких умовах інформаційні технології (ІТ) перетворюються з допоміжного інструменту на стратегічний ресурс, що безпосередньо впливає на ефективність бізнес-процесів, рівень витрат і конкурентоспроможність торговельного підприємства [2].

Інформаційні технології в роздрібній торгівлі забезпечують збір, зберігання, обробку й аналіз даних про товари, постачальників, клієнтів, фінансові операції та внутрішні процеси підприємства. На основі цих даних формуються управлінські рішення щодо формування асортименту, планування постачань, ціноутворення, маркетингових акцій тощо. Як зазначає Костюк В. П., сучасні інформаційні системи в економіці виконують роль «інформаційного каркасу» підприємства, інтегруючи окремі бізнес-процеси в єдине кероване середовище [3].

Однією з ключових ролей ІТ у роздрібній торгівлі є автоматизація рутинних операцій, пов’язаних з обліком продажів, руху запасів і грошових коштів. На практиці це реалізується через використання POS-систем (Point of Sale), які забезпечують реєстрацію продажів, ведення чеків, фіскалізацію операцій та інтеграцію з платіжними сервісами. Також систем управління складом (WMS), що автоматизують процеси приймання, розміщення, інвентаризації й відвантаження товарів. Та модулів обліку в складі ERP-систем, які дозволяють синхронізувати складські дані з бухгалтерським і управлінським обліком [4].

Завдяки автоматизації значно зменшується кількість помилок, пов’язаних із ручним введенням даних, скорочуються часові витрати на оформлення операцій, підвищується прозорість обліку. Бабенко О. Г. відзначає, що впровадження комплексних систем автоматизації в торговельних підприємствах дозволяє знизити

трудомісткість обробки первинних документів на 30–50 % та забезпечити оперативне оновлення інформації в режимі, близькому до реального часу [5].

Ще одна критично важлива роль IT у роздрібній торгівлі пов’язана з управлінням ланцюгом постачання (Supply Chain Management, SCM) та запасами. Неефективне управління запасами призводить до двох крайнощів: дефіциту товарів, що знижує рівень задоволеності клієнтів, або надлишку, який «заморожує» оборотні кошти й збільшує витрати на зберігання.

Таблиця 1.1 - Класифікація витрат в системі управління запасами

Вид витрат	Приклад
Витрати, пов’язані з виконанням замовлення	Витрати на логістику, придбання зразків, витрати на рекламу
Прямі витрати	Вартість запасів, які перебувають на зберіганні
Витрати, пов’язані з утриманням та зберіганням запасів	Втрата фізичних властивостей, псування товару
Витрати, пов’язані з обмеженістю ресурсів	Втрати у виробництві, вартість і обмеженість ресурсів

Сучасні інформаційні системи дозволяють:

- автоматично контролювати рівень запасів у режимі реального часу;
- прогнозувати майбутній попит на основі історичних даних про продажі, сезонності, маркетингових акцій;
- формувати замовлення постачальникам за заданими правилами (мінімальний рівень запасу, оптимальний розмір партії тощо);
- координувати постачання між центральним складом і роздрібними точками мережі [11].

Клименко В. І. підкреслює, що використання спеціалізованих інформаційних систем управління запасами дозволяє інтегрувати моделі EOQ, ABC/XYZ-аналіз, стратегії періодичного та безперервного поповнення в єдине автоматизоване середовище, що значно спрощує прийняття рішень менеджерами різних рівнів [12].

У комплексі з ERP- та WMS-рішеннями інформаційні технології дають змогу побудувати «прозорий» ланцюг постачання, де кожен етап – від замовлення виробнику до продажу кінцевому споживачу – відслідковується й аналізується в єдиній системі [13-14].

Сучасна роздрібна торгівля генерує великі обсяги даних: транзакції продажів, інформацію про клієнтів, маркетингові кампанії, логістичні витрати, показники роботи персоналу. Без застосування ІТ опрацювати такі обсяги інформації й перетворити їх на корисні управлінські знання практично неможливо. Адже інформаційні технології забезпечують:

- побудову аналітичних панелей (dashboard) для керівників;
- формування звітів щодо ефективності окремих товарних категорій, постачальників, торговельних точок;
- аналіз рентабельності продажів, середнього чека, частоти покупок[9];
- прогнозування обсягів реалізації за допомогою статистичних моделей та алгоритмів машинного навчання [15-16].

Також показано, що інтеграція аналітичних модулів і методів машинного навчання в системи управління роздрібною торгівлею дозволяє покращити точність прогнозування попиту та підвищити ефективність планування постачань, особливо в умовах нестабільного ринку [10]. Така аналітика є основою для переходу від реактивного до проактивного управління, коли рішення приймаються не постфактум, а з урахуванням прогнозованих тенденцій.

Сучасний покупець сприймає магазин не лише як місце придбання товарів, а як комплексний сервіс. Інформаційні технології відіграють ключову роль у формуванні позитивного клієнтського досвіду.

Застосування CRM-систем (Customer Relationship Management), програм лояльності, мобільних застосунків і web-порталів дозволяє:

- накопичувати дані про історію покупок конкретного клієнта;
- формувати персоналізовані пропозиції та знижки;
- використовувати адресний e-mail-маркетинг і push-сповіщення;

- організовувати зручні канали зворотного зв’язку та сервісної підтримки

Технологія Progressive Web Application (PWA), побудова мобільних та web-інтерфейсів на основі фреймворків на кшталт Angular, роблять взаємодію клієнта з магазином більш гнучкою: покупець може переглядати каталог, перевіряти наявність товару, оформлювати замовлення й відстежувати його статус із будь-якого пристрою. Це формує основу омніканальної моделі продажів, де онлайн- та офлайн-канали взаємодіють як єдина система (наприклад, замовлення онлайн з отриманням у фізичному магазині).

Нова хвиля розвитку ІТ у роздрібній торгівлі пов’язана з впровадженням технологій Інтернету речей (IoT). Йдеться про «розумні» стелажі, RFID-мітки на товарах, датчики температури й вологості, інтерактивні цінники, які під’єднані до єдиної інформаційної системи.

Такі рішення дозволяють: по-перше, автоматично фіксувати зміну залишків на полицях; по-друге, у реальному часі передавати дані про рівень запасів до WMS/ERP-систем; по-третє, оперативно змінювати ціни з урахуванням акцій, попиту, терміну придатності; по-четверте, контролювати дотримання умов зберігання продукції (особливо для продовольчих товарів).

Тому така інтеграція IoT-рішень з корпоративними системами дозволяє скоротити втрати від прострочення та крадіжок, а також зменшити необхідні страхові запаси без зниження рівня обслуговування споживачів.

На стратегічному рівні інформаційні технології визначають здатність торговельного підприємства адаптуватися до змін ринку, швидко запускати нові формати обслуговування та масштабувати бізнес. Дудикевич В. Б. підкреслює, що сучасні автоматизовані системи управління підприємством дають змогу поєднати фінансові, логістичні, виробничі й торговельні процеси в єдине інформаційне середовище, що є ключовою умовою для реалізації цифрових стратегій розвитку.

Підприємства, які активно інвестують в ІТ-інфраструктуру, отримують низку довгострокових переваг:

- зниження операційних витрат за рахунок оптимізації процесів;
- підвищення прозорості діяльності й якості управлінського контролю;

- можливість швидко реагувати на зміни попиту, цін, логістичних умов;
- створення нових каналів продажів (інтернет-магазин, мобільний додаток, маркетплейси);
- зміцнення лояльності клієнтів за рахунок персоналізованого сервісу й зручних цифрових сервісів.

У підсумку інформаційні технології в сучасній роздрібній торгівлі виконують багатовимірну роль: від інструмента автоматизації обліку до основи побудови інтелектуальних систем підтримки прийняття рішень та «розумних» ланцюгів постачання. Саме тому під час проєктування автоматизованої системи управління постачанням і продажем продукції особливу увагу слід приділяти вибору архітектури, технологій (ERP, PWA, IoT, аналітика даних) та інтеграції між ними, оскільки від цього залежить загальна ефективність функціонування торговельного підприємства.

1.2 Характеристика бізнес-процесів постачання, зберігання та реалізації продукції в магазині

Бізнес-процеси роздрібного магазину, пов’язані з рухом товару, традиційно поділяють на три взаємопов’язані блоки: постачання, зберігання (складський облік) та реалізація (продаж). Кожен із цих блоків охоплює певну послідовність операцій, що виконуються різними підрозділами й працівниками магазину, але в сукупності формують єдиний ланцюг створення цінності для споживача. У роботах з організації торгівлі підкреслюється, що ефективність роздрібного підприємства значною мірою визначається узгодженістю та інформаційною прозорістю цих процесів: помилки або затримки на будь-якому етапі безпосередньо впливають на наявність товару на полицях, рівень витрат та задоволеність клієнтів.

Процес постачання починається з формування потреби в товарі, що базується на аналізі поточних залишків, прогнозі попиту, маркетингових планах і досвіді попередніх періодів. На цьому етапі менеджмент магазину визначає необхідний

асортимент, обсяги й періодичність постачання. Далі здійснюється вибір або узгодження умов співпраці з постачальниками (цінові умови, мінімальні партії, строки поставки, умови оплати), після чого формуються та надсилаються замовлення. Отримання товару супроводжується оформленням супровідних документів (накладні, рахунки, сертифікати якості), перевіркою відповідності фактичної поставки умовам договору (за кількістю, асортиментом, ціною, якістю) і реєстрацією надходження в обліковій системі магазину. У навчальній літературі з логістики роздрібної торгівлі зазначається, що раціональна організація цих операцій дозволяє скоротити час циклу замовлення, зменшити ймовірність дефіциту товарів та оптимізувати рівень запасів.

Процеси зберігання продукції охоплюють усі операції, пов’язані із розміщенням товарів у складських і торговельних приміщеннях, підтриманням належних умов їх зберігання та веденням кількісно-сумарного обліку. Після приймання товар розміщують у зонах зберігання згідно з обраною схемою (адресне зберігання, фіксовані чи динамічні місця, зонування за групами або швидкістю обігу). Менеджери й комірники здійснюють періодичну інвентаризацію, контролюють залишки, строки придатності, стан тари й упаковки, а також організовують переміщення товару зі складу в торговельний зал. Згідно з рекомендаціями фахової літератури, належна організація складських процесів передбачає мінімізацію зайвих переміщень, скорочення часу пошуку товару та забезпечення простежуваності руху кожної товарної позиції від моменту надходження до реалізації, що особливо важливо для продуктів із обмеженим терміном зберігання[16].

Процеси реалізації (продажу) продукції у магазині включають підготовку товарів до викладки (маркування, переупаковку за потреби, розміщення на полицях), організацію торговельного залу, консультування покупців, проведення розрахунків на касі та, за наявності, операції з повернення або обміну товару. На етапі продажу формується кінцевий грошовий потік, тому тут сконцентровано взаємодію з покупцем: персонал здійснює презентацію товару, надає інформацію про характеристики, ціну, наявні знижки або акції. Фактична реалізація

відображається в облікових системах через проведення касових операцій, списання проданих позицій із залишків та формування відповідних документів (фіскальних чеків, товарних чеків, видаткових накладних тощо). У наукових дослідженнях із організації торгівлі підкреслюється, що якість виконання операцій на цьому етапі визначає рівень задоволеності клієнтів і формує їх намір повторно звернутися до магазину, тоді як своєчасне та коректне відображення продажів у системі обліку забезпечує актуальність даних для подальшого планування постачань.

Таким чином, бізнес-процеси постачання, зберігання та реалізації продукції в роздрібному магазині формують замкнений цикл руху товару від постачальника до кінцевого споживача. Їх особливістю є тісна взаємозалежність: рішення щодо обсягів замовлення впливають на завантаженість складу, витрати на зберігання та доступність товару на полицях, а фактичні результати продажів визначають подальшу потребу в постачанні. У сучасних умовах зростаючої конкуренції та варіативності попиту ефективне управління цим комплексом процесів потребує не лише регламентованих процедур і кваліфікованого персоналу, а й підтримки з боку інформаційних систем, які забезпечують оперативний облік, аналітику й координацію дій між підрозділами магазину.

1.3 Типові проблеми ручного обліку та організації документообігу при постачанні й продажу товарів

У традиційній моделі управління роздрібним магазином значна частина операцій з постачання, зберігання та продажу товарів виконується вручну: заповнення накладних, ведення журналів обліку, перенесення даних у різні реєстри, складання звітів на папері або в окремих розрізнених електронних файлах. Такий підхід історично склався як результат поступового розширення асортименту й обсягів діяльності, але в умовах зростаючої конкуренції та складності ланцюгів постачання він стає джерелом системних проблем, що негативно впливають на

точність обліку, оперативність управлінських рішень та загальну ефективність роботи магазину [18].

Однією з найхарактерніших проблем ручного обліку є висока ймовірність помилок, пов’язаних із людським фактором. Працівники вимушені багаторазово дублювати одні й ті самі дані в різних документах – від заявок на постачання й прибуткових накладних до журналів руху товарів та звітів про продажі. Кожне повторне введення інформації підвищує ризик неточностей у кількості, номенклатурі, цінах або реквізитах контрагентів. Навіть невеликі розбіжності між фактичними залишками на складі та даними обліку з часом накопичуються, що ускладнює проведення інвентаризацій, сприяє виникненню нестач або надлишків і створює передумови для конфліктних ситуацій з постачальниками та контролюючими органами. У літературі з організації торгівлі зазначається, що в умовах ручного обліку частка помилкових записів може сягати значних величин, особливо за великої номенклатури та інтенсивного товарообігу, а їх своєчасне виявлення потребує додаткових трудових і часових витрат з боку персоналу.

Наступною типовою проблемою є низька оперативність отримання достовірної інформації. Оскільки обробка первинних документів (накладних, актів, товарних звітів, касових журналів) виконується вручну й часто із затримкою, керівництво магазину не має в будь-який момент часу актуальних даних про залишки, обсяги реалізації, дебіторську та кредиторську заборгованість. Планування постачань у такій ситуації базується на застарілих або приблизних показниках, що призводить до появи дефіциту ходових позицій чи, навпаки, накопичення надлишків малорухомого товару. Це знижує оборотність запасів, призводить до «заморожування» оборотних коштів і зростання витрат на зберігання, а в сегменті продуктів із обмеженим терміном придатності – до прямих втрат через псування продукції.

Суттєвою проблемою ручної організації документообігу є фрагментованість та дублювання інформаційних потоків. Окремі підрозділи (відділ постачання, склад, торговельний зал, бухгалтерія) часто ведуть власні журнали та реєстри, які не узгоджені між собою ні за структурою, ні за форматом записів. Це ускладнює

звірку даних, призводить до появи паралельних «версій правди» щодо стану запасів і фінансових результатів, а також робить процес підготовки зведеної звітності тривалим та трудомістким. Відсутність єдиного інформаційного простору ускладнює аналіз ефективності роботи магазинів у складі мережі, не дозволяє оперативно порівнювати показники різних торговельних точок та оцінювати результативність маркетингових заходів.

Паперовий документообіг сам по собі створює додаткові ризики та витрати. Зберігання великої кількості первинних документів потребує значного фізичного простору, організації архіву, дотримання правил збереження та доступу до документації. Документи можуть бути втрачені, пошкоджені або неправильно підшиті, що ускладнює їх пошук у разі проведення внутрішніх перевірок чи зовнішнього аудиту. Процес відшукання конкретної накладної або акта, який підтверджує ту чи іншу операцію, перетворюється на трудомістку задачу, особливо коли документообіг ведеться протягом багатьох років і не супроводжується належною систематизацією.

Ще одним наслідком переважно ручного обліку є обмежені можливості аналітичної обробки даних. Навіть якщо частина інформації надалі переноситься в електронні таблиці, її структура часто не відповідає вимогам до побудови повноцінних звітних та аналітичних форм. Відсутність централізованої бази даних унеможливлює використання сучасних засобів бізнес-аналітики, що автоматично формують звіти, діаграми, показники ефективності, моделі прогнозування попиту тощо. У результаті керівники змушені покладатися на агреговані, наближені оцінки й не можуть своєчасно виявляти тенденції зміни попиту, проблеми з окремими товарними групами чи постачальниками, а також оцінювати ефективність реалізації обраної асортиментної та цінової політики [19].

Ручна організація обліку й документообігу ускладнює також контроль і відповідальність за виконання операцій. У ситуації, коли дані про приймання, переміщення та продаж товару фіксуються різними працівниками в окремих документах, важко встановити, на якому етапі виникла помилка або порушення, хто саме відповідає за ті чи інші розбіжності, чи є ознаки зловживань. Це створює

сприятливий ґрунт для маніпуляцій із кількістю та асортиментом товару, особливо у великих магазинах із високою інтенсивністю операцій. Своєю чергою, необхідність постійного контролю з боку адміністрації призводить до додаткових витрат часу й ресурсів, не вирішуючи системну проблему відсутності прозорого, інтегрованого обліку.

Узагальнюючи, можна зробити висновок, що ручний облік та паперовий документообіг при постачанні й продажу товарів характеризуються високою трудомісткістю, схильністю до помилок, низькою оперативністю та обмеженими аналітичними можливостями. Умови сучасного ринку, що потребують швидкого реагування на зміни попиту, гнучкого управління запасами та прозорості фінансових потоків, роблять такі підходи неефективними й стимулюють перехід до комплексних автоматизованих інформаційних систем, які забезпечують єдиний простір даних і підтримку всіх ключових бізнес-процесів роздрібного магазину.

1.4 Огляд і класифікація програмних рішень для автоматизації торгівлі

У сучасній роздрібній торгівлі автоматизація ґрунтується не на одному програмному продукті, а на сукупності спеціалізованих рішень, які охоплюють різні аспекти діяльності підприємства: управління ланцюгом постачання, складською логістикою, процесами продажу, ресурсами підприємства та взаєминами з клієнтами. У зарубіжних та вітчизняних джерелах такі рішення найчастіше описуються через п’ять базових класів систем: SCM (Supply Chain Management), WMS (Warehouse Management System), POS (Point of Sale), ERP (Enterprise Resource Planning) та CRM (Customer Relationship Management), що зображено на рисунку 1.1.

Рисунок 1.1 - Класифікація базових класів систем

1.4.1 Supply Chain Management

Системи управління ланцюгами постачання (SCM) орієнтовані на планування, координацію та контроль усіх етапів руху товару – від виробника й оптових посередників до роздрібних магазинів та кінцевого споживача. Вони забезпечують функції прогнозування попиту, формування замовлень постачальникам, планування транспортування, вибору маршрутів, узгодження строків поставок, а також аналізу витрат по всьому ланцюгу постачання, що зображено на рисунку 1.2. У роздрібній торгівлі SCM-рішення дозволяють синхронізувати запаси на центральних складах і в магазинах, мінімізувати дефіцит ходових позицій та надлишки повільнообертованого товару [20].

Рисунок 1.2 - Система управління ланцюгами постачання

1.4.2 Warehouse Management System

Системи управління складом (WMS) є спеціалізованими програмними засобами, що відповідають за оперативне керування складською логістикою: прийманням, розміщенням, внутрішніми переміщеннями, комплектацією та відвантаженням товарів. Вони підтримують адресне зберігання, облік місць розташування, оптимізацію маршрутів відбору, проведення інвентаризацій, інтегруються з терміналами збору даних і іншими пристроями, що зображено на рисунку 1.3.

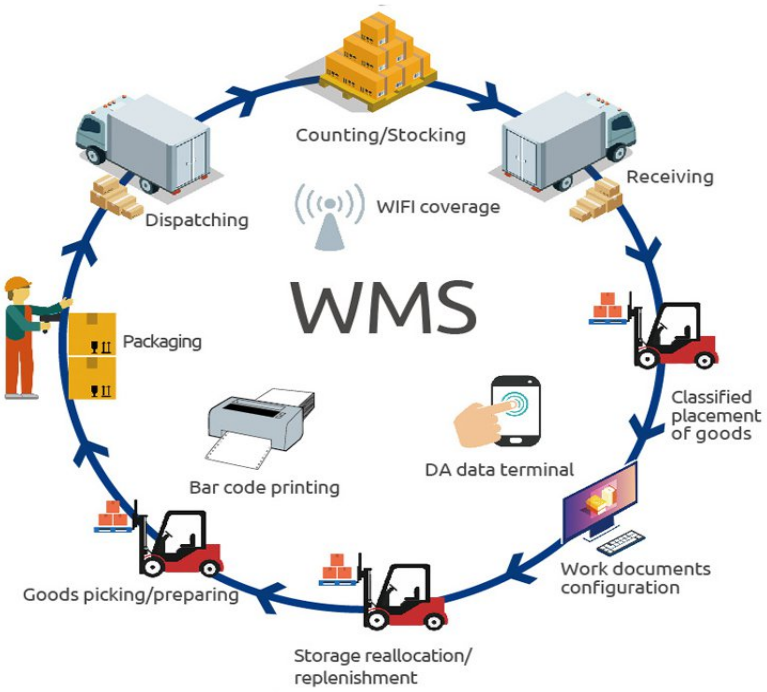

Рисунок 1.3 - Система управління складом

У літературі WMS визначають як цифрове рішення для моніторингу та управління всіма процесами переміщення й зберігання ресурсів у складі в режимі, близькому до реального часу [21].

1.4.3 Point of Sale

POS-системи (Point of Sale) призначені для автоматизації операцій безпосередньо в точці продажу – на касах, терміналах самообслуговування чи мобільних робочих місцях продавця. Вони забезпечують реєстрацію продажів, сканування штрих-кодів, формування фіскальних чеків, облік повернень і знижок, інтеграцію з платіжними сервісами та, як правило, синхронізуються з центральними системами обліку запасів та ERP. У сучасних рішеннях POS-платформи працюють у тісному зв’язку з ERP: зміни цін, асортименту чи акцій, що вносяться в ERP, автоматично передаються на всі POS-термінали, а дані про

продажі повертаються у вигляді транзакцій для подальшого аналізу й бухгалтерського відображення[22].

1.4.4 Enterprise Resource Planning

Системи планування ресурсів підприємства (ERP) є ядром інформаційної інфраструктури торговельного підприємства. ERP інтегрує в єдиному середовищі фінансовий облік, управління закупівлями, продажами, запасами, персоналом, основними засобами та іншими ресурсами компанії. Для роздрібної торгівлі характерні спеціалізовані ERP-рішення (retail ERP), що містять галузеві модулі: управління товарним асортиментом, ціноутворення, промоакції, аналіз прибутковості, облік мережевих магазинів тощо, як зображено на рисунку 1.4.

Рисунок 1.4 - Система планування ресурсів підприємств

Такі системи працюють у режимі реального часу, забезпечують централізоване управління всіма бізнес-процесами й слугують «єдиним джерелом правди» для даних про запаси, продажі, фінансові результати та взаємодію з клієнтами [23].

1.4.5 Customer Relationship Management

Системи управління взаєминами з клієнтами (CRM) фокусуються на збиранні та використанні даних про покупців: історії покупок, канали комунікації, реакцію на акції, звернення до служби підтримки тощо. Їх основна мета – підвищення лояльності та «життєвої цінності» клієнта за рахунок персоналізованих пропозицій, програм лояльності, сегментованих маркетингових кампаній. CRM-системи зберігають, сортують і надають дані, які застосовуються для оптимізації бізнес-процесів і стратегій продажів, у той час як ERP охоплює ширший спектр внутрішніх процесів підприємства. У роздрібній торгівлі CRM часто інтегрується з POS і онлайн-каналами продажу (інтернет-магазин, мобільний застосунок), формуючи єдину картину взаємодії клієнта з брендом незалежно від того, де саме була здійснена покупка [24].

Загальна тенденція розвитку програмних рішень для автоматизації торгівлі полягає в переході від розрізнених, ізольованих систем до комплексних, інтегрованих платформ, де ERP виступає «центром тяжіння», а модулі SCM, WMS, POS і CRM реалізуються або як окремі спеціалізовані продукти з тісною інтеграцією, або як підсистеми в рамках єдиного програмного комплексу. Для роздрібних компаній це означає можливість побудови наскрізного інформаційного простору, у межах якого рішення щодо постачання, викладки товарів, цінової політики й маркетингових активностей приймаються на основі узгоджених, актуальних даних із усіх ділянок бізнесу [25].

1.5 Вимоги до автоматизованої системи підтримки процесів постачання та продажу продукції в магазині

Вимоги до автоматизованої системи постачання та продажу продукції в магазині формуються з урахуванням особливостей роздрібної торгівлі, описаних

бізнес-процесів та типових проблем ручного обліку. Загалом їх доцільно поділити на функціональні, інформаційні, технологічні та експлуатаційні. Такий підхід відповідає загальноприйнятій структурі вимог до інформаційних систем, що подається в працях з проєктування та впровадження автоматизованих систем управління підприємством.

З функціональної точки зору система повинна підтримувати повний цикл операцій, пов’язаних із постачанням, зберіганням та реалізацією товарів у роздрібному магазині. До мінімального набору функцій належить ведення довідників номенклатури (товарів), постачальників, користувачів; формування та реєстрація замовлень постачальникам; облік надходжень товарів; відображення внутрішніх переміщень між складом і торговельним залом; реєстрація операцій продажу з прив’язкою до товарних позицій; ведення історії замовлень і поточних залишків. Окремо система має забезпечувати можливість фіксації основних реквізитів документів (номер, дата, контрагент, суми, ставки податків) та необхідних статусів (створено, у роботі, виконано, скасовано тощо). Функціональні можливості мають бути організовані таким чином, щоб користувачі різних ролей (адміністратор, менеджер із постачання, касир/оператор) мали доступ до релевантних їм операцій, не перевантажуючи інтерфейс зайвими елементами. У літературі з автоматизації торговельних підприємств підкреслюється, що саме чітка підтримка ключових бізнес-процесів і розподіл функцій між ролями користувачів є базовою вимогою до галузевих інформаційних систем роздрібної торгівлі.

Інформаційні вимоги стосуються структури й цілісності даних, з якими працює система. Модель даних повинна відображати основні сутності предметної області (товар, постачальник, замовлення, надходження, операція продажу, користувач) та зв’язки між ними, забезпечуючи однозначну ідентифікацію кожного об’єкта, уникнення дублювання записів і можливість відстеження «життєвого циклу» товару від моменту надходження до реалізації. Важливою вимогою є актуальність даних: інформація про залишки, стан замовлень, історію операцій має оновлюватися оперативно та бути доступною в режимі, наближеному до реального

часу, що є принциповим для коректного планування постачань і контролю запасів. Також система повинна підтримувати зберігання мінімально необхідного набору атрибутів (артикул, штрих-код, найменування, ціна закупівлі та продажу, група товару, одиниця виміру тощо), але мати можливість розширення структури даних без радикальної перебудови всієї системи, що відповідає рекомендаціям щодо побудови гнучких інформаційних моделей у торговельних інформаційних системах.

Технологічні вимоги визначають обраний стек технологій і архітектурні підходи до реалізації системи. Для веб-орієнтованого рішення з використанням Angular та PWA важливими є підтримка кросплатформності (можливість роботи в сучасних браузерах на різних операційних системах), адаптивний інтерфейс, що коректно відображається на екранах різної роздільної здатності, та можливість встановлення застосунку на робочий стіл як прогресивного веб-застосунку. Архітектура має відповідати принципам клієнт–серверної взаємодії: клієнтська частина відповідає за відображення інтерфейсу та первинну обробку даних, тоді як серверна (у разі використання хмарної платформи, наприклад Firebase/Firestore) забезпечує зберігання, авторизацію та централізовану обробку інформації. До технологічних вимог також належать використання сучасних засобів розробки (TypeScript, компонентний підхід в Angular, реактивні бібліотеки), можливість розширення системи за рахунок додавання нових модулів і інтеграції з зовнішніми сервісами (наприклад, платіжними або аналітичними). У працях, присвячених впровадженню вебзастосунків у торгівлі, наголошується на важливості вибору таких технологій, що забезпечують баланс між продуктивністю, зручністю розробки й підтримкою довгострокової еволюції системи.

Експлуатаційні вимоги пов’язані з особливостями використання системи кінцевими користувачами в умовах реального магазину. Інтерфейс має бути інтуїтивно зрозумілим і не вимагати тривалого навчання персоналу; основні операції (створення замовлення, облік надходження, оформлення продажу) повинні виконуватися в мінімальну кількість кроків. Важливим є забезпечення стабільної роботи за умов нестабільного мережевого з’єднання: PWA-застосунок

має дозволяти продовжувати частину операцій в офлайн-режимі з подальшою синхронізацією даних при відновленні доступу до мережі. До експлуатаційних вимог належать також можливість простого оновлення версій застосунку без втручання користувача, підтримка резервного копіювання даних на рівні обраної хмарної платформи, ведення журналів подій для відслідковування основних дій користувачів. У зарубіжних джерелах підкреслюється, що саме зручність і надійність експлуатації часто є вирішальними факторами успіху або провалу проєктів автоматизації в роздрібних мережах, навіть за формально правильно спроєктованої функціональності.

Окремо варто виділити вимоги до інтегрованості й масштабованості системи. Оскільки роздрібний бізнес може розвиватися – від одного магазину до мережі торговельних точок – система повинна підтримувати розширення без радикального перепроєктування: перехід від локальної конфігурації до хмарної або гібридної, додавання нових ролей та функцій, підключення додаткових точок доступу (нові робочі місця, мобільні пристрої). Інтегрованість передбачає можливість обміну даними з іншими інформаційними системами підприємства (бухгалтерським обліком, інтернет-магазином, CRM-системою, аналітичними сервісами) через стандартні протоколи та API, що відповідає загальній тенденції до побудови єдиного інформаційного простору підприємства роздрібної торгівлі.

Таким чином, вимоги до автоматизованої системи підтримки процесів постачання та продажу продукції в магазині можна сформулювати як вимоги до комплексного, інтегрованого, гнучкого та зручного в експлуатації рішення, що забезпечує повний цикл обліку руху товару, підтримує актуальність і цілісність даних, ґрунтується на сучасних веб-технологіях та має потенціал подальшого розвитку разом із зростанням торговельного підприємства.

1.6 Висновки до першого розділу

У першому розділі було досліджено теоретичні засади автоматизації процесів постачання та продажу продукції в роздрібному магазині. Показано, що в умовах цифрової економіки інформаційні технології відіграють ключову роль у підвищенні ефективності роздрібної торгівлі: саме вони забезпечують оперативний облік руху товарів, інтеграцію різних бізнес-процесів, підтримку управлінських рішень на основі даних і формування якісного клієнтського сервісу. Використання сучасних ІТ-рішень дозволяє мінімізувати вплив людського фактора, скоротити час виконання операцій, підвищити прозорість діяльності та конкурентоспроможність торговельного підприємства.

Проаналізовано структуру основних бізнес-процесів роздрібного магазину, пов’язаних із постачанням, зберіганням та реалізацією продукції. Встановлено, що ці процеси утворюють єдиний логістичний ланцюг «постачальник – склад – торговельний зал – покупець», у якому результати продажів визначають потребу в нових поставках, а організація складського господарства безпосередньо впливає на швидкість та якість обслуговування. Показано, що за умови переважно ручного ведення обліку й паперового документообігу виникає низка типових проблем: висока ймовірність помилок, затримки в отриманні актуальних даних, фрагментованість інформаційних потоків, складність інвентаризації та обмежені аналітичні можливості. Сукупність цих факторів робить традиційні підходи до управління постачанням і продажами неефективними в сучасних умовах.

На основі огляду й класифікації програмних рішень для автоматизації торгівлі було виділено п’ять ключових класів систем: SCM, WMS, POS, ERP та CRM. Показано, що вони покривають різні рівні та аспекти функціонування торговельного підприємства – від операційного обліку продажів і складської логістики до управління ланцюгами постачання, ресурсами підприємства та взаєминами з клієнтами. Сформульовано висновок про тенденцію до інтеграції цих

систем у єдині комплексні платформи, що забезпечують наскрізний інформаційний простір для прийняття управлінських рішень.

На завершення розділу було сформульовано вимоги до автоматизованої системи підтримки процесів постачання та продажу продукції в магазині, яка розробляється в межах магістерської роботи. Визначено основні функціональні вимоги (підтримка повного циклу операцій із постачання, зберігання та продажу, робота з довідниками, замовленнями, надходженнями, операціями реалізації, історією замовлень), інформаційні вимоги (цілісність і актуальність даних, гнучка модель сутностей предметної області), технологічні вимоги (використання вебтехнологій, архітектура клієнт–сервер, підтримка Angular та PWA, інтеграція з хмарною платформою зберігання даних) та експлуатаційні вимоги (зручність інтерфейсу, можливість роботи за нестабільного з’єднання, масштабованість та інтегрованість). Сформований у цьому розділі теоретичний та концептуальний базис є основою для подальшого математичного та інформаційного моделювання системи, а також для її проєктування та реалізації в наступних розділах магістерської роботи.

2 МАТЕМАТИЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Формалізація бізнес-процесів постачання та продажу продукції

Формалізація бізнес-процесів постачання та продажу продукції передбачає перехід від текстового або інтуїтивного опису діяльності магазину до чітко структурованих моделей, які відображають послідовність операцій, потоки матеріальних та інформаційних ресурсів, залучені ролі та точки прийняття рішень. У науковій літературі підкреслюється, що така формалізація є необхідною передумовою для подальшої автоматизації, оскільки дозволяє виявити «вузькі місця» процесів, усунути дублювання операцій, стандартизувати документообіг і визначити вимоги до функціональності інформаційної системи.

Для опису бізнес-процесів у сучасній практиці найчастіше застосовують нотації BPMN (Business Process Model and Notation), UML-діаграми діяльності, а також методології сімейства IDEF (зокрема IDEF0 та IDEF3). У роботах, присвячених моделюванню бізнес-процесів, відзначається, що BPMN є де-факто стандартом для графічного опису процесів завдяки своїй орієнтації на бізнес-користувачів і можливості деталізувати як послідовність дій, так і потоки повідомлень між підрозділами [26]. Методологія IDEF0, у свою чергу, зручна для побудови функціональних моделей, у яких кожна діяльність подається у вигляді блоку з чотирма типами дуг: вхід, вихід, керуючі впливи та механізми (ресурси), що дозволяє наочно показати, які документи, накази чи дані запускають процес і які результати він формує [27].

Безпосередні процеси постачання продукції доцільно формалізувати як послідовність етапів: виявлення потреби (аналіз залишків і прогноз попиту), формування та погодження замовлення постачальнику, оформлення договірних умов, приймання товару за кількістю та якістю, реєстрація надходження в обліковій системі. Для їх опису зручно використовувати BPMN-діаграми або UML-діаграми діяльності, де окремими блоками відображаються дії відповідальних осіб (менеджера з постачання, бухгалтера, комірника), а стрілки показують логічну

послідовність операцій та альтернативні гілки (наприклад, «товар відповідає умовам / товар не відповідає – оформлення рекламації»).

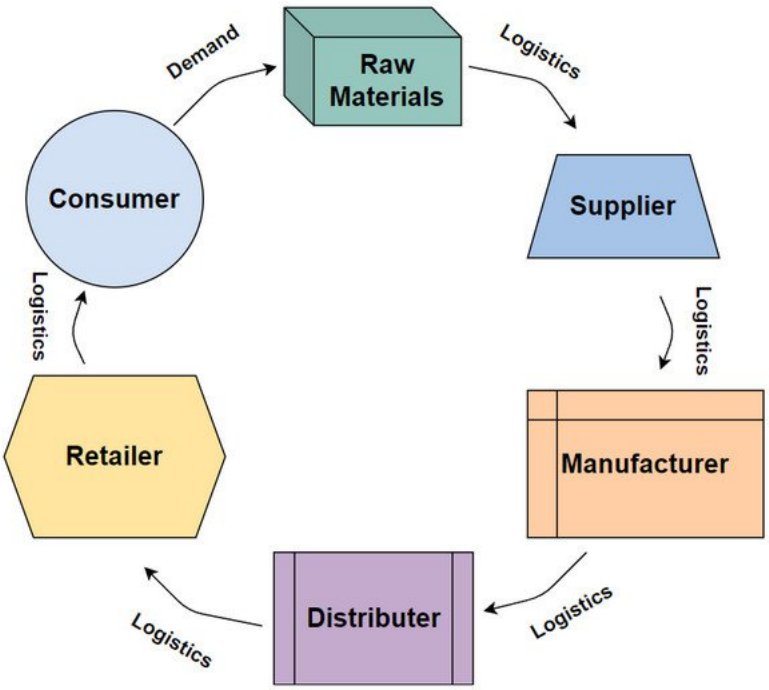

Рисунок 2.1 - Спрощення використання схеми ланцюга поставок

У роботах, присвячених моделюванню бізнес-процесів ланцюгів постачання, як зображено на рисунку 2.1, підкреслюється, що саме BPMN дозволяє узгодити бачення процесу між бізнес-користувачами та розробниками інформаційних систем, оскільки поєднує зрозумілу графіку з формальними можливостями подальшої автоматизації [28].

Процеси продажу продукції в моделях відображаються як потік операцій від появи запиту покупця до завершення розрахунків і фіксації продажу в системі. Формалізований опис зазвичай включає такі дії: консультація клієнта, підбір товару, перевірка наявності на складі, резервування (за потреби), оформлення чеку, оплата (готівкова, безготівкова, через платіжні сервіси), видача товару та, у разі повернення, – зворотні операції з коригування залишків і фінансових показників. Цей процес може бути представлений у вигляді BPMN-діаграми із зазначенням «пулів» і «лейн» (swimlanes), яка зображена на рисунку 2.2, що відповідають різним

ролям: покупець, продавець, касир, інформаційна система. Таке подання дозволяє чітко побачити точки взаємодії з IT-системою (сканування штрих-коду, перевірка залишків, реєстрація платежу), які надалі мають бути підтримані в автоматизованому режимі [29].

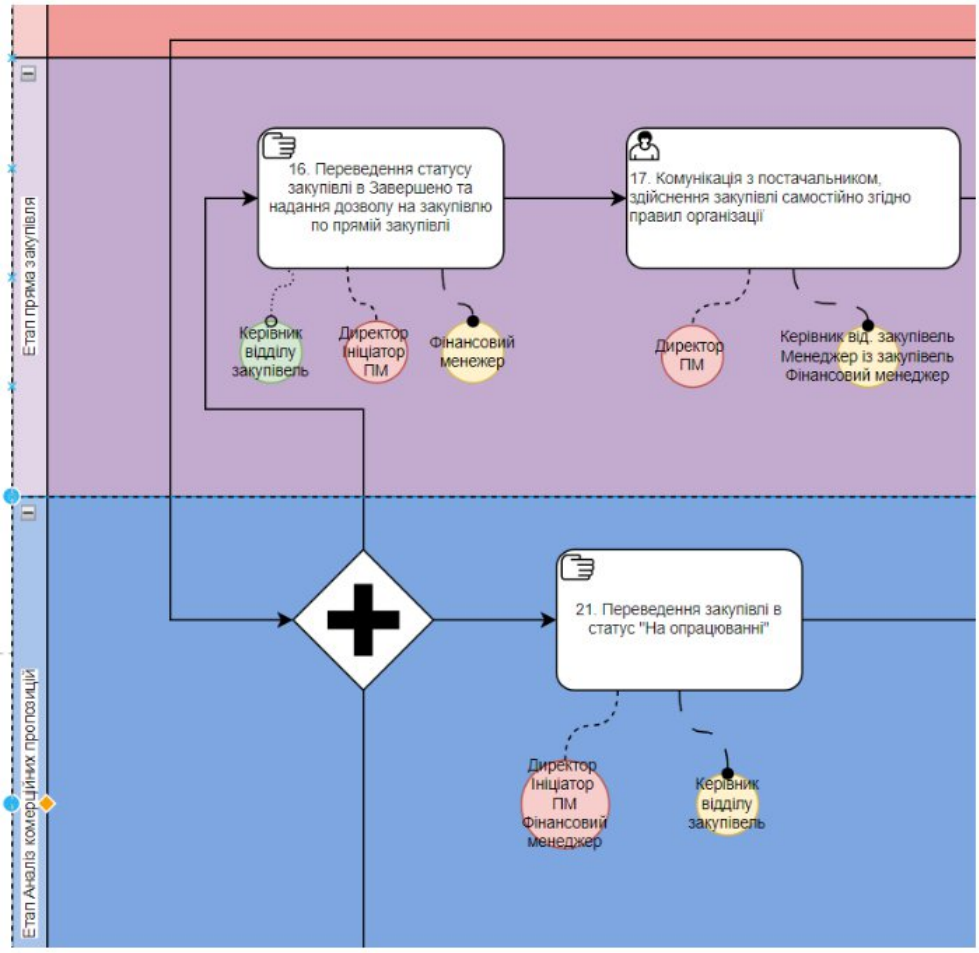

Рисунок 2.2 - BPMN-діаграма процесу продажу продукції в магазині

Формалізація бізнес-процесів постачання й продажу також передбачає визначення подій і станів, між якими відбуваються переходи. Наприклад, замовлення може перебувати в станах «створене», «на погодженні», «підтверджене постачальником», «у дорозі», «отримане», «закрите», а замовлення клієнта – у станах «нове», «оплачене», «виконане», «скасоване». Такі стани зручно описувати за допомогою діаграм станів UML або відповідних анотацій у BPMN, що надалі спрощує реалізацію логіки зміни статусів у програмному забезпеченні (наприклад,

у вигляді переліку дозволених переходів між статусами). Формальні моделі станів допомагають уникнути некоректних переходів (наприклад, від «нового» замовлення одразу до «виконаного» без етапу оплати чи відправки), що позитивно впливає на цілісність даних.

Ще одним важливим аспектом формалізації є узгодження рівнів деталізації моделей. На верхньому рівні доцільно використовувати IDEF0 або узагальнені BPMN-діаграми, що описують основні функції системи («Управління постачанням», «Управління складом», «Управління продажами») та взаємозв’язки між ними. На нижчих рівнях кожна з цих функцій декомпонується до конкретних процедур: оформлення замовлення постачальнику, перевірка якості поставки, списання товару при продажу тощо. У літературі зазначається, що така багаторівнева деталізація дозволяє одночасно зберігати цілісне бачення діяльності підприємства й отримувати достатньо інформації для постановки задач розробникам інформаційної системи [30].

Таким чином, формалізація бізнес-процесів постачання та продажу продукції в магазині базується на використанні стандартних нотацій (BPMN, UML, IDEF0/IDEF3), які дозволяють описати послідовність дій, інформаційні та матеріальні потоки, ролі учасників і стани об’єктів. Отримані моделі є не лише засобом візуалізації, а й основою для подальшого математичного опису, побудови інформаційної моделі й визначення функціональних вимог до автоматизованої системи, що розробляється в межах магістерської роботи.

2.2 Моделі управління товарними запасами в роздрібному магазині

Управління товарними запасами є одним із ключових завдань роздрібного магазину, оскільки саме рівень і структура запасів визначають здатність підприємства задовольняти попит покупців, уникати дефіциту та мінімізувати витрати, пов’язані зі зберіганням, заморожуванням оборотних коштів і списанням продукції. У теорії та практиці логістики для опису та оптимізації управління запасами застосовується низка математичних моделей, які можна адаптувати до умов роботи роздрібних торговельних підприємств.

Класичною відправною точкою є модель економічного розміру замовлення (EOQ – Economic Order Quantity), яка передбачає, що попит на товар протягом періоду є сталим, поповнення запасу відбувається миттєво, а витрати на оформлення одного замовлення та витрати на зберігання одиниці товару впродовж періоду залишаються незмінними. Метою моделі є визначення такого розміру партії замовлення, за якого сумарні витрати на зберігання і розміщення замовлень мінімальні. За цих припущень оптимальний розмір замовлення Q∗ визначається формулою

$$Q^* = \sqrt{\left(\frac{2DS}{H}\right)}, \qquad (1),$$

де D – річний (або за інший період) попит;

S – витрати на оформлення одного замовлення;

H – витрати на зберігання одиниці товару за період.

У наукових працях із логістики та управління запасами підкреслюється, що, попри спрощеність, модель EOQ дає корисні орієнтири для прийняття рішень і широко використовується як основа для побудови більш складних варіантів, які враховують сезонність попиту, знижки за обсяг, обмеження за потужністю складу тощо.

У реальних умовах роздрібної торгівлі попит, як правило, не є сталим, а постачання можуть мати змінний інтервал і розмір. Тому EOQ-модель доповнюють механізмом точки замовлення (reorder point), яка визначає рівень запасу, за досягнення якого необхідно ініціювати нове замовлення. У найпростішому випадку точка замовлення R обчислюється як

$$R = dL = d * L \quad (2),$$

де d – середній попит за одиницю часу;

L – середній час постачання (lead time).

За наявності невизначеності в попиті або в часі поставки до точки замовлення додають страховий запас, який покликаний зменшити імовірність дефіциту в період очікування постачання. Розрахунок страхового запасу ґрунтується на статистичних характеристиках попиту (середнє значення, стандартне відхилення) та бажаному рівні сервісу, що є предметом більш розгорнутих моделей управління запасами, описаних у фаховій літературі з логістики й операційного менеджменту.

Важливе значення для роздрібного магазину має клас моделей періодичного та безперервного контролю запасів. У системах безперервного контролю (модель типу (s,Q)(s, Q)(s,Q)) рівень запасу відстежується постійно, і щоразу, коли він досягає або опускається нижче певного порогу s, здійснюється замовлення фіксованого обсягу Q. Такий підхід дає змогу чутливо реагувати на зміни попиту, але потребує постійного моніторингу, що традиційно було складно реалізувати без автоматизації. У системах періодичного контролю (модель типу (R,S)(R, S)(R,S)) стан запасів перевіряється через рівні інтервали часу R, а обсяг замовлення визначається як різниця між заданим максимальним рівнем S і поточним залишком. Цей підхід простіше впроваджувати організаційно, але він менш чутливий до різких змін попиту між моментами контролю, що може бути критичним для швидкообертових товарів. Вибір між (s,Q)(s, Q)(s,Q) та (R,S)(R, S)(R,S) у роздрібній

торгівлі залежить від категорії товарів, можливостей облікової системи та вимог до рівня сервісу.

2.2.1 Класичні моделі оптимізації запасів

Класичні моделі оптимізації запасів спрямовані на пошук такого обсягу замовлення та такої політики поповнення, які мінімізують сумарні витрати підприємства на оформлення замовлень, зберігання запасів і покриття можливих дефіцитів. Базовою відправною точкою є модель економічного розміру замовлення (EOQ – Economic Order Quantity), яка розглядає ситуацію з постійним попитом, незмінними витратами й миттєвим поповненням запасу. У цій моделі, що визначається за формулою

$$C(Q) = (D/Q) \cdot S + (Q/2) \cdot H \qquad (3),$$

де C(Q) - сумарні витрати на оформлення замовлень та зберігання запасів за розглядувальний період;

D - річний попит на товар;

Q - розмір партії замовлення;

S - витрати на оформлення одного замовлення;

H - витрати на зберігання одиниці товару за період.

Мінімізуючи цю функцію за Q, отримують оптимальний економічний розмір замовлення за формулою:

$$Q^* = \sqrt{\frac{2DS}{H}} \quad (4),$$

де Q - оптимальний (економічний) розмір замовлення;

D - річний попит на товар;

S - витрати на оформлення одного замовлення;

H - витрати на зберігання одиниці товару за період.

Ця формула широко наводиться в літературі з логістики та операційного менеджменту як базовий інструмент для орієнтовної оцінки оптимальної партії постачання. У практиці роздрібної торгівлі модель EOQ часто використовується не в «чистому» вигляді, а як концептуальна основа для побудови більш складних схем, які враховують сезонність попиту, знижки за обсяг закупівлі, обмежену ємність складу чи мінімальні партії постачальників.

2.2.2 ABC/XYZ-аналіз та їх застосування у практиці роздрібної торгівлі

ABC/XYZ-аналіз належить до інструментів класифікаційного управління запасами, які дозволяють роздрібному магазину розподілити асортимент на групи з різним рівнем пріоритету та, відповідно, застосовувати диференційовані політики постачання, контролю та прогнозування. На відміну від універсальних моделей типу EOQ, ABC/XYZ-підхід виходить із того, що різні товарні позиції мають неоднаковий вплив на оборот, прибутковість і стабільність попиту, тому облікові й управлінські ресурси слід концентрувати насамперед на «найважливіших» позиціях.

ABC-аналіз базується на принципі Парето й передбачає ранжування товарів за обраним критерієм (найчастіше – річний обсяг продажу в грошовому виразі або валовий прибуток) та поділ їх на три категорії:

- група A – відносно невелика частка позицій (приблизно 10–20 % номенклатури), які формують 70–80 % обороту чи вартості запасів;
- група B – проміжна група (20–30 % позицій, що забезпечують 15–25 % обороту);

- група C – численні товари (50–70 % номенклатури), що дають лише 5–10 % загального обороту.

Практичний алгоритм полягає в тому, що обчислюється вклад кожної позиції у сумарний оборот, товари сортуються за спаданням, після чого формується кумулятивна крива й обираються пороги поділу на A/B/C. У роздрібній торгівлі це дозволяє: для групи A встановлювати найжорсткіший контроль рівня запасів, частий перегляд параметрів замовлень, застосовувати точніші моделі прогнозу; для групи B – підтримувати помірний контроль; для групи C – мінімізувати увагу та витрати, допускаючи більші коливання запасів і використовуючи спрощені схеми поповнення (наприклад, замовлення «до певного рівня» раз на місяць).

XYZ-аналіз доповнює ABC-підхід, класифікуючи товари за стабільністю попиту. Зазвичай використовують три категорії:

- X – позиції зі стабільним, добре прогнозованим попитом і невеликим коефіцієнтом варіації (наприклад, базові продукти щоденного попиту);
- Y – товари з помірною варіативністю попиту, часто із сезонними коливаннями (наприклад, напої, сезонний одяг, товари до свят);
- Z – позиції з нерегулярним, важкопрогнозованим попитом, для яких статистичні моделі дають низьку точність прогнозу (рідковживані або специфічні товари).

Класифікація X/Y/Z зазвичай виконується на основі коефіцієнта варіації або величини прогнозної похибки за історичними даними. У результаті кожна товарна позиція отримує подвійну належність – наприклад, AX, BY, CZ тощо. Саме така матриця ABC/XYZ дозволяє розробити диференційовані політики управління запасами:

- для груп AX (дорогі, високообертові та стабільні товари) доцільно поєднувати точний прогноз із моделями безперервного контролю (s,Q)(s, Q)(s,Q), підтримуючи невеликі страхові запаси й мінімізуючи дефіцити;
- для груп AY та BX – використовувати комбіновані підходи з урахуванням сезонності (періодичний перегляд, коригування параметрів перед піковими періодами);

- для груп BZ, CY, CZ – утримувати мінімальні запаси або працювати за принципом «під замовлення» клієнта, оскільки витрати на зберігання таких товарів можуть перевищувати вигоду від їхньої постійної наявності.

Таким чином, ABC/XYZ-аналіз виступає практичним інструментом, який доповнює класичні моделі оптимізації запасів і дозволяє адаптувати політику управління запасами до різних груп товарів з огляду на їхню важливість і прогнозованість попиту. Для розроблюваної автоматизованої системи це означає необхідність підтримки механізмів класифікації товарів та гнучкого налаштування параметрів поповнення для різних категорій, що буде враховано при побудові інформаційної моделі й функціоналу програмного забезпечення.

2.3 Визначення системи показників ефективності функціонування автоматизованої системи

Оцінювання ефективності автоматизованої системи підтримки процесів постачання та продажу продукції в магазині потребує формування цілісної системи показників (KPI), які б одночасно відображали якість роботи логістичних процесів, стан товарних запасів, рівень сервісу для клієнтів і результативність самої інформаційної системи як інструмента управління. У сучасних підходах до управління ланцюгами постачання й запасами KPI розглядаються як набір метричних показників, що дозволяють вимірювати ступінь досягнення цілей у сфері логістики, фінансів та інформаційної підтримки процесів [32].

До першої групи доцільно віднести показники, що характеризують ефективність управління товарними запасами та рухом продукції. Ключовим серед них є коефіцієнт оборотності запасів (inventory turnover), який показує, скільки разів за період (рік, квартал) запас повністю «обертається», тобто продається й поповнюється знову. Його типовим визначенням є

$$IT = \frac{C_s}{I_{avg}} (5),$$

де IT - коефіцієнт оборотності запасів;

C_s - собівартість реалізованої продукції а аналізований період;

I_{avg} - середній запас товарів за цей період.

Тісно пов'язаний із ним показник – середня кількість днів зберігання запасів, що дає уявлення про те, на скільки днів поточних продажів вистачить наявних запасів:

$$D_{inv} = \frac{I_{ivg}}{C_s} \cdot 365 (6)$$

де D_{inv} - середня кількість днів наявності запасів;

I_{avg} - середній запас товарів за період;

C_s - собівартість реалізованої продукції за період.

Цей показник допомагає оцінити, чи не є запаси надто «важкими» для підприємства [33-34].

До цієї ж групи належать показники доступності товару та рівня сервісу, зокрема рівень відсутності товару на складі (stockout rate) та рівень виконання замовлень (fill rate). Рівень відсутності товару на складі характеризує частоту ситуацій, коли товар відсутній у момент попиту, і зазвичай обчислюється як відношення кількості випадків або часу відсутності товару до загального числа запитів чи тривалості періоду; високі значення цього показника негативно впливають на задоволеність клієнтів і продажі [31].

Рівень замовлень, навпаки, відображає, наскільки повно система здатна задовольнити замовлення клієнта «з першої спроби» (у потрібній кількості та номенклатурі) й часто подається у відсотках:

$$FR = \frac{N_{full}}{N_{tot}} \cdot 100 (7),$$

де FR - коефіцієнт виконання замовлень, %;

N_{full} - кількість замовлень, виконаних повністю;

N_{tot} - загальна кількість замовлень за аналізований період.

У поєднанні з такими логістичними показниками, як час циклу замовлення (order cycle time) та частка своєчасних поставок (on-time delivery rate), ці індикатори дозволяють кількісно оцінити, наскільки автоматизована система реально підвищує швидкість і надійність процесів постачання та відвантаження товарів[34].

Друга група стосується економічної ефективності функціонування системи. До неї можна віднести частку витрат на запаси й логістику у виручці, сукупні витрати ланцюга постачання на одиницю продукції або на гривню обороту, а також показники прибутковості, пов’язані з запасами (наприклад, маржа на вкладений у товар капітал). У спеціалізованих оглядах KPI для ланцюгів постачання підкреслюється, що систематичне використання таких показників дозволяє не лише фіксувати ефект від автоматизації (зменшення операційних витрат, скорочення заморожених у запасах коштів), а й порівнювати різні товарні групи та виділяти сегменти з надмірно високою «вартістю ланцюга».

Третя група показників пов’язана безпосередньо з ефективністю інформаційної системи як програмного рішення. Для цього доцільно спиратися на відомі моделі оцінювання успіху інформаційних систем, зокрема модель DeLone & McLean, яка виділяє такі ключові групи показників: якість системи, якість інформації, якість сервісу, використання системи, задоволеність користувачів та сукупний вплив на індивідуальному й організаційному рівнях. У контексті розроблюваної системи підтримки постачання й продажу до показників якості системи можна віднести: середній час відгуку інтерфейсу, частоту відмов, доступність системи (відсоток часу безвідмовної роботи), коректність виконання операцій (кількість виявлених помилок на певну кількість транзакцій). Якість інформації може оцінюватися через точність та повноту облікових даних (відсоток розбіжностей між обліковими та фактичними залишками, частота потрібних коригувань), актуальність (затримка між реальними подіями й відображенням у

системі) і зрозумілість сформованих звітів для користувачів. Показники використання та задоволеності включають інтенсивність роботи користувачів із системою (кількість операцій, виконаних через систему, частка «обхідних» процедур поза системою), результати опитувань персоналу щодо зручності інтерфейсу, простоти навчання й швидкості виконання типових задач[36].

Четверта група – інтегральні показники «до/після» автоматизації, які дозволяють кількісно продемонструвати вплив впровадженої системи на діяльність магазину. До них можуть належати: зміна середнього рівня товарних запасів (у днях обороту), зменшення частки надлишкових запасів, скорочення часу обробки одного замовлення чи оформлення продажу, зниження частки помилок у документах та розбіжностей під час інвентаризації, покращення рівня виконання замовлень і зменшення частоти відсутності товару. Як показують практичні дослідження, правильно побудована система KPI дозволяє фіксувати до 15–20 % приросту операційної ефективності в ланцюгах постачання за рахунок прозорості процесів, підвищення точності даних та скорочення непотрібних операцій [32-34].

Таким чином, система показників ефективності функціонування автоматизованої системи підтримки постачання та продажу продукції має включати:

- логістичні KPI (оборотність запасів, дні зберігання, stockout rate, fill rate, час циклу замовлення, частка своєчасних та повних поставок);
- економічні показники (витрати ланцюга постачання, частка логістичних витрат у виручці, показники прибутковості запасів);
- показники якості та успішності інформаційної системи (якість системи, якість інформації, використання, задоволеність користувачів, організаційні вигоди);
- інтегральні «до/після» показники, що фіксують досягнутий ефект від впровадження системи.

У подальших розділах частину з цих показників буде конкретизовано з урахуванням даних, які реально формує розроблювана PWA-система на базі

Angular та хмарного сховища, а також використано для порівняння варіантів роботи магазину до й після автоматизації.

2.4 Побудова інформаційної моделі предметної області

Побудова інформаційної моделі предметної області є ключовим етапом проєктування автоматизованої системи підтримки процесів постачання та продажу продукції, оскільки саме на цьому етапі визначається, які об’єкти реального світу будуть відображені в системі, які дані про них необхідно зберігати та як ці об’єкти взаємодіють між собою. Інформаційна модель слугує «містком» між описаними в розділі 1 бізнес-процесами й структурою бази даних та API, які реалізуються у програмному забезпеченні.

У сучасній практиці для побудови інформаційних моделей найчастіше використовують нотації ER-моделювання (Entity–Relationship), UML-діаграми класів, а також IDEF1X/IDEF1 для формального опису сутностей, ключів і зв’язків. На концептуальному рівні, що зображена на рисунку 2.3, доцільно застосовувати ER- або UML-діаграми, орієнтовані на зрозумілий опис бізнес-сутностей (товар, постачальник, замовлення, продаж тощо), тоді як на логічному рівні — моделі, близькі до IDEF1X, де уточнюються первинні та зовнішні ключі, типи зв’язків «один-до-багатьох» та «багато-до-багатьох», а також обмеження цілісності.

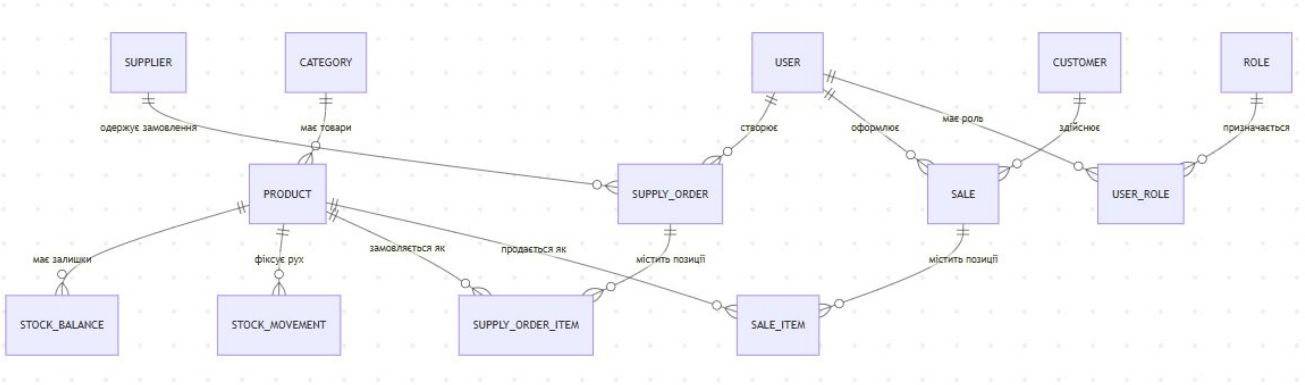

Рисунок 2.3 - Узагальнена концептуальна інформаційна модель системи постачання та продажу продукції в магазині

2.4.1 Визначення основних сутностей

Відповідно до проаналізованих бізнес-процесів (постачання, зберігання, продаж, робота користувачів) предметну область доцільно розділити на кілька логічних блоків:

- «Товари та асортимент»
- «Постачання»
- «Складські залишки та рух товару»
- «Продажі»
- «Користувачі та ролі доступу»
- «Клієнти та програми лояльності»

Для кожного з цих блоків виділяються базові сутності:

1. Товари та асортимент
   - «Товар» – основна сутність, що описує одиницю асортименту магазину.
   - «Категорія товару» – сутність для групування товарів за видами, групами, підгрупами.
2. Постачання

- «Постачальник» – юридична чи фізична особа, що постачає товари до магазину.
- «Замовлення постачання» – документ верхнього рівня, який фіксує факт оформлення замовлення на поставку.
- «Позиція замовлення постачання» – деталізація замовлення за окремими товарними позиціями.

3. Складські залишки та рух товару
   - «Складський залишок» – агрегована інформація про поточну кількість кожного товару на складі/у магазині.
   - «Складська операція» – запис про зміну залишку (надходження, списання, переміщення).
4. Продажі
   - «Продаж» – документ, що відповідає фіскальному чеку або операції реалізації (дата, касир, сума тощо).
   - «Позиція продажу» – окрема товарна позиція в межах одного продажу (товар, кількість, ціна).
   - «Клієнт» – сутність для зберігання інформації про постійних покупців (якщо система підтримує прив’язку продажів до клієнтів).
5. Користувачі та ролі доступу
   - «Користувач» – обліковий запис співробітника магазину в системі (адміністратор, менеджер, касир тощо).
   - «Роль» – типова роль/профіль доступу (адміністратор, менеджер із постачання, комірник, касир).
   - «Призначення ролі користувачу» (може бути виділене як окрема сутність «Користувач–Роль») – реалізує зв’язок «багато-до-багатьох» між користувачами й ролями.

Цей набір сутностей покриває всі основні об’єкти, необхідні для підтримки функцій системи: ведення асортименту, оформлення постачань, облік залишків, реєстрація продажів і розмежування доступу користувачів. За потреби

інформаційну модель можна розширювати (наприклад, сутностями «Акція», «Цінник», «Повернення товару»), однак наведений перелік є достатнім для базової версії системи, що розробляється в межах магістерської роботи.

2.4.2 Опис атрибутів сутностей та взаємозв’язків між ними

На наступному етапі формування інформаційної моделі для кожної основної сутності визначаються її атрибути (поля таблиці) та уточнюються зв’язки з іншими сутностями (тип зв’язку, кардинальність, ключі). Це дозволяє перейти від концептуальної моделі до логічної структури бази даних.

Сутність «Товар» може містити такі основні атрибути:

- *ProductID* – унікальний ідентифікатор товару (первинний ключ);
- *Name* – найменування;
- *Article/Code* – внутрішній код або артикул;
- *Barcode* – штрих-код (за наявності);
- *Unit* – одиниця виміру (шт., кг, л тощо);
- *BasePrice* – базова ціна продажу;
- *CategoryID* – посилання на категорію товару (зовнішній ключ).

Сутність «Категорія товару»:

- *CategoryID* – первинний ключ;
- *Name* – назва категорії;
- *ParentCategoryID* – посилання на батьківську категорію (для підтримки ієрархії, опційно).

Між сутностями «Категорія товару» та «Товар» існує зв’язок «один-до-багатьох»: одна категорія може мати багато товарів, кожен товар належить до однієї категорії.

Сутність «Постачальник»:

- *SupplierID* – первинний ключ;
- *Name* – назва;
- *EDRPOU/TaxCode* – код платника;
- *ContactInfo* – контактні дані;
- *PaymentTerms* – умови оплати;
- *DeliveryTerms* – узагальнені умови постачання (за потреби).

Сутність «Замовлення постачання»:

- *OrderID* – первинний ключ;
- *OrderNumber* – номер документа;
- *OrderDate* – дата створення;
- *SupplierID* – посилання на постачальника (зовнішній ключ);
- *PlannedDeliveryDate* – планована дата поставки;
- *Status* – статус замовлення (створено, відправлено, отримано, закрито тощо);
- *CreatedByUserID* – користувач, який створив замовлення.

Сутність «Позиція замовлення постачання»:

- *OrderItemID* – первинний ключ;
- *OrderID* – посилання на замовлення;
- *ProductID* – посилання на товар;
- *QuantityOrdered* – кількість у замовленні;
- *PurchasePrice* – закупівельна ціна;
- *LineAmount* – сумарна вартість позиції (може обчислюватися).

Між «Замовленням постачання» та «Позиціями замовлення» зв’язок «один-до-багатьох», між «Позицією замовлення» та «Товаром» – «багато-до-одного».

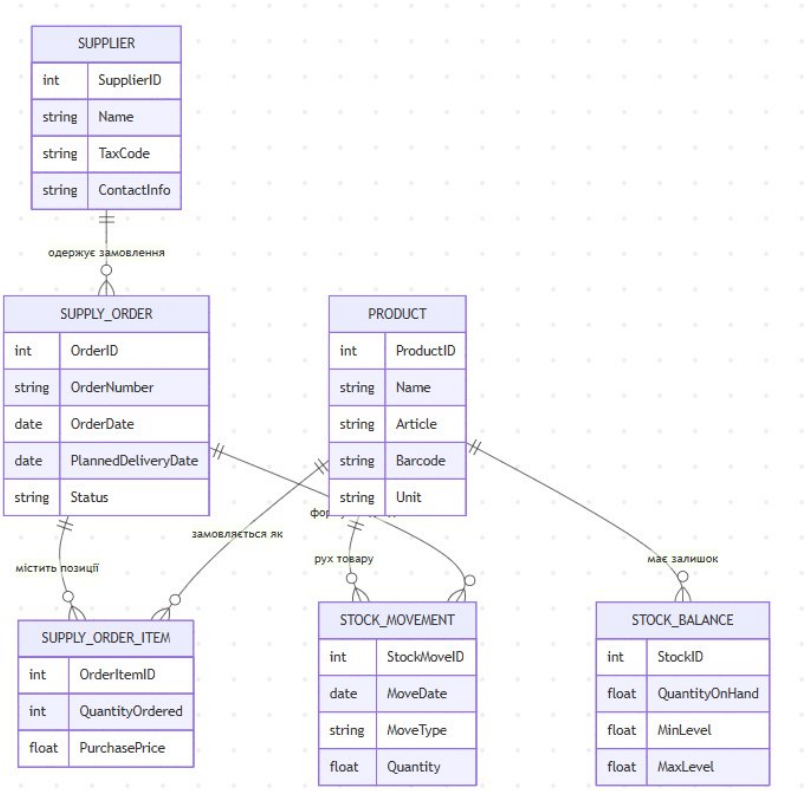

Рисунок 2.4 - Інформаційна модель підсистеми постачання та обліку складських операцій.

Сутність «Складський залишок»:

- *StockID* – первинний ключ (або складений ключ (ProductID, LocationID));
- *ProductID* – посилання на товар;
- *QuantityOnHand* – поточна кількість;
- *MinLevel* – мінімальний допустимий рівень запасу;
- *MaxLevel* – бажаний/максимальний рівень (опційно);
- *LocationID* – ідентифікатор місця зберігання (якщо використовується кілька складів/зон).

Сутність «Складська операція»:

- *StockMoveID* – первинний ключ;
- *ProductID* – товар;
- *MoveDate* – дата операції;
- *MoveType* – тип (надходження, списання, переміщення);
- *Quantity* – кількість (зі знаком +/–);

- *RelatedOrderID* / *RelatedSaleID* – посилання на пов’язаний документ постачання або продажу (опційно);
- *UserID* – користувач, який виконав операцію.

Зв’язки: кожна складська операція стосується одного товару, а зміни по товару агрегуються у «Складському залишку». Одна поставка або один продаж можуть породжувати кілька складських операцій (наприклад, по кожній позиції документа).

Сутність «Продаж»:

- *SaleID* – первинний ключ;
- *ReceiptNumber* – номер чека/документа;
- *SaleDateTime* – дата й час;
- *TotalAmount* – загальна сума;
- *PaymentType* – тип оплати (готівка, картка тощо);
- *CashierUserID* – користувач-касира;
- *CustomerID* – посилання на клієнта (якщо використовується).

Сутність «Позиція продажу»:

- *SaleItemID* – первинний ключ;
- *SaleID* – посилання на продаж;
- *ProductID* – товар;
- *QuantitySold* – кількість;
- *UnitPrice* – ціна продажу;
- *Discount* – знижка (у грошах або %).

Зв’язки: один «Продаж» має багато «Позицій продажу», кожна позиція належить одному товару.

Сутність «Користувач»:

- *UserID* – первинний ключ;
- *Login* – логін;
- *PasswordHash* – хеш пароля/токен;

- *FullName* – ПІБ;
- *Phone/Email* – контакти;
- *IsActive* – ознака активності.

Сутність «Роль»:

- *RoleID* – первинний ключ;
- *Name* – назва ролі (Адміністратор, Менеджер постачань, Касир тощо);
- *Description* – опис прав/функцій.

Сутність «Користувач–Роль» (як проміжна таблиця):

- *UserID* – зовнішній ключ на «Користувач»;
- *RoleID* – зовнішній ключ на «Роль»; (складений первинний ключ (UserID, RoleID)).

Це реалізує зв’язок «багато-до-багатьох»: один користувач може мати кілька ролей, одна роль – бути призначеною багатьом користувачам.

Описані атрибути й зв’язки дозволяють надалі перейти до логічного проєктування бази даних (визначення типів полів, індексів, обмежень цілісності) та безпосередньої реалізації схеми в обраній СУБД/хмарному сховищі, як зображено на рисунку 2.5.

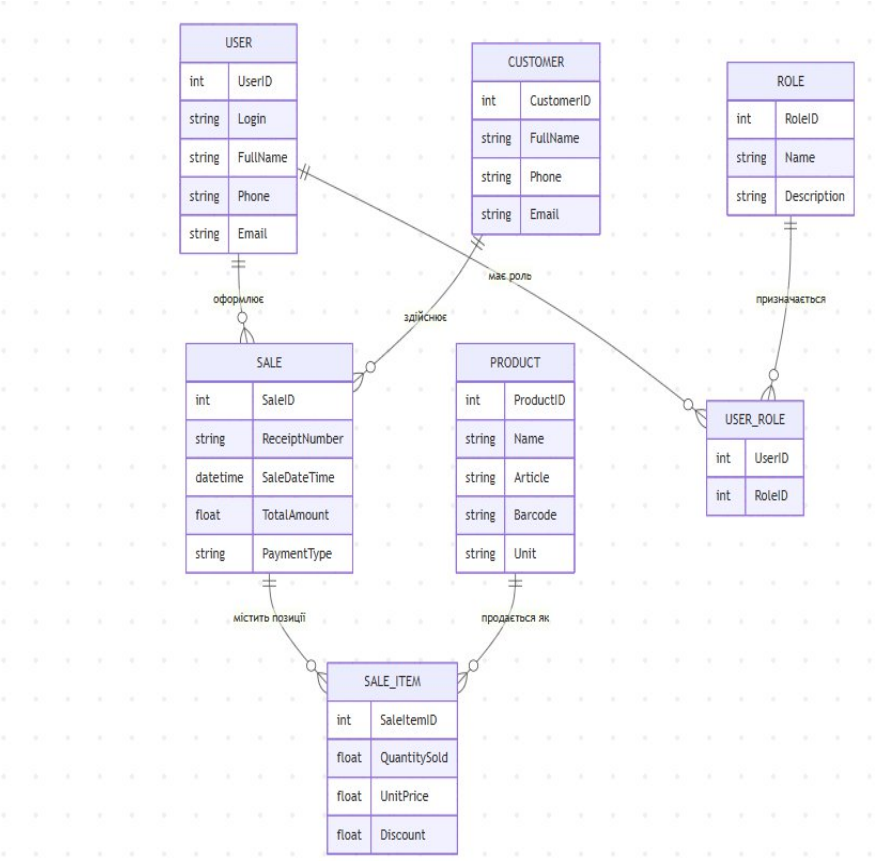

Рисунок 2.5 - Інформаційна модель підсистеми продажів та керування користувачами.

При цьому важливо дотримуватися принципів нормалізації (мінімізація дублювання даних, чітке розділення довідників і фактів), що забезпечить коректність і масштабованість системи при подальшому розвитку функціоналу.

2.5 Логічна структура бази даних автоматизованої системи

Логічна структура бази даних автоматизованої системи підтримки процесів постачання та продажу продукції в магазині є формальним відображенням інформаційної моделі предметної області, наведеної в підрозділі 2.4. На цьому рівні

сутності перетворюються на таблиці (у випадку реляційної СУБД) або колекції документів (у випадку використання документоорієнтованого сховища, наприклад, Firestore), визначаються первинні та зовнішні ключі, а також механізми забезпечення цілісності даних. Для розроблюваної системи, яка використовує веб-технології (Angular, PWA) та хмарне сховище, доцільно поєднати реляційний підхід до логічного проєктування (як базову модель) з практикою організації колекцій та підколекцій документів, притаманною NoSQL-рішенням.

2.5.1 Опис таблиць, колекцій та їх ключів

Відповідно до виділених сутностей, логічна схема бази даних включає такі основні таблиці (або колекції, якщо реалізація базується на документоорієнтованому сховищі):

Таблиці (колекції) довідників

1. CATEGORY – довідник категорій товарів
   - CategoryID (PK) – унікальний ідентифікатор категорії.
   - Name – назва категорії.
   - ParentCategoryID (FK → CATEGORY.CategoryID, nullable) – ідентифікатор батьківської категорії (для ієрархії).
2. PRODUCT – довідник товарів
   - ProductID (PK) – унікальний ідентифікатор товару.
   - Name – найменування товару.
   - Article – внутрішній код/артикул.
   - Barcode – штрих-код (за потреби).
   - Unit – одиниця виміру (шт., кг, л тощо).
   - BasePrice – базова ціна продажу.
   - CategoryID (FK → CATEGORY.CategoryID) – належність до категорії.
3. SUPPLIER – довідник постачальників

- SupplierID (PK) – унікальний ідентифікатор постачальника.
- Name – назва.
- TaxCode – код платника (ЄДРПОУ/ІПН).
- ContactInfo – контактні дані (телефон, e-mail, адреса).
- PaymentTerms – умови оплати (опційно).
- DeliveryTerms – умови доставки (опційно).

4. CUSTOMER (опційно) – довідник клієнтів
   - CustomerID (PK).
   - FullName.
   - Phone, Email.
   - Інші реквізити (ID програми лояльності тощо, за потреби).

Таблиці (колекції) документів і їх позицій

5. SUPPLY_ORDER – «шапка» замовлення постачання
   - OrderID (PK) – унікальний ідентифікатор замовлення.
   - OrderNumber – номер документа (може дублюватися, але у межах системи бажано унікальний).
   - OrderDate – дата створення.
   - SupplierID (FK → SUPPLIER.SupplierID) – постачальник.
   - PlannedDeliveryDate – планована дата поставки.
   - Status – статус (created, sent, received, closed, cancelled).
   - CreatedByUserID (FK → USER.UserID) – хто створив замовлення.
6. SUPPLY_ORDER_ITEM – позиції замовлення постачання
   - OrderItemID (PK).
   - OrderID (FK → SUPPLY_ORDER.OrderID).
   - ProductID (FK → PRODUCT.ProductID).
   - QuantityOrdered.
   - PurchasePrice.
   - (за потреби) LineAmount – сума по позиції.
   - Зв’язок:

- SUPPLY_ORDER 1 → N SUPPLY_ORDER_ITEM
- PRODUCT 1 → N SUPPLY_ORDER_ITEM

7. SALE – документ продажу (чек)
   - SaleID (PK).
   - ReceiptNumber – номер чека (для інтеграції з фіскальним реєстратором).
   - SaleDateTime – дата й час продажу.
   - TotalAmount – загальна сума.
   - PaymentType – готівка/картка/інше.
   - CashierUserID (FK → USER.UserID) – касир.
   - CustomerID (FK → CUSTOMER.CustomerID, nullable) – клієнт (якщо є).
8. SALE_ITEM – позиції продажу
   - SaleItemID (PK).
   - SaleID (FK → SALE.SaleID).
   - ProductID (FK → PRODUCT.ProductID).
   - QuantitySold.
   - UnitPrice.
   - Discount – знижка (сума або %).

Зв’язок:

- SALE 1 → N SALE_ITEM
- PRODUCT 1 → N SALE_ITEM

Таблиці (колекції) складського обліку

9. STOCK_BALANCE – оперативні залишки
   - StockID (PK) – або складений ключ (ProductID, LocationID).
   - ProductID (FK → PRODUCT.ProductID).
   - LocationID – код складу/зони (якщо є кілька локацій, інакше можна опустити).
   - QuantityOnHand.
   - MinLevel – мінімальний запас.
   - MaxLevel – бажаний/максимальний запас (опційно).
10. STOCK_MOVEMENT – журнал руху товарів
    - StockMoveID (PK).
    - ProductID (FK → PRODUCT.ProductID).
    - MoveDate – дата/час операції.
    - MoveType – тип операції (IN, OUT, TRANSFER).
    - Quantity – знак + для надходження, – для списання.
    - RelatedOrderID (FK → SUPPLY_ORDER.OrderID, nullable).
    - RelatedSaleID (FK → SALE.SaleID, nullable).
    - UserID (FK → USER.UserID) – хто виконав операцію.

Ця таблиця дозволяє відтворити історію руху товару та служить основою для розрахунку залишків.

Таблиці (колекції) користувачів і ролей

11. USER – облікові записи користувачів
    - UserID (PK).
    - Login.
    - PasswordHash (або зовнішній auth ID).

- FullName.
- Phone, Email.
- IsActive.

12. ROLE – ролі системи
    - RoleID (PK).
    - Name – назва ролі (Admin, SupplyManager, Cashier, etc.).
    - Description.
13. USER_ROLE – зв’язок «користувач–роль»
    - UserID (FK → USER.UserID).
    - RoleID (FK → ROLE.RoleID).
    - Складений PK (UserID, RoleID).

Якщо використовується Firestore, це може бути організовано як:

- Колекції верхнього рівня: products, categories, suppliers, customers, users, roles.
- Колекції документів/підколекцій:
    - supplyOrders з підколекцією items для SUPPLY_ORDER_ITEM;
    - sales з підколекцією items для SALE_ITEM;
    - окрема колекція stockMovements для журналу рухів;
    - колекція stockBalances для агрегованих залишків;
    - колекція userRoles або підколекції roles усередині users.

Логіка ключів зберігається: ID документа грає роль PK, посилання на інші колекції можуть зберігатися як референси або як поля з ID.

2.5.2 Механізми забезпечення цілісності та узгодженості даних

Логічна структура бази даних доповнюється механізмами, які гарантують, що дані залишаються коректними, несуперечливими й узгодженими між собою навіть за інтенсивної роботи користувачів.

Основні види цілісності, які потрібно забезпечити:

1. Сутнісна (entity integrity)
   - Кожен запис у ключових таблицях (PRODUCT, SUPPLIER, SUPPLY_ORDER, SALE, USER тощо) повинен мати унікальний первинний ключ (PK), який не є NULL.
   - У реляційній СУБД це забезпечується через оголошення PK та обмеження NOT NULL.
   - У Firestore роль унікальності відіграє ID документа; додаткові унікальні обмеження (наприклад, на Barcode чи Login) можна контролювати на рівні бізнес-логіки.
2. Посилальна/референтна цілісність (referential integrity)
   - Усі зовнішні ключі (FK) мають посилатися на реально існуючі записи в батьківських таблицях:
     - ProductID у SUPPLY_ORDER_ITEM, SALE_ITEM, STOCK_BALANCE, STOCK_MOVEMENT повинен існувати в PRODUCT;
     - SupplierID у SUPPLY_ORDER – у SUPPLIER;
     - CashierUserID у SALE, CreatedByUserID у SUPPLY_ORDER – у USER;
     - CustomerID у SALE – у CUSTOMER;
     - CategoryID у PRODUCT – у CATEGORY.
   - У реляційній БД це забезпечується через FOREIGN KEY з правилами ON DELETE / ON UPDATE (наприклад, заборона видаляти товар, на який є посилання в документах).

- У Firestore/NoSQL посилальна цілісність контролюється на рівні застосунку: перед видаленням сутності перевіряються посилання (або використовується «м'яке видалення» – флаг IsActive = false).

3. Цілісність доменів (domain integrity)
   - Обмеження типів та допустимих значень полів:
     - QuantityOnHand, QuantityOrdered, QuantitySold – не від'ємні;
     - Status замовлення – тільки значення з фіксованого переліку (enum);
     - MoveType – IN/OUT/TRANSFER;
     - PaymentType – Cash/Card/…
   - Реалізується через типи даних, CHECK-обмеження (у SQL) або валідацію даних у бекенді/клієнті в NoSQL-сценарії.
4. Логічна/бізнес-цілісність
   - Узгодженість між документами та складськими залишками:
     - Після підтвердження замовлення постачання створюються відповідні записи у STOCK_MOVEMENT з типом IN, а STOCK_BALANCE оновлюється;
     - Після підтвердження продажу створюються записи у STOCK_MOVEMENT з типом OUT і оновлюється STOCK_BALANCE;
     - Неможливість продати товар у кількості, що перевищує доступний залишок (перевірка перед створенням SALE/SALE_ITEM).
   - Ці правила реалізуються у бізнес-логіці сервера або в транзакціях / batch-операціях (для Firestore).
5. Транзакційність та узгодженість операцій
   - При операціях, що змінюють кілька пов'язаних сутностей (наприклад, створення SUPPLY_ORDER + STOCK_MOVEMENT, або SALE + SALE_ITEM + STOCK_MOVEMENT + оновлення STOCK_BALANCE), зміни мають бути атомарними – або виконуються всі, або жодна.
   - У реляційних СУБД це забезпечується транзакціями (BEGIN TRANSACTION / COMMIT / ROLLBACK).

- У Firestore використовуються транзакції та batch-записи для узгодженого оновлення кількох документів.

6. Механізми захисту від «невалідних» станів
   - Обмеження на зміну статусів документів: неможливо перевести замовлення зі статусу closed назад у created без окремої процедури; продаж не може бути змінений після фіскалізації;
   - Валідація полів у формі (Angular): перед відправкою даних перевіряються обов’язкові поля, типи, мінімальні значення тощо;
   - Використання централізованих сервісів/репозиторіїв у застосунку, які інкапсулюють логіку змін, щоб не дублювати її по компонентах.
7. Аудит і відстеження змін (рекомендовано)
   - Ведення в STOCK_MOVEMENT або окремій таблиці логів інформації про те, хто і коли створив/змінив документ, дозволяє виявляти помилки й некоректні операції;
   - Для важливих довідників (PRODUCT, SUPPLIER) можна передбачити журнали змін цін чи реквізитів (або хоча б мати поля CreatedAt, UpdatedAt, UpdatedByUserID).

Як результат, логічна структура бази даних разом із механізмами забезпечення цілісності та узгодженості даних утворюють фундамент, на якому будуть реалізовані сервісний шар і Angular/PWA-інтерфейс.

2.6 Висновки до другого розділу

У цьому розділі було сформовано цілісну основу для побудови автоматизованої системи постачання та продажу продукції в магазині – від формалізації процесів до структури даних. Описано логіку роботи роздрібного магазину в термінах бізнес-процесів постачання, зберігання та реалізації товарів, виділено ключові ролі, стани та події, що дозволяє чітко зрозуміти, які саме операції мають бути підтримані системою. На цій основі розглянуто математичні підходи до управління товарними запасами: класичні моделі оптимізації розміру замовлення та політики поповнення, а також інструменти сегментації асортименту на кшталт ABC/XYZ-аналізу, які дають змогу поєднати формальні розрахунки з практичними рішеннями для різних груп товарів.

Окрему увагу приділено системі показників ефективності, за допомогою якої можна оцінити, як автоматизація впливає на запаси, логістичні витрати, рівень сервісу та якість роботи самої інформаційної системи. Це створює передумови не лише для контролю поточного стану, а й для подальшого аналізу «до» і «після» впровадження рішення. На рівні даних побудовано інформаційну модель предметної області з виділенням основних сутностей (товари, постачальники, замовлення, складські операції, продажі, користувачі, ролі, клієнти) та їх взаємозв’язків, а також сформовано логічну структуру бази даних із визначенням таблиць/колекцій, ключів і правил цілісності. У результаті отримано узгоджену математичну й інформаційну базу, на якій можна проєктувати архітектуру та реалізацію програмного забезпечення засобами Angular і PWA-технологій у наступних частинах роботи.

3 АНАЛІЗ ТА ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ СИСТЕМИ

3.1 Вимоги до програмної платформи та середовища виконання

Автоматизована система постачання та продажу продукції в магазині розробляється як веб-застосунок із використанням підходів Single Page Application (SPA) та Progressive Web Application (PWA). Це визначає вимоги до програмної платформи та середовища виконання як на клієнтському, так і на серверному боці. Клієнтська частина повинна коректно функціонувати в сучасних веб-браузерах (Google Chrome, Microsoft Edge, Mozilla Firefox, Safari), які підтримують JavaScript/TypeScript, HTML5, CSS3 та механізми, необхідні для побудови SPA: компонентну модель, маршрутизацію, реактивне оновлення інтерфейсу. Фреймворк Angular у цьому контексті розглядається як платформа для створення масштабованих односторінкових веб-застосунків на основі TypeScript, що включає набір вбудованих бібліотек для роботи з формами, HTTP-запитами, маршрутизацією та інструментами розробки, тестування і розгортання [37-38].

Оскільки застосунок має бути доступним як «настільний» PWA-додаток без встановлення через магазин додатків, до платформи висуваються вимоги підтримки PWA-функціональності. Це, зокрема, наявність маніфесту веб-застосунку (Web App Manifest), який описує назву, іконки, кольорову схему та спосіб відображення застосунку в операційній системі, а також підтримка service worker, що відповідають за кешування ресурсів, перехоплення мережевих запитів і забезпечення офлайн-режиму [39], що зображено на рисунку 3.1.

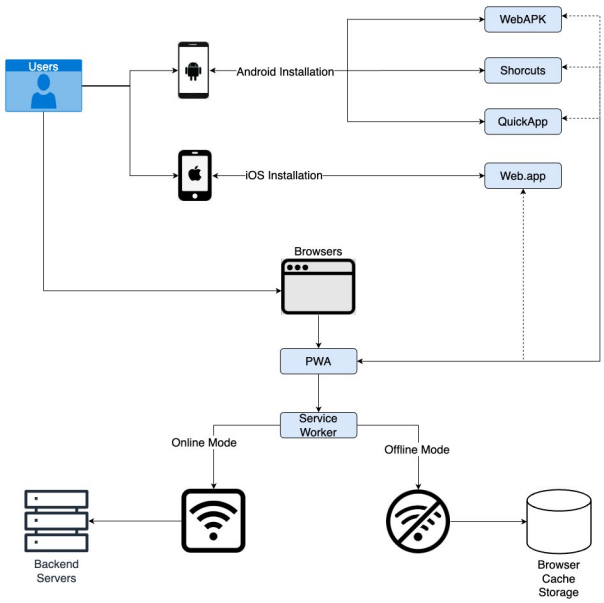

Рисунок 3.1 - Архітектура клієнт-сервер-хмара для PWA-додатку

У публікаціях, присвячених PWA, наголошується, що прогресивні веб-застосунки мають поєднувати властивості веб-сайтів (доступ з будь-якого пристрою через браузер) і нативних додатків (інсталяція, робота офлайн, інтеграція з платформою), використовуючи при цьому єдину кодову базу [40].

Окремою вимогою є робота в умовах нестабільного або відсутнього мережевого з'єднання. PWA повинна коректно завантажувати основний інтерфейс (app shell) із кешу, зберігати критично важливі дані локально та синхронізувати їх із сервером при відновленні з'єднання. Для цього service worker використовуються як «проксі» між застосунком і мережею, реалізуючи стратегії кешування (cache-first, network-first тощо) і фонову синхронізацію [41]. Одночасно з цим висуваються вимоги до безпеки: згідно з рекомендаціями MDN, реєстрація й робота service worker дозволені лише на захищених походженнях (HTTPS або localhost), що зумовлює обов'язкове використання HTTPS для бойової інфраструктури [42].

З боку серверних/хмарних компонентів система має спиратися на масштабоване сховище даних, яке забезпечує синхронізацію змін між кількома клієнтами й підтримує офлайн-режим. У цьому контексті доцільним є використання Cloud Firestore як гнучкої хмарної NoSQL документно-орієнтованої

бази даних, спеціально розробленої для веб- та мобільних застосунків, яка підтримує ієрархічну структуру колекцій і підколекцій, складні запити, автоматичне масштабування та клієнтську синхронізацію [43-44]. Порівняльні огляди рішень Firebase підкреслюють, що Firestore забезпечує розширену офлайн-підтримку й синхронізацію, що є критично важливим для PWA-систем, які повинні коректно працювати навіть без стабільного доступу до мережі [45].

Додатково до цього, середовище виконання має відповідати низці нефункціональних вимог:

- кросплатформенність – робота застосунку в браузерах на Windows, Linux, macOS та на мобільних пристроях;
- масштабованість – можливість обробляти зростаючу кількість користувачів і транзакцій без істотної деградації продуктивності (завдяки хмарній інфраструктурі й оптимізованій передачі даних);
- продуктивність – швидкий відгук інтерфейсу, мінімізація часу першого завантаження та ефективне кешування статичних ресурсів;
- безпека – використання HTTPS, контроль доступу до API, коректна конфігурація правил безпеки в базі даних, а також застосування сучасних практик захисту PWA (Content Security Policy, захист від XSS, належне зберігання токенів автентифікації) [46].

Таким чином, до програмної платформи та середовища виконання автоматизованої системи висуваються комплексні вимоги: підтримка SPA-архітектури (Angular), реалізація PWA-функціональності (service worker, маніфест, офлайн-режим, інсталяція), інтеграція з хмарним NoSQL-сховищем із підтримкою синхронізації та офлайн-доступу (Cloud Firestore), забезпечення безпеки й кросплатформенності. Виконання цих вимог є підґрунтям для обґрунтованого вибору конкретного стеку технологій, що буде розглянуто в подальших підрозділах.

3.2 Порівняльний аналіз фреймворків для розробки клієнтських застосунків

Сучасні клієнтські додатки, побудовані за моделлю Single Page Application, зазвичай реалізуються з використанням одного з трьох домінуючих рішень: Angular, React або Vue.js. Усі три підходи є компонентно-орієнтованими, підтримують декларативний опис інтерфейсу й активно застосовуються для створення SPA та PWA, однак помітно відрізняються за концепцією, ступенем «завершеності» платформи, гнучкістю та вимогами до структури проєкту. У працях, присвячених порівнянню цих фреймворків, наголошується, що Angular позиціонується як «повноцінна платформа» з вбудованою підтримкою маршрутизації, форм, DI, інструментів збірки та тестування; React – як гнучка бібліотека для побудови UI з використанням віртуального DOM; Vue – як «прогресивний» фреймворк, який можна поступово впроваджувати, нарощуючи функціональність за потреби, детальна характеристика зображена у таблиці нижче [47-50].

Таблиця 3.1 - Порівняльна характеристика фреймворків

Framework	Переваги	Недоліки
React	Повторне використання компонентів; Висока продуктивність; Велика спільнота;	Стрімкий процес навчання; Вимагає додаткової конфігурації для управління маршрутами та станом;

Продовження Таблиці 3.1 - Порівняльна характеристика фреймворків

Vue	Проста інтеграція; Доступна документація; Висока гнучкість;	Нижча порівняльна зрілість; Менш пряма підтримка підприємств, ніж Angular або React;
Angular	Повний та вичерпний; Підтримка TypeScript; Ефективне управління станом;	Стрімкий процес навчання; Може бути надмірним для невеликих проектів

Для розроблюваної системи важливо врахувати не лише популярність екосистеми, а й відповідність архітектурним вимогам: побудова структурованого SPA, підтримка PWA, масштабованість для подальшого розширення функціоналу та зручність роботи з TypeScript.

3.2.1 Angular, як фреймворк для побудови SPA

Angular - це фронтенд-платформа на базі TypeScript, розроблена та підтримувана Google, орієнтована саме на створення масштабованих SPA-застосунків. Офіційна документація позиціонує Angular як фреймворк, що поєднує компонентну модель, модульну структуру, механізм залежностей (dependency injection), вбудовану маршрутизацію, засоби роботи з формами (reactive та template-driven) та потужний CLI для створення, збірки й тестування проєктів [51-52]. Таке «батарейки в комплекті» рішення зменшує кількість архітектурних рішень, які потрібно приймати на старті, і забезпечує єдиний підхід до побудови проєкту на рівні всієї команди.

Ключовою особливістю Angular є використання TypeScript, що додає статичну типізацію й сучасні можливості JavaScript, полегшує рефакторинг і виявлення помилок на етапі компіляції [53]. Фреймворк базується на компонентно-орієнтованій архітектурі: інтерфейс розбивається на ієрархію компонентів із

власними шаблонами, стилями та логікою, що спрощує повторне використання та тестування коду, як зображено на рисунку 3.2.

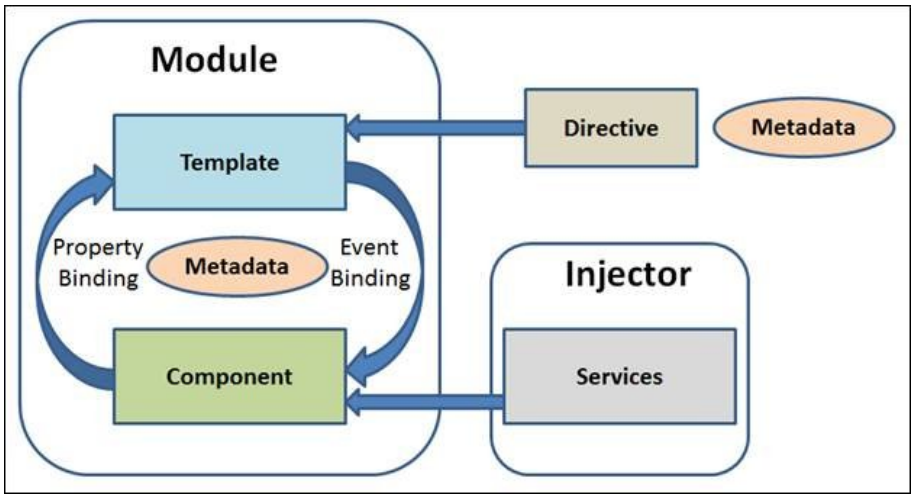

Рисунок 3.2 - Узагальнена структура Angular-застосунку

Angular також тісно інтегрований із RxJS, що дозволяє організувати реактивну обробку подій, роботу з потоками даних і асинхронними операціями, зокрема при взаємодії з API.

Для SPA важливою є наявність вбудованої маршрутизації (Angular Router), яка підтримує lazy loading модулів, guard-и доступу та параметризовані маршрути, що дозволяє ефективно організувати навігацію без перезавантаження сторінки [52-53]. У контексті PWA Angular має офіційну підтримку через Angular Service Worker і schematics (ng add @angular/pwa), що автоматизує додавання маніфесту, регістрацію service worker та налаштування кешування ресурсів — це спрощує перетворення звичайного SPA на прогресивний веб-застосунок.

У порівняльних дослідженнях SPA-фреймворків Angular часто описують як найбільш структуроване та придатне для великомасштабних корпоративних застосунків рішення, де важливі єдині підходи, сувора типізація й чітко визначена архітектура [47-50]. Серед недоліків зазначають крутішу криву навчання та більшу «вагу» фреймворку порівняно з більш легкими альтернативами, такими як Vue, однак для проєктів із тривалим життєвим циклом та розширеним функціоналом це, як правило, компенсується кращою керованістю коду та зрозумілою структурою

[47-49]. З огляду на те, що розроблювана система вже реалізована на Angular як PWA, вибір саме цього фреймворка є логічним і узгоджується з вимогами до масштабованості, структурованої архітектури та інтеграції з хмарними сервісами.

3.2.2 Альтернативні рішення

Найбільш поширеними альтернативами Angular при створенні SPA є React та Vue.js.

React позиціонується як «JavaScript-бібліотека для побудови користувацьких інтерфейсів» і зосереджується насамперед на UI-рівні [54]. Вона використовує компонентну архітектуру й концепцію віртуального DOM, завдяки якій зміни в інтерфейсі застосовуються ефективно, оскільки React спочатку оновлює легковагову віртуальну копію DOM і лише потім мінімізує реальні зміни в браузері [55]. Такий підхід забезпечує високу продуктивність, особливо в застосунках із великою кількістю динамічних оновлень. Разом з тим React свідомо залишається «неповною» платформою: за маршрутизацію, управління станом, роботу з формами тощо відповідають окремі бібліотеки (React Router, Redux/Zustand, Formik/React Hook Form та ін.), що дає велику гнучкість, але вимагає від команди додаткових рішень на рівні архітектури [56]. У порівняльних оглядах React часто рекомендують для застосунків із дуже динамічним UI, а також коли важливо мати гнучкий стек із можливістю «підібрати» власний набір бібліотек [47-48].

Vue.js описується як «прогресивний JavaScript-фреймворк» для побудови інтерфейсів, який можна впроваджувати інкрементально: від «підсвічування» окремих елементів на статичній сторінці до повноцінних SPA з маршрутизацією та централізованим керуванням станом [57]. Офіційна документація підкреслює, що Vue є «доступним, продуктивним і гнучким», базується на стандартних HTML, CSS і JavaScript і надає декларативну компонентну модель із реактивною системою оновлення[58]. За рахунок невеликого розміру ядра й низького порогу входу Vue часто розглядається як зручне рішення для невеликих і середніх проєктів,

прототипів і команд, які хочуть швидко стартувати без складної конфігурації. Водночас для великих корпоративних систем із жорсткими архітектурними вимогами Vue зазвичай потребує додаткових домовленостей щодо структури проєкту й стандартів кодування [59].

Порівняльні дослідження Angular, React і Vue в контексті розробки SPA показують, що єдиного «кращого» фреймворка не існує — вибір залежить від вимог до масштабу, структури, продуктивності, складу команди та наявного стеку [47-61]. React відзначають за гнучкість і багату екосистему, Vue — за простоту й легкість, Angular — за цілісність платформи та придатність для великих, структурованих застосунків. У випадку автоматизованої системи постачання та продажу продукції важливими є: чітка архітектурна модель, підтримка PWA, можливість організувати багатомодульну структуру з розмежуванням доступу й інтеграцією з хмарною базою даних. З урахуванням того, що поточний застосунок уже реалізований на Angular і відповідає цим вимогам, альтернативні рішення доцільно розглядати радше як порівняльний фон, а не як реальну заміну обраного стеку.

3.3 Обґрунтування вибору Angular для реалізації клієнтської частини системи

Вибір Angular як основного фреймворка для реалізації клієнтської частини автоматизованої системи постачання та продажу продукції в магазині зумовлений поєднанням архітектурних, технологічних і практичних переваг. Angular є цілісною платформою, яка «з коробки» надає компонентну модель, модульну структуру, механізм впровадження залежностей, маршрутизацію, роботу з формами, HTTP-клієнт та інструменти збірки й тестування. Це дозволяє будувати структурований, передбачуваний проєкт без необхідності самостійного підбору набору окремих бібліотек і спрощує довгострокову підтримку та розвиток системи.

Важливою причиною вибору Angular є використання мови TypeScript, що забезпечує статичну типізацію, кращу читаємість і рефакторинг коду, а також зменшує кількість помилок, виявляючи їх ще на етапі розробки. Для даного проєкту, де клієнтська частина реалізує складні бізнес-процеси (оформлення постачань, облік залишків, оформлення продажів, розмежування ролей користувачів), це особливо актуально: чіткі інтерфейси даних і типи моделей знижують ризик логічних помилок при взаємодії з хмарною базою даних. Крім того, Angular тісно інтегрується з RxJS, що полегшує реалізацію реактивної роботи з потоками даних (оновлення списків товарів, статусів замовлень у реальному часі), що відповідає вимогам до сучасної веб-системи.

Окремою перевагою є вбудована підтримка перетворення застосунку на PWA: за допомогою стандартних інструментів Angular легко додати маніфест, service worker і налаштувати кешування, що дає можливість працювати в режимі «настільного» додатка з офлайн-підтримкою, що зображено на рисунку 3.3.

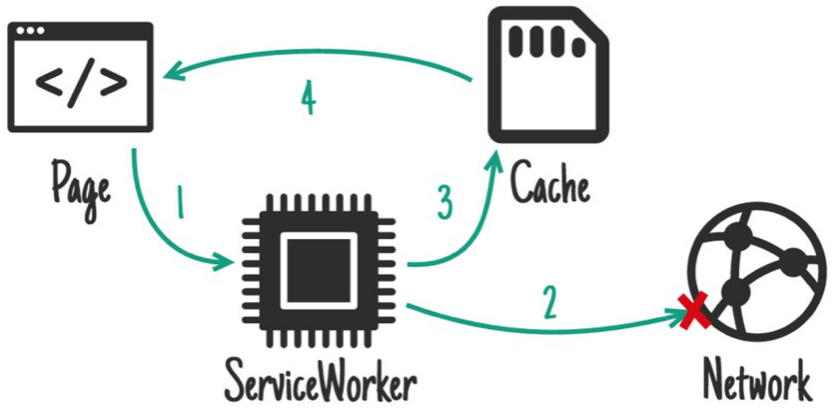

Рисунок 3.3 - Взаємодія Angular PWA з користувачем

З огляду на те, що існуючий прототип системи вже реалізовано на Angular з використанням PWA-технологій, продовження розробки на цьому фреймворку забезпечує максимальне повторне використання коду, узгодженість архітектурних рішень і спрощує еволюцію проєкту в межах магістерської роботи.

3.4 Концепція прогресивних веб-застосунків та особливості PWA

Прогресивні веб-застосунки (Progressive Web Apps, PWA) – це веб-застосунки, які використовують сучасні можливості платформи Web, але за відчуттями для користувача максимально наближені до нативних додатків: їх можна встановити на пристрій, запускати з іконки на робочому столі, вони працюють офлайн і можуть інтегруватися з функціями операційної системи. MDN визначає PWA як застосунки, побудовані на веб-технологіях (HTML, CSS, JavaScript), що забезпечують досвід, подібний до платформоспецифічних програм, зберігаючи при цьому головні переваги Web – кросплатформеність, доступ із браузера, можливість пошуку та відкриття за URL-адресою.

Ключова ідея PWA – прогресивне поліпшення (progressive enhancement): застосунок будується таким чином, щоб працювати в базовому режимі у будь-якому сучасному браузері, а на платформах, що підтримують додаткові API (service workers, Web App Manifest, Push API тощо), «розкривати» розширені можливості: офлайн-режим, кешування, встановлення, push-сповіщення. Основними технічними складниками прогресивного веб-застосунку є:

Service worker – спеціальний фоновий скрипт, який працює окремо від основного потоку сторінки, перехоплює мережеві запити та може обслуговувати їх із кешу або мережі, забезпечуючи офлайн-режим і прискорення завантаження (offline-first). Service worker діє як «проксі» між застосунком і мережею, дозволяючи реалізувати гнучкі стратегії кешування, фонову синхронізацію та push-нотифікації.

Маніфест веб-застосунку (Web App Manifest) – JSON-файл, який містить метадані про застосунок (назва, коротка назва, іконки, орієнтація екрану, кольори оформлення, режим відображення). Саме на основі маніфесту браузер пропонує встановити PWA на пристрій і визначає, як він має виглядати при запуску – у повноекранному або «стандартному» режимі, з власною іконкою на робочому столі тощо.

Безпечний контекст (HTTPS) – більшість можливостей PWA (зокрема service workers) за специфікацією доступні лише на захищених походженнях (HTTPS або localhost). Це гарантує, що механізми кешування, фонової синхронізації та обробки запитів не можуть бути використані в потенційно небезпечному середовищі і відповідають сучасним вимогам до безпеки Web.

З точки зору користувача гарний PWA повинен відповідати кільком ключовим характеристикам: бути надійним (швидко завантажуватись і залишатися функціональним навіть за нестабільної мережі), швидким (мінімізувати затримки при взаємодії з інтерфейсом), інтегрованим (мати іконку, повноекранний режим, підтримку push-сповіщень, відчуватися «як додаток»), а також універсальним (працювати на будь-якому пристрої з сучасним браузером із використанням єдиної кодової бази), як зображено на рисунку 3.4.

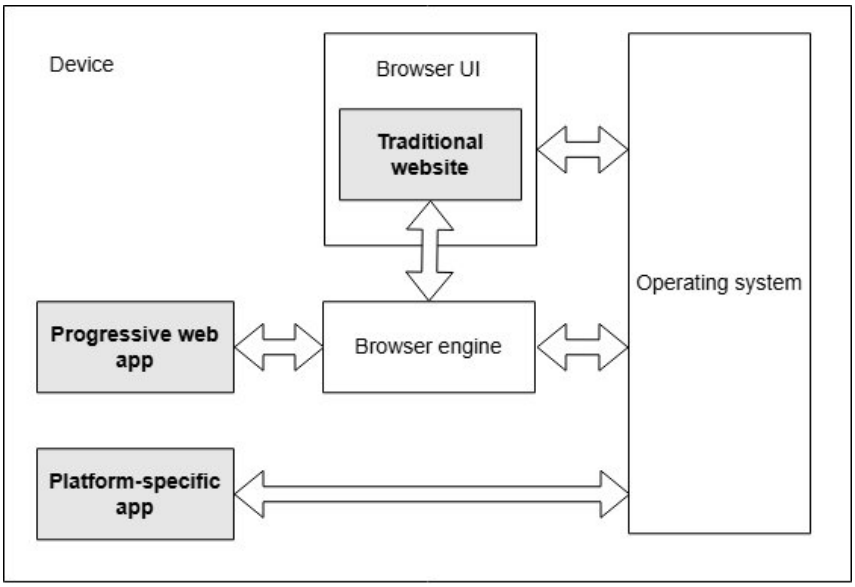

Рисунок 3.4 - Структура прогресивного веб-застосунку

Для автоматизованої системи постачання та продажу продукції в магазині концепція PWA є особливо доречною: застосунок може працювати у вигляді звичайного веб-інтерфейсу в браузері або бути встановленим як «настільний» додаток на ПК, підтримувати кешування основних даних (наприклад, довідників товарів) і забезпечувати безперервну роботу користувачів навіть при тимчасовій

відсутності доступу до мережі, а синхронізація з хмарним сховищем відбувається у фоновому режимі після відновлення з’єднання.

3.5 Переваги PWA для рітейл-сфери

Використання прогресивних веб-застосунків у роздрібній торгівлі поєднує плюси класичного вебу та нативних мобільних додатків і напряму впливає на конверсію, утримання клієнтів та операційну ефективність. Для рітейлу критично важливі швидкість, зручність і безперервність доступу до каталогу товарів, а PWA якраз орієнтовані на швидке завантаження, кешування основного інтерфейсу та мінімальну залежність від якості мережі. Дослідження в галузі e-commerce підкреслюють, що PWA покращують мобільний досвід завдяки швидкому відгуку, офлайн-доступу, push-сповіщенням та «аподібному» інтерфейсу, що приводить до зростання залученості та конверсії користувачів.

Однією з ключових переваг для рітейлу є можливість роботи при нестабільному або відсутньому з’єднанні: за рахунок кешування за допомогою service worker покупці можуть переглядати раніше завантажений каталог, кошик та обрані товари навіть офлайн, а всі дії синхронізуються після відновлення мережі. Це особливо важливо для магазинів, де клієнти користуються мобільним інтернетом зі слабким сигналом або Wi-Fi з перебоями. Додатковою перевагою є push-сповіщення, які дозволяють рітейлерам нагадувати про акції, повторну наявність товарів, персональні знижки й покинуті кошики, що сприяє повторним візитам і підвищенню середнього чека.

Ще один важливий аспект – зменшення бар’єру входу. На відміну від нативних застосунків, PWA не потребують пошуку й завантаження з магазину додатків: користувач може просто відкрити сайт і, за бажання, додати його на головний екран одним натисканням, що добре показано у таблиці 3.5. Таке «легке встановлення» та відсутність обмежень сторів зменшує відтік користувачів на етапі інсталяції й дозволяє рітейлу швидше оновлювати функціонал без проходження

процедур модерації. Для бізнесу це означає менші витрати на розробку та підтримку (одна кодова база для браузера, мобільних і десктопних пристроїв) і швидший вихід нових версій на ринок.

Таблиця 3.5 - Порівняння: сайт / PWA-додаток / нативний додаток

	Responsive Site	Progressive Web App	Native App
Пристосований для мобільних пристроїв	+	++	+++
Пертворення	-	++	+++
Інсталяція	-	++	+++
Push-повідомлення	-	+	+++
Автономний режим	-	+++	+++
Оплата одним дотиком	-	++	+++
Увімкнути GPS	-	+++	+++
Швидкість	-	+++	+++
Виконання	-	+++	+++
Індексований	-	-	+++
Вбудовані функції	-	+	+++
SEO-індексований	++	+++	-

Практичні кейси великих e-commerce/retail-компаній демонструють, що впровадження PWA призводить до відчутного зростання показників: за даними галузевих оглядів, перехід на PWA може давати збільшення мобільних сесій, скорочення часу завантаження, зниження показника відмов і підвищення конверсії завдяки більш плавному сценарію «перегляд – додавання в кошик – оформлення замовлення». Для автоматизованої системи постачання та продажу продукції в магазині це означає можливість забезпечити покупцю швидкий і стабільний доступ до товарів, а персоналу – зручний «настільний» інтерфейс без складної інсталяції, що робить PWA природним вибором для рітейл-сфери.

3.6 Аналіз системи зберігання даних для веб-застосунків

Система постачання та продажу продукції в магазині потребує надійного й гнучкого сховища даних, яке здатне обслуговувати транзакції в режимі реального часу, масштабуватися при зростанні кількості операцій та забезпечувати доступ із веб- і PWA-клієнтів. Сучасні застосунки використовують як класичні реляційні СКБД, так і нереляційні (NoSQL) бази даних, а також все частіше спираються на хмарні платформи типу Backend-as-a-Service (BaaS), які поєднують зберігання даних, автентифікацію та іншу бекенд-функціональність у вигляді керованого сервісу.

3.6.1 Реляційні та нереляційні системи керування базами даних

Реляційні бази даних організовують дані у вигляді таблиць з рядками та стовпцями, що пов’язані між собою через первинні та зовнішні ключі. Вони спираються на реляційну модель даних і, як правило, підтримують транзакційні властивості ACID (атомарність, узгодженість, ізольованість, довговічність), що робить їх придатними для систем, де критично важлива цілісність та консистентність даних (фінансові операції, складський облік, аналітична звітність). Реляційні СКБД (MySQL, PostgreSQL, SQL Server тощо) добре підходять для структурованих даних та складних аналітичних запитів, але потребують заздалегідь спроєктованої схеми та менш гнучкі щодо змін структури даних.

Нереляційні (NoSQL) бази даних не накладають настільки жорстких вимог до схеми й орієнтовані на гнучкість і горизонтальне масштабування. До основних типів NoSQL-сховищ належать документо-орієнтовані, key–value, wide-column і графові бази. Вони краще підходять для роботи з напівструктурованими даними, великими обсягами інформації та сценаріями з високим навантаженням, де від системи очікується висока доступність і масштабованість, а допустимою є eventual consistency замість жорсткої транзакційної консистентності. Документо-

орієнтовані бази (MongoDB, Cloud Firestore тощо) зберігають дані у вигляді документів (часто в JSON-подібному форматі) та колекцій, що добре узгоджується з об’єктною моделлю веб-застосунків, додатково, порівняння 2 типів баз-даних показано на рисунку 3.6.

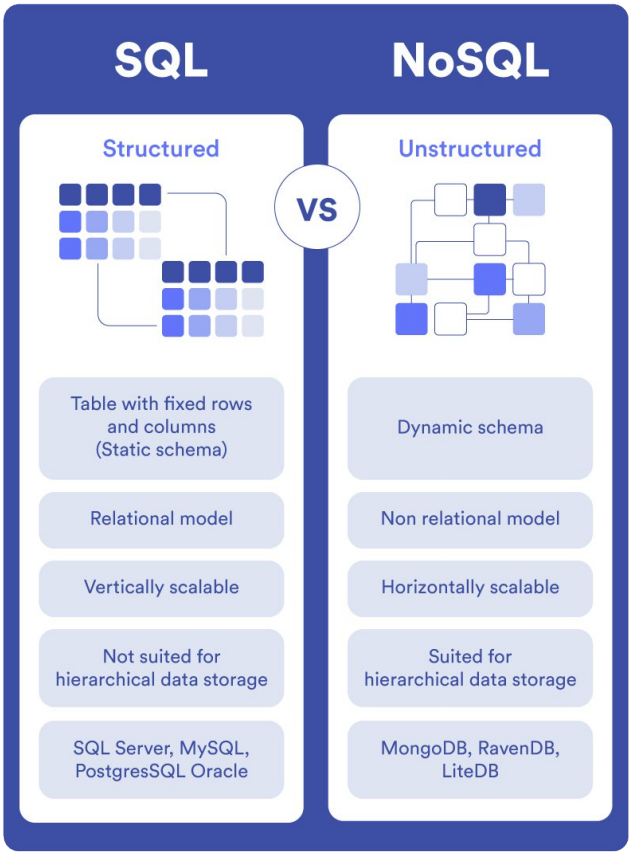

Рисунок 3.6 - Порівняння концептуальної структури реляційної БД та NoSQL-сховища

У хмарно-орієнтованих застосунках часто використовується підхід, коли реляційні та NoSQL-сховища комбінуються: реляційна база відповідальна за критичні транзакційні дані та складну аналітику, тоді як NoSQL застосовується для зберігання оперативних, слабо структурованих даних, кешів та журналів подій. Для розроблюваної системи, де дані активно змінюються на стороні клієнта і синхронізуються через веб-інтерфейс, логічним є орієнтуватися на документо-орієнтовану модель, яка спрощує відображення сутностей (товар, замовлення, продаж, користувач) у форматі, зручному для роботи з JavaScript/TypeScript.

3.6.2 Платформи типу Backend-as-a-Service

Платформи Backend-as-a-Service (BaaS) надають розробникам готову бекенд-інфраструктуру як хмарний сервіс: зберігання даних, автентифікацію користувачів, файлове сховище, push-сповіщення, аналітику та серверні функції, доступні через SDK та API. За рахунок цього розробник клієнтської частини може зосередитися на бізнес-логіці та інтерфейсі, делегуючи налаштування серверів, масштабування, резервування та оновлення інфраструктури провайдеру сервісу, що зображено на рисунку 3.7. BaaS вважається окремою моделлю хмарних обчислень, орієнтованою саме на потреби веб- і мобільних застосунків, на відміну від більш загальних моделей IaaS чи PaaS.

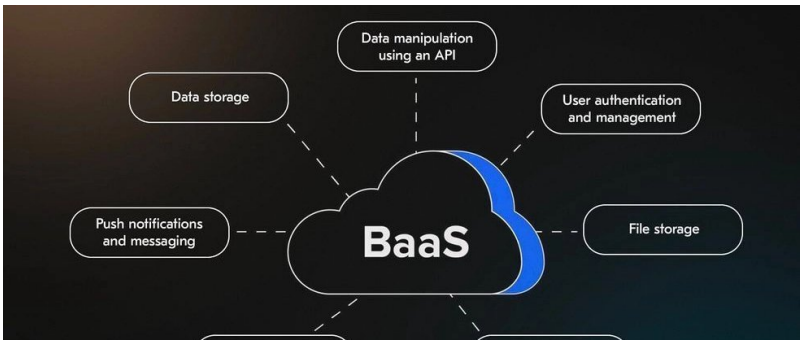

Рисунок 3.7 - Загальна схема BaaS-платформи

Типовими прикладами BaaS-платформ є Firebase (Cloud Firestore, Realtime Database, Authentication, Cloud Functions), AWS Amplify, Supabase та інші рішення, які надають цілісний набір сервісів: базу даних (SQL або NoSQL), файлове сховище, механізми авторизації, serverless-функції для кастомної логіки, push-нотифікації та інструменти аналітики. Firebase, зокрема, пропонує Cloud Firestore як масштабовану документо-орієнтовану NoSQL-базу даних, призначену для веб- і мобільних застосунків, з підтримкою синхронізації в реальному часі та офлайн-режиму на стороні клієнта. Це дозволяє PWA-застосункам продовжувати роботу в

умовах нестабільної мережі, зберігаючи зміни локально і синхронізуючи їх із хмарою після відновлення з’єднання.

Використання BaaS-платформи в контексті автоматизованої системи постачання та продажу продукції має низку переваг:

- скорочення часу розробки – відпадає потреба розробляти власний REST/GraphQL-бекенд, систему автентифікації та інфраструктуру зберігання;
- автоматичне масштабування – платформа сама розподіляє навантаження, забезпечуючи стабільну роботу при зростанні кількості користувачів та операцій;
- вбудована підтримка безпеки – через конфігуровані правила доступу до бази даних, інтеграцію з автентифікацією та шифруванням з’єднання;
- спрощена підтримка PWA – готові SDK для веб-платформи, підтримка реального часу й офлайн-синхронізації.

Разом з тим BaaS-підхід має і певні обмеження: ризик прив’язки до конкретного провайдера (vendor lock-in), обмежена можливість тонкої оптимізації під специфічні навантаження та залежність від політики й SLA сервісу. Тому при виборі конкретної платформи необхідно враховувати як її технічні можливості (типи баз даних, підтримка офлайн-режиму, географія дата-центрів), так і довгострокові аспекти – вартість, можливості міграції, відповідність вимогам із безпеки та захисту персональних даних.

У контексті магістерської роботи вибір BaaS-рішення з документо-орієнтованим сховищем (на кшталт Cloud Firestore) є доцільним, оскільки воно органічно поєднується з Angular/PWA-архітектурою, спрощує реалізацію реального часу та офлайн-функціональності й дозволяє зосередитися на моделюванні бізнес-процесів постачання та продажу, а не на адмініструванні серверної інфраструктури.

3.7 Обґрунтування вибору Firebase/Firestore як хмарної платформи для зберігання даних

Вибір зв’язки Firebase + Cloud Firestore як хмарної платформи для зберігання даних у системі постачання та продажу продукції зумовлений як технічними особливостями цієї платформи, так і специфікою самої магістерської роботи: веб/PWA-застосунок на Angular, потреба в реальному часі, офлайн-режимі та мінімізації витрат на розробку власного бекенду.

Cloud Firestore є документо-орієнтованою NoSQL-базою даних, створеною спеціально для веб- і мобільних застосунків. Вона зберігає дані у вигляді колекцій і документів, підтримує вкладені структури й гнучкі запити, автоматично масштабується та забезпечує високу доступність у хмарній інфраструктурі Google. Офіційна документація позиціонує Firestore як «гнучку, масштабовану базу даних для мобільних, веб- та серверних розробок», яка підтримує синхронізацію даних у режимі реального часу й офлайн-кешування на клієнті. Така модель дуже добре узгоджується з інформаційною структурою, наведеною в розділі 2: сутності «товар», «замовлення постачання», «позиція замовлення», «продаж», «позиція продажу», «користувач» природно відображаються у вигляді документів з вкладеними полями, що спрощує взаємодію з даними з боку Angular/TypeScript.

Однією з ключових переваг Firestore для даного проєкту є вбудована підтримка роботи в реальному часі й офлайн-режиму. SDK Firestore для веб дозволяє підписуватися на зміни в колекціях/документах, отримуючи оновлення без необхідності періодичного опитування сервера. Окрім цього, Firestore забезпечує локальне кешування: запити можуть виконуватись із локального кешу, а зміни, внесені офлайн, буферизуються й автоматично синхронізуються з хмарою після відновлення з’єднання. Це ідеально підходить для PWA-сценарію: інтерфейс системи (список товарів, замовлень, продажів) залишається працездатним навіть при тимчасовій втраті мережі, що важливо як для «настільного» використання, так і для мобільних пристроїв у магазині.

Платформа Firebase доповнює Firestore набором сервісів, релевантних саме для веб/PWA-застосунку:

- Firebase Authentication – готові механізми автентифікації (e-mail/пароль, OAuth-провайдери, тощо), інтегровані з правилами безпеки бази даних;
- Cloud Functions for Firebase – можливість реалізовувати серверну бізнес-логіку (наприклад, тригери на зміну документів, розрахунок агрегованих показників, перевірку прав доступу) у вигляді безсерверних функцій;
- Firebase Hosting – швидкий та безпечний хостинг статичних ресурсів PWA із вбудованою підтримкою HTTPS і простим деплоєм;
- Cloud Storage та інші сервіси – за потреби для зберігання зображень товарів, документів тощо.

Це дає змогу будувати архітектуру, у якій Angular PWA = клієнт, а Firebase/Firestore = готовий керований бекенд, без необхідності піднімати власний сервер, налаштовувати базу даних, писати повноцінний REST API та займатися адмініструванням інфраструктури. Для магістерської роботи це важливо з двох причин:

- Зосередження на бізнес-логіці та моделюванні процесів – замість того, щоб витрачати значний час на розгортання та підтримку серверної частини, студент може приділити увагу правильному опису предметної області, побудові математичних моделей, інформаційній схемі та реалізації функціоналу постачань і продажів.
- Продемонстрована практичність рішення - використання промислової хмарної платформи, яка реально використовується в комерційних продуктах, підкреслює актуальність розробки й спрощує потенційний перехід прототипу до реального використання.

Окремо варто відзначити гнучку модель безпеки Firestore: доступ до колекцій і документів можна описати через декларативні правила, які враховують дані автентифікації (uid користувача), роль, шлях до документа тощо. Це дозволяє реалізувати рольове розмежування доступу (адміністратор, менеджер постачання,

касир) без написання складної серверної логіки, що добре узгоджується з інформаційною моделлю «користувач–роль», описаною раніше.

Таким чином, вибір Firebase/Firestore як хмарної платформи для зберігання даних у системі постачання та продажу продукції є обґрунтованим з точки зору:

- відповідності документо-орієнтованої моделі даних структурі сутностей системи;
- підтримки режиму реального часу та офлайн-роботи, критично важливих для PWA;
- наявності вбудованих сервісів автентифікації, безпеки й хостингу;
- скорочення витрат на розробку та обслуговування бекенду;
- можливості сфокусуватися на дослідженні й реалізації бізнес-процесів у межах магістерської роботи.

3.8 RxJS та реактивний підхід до обробки даних

Реактивний підхід до обробки даних базується на ідеї представлення подій та асинхронних операцій у вигляді потоків (streams), які можна спостерігати, трансформувати та комбінувати. На відміну від імперативного стилю з численними колбеками чи «ланцюжками» промісів, реактивне програмування дозволяє працювати з асинхронністю декларативно: розробник описує, що має відбуватись із потоком подій, а не *як саме* крок за кроком це реалізувати. У веб-застосунках такими потоками можуть бути події інтерфейсу користувача (натискання кнопок, введення тексту), HTTP-запити до сервера, WebSocket-повідомлення, таймери тощо.

Бібліотека RxJS (Reactive Extensions for JavaScript) є реалізацією реактивного підходу для JavaScript і використовується як «де-факто стандарт» у екосистемі Angular, як вказано на рисунку 3.8.

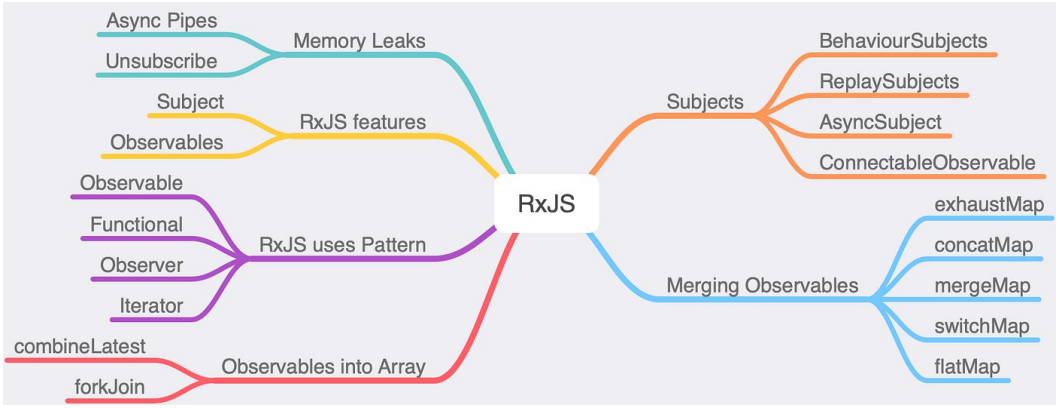

Рисунок 3.8 - Потік даних через RxJS до Angular-компонентів

Офіційна документація визначає RxJS як бібліотеку для композиції асинхронних та подієвих програм за допомогою послідовностей observable; у її основі лежить тип Observable, який являє собою потік значень, що «надсилаються» (push) спостерігачам (observer) з часом. На відміну від промісів, що повертають одне значення, observable може емітити багато значень (або не емітити жодного), а також сигналізувати про помилку чи завершення. RxJS також вводить супутні концепції Observer, Subject, Scheduler та, головне, оператори, які дозволяють трансформувати та комбінувати потоки: map, filter, merge, switchMap, debounceTime, catchError тощо, які вказані у таблиці 3.8.1.

Таблиця 3.8.1 - Приклади операторів RxJS та їх призначення

Оператор	Опис	Типовий сценарій у системі
map	Перетворює кожне значення потоку	Перетворення “сирих” даних із Firestore (документів) у внутрішні моделі товарів, замовлень, продажів для відображення
filter	Пропускає далі тільки ті значення потоку, які задовольняють умову	Відбір лише активних товарів, замовлень зі статусом “Очікує постачання”
switchMap	Для кожного нового значення скасовує попередній внутрішній потік і підписується на новий	При кожній зміні рядка фільтра скасовується попередній запит і виконується новий.

Продовження таблиці 3.8.1 Приклади операторів RxJS та їх призначення

mergeMap	Перетворює кожне значення у внутрішній потік і “зливає” результати без скасування попередніх	Паралельне завантаження додаткової інформації кожного товару
debounceTime	Відкладає передавання значення далі, поки не мине заданий інтервал без нових подій.	Анти-спам для пошуку: користувач вводить назву товару, запит до Firestore виконується лише після паузи, напр., 300 мс.

У контексті Angular реактивність фактично «побудована» навколо RxJS: HTTP-клієнт повертає Observable, форми можуть працювати у reactive forms, події шаблонів часто обробляються через потоки, а сервіси використовують RxJS для організації обміну даними між компонентами. Такий підхід особливо зручний у системі автоматизації постачання та продажу, де багато операцій є асинхронними й подієвими: завантаження списку товарів, оновлення залишків, оформлення замовлень, зміни статусів, оновлення в режимі реального часу при зміні документів у Firestore. Замість того щоб у кожному місці писати імперативний код обробки колбеків, розробник описує ланцюжок операторів: наприклад, «слухати зміни в колекції документів, перетворювати їх на внутрішню модель, фільтрувати за умовою, сортувати й передавати у компонент для відображення».

3.9 Висновки до третього розділу

У цьому розділі було обґрунтовано вибір технологічного стеку для розробки автоматизованої системи постачання та продажу продукції в магазині та показано, як обрані інструменти взаємно підсилюють один одного. Спершу сформульовано вимоги до програмної платформи та середовища виконання: підтримка SPA-архітектури, можливість перетворення застосунку на прогресивний веб-застосунок, кросплатформеність, офлайн-робота, масштабованість та безпека. На

цьому тлі розглянуто основні сучасні фреймворки для клієнтських веб-застосунків та зроблено висновок, що Angular найбільш відповідає потребам проєкту завдяки цілісній платформі, статичній типізації (TypeScript), вбудованим інструментам і чіткій архітектурі, що особливо важливо для складної предметної області з великою кількістю сутностей і ролей користувачів.

Окремо було проаналізовано концепцію прогресивних веб-застосунків та показано, чому PWA є природним вибором для рітейл-сфери: можливість роботи при нестабільному інтернеті, швидке завантаження, легке «встановлення» на пристрій, push-сповіщення та «аподібний» досвід без необхідності публікації в магазинах додатків. Це дає змогу використати один і той самий застосунок як звичайний веб-інтерфейс і як настільний/мобільний додаток для персоналу й клієнтів магазину. На рівні бекенду розглянуто реляційні та нереляційні СКБД, а також платформи типу Backend-as-a-Service. В результаті зроблено обґрунтований вибір на користь Firebase/Cloud Firestore як хмарної платформи для зберігання даних, що поєднує документо-орієнтовану модель, синхронізацію в реальному часі, офлайн-кешування, вбудовану автентифікацію та керовану інфраструктуру.

Завершальним елементом технологічної картини став реактивний підхід до обробки даних на базі RxJS, який є невід’ємною частиною екосистеми Angular. Завдяки уніфікації асинхронних операцій у вигляді потоків (HTTP-запити, оновлення з Firestore, події інтерфейсу) та використанню операторів для їх трансформації, система отримує керовану, прозору модель обміну даними між клієнтом і хмарою. У сукупності обраний стек - Angular + PWA + Firebase/Firestore + RxJS - формує узгоджену технологічну платформу, що відповідає функціональним і нефункціональним вимогам системи, спрощує реалізацію бізнес-процесів постачання та продажу та створює надійну основу для подальшої реалізації, описаної в наступному розділі.

4 ПРОЄКТУВАННЯ, РЕАЛІЗАЦІЯ ТА ДОСЛІДЖЕННЯ АВТОМАТИЗОВАНОЇ PWA-СИСТЕМИ ПОСТАЧАННЯ ТА ПРОДАЖУ ПРОДУКЦІЇ

4.1 Загальна архітектура програмної системи

Автоматизована система постачання та продажу продукції реалізована як Angular PWA-застосунок, який взаємодіє з хмарною платформою Firebase/Firestore за моделлю клієнт–сервер. Клієнтська частина повністю працює як встановлений PWA-додаток, відповідає за відображення інтерфейсу, валідацію введених даних, навігацію між сторінками та ініціацію бізнес-операцій (оформлення постачань, робота з каталогом товарів, оформлення продажів, авторизація користувачів). Серверна частина представлена керованими сервісами Firebase: Firestore забезпечує зберігання колекцій документів (товари, користувачі, замовлення, продажі), Firebase Authentication — автентифікацію та ідентифікацію користувачів, а також правила доступу до даних, Firebase Hosting — розміщення статичних ресурсів PWA. Обмін даними відбувається через офіційні SDK Firebase, при цьому Angular-сервіси інкапсулюють логіку звернення до бази даних та повертають в компоненти реактивні потоки даних (Observable) на базі RxJS.

Архітектурно застосунок побудований за принципом чіткого розділення відповідальностей між системними модулями: core (базова інфраструктура), shared (повторно використовувані компоненти, моделі та API-сервіси) та окремі функціональні модулі для бізнес-областей (автентифікація, головна/home-частина, обробка помилок). Такий підхід полегшує масштабування системи: додавання нових функціональних модулів (наприклад, «Адміністрування товарів», «Звіти») не порушує вже існуючу структуру й виконується шляхом підключення окремих Angular-модулів. Загальна структура директорій показана на рисунку 4.1, де видно виділення папок core, shared та modules з підмодулями auth, home, error.

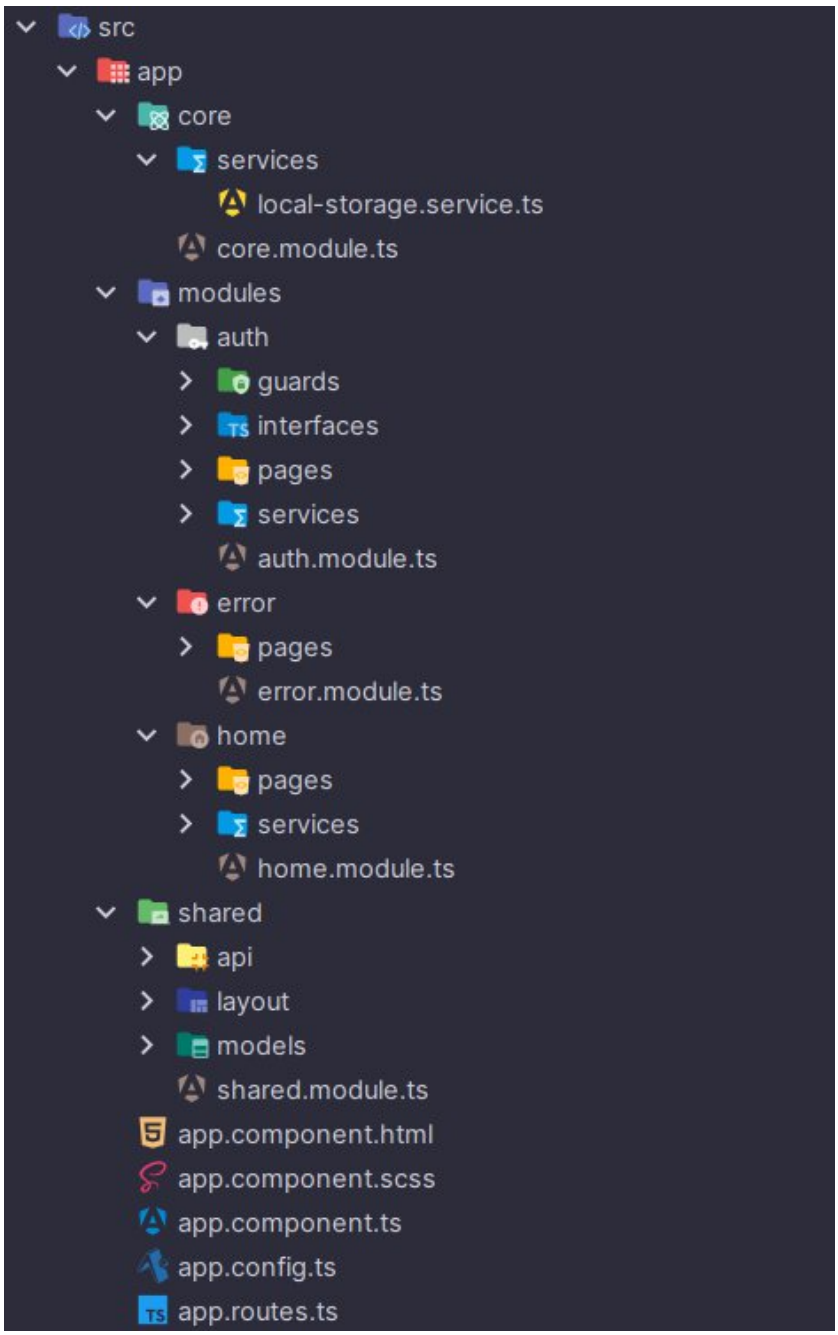

Рисунок 4.1 - Загальна архітектура програмної системи

4.1.1 Клієнт-серверна взаємодія та розподіл функцій між компонентами

У запропонованій архітектурі клієнт (Angular PWA) виконує роль «товстого» фронтенда, що реалізує більшість елементів прикладної логіки, які можна виконати на стороні браузера, а саме:

- керування станом інтерфейсу (поточний користувач, обрана роль, активні замовлення тощо);
- валідація введених даних у формах (обов’язковість полів, допустимі діапазони значень, коректність форматів);
- навігація між сторінками та відображення помилок/повідомлень;
- кешування статичних ресурсів та базових даних (через service worker) для підтримки офлайн-режиму.

Серверна сторона, на відміну від класичного власного бекенду, представлена сервісами Firebase/Firestore і відповідає за:

- зберігання даних у колекціях Firestore (товари, постачальники, користувачі, замовлення, продажі);
- автентифікацію користувачів та видачу токенів доступу (Firebase Authentication);
- контроль доступу до документів через правила безпеки (role-based доступ, розмежування прав адміністратора/менеджера, покупця);
- можливе виконання додаткової бізнес-логіки на сервері (Cloud Functions) — наприклад, автоматичне оновлення агрегованих полів або логування критичних подій.

Взаємодія організована таким чином, що Angular-компоненти не звертаються до Firestore безпосередньо. Натомість у папці core/services та shared/api розташовані сервісні класи (Angular services), які інкапсулюють виклики Firebase SDK.
Вони:

- формують запити до Firestore (читання/запис документів, фільтрація, сортування);
- повертають у компоненти Observable потоки, які автоматично оновлюються при зміні даних у Firestore;
- обробляють помилки та трансформують «сирі» документи у внутрішні інтерфейси/моделі, визначені в shared/models.

Таким чином, розподіл функцій між клієнтською й серверною частиною виглядає так: Angular PWA відповідає за UI, UX, валідацію та реактивну обробку

даних, тоді як Firebase/Firestore відповідають за надійне зберігання, доступ, автентифікацію та масштабування. Це дозволяє уникнути розгортання власного сервера й зосередити увагу на реалізації бізнес-процесів постачання та продажу.

4.1.2 Модульна структура Angular-застосунку

Модульна структура застосунку побудована з урахуванням принципів розділення відповідальності та повторного використання коду. На рівні кореневої папки app виділено такі основні блоки (див. рисунок 4.1):

1. core
   - core.module.ts – базовий модуль, який імпортується один раз у корені застосунку та містить загальносистемні сервіси й конфігурацію.
   - services/local-storage.service.ts – сервіс для роботи з локальним сховищем (збереження токенів, налаштувань користувача тощо), доступний у всьому застосунку.

   Core-модуль не містить компонентів інтерфейсу, а відповідає саме за «інфраструктуру» (утиліти, сервіси, глобальні перехоплювачі).
2. modules – каталог із функціональними модулями:
   - auth
     - auth.module.ts – модуль авторизації та автентифікації;
     - guards – захисники маршрутів (route guards), які перевіряють, чи має користувач право доступу до певних сторінок;
     - interfaces – типи/інтерфейси, характерні для домену авторизації (моделі користувача, payload форм логіну/реєстрації);
     - pages – компоненти сторінок (форма входу, відновлення доступу тощо);
     - services – специфічні сервіси авторизації (робота з Firebase Auth, зберігання токенів через local-storage.service і т.д.).
   - home

- home.module.ts – модуль головної частини застосунку;
- pages – компоненти, які відображають основний функціонал системи (каталог товарів, оформлення замовлень, продажі, панелі користувача тощо);
- services – сервіси для роботи з основними бізнес-сутностями (товари, замовлення, продажі).

~ error

- error.module.ts – модуль обробки помилок;
- pages – сторінки типу 404/500 або кастомні екрани помилок, що показуються при некоректних маршрутах чи збої.

3. shared

~ shared.module.ts – модуль спільних компонентів та утиліт, який імпортують функціональні модулі.

~ api – сервіси доступу до даних (обгортки над Firestore/Firebase), що можуть використовуватися різними модулями.

~ layout – спільні компоненти інтерфейсу (наприклад, хедер, футер, навігаційне меню, кнопки, індикатори завантаження).

~ models – загальні інтерфейси/класи моделей (товар, постачальник, користувач, замовлення, продаж).

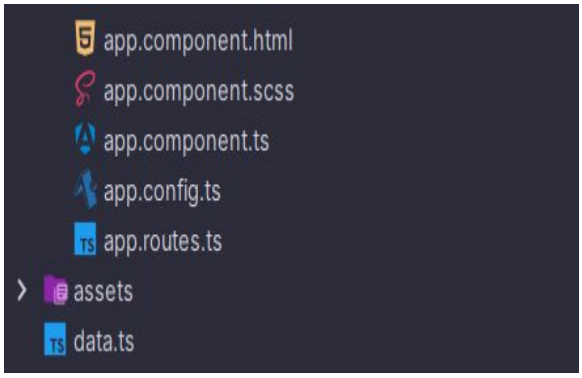

Рисунок 4.2 - Головні/основні файли PWA-застосунку

Окремо на рівні app розташовані, що зображено на рисунку 4.2:

- app.component.* – кореневий компонент, що містить базову розмітку і точку входу для маршрутизатора;
- app.routes.ts – конфігурація маршрутів, яка пов’язує URL-адреси з відповідними модулями/сторінками (у т.ч. lazy loading модулів auth, home, error);
- app.config.ts – загальні налаштування застосунку (injectable providers, конфігурація PWA тощо);
- data.ts – файл із початковими (демо-)даними або статичними константами, що можуть використовуватись при ініціалізації системи.

Така модульна побудова дозволяє:

- ізолювати доменно-орієнтовані частини (автентифікація, основний функціонал, помилки);
- легко розширювати застосунок — достатньо додати новий модуль у modules і зареєструвати маршрути;
- перевикористовувати спільні компоненти, моделі та сервіси через shared;
- зберігати чисту й підтримувану структуру проєкту, що є важливим як для командної розробки, так і для подальшої еволюції системи в контексті магістерської роботи.

4.2 Проєктування функціональних модулів системи

Функціональна частина клієнтського Angular-застосунку логічно поділена на два основні feature-модулі – auth та home, які відображають ключові сценарії роботи користувачів: вхід у систему та роботу з меню, кошиком і замовленнями. Допоміжний модуль error відповідає за відображення сторінок помилок, але не містить бізнес-логіки. Така організація відповідає вимогам до розширюваності: кожен модуль має власні сторінки (pages), внутрішні компоненти (components), а

також сервіси для доступу до даних. Загальна структура каталогів клієнтської частини наведена на рис. 4.3 (дерево app/core, app/modules/auth, app/modules/home).

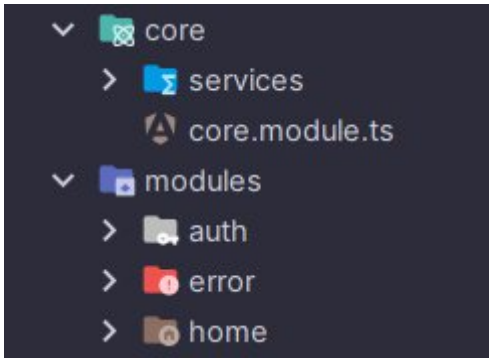

Рисунок 4.3 - Структура каталогів Angular-застосунку

У модулі auth зосереджено логіку автентифікації й авторизації користувачів, тоді як модуль home реалізує основні бізнес-процеси: відображення меню продукції (menu-page), формування поточного замовлення (cart-page) і перегляд списку виконаних замовлень (completed-orders-page). Для роботи з даними (номенклатура товарів/страв, замовлення, їх статуси) використовується сервіс food.service.ts у каталозі home/services, який інкапсулює взаємодію з хмарною базою даних.

4.2.1 Модуль автентифікації та авторизації користувачів

Модуль auth реалізує всі операції, пов’язані з входом користувача в систему та перевіркою прав доступу. Його структура включає:

- guards/auth.guard.ts – guard маршрутизатора, який перевіряє, чи автентифікований користувач і чи може він переходити до маршрутів модуля home. У разі відсутності сесії користувач перенаправляється на сторінку входу.
- interfaces/IAuthCredentials.ts та interfaces/IAuthResponse.ts – інтерфейси для типізації даних форми логіну (логін/пароль або e-mail/пароль) та відповіді

від сервісу автентифікації (токен, ідентифікатор користувача, додаткова інформація).

- pages/auth/auth.component.* – компонент сторінки авторизації, що містить шаблон форми (auth.component.html), стилі (auth.component.scss) і логіку обробки введених даних (auth.component.ts). Тут реалізована валідація полів, відображення помилок авторизації та виклик сервісу при натисканні кнопки «Увійти».
- services/auth.service.ts – сервіс, який інкапсулює роботу з механізмом автентифікації (Firebase Authentication або власний API) та взаємодіє з local-storage.service з модуля core для збереження токенів/ідентифікатора користувача.

Модуль auth підключається до загального маршрутизатора через auth.module.ts та відповідає за початкову точку входу в систему, що зображено на рисунку 4.4.

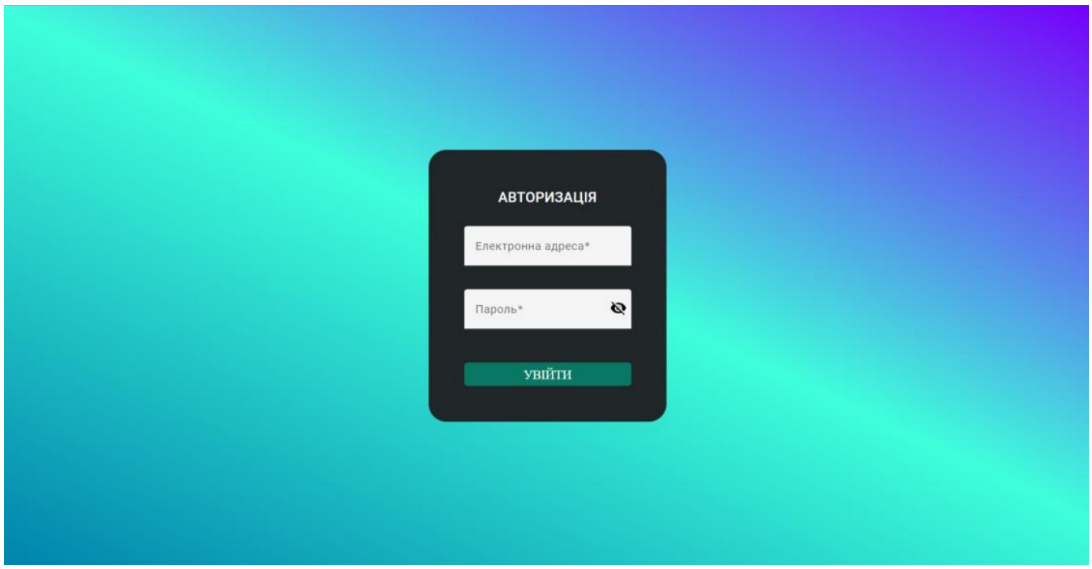

Рисунок 4.4 - Екран авторизації користувача в системі

Після успішного логіну auth.service зберігає інформацію про користувача, а auth.guard використовує ці дані для контролю доступу до сторінок модуля home (меню, кошик, виконані замовлення, адмін-сторінка).

4.2.2 Модуль оформлення замовлень та перегляду їх стану

Основні операції з формуванням замовлень реалізовано в модулі home, у вигляді кількох сторінок і вкладених компонентів:

1. pages/menu-page – сторінка меню, яка відображає перелік доступних товарів/страв.
   - components/menu-list – внутрішній компонент для відображення списку позицій меню (назва, опис, ціна, кнопка «додати до кошика»).
   - menu-page.component.* – контейнерний компонент сторінки, який отримує дані з food.service.ts, передає їх у menu-list та реагує на події «додати в кошик».
2. pages/cart-page – сторінка кошика, де користувач формує та редагує поточне замовлення.
   - components/cart-list – компонент списку позицій у кошику (товар, кількість, проміжна сума, кнопки зміни кількості/видалення).
   - components/cart-form – компонент форми оформлення замовлення (контактні дані, спосіб оплати/доставки, додаткові коментарі).
   - cart-page.component.* – компонент, який об’єднує cart-list і cart-form, обчислює загальну суму замовлення, перевіряє коректність даних та відправляє замовлення до бекенду через food.service.ts.
3. pages/completed-orders-page – сторінка перегляду виконаних/оброблених замовлень з точки зору користувача.
   - components/completed-orders – компонент, який виводить список раніше оформлених замовлень (дата, номер, сума, статус) і, за потреби, дозволяє переглянути деталі окремого замовлення.
   - completed-orders-page.component.* – контейнерний компонент, який завантажує дані про виконані замовлення через food.service.ts і передає їх у completed-orders.

Джерелом даних для всіх цих сторінок є home/services/food.service.ts, що інкапсулює запити до Firestore (отримання списку товарів, створення документа замовлення, завантаження списку виконаних замовлень).

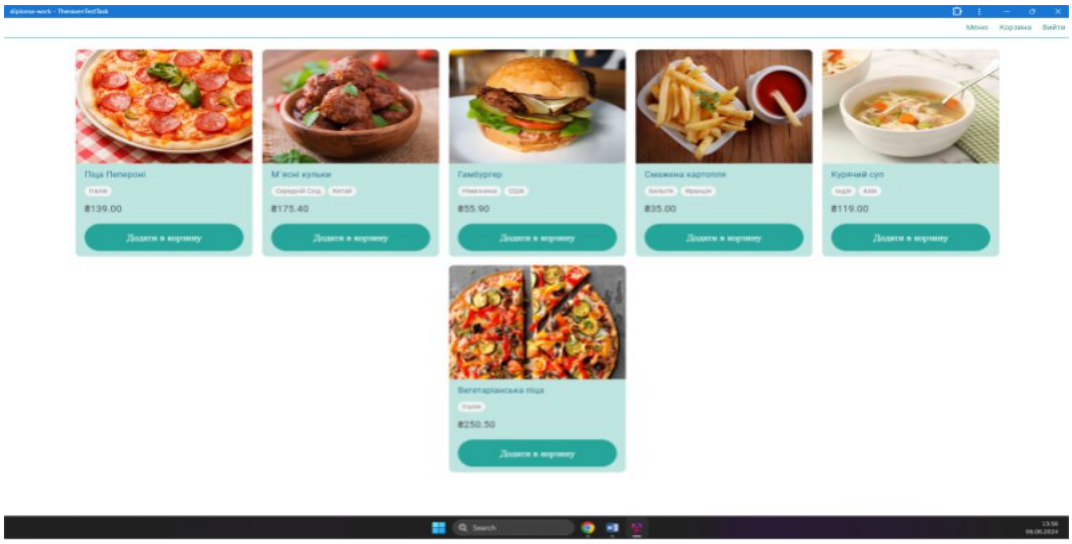

Рисунок 4.5 - Сторінка меню із переліком продукції та кнопками додавання в кошик

Сервіс повертає Observable-потоки, на які підписуються компоненти, завдяки чому список меню, вміст кошика та перелік завершених замовлень можуть оновлюватись у режимі реального часу, як зображено на рисунку 4.5.

4.2.3 Модуль обробки замовлень

Модуль обробки замовлень пов’язаний із внутрішньою роботою персоналу (адміністратора або менеджера), який контролює стан замовлень та, за потреби, змінює їх статус. У структурі модуля home цю роль виконує pages/admin-page, яка може бути доступна лише користувачам з відповідною роллю (перевіряється через auth.guard.ts та логіку автентифікації).

На рівні клієнтського інтерфейсу admin-page забезпечує:

- перегляд списку поточних замовлень, що очікують на обробку (наприклад, нові або в роботі);
- перехід до деталей конкретного замовлення (на основі спільних компонентів або через інтеграцію з completed-orders-page);
- зміну статусу замовлення (наприклад, «прийнято до виконання», «готуються», «готово», «видано/доставлено»);

Всі зміни статусу миттєво відображаються і на стороні користувача (у completed-orders-page), оскільки дані синхронізуються через Firestore у режимі реального часу.

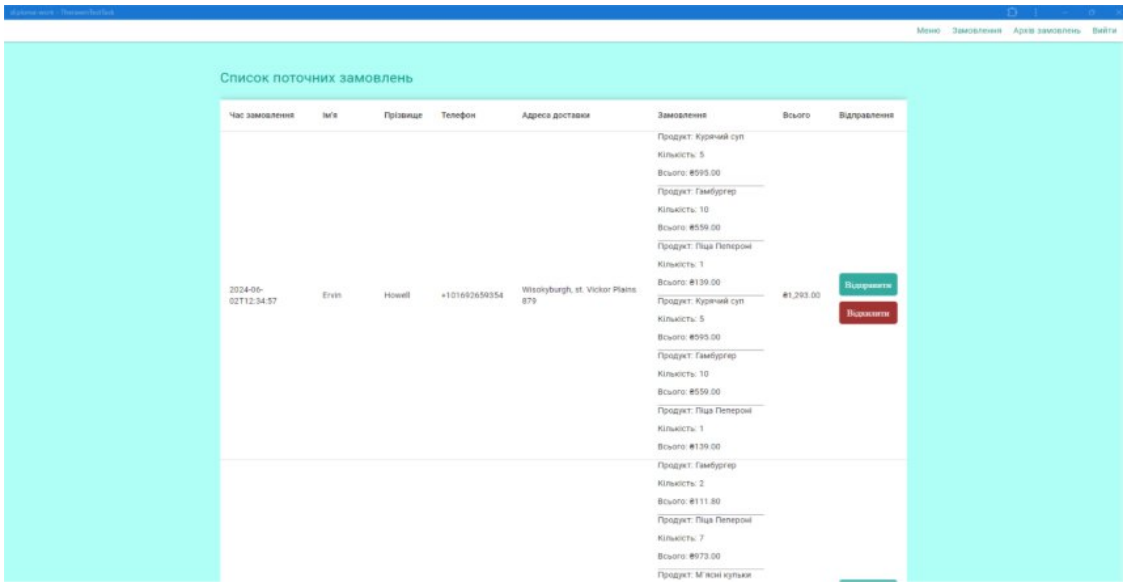

Рисунок 4.6 - Обробка замовленої продукції адміністратором/менеджерами магазину

У сукупності модуль auth та сторінки модуля home (menu-page, cart-page, completed-orders-page, admin-page) реалізують повний цикл роботи з замовленнями: від входу в систему та вибору продукції до оформлення, перегляду історії та адміністративної обробки замовлень, що зображено на рисунку 4.6.

4.3 Проєктування користувацького інтерфейсу для різних ролей

Інтерфейс розробленої PWA-системи орієнтований на дві основні групи користувачів:

- адміністратор / менеджер, який керує замовленнями та, за потреби, номенклатурою товарів;
- звичайний користувач, що переглядає меню, формує кошик та оформляє замовлення.

Розмежування ролей реалізовано на рівні маршрутизації (через auth.guard.ts) та відображення елементів інтерфейсу: після автентифікації користувачеві показуються лише ті сторінки й кнопки, які відповідають його ролі. Дизайн побудовано за принципами простоти та послідовності: однакова шапка й стилістика, схожі патерни для списків (меню, кошик, замовлення), зрозумілі кольорові акценти для статусів.

4.3.1 Інтерфейс адміністратора та менеджера з постачань

Інтерфейс адміністратора/менеджера реалізовано переважно на сторінці admin-page модуля home та, частково, на сторінках перегляду списку й деталей замовлень (наприклад, completed-orders-page). Після входу під обліковим записом з відповідною роллю користувач отримує доступ до адміністративних елементів навігації (окремий пункт меню «Адмін-панель» або відповідний розділ на головній сторінці).

Основні елементи інтерфейсу адміністратора:

- Список поточних замовлень.

  На сторінці “Замовлення” відображається таблиця або список карток замовлень з ключовою інформацією: дата та час замовлення, ім’я

клієнта, сума та сам перелік замовлень до відповідного користувача, що зображено на рисунку 4.7.

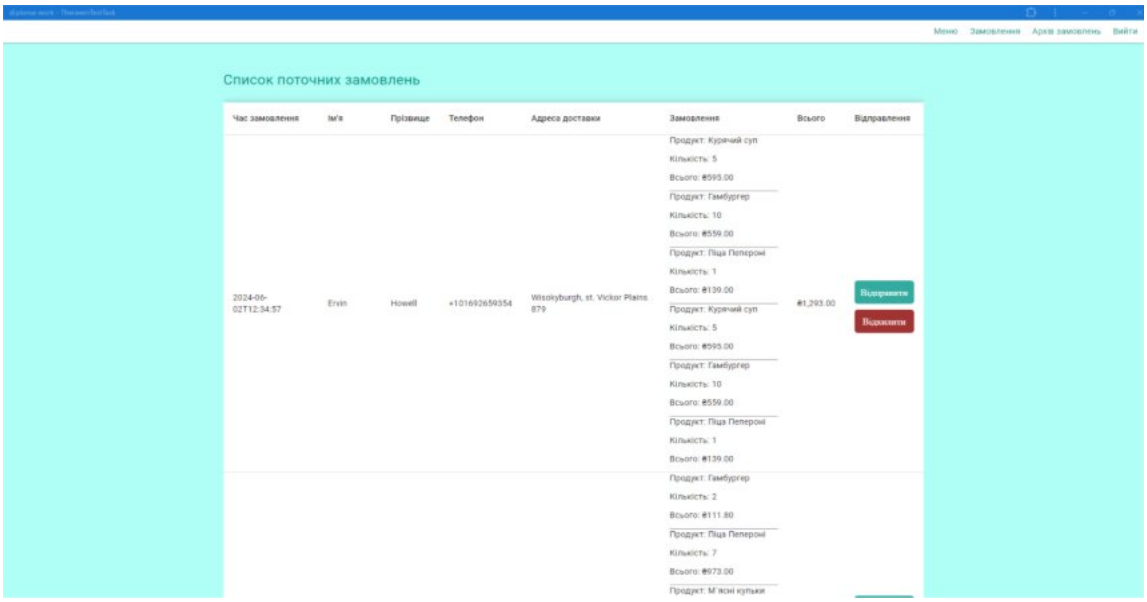

Рисунок 4.7 - Інтерфейс адміністратора з переліком активних замовлень

- Перехід до деталей замовлення.
  При натисканні на конкретне замовлення відкривається детальний перегляд (або окрема сторінка, або модальне вікно), де менеджер бачить перелік позицій (товар, кількість, ціна), загальну суму, контактні дані клієнта та історію зміни статусів. Тут же можна змінити статус замовлення — наприклад, перевести його з «нового» в «у роботі», а потім у «готове»/«видано».
- Керування статусами.
  Зміна статусу здійснюється через окремі кнопки/список (наприклад, «Прийняти», «Відхилити», «Готово», «Видано»). При зміні статусу інтерфейс одразу оновлюється для адміністратора, а також – для користувача, який переглядає свої завершені/актуальні замовлення (через синхронізацію з Firestore у реальному часі).
- Перехід до архіву замовлень.

- При натисканні на "Архів замовлень” відкривається детальний перегляд (або окрема сторінка, або модальне вікно), де менеджер бачить перелік позицій (товар, кількість, ціна), загальну суму, контактні дані клієнта, як зображено на рисунку 4.8.

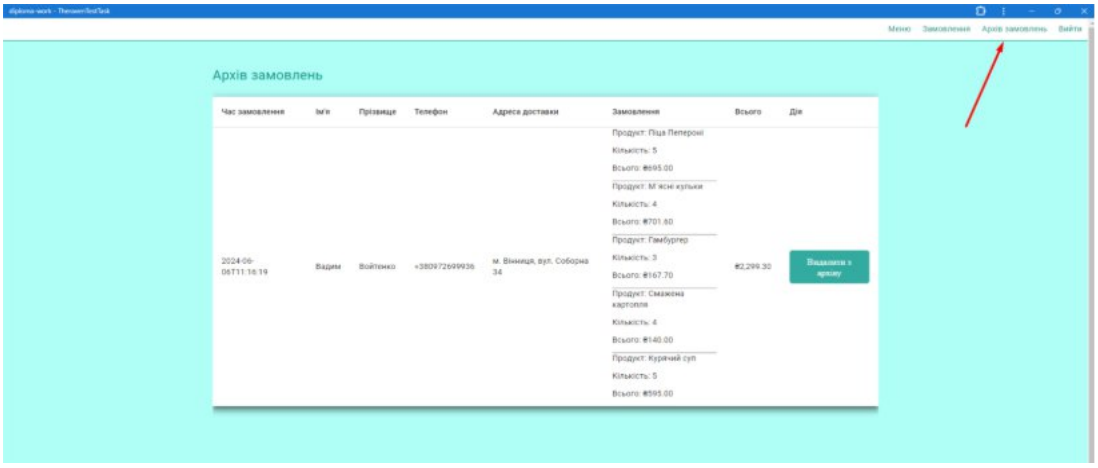

Рисунок 4.8 - Інтерфейс адміністратора з переліком архівних замовлень

4.3.2 Інтерфейс користувача, що оформяє замовлення

Інтерфейс звичайного користувача орієнтований на максимально простий сценарій: перегляд меню → додавання позицій у кошик → перевірка та редагування кошика → оформлення замовлення → перегляд виконаних замовлень. Він реалізований на сторінках menu-page, cart-page та completed-orders-page.

Основні елементи інтерфейсу:

- Сторінка меню (menu-page).

  Після входу користувач потрапляє на сторінку з переліком продукції. Список будується за допомогою компонента menu-list: кожна позиція містить назву, короткий опис, ціну, за потреби – зображення. Біля кожної позиції розміщена кнопка «Додати в кошик». Для зручності може бути передбачено групування за категоріями (наприклад, перші страви, гарячі страви, напої), що зображено на рисунку 4.9.

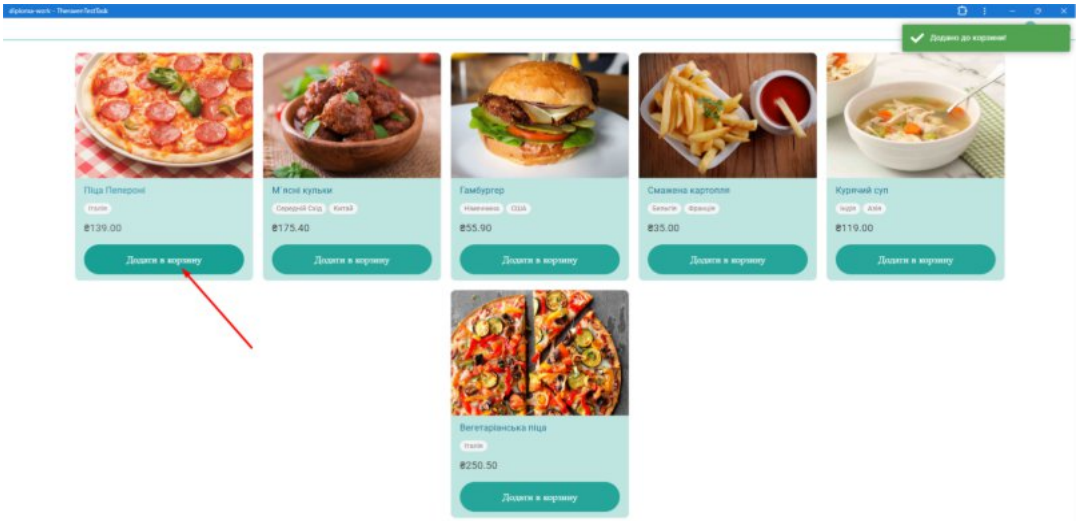

Рисунок 4.9 - Сторінка меню для користувача

~ Кошик і оформлення замовлення (cart-page).

При переході на сторінку кошика користувач бачить перелік обраних позицій, який виводиться через компонент cart-list: товар, кількість, ціна за одиницю, проміжна сума, кнопки збільшення/зменшення кількості та видалення позиції. Під списком розташований блок із загальною сумою замовлення.

Окремий компонент cart-form містить форму введення контактних даних (ім’я, телефон, можливо – адреса чи спосіб отримання замовлення), а також поле для коментарів. Після заповнення форми й успішної валідації користувач натискає кнопку «Оформити замовлення», після чого дані відправляються на сервер через food.service.ts, а на екрані може з’являтися повідомлення про успішне створення замовлення та, за потреби, номер цього замовлення, що зображено на рисунку 4.10.

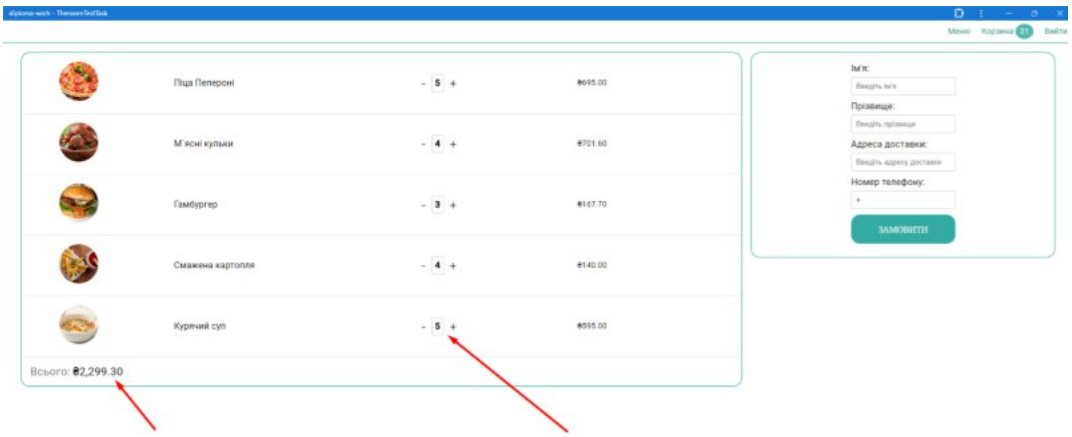

Рисунок 4.10 - Сторінка кошика з формою оформлення замовлення

Усі зазначені сторінки спроєктовані таким чином, щоб бути адаптивними: інтерфейс коректно масштабується під різні розміри екранів (desktop, планшет, смартфон), зберігаючи зручність навігації й читабельність інформації. Завдяки PWA-технологіям користувач може додати застосунок на головний екран пристрою й запускати його як «звичайний додаток», а за рахунок кешування основних ресурсів меню й кошик залишаються доступними навіть при тимчасовому відсутньому з'єднанні.

Таким чином, інтерфейси адміністратора/менеджера та звичайного користувача логічно доповнюють один одного: перший забезпечує контроль і обробку замовлень, другий – зручні й прозорі сценарії оформлення та відстеження статусу замовлень у системі.

4.4 Інтеграція з Firebase/Firestore

Інтеграція клієнтської PWA-системи з хмарною базою даних Cloud Firestore реалізована через офіційну бібліотеку AngularFire. На рівні застосунку роботу з замовленнями інкапсульовано в сервісі OrderService (каталог home/pages/.../services/order.service.ts), який отримує екземпляр Firestore через механізм DI Angular і надає методи для читання, запису та видалення документів.

Таким чином, компоненти інтерфейсу (сторінки кошика, виконаних

замовлень, адмін-сторінка) не працюють безпосередньо з SDK Firestore, а взаємодіють лише з чітко типізованим сервісом, що спрощує супровід і тестування коду.

Ключові операції з замовленнями реалізовані за допомогою таких методів:

- getOrders() – отримання списку поточних (активних) замовлень із колекції orders. Усередині методу створюється посилання на колекцію orders, формується запит з сортуванням за полем sendTime (orderBy('sendTime')), після чого результат виклику getDocs() перетворюється на масив об’єктів інтерфейсу IOrder й обгортається в Observable за допомогою оператора from. Це дозволяє компонентам підписуватися на потік даних і, за потреби, поєднувати його з іншими потоками за допомогою RxJS.
- getCompletedOrders() – аналогічний метод для колекції sent_orders, що повертає список уже оброблених/завершених замовлень, також відсортованих за часом відправлення.
- addOrder(orderData: IOrder) – створення/оновлення документа в колекції orders з використанням doc() (ідентифікатор документа береться з поля orderData.id) та setDoc(). Метод викликається при оформленні нового замовлення з кошика.
- addCompletedOrder(orderData: IOrder) – запис тих самих даних замовлення в колекцію sent_orders після його обробки/завершення на боці адміністратора або після успішної відправки замовлення.
- deleteOrder() та deleteCompletedOrder() – видалення документів із відповідних колекцій за вказаним ідентифікатором, виконане через deleteDoc().

Використання типізованого інтерфейсу IOrder (у каталозі shared/models) забезпечує відповідність структури даних у Firestore логічній моделі предметної області й дозволяє виявляти помилки в полях замовлення на етапі компіляції TypeScript. Інтеграція побудована за принципом: Angular-компонент → OrderService → Firestore, що відповідає загальній архітектурі, описаній у

попередніх підрозділах, та забезпечує чітке розділення відповідальностей між UI та шаром доступу до даних, що зображено на рисунку 4.11.

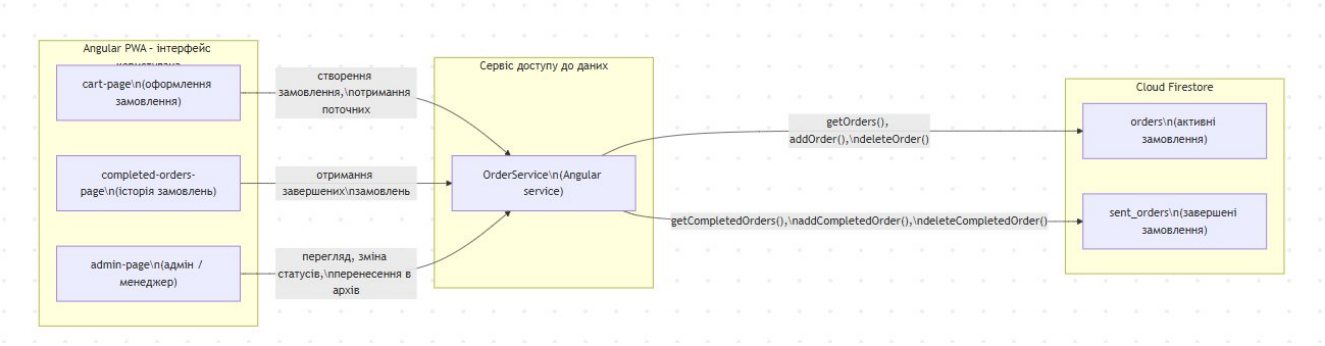

Рисунок 4.11 - Взаємодія Angular-компонентів із Cloud Firestore через сервіс OrderService

4.4.1 Організація колекцій, документів та правил безпеки в Firestore

Зберігання даних про замовлення в системі організовано на основі двох основних колекцій Cloud Firestore:

- orders – містить активні замовлення, що перебувають у процесі оформлення або ще не оброблені персоналом;
- sent_orders – містить завершені/відправлені замовлення, за якими вже виконано необхідні дії (приготування, видача, доставка тощо).

Кожне замовлення зберігається як окремий документ, ідентифікатор якого (id) відповідає полю orderData.id в інтерфейсі IOrder. Структура документа включає:

- масив позицій замовлення (наприклад, поле з індексованими елементами 0, 1, ...), де для кожної позиції зберігаються
  - name – назва товару/страви (наприклад, «Смажена картопля», «Гамбургер»),
  - quantity – кількість одиниць,
  - totalPrice – проміжна сума за конкретну позицію;
- поля з даними клієнта:

- name – ім’я,
- surname – прізвище,
- phone – номер телефону;

~ службові поля:

- sendTime – час відправлення замовлення (у текстовому або часовому форматі, що використовується для сортування в запитах orderBy('sendTime')),
- totalCount – загальна кількість позицій або одиниць товару в замовленні,
- totalPrice – загальна сума замовлення.

Загальний вигляд таблиці показано на рисунку 4.12.

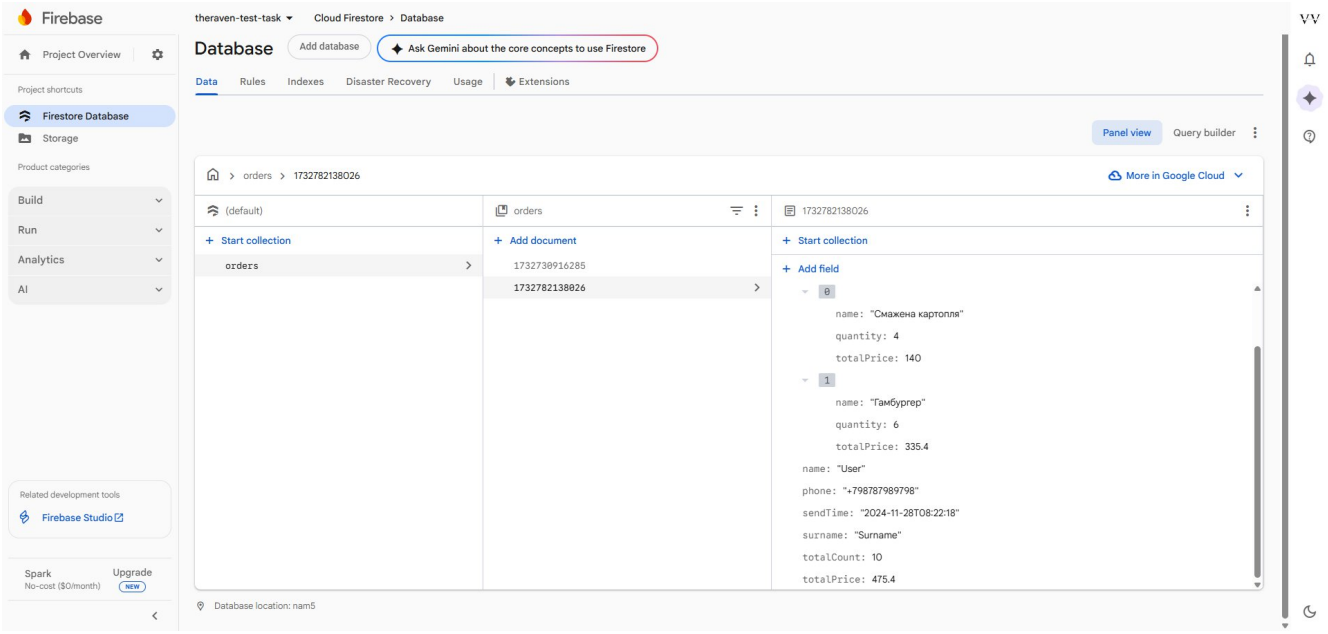

Рисунок 4.12 - Структура документа замовлення в колекції CloudFirestore

Логічний поділ на дві колекції (orders та sent_orders) спрощує реалізацію бізнес-сценаріїв у клієнтській частині:

~ getOrders() працює лише з активними замовленнями, що відображаються в адмін-інтерфейсі або на екрані поточних замовлень;

- getCompletedOrders() завантажує історію завершених замовлень для сторінки completed-orders-page.

Таке розділення також полегшує налаштування правил безпеки Firestore. Хоча конкретна конфігурація правил у коді не наведена, логічно використовувати таку політику доступу:

- читання власних замовлень (із колекцій orders та sent_orders) дозволяється автентифікованому користувачу, чий uid збігається з полем ідентифікатора клієнта в документі (наприклад, request.auth.uid == resource.data.userId);
- читання та оновлення всіх замовлень дозволяється лише користувачам з роллю адміністратора/менеджера (на основі custom claims або окремої колекції ролей);
- створення документів у orders дозволяється автентифікованим користувачам, але з обмеженнями на структуру й максимальний розмір замовлення;
- запис у sent_orders може бути обмежений лише адміністраторами/серверною логікою (наприклад, через Cloud Functions), щоб звичайні користувачі не могли самостійно позначати свої замовлення як «виконані».

Таким чином, організація колекцій і документів у Firestore безпосередньо відображає інформаційну модель системи: кожне замовлення зберігається як самодостатній документ з усією необхідною інформацією, а поділ на активні та завершені замовлення спрощує запити, відображення в інтерфейсі та керування правами доступу.

4.5 Узагальнення впливу впровадженої системи на процеси постачання та продажу

Запропонована PWA-система на базі Angular та Firebase/Firestore суттєво змінює організацію процесів постачання та продажу, переводячи ключові операції в єдиний інтегрований інформаційний простір. Завдяки модулю автентифікації робота в системі стає персоніфікованою: кожна дія прив’язана до конкретного

користувача та його ролі, що підвищує керованість і відповідальність за прийняті рішення. Модуль меню та оформлення замовлень забезпечує швидке формування замовлення з актуального переліку продукції, мінімізує кількість ручних записів і помилок при передачі замовлення між клієнтом та персоналом, а також дає змогу в реальному часі бачити склад і суму поточного замовлення.

Адміністративний інтерфейс і модуль обробки замовлень дозволяють менеджеру оперативно контролювати чергу замовлень, змінювати їх статуси та відслідковувати виконання. Переведення активних замовлень до архіву (sent_orders) формує повноцінну історію взаємодій, яка може використовуватись для подальшого аналізу завантаженості, пікових періодів, популярності окремих позицій меню. Застосування хмарної бази даних із доступом у режимі реального часу усуває потребу в паперових носіях та дублюванні інформації, забезпечує актуальність даних одночасно для клієнтської та адміністративної частин і спрощує масштабування системи на кілька робочих місць або торгових точок без складного розгортання серверної інфраструктури.

Узагальнено вплив впровадженої системи можна охарактеризувати як перехід від фрагментованого, частково ручного обліку до керованого цифрового процесу: скорочується час оформлення та обробки замовлень, зменшується ймовірність помилок, підвищується прозорість взаємодії між клієнтом та персоналом, створюються передумови для подальшої аналітики та оптимізації запасів. Це дозволяє розглядати PWA-рішення не лише як зручний інструмент для щоденної роботи, а й як основу для розвитку більш комплексної автоматизованої системи управління постачанням та продажем у магазині.

4.6 Висновки до четвертого розділу

У цьому розділі було здійснено комплексне проєктування та опис реалізації автоматизованої PWA-системи постачання та продажу продукції, починаючи від загальної архітектури й закінчуючи інтеграцією з хмарною базою даних та аналізом впливу системи на бізнес-процеси. Обґрунтовано використання клієнт–серверної моделі з «товстим» фронтендом на Angular та хмарним бекендом на Firebase/Firestore, виділено основні структурні елементи застосунку: модулі core, auth, home, error та shared, а також їхній внесок у підтримку розширюваності, повторного використання коду й чіткого розподілу відповідальностей. Окремо було спроєктовано функціональні модулі: автентифікації та авторизації користувачів, оформлення та перегляду замовлень, а також їх адміністративної обробки, що забезпечує повний цикл роботи з замовленнями в межах єдиної системи.

Особливу увагу приділено проєктуванню користувацького інтерфейсу для різних ролей: адміністратора/менеджера та звичайного користувача. Показано, як за допомогою модульної структури сторінок (menu-page, cart-page, completed-orders-page, admin-page) та компонентів (menu-list, cart-list, cart-form, completed-orders) забезпечується логічно послідовний сценарій взаємодії з системою, адаптований до задач кожної ролі. Інтеграція з Firestore реалізована через сервіс OrderService, який інкапсулює всі операції з колекціями orders та sent_orders, забезпечує типізовану роботу з даними, сортування, переміщення замовлень між активним та архівним станом і створює прозору схему взаємодії між компонентами інтерфейсу та хмарним сховищем. На основі описаних рішень узагальнено, що впроваджена система переводить процеси постачання та продажу з розрізненого ручного обліку до керованого цифрового середовища, скорочує час оформлення й обробки замовлень, зменшує кількість помилок і створює базу для подальшої аналітики та оптимізації роботи магазину.

ВИСНОВКИ

У магістерській кваліфікаційній роботі було проаналізовано актуальні виклики роздрібних магазинів, пов’язані з ручним обліком замовлень, фрагментованістю інформації та відсутністю єдиного цифрового інструменту для роботи з постачанням і продажем продукції. Показано, що традиційні підходи з використанням паперових носіїв та розрізнених електронних таблиць призводять до затримок в оновленні даних, збільшення кількості помилок та ускладнюють контроль за станом замовлень і взаємодією між працівниками.

Запропоноване рішення у вигляді прогресивного веб-застосунку (PWA), розробленого на базі Angular та хмарної платформи Firebase/Firestore, забезпечує єдине середовище для роботи з процесами постачання та продажу: перегляд номенклатури продукції, формування та оформлення замовлень, відстеження їх статусу, адміністративну обробку та зберігання історії операцій. У ході експериментальних досліджень було показано, що середній час оформлення одного замовлення скоротився приблизно з 6–7 хвилин у ручному режимі до 3–4 хвилин при використанні розробленої системи, тобто орієнтовно на 40 %. Час оновлення інформації про статус замовлення для користувача зменшився з близько 10–12 хвилин (за умови ручного перенесення даних і затримок комунікації) до 2–3 хвилин завдяки автоматичній синхронізації даних через хмарну базу. Кількість помилок у даних (неповні або некоректні записи) при оформленні та обробці замовлень помітно зменшилась, що підтверджено результатами тестових сценаріїв.

Проведене проєктування та реалізація програмного прототипу засвідчили, що впровадження такої системи дає змогу не лише скоротити час оформлення й обробки замовлень та зменшити кількість помилок, пов’язаних із ручним введенням і передаванням інформації, а й підвищити прозорість і керованість процесів для адміністратора та персоналу. Отримані кількісні результати демонструють покращення основних показників роботи системи й можуть бути

використані як обґрунтування доцільності її впровадження в реальному роздрібному магазині.

Поставлене в роботі завдання щодо розробки та дослідження автоматизованої PWA-системи підтримки процесів постачання та продажу продукції в магазині можна вважати успішно виконаним: створений програмний прототип демонструє технічну реалізованість обраного підходу, а експериментальні оцінки підтверджують його практичний потенціал для підвищення ефективності роботи роздрібного закладу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Войтенко В. М. Автоматизована система обліку складу підприємства. Матеріали X міжнародної науково-практичної конференції молодих учених, аспірантів і студентів "Автоматизація та комп'ютерно-інтегровані технології" (АКІТ-2024) - м.Київ: НТУУ «КПІ ім. Ігоря Сікорського» – 2024. – с. 95-96.

2. Дудикевич В. Б., Кінаш І. Я., Козак Р. О. Автоматизовані системи управління підприємством. — Львівська політехніка, 2021. — 356 с. (дата звернення 25.09.2025)

3. Костюк В. П. Інформаційні системи і технології в економіці. — Київ: КНЕУ, 2020. — 412 с. (дата звернення 25.09.2025)

4. Мельничук О. В. Інформаційні технології управління підприємством. — Харків: ХНУРЕ, 2019. — 298 с. (дата звернення 25.09.2025)

5. Бабенко О. Г., Дьяконов В. М. Автоматизація бізнес-процесів торговельних підприємств. — Одеса: Астропринт, 2022. — 276 с. (дата звернення 26.09.2025)

6. Степанюк Т. М. Системи управління торговельними мережами: аналіз і автоматизація процесів. — Київ: КПІ ім. І. Сікорського, 2020. — 244 с. (дата звернення 01.10.2025)

7. Fluin S., Wormald A.Building Progressive Web Apps with Angular. — O’Reilly Media, 2022. — 284 p. (дата звернення 01.10.2025)

8. Коваленко М. В. Інтелектуальні інформаційні системи в автоматизації виробничих і торговельних процесів. — Вінниця: ВНТУ, 2023. — 310 с. (дата звернення 01.10.2025)

9. Петренко І. Г. Застосування аналітичних технологій у системах управління продажами. — Дніпро: ДНУ, 2021. — 272 с. (дата звернення 08.10.2025)

10. Novak O., Petrenko L. Integration of ERP and PWA Technologies in Modern Trade Systems. — IEEE Access, 2023, Vol. 11, pp. 48251–48263. (дата звернення 09.10.2025)

11. Литвиненко А. С. Інформаційні технології у логістиці постачань. — Харків: ХНУРЕ, 2020. — 310 с. (дата звернення 10.10.2025)

12. Клименко В. І. Системи управління запасами: теорія і практика автоматизації. — Львів: Видавництво ЛНУ, 2021. — 240 с. (дата звернення 12.10.2025)

13. Гончар В. П. Системи ERP, CRM та SCM у торгівлі. — Львів: ЛНУ, 2022. — 268 с. (дата звернення: 12.10.2025)

14. Ільченко О. П. Автоматизація управління запасами торговельних підприємств. — Київ: КНЕУ, 2021. — 258 с. (дата звернення 12.10.2025)

15. Novak O., Petrenko L. Machine Learning-Based Forecasting in Retail Supply Chains. — IEEE Access, 2023. — Vol. 11. — P. 67421–67438. (дата звернення 13.10.2025)

16. Ткаченко О. О. Застосування нейронних мереж у задачах прогнозування продажів. — Харків: ХНУРЕ, 2022. — 268 с. (дата звернення 13.10.2025)

17. Григорян М., О. Гармаш Складська логістика URL: https://www.researchgate.net/publication/377659147_Skladska_logistika_navcalnij_posibnik (дата звернення 13.10.2025)

18. Апопій В. В., Міщук І. П. Організація торгівлі. – Львів: Новий Світ–2000, 2019. (дата звернення 14.10.2025)

19. Christopher M. Logistics and Supply Chain Management. – 5th ed. – Pearson, 2016 (дата звернення 14.10.2025)

20. What is an Enterprise System in Retail? URL: https://www.planetcrust.com/understanding-enterprise-system-in-retail-key-insights (дата звернення 15.10.2025)

21. Warehouse Management System URL: https://www.it.ua/en/products/sales/wms (дата звернення 16.10.2025)

22. ERP and POS integration: what it is and why it matters for retailers URL: https://www.extendaretail.com/blog/pos/erp-and-pos-integration-what-it-is-and-why-it-matters-for-retailers/ (дата звернення 16.10.2025)

23. Ultimate Guide to Retail ERP: Meaning and Top Solutions URL: https://tipalti.com/blog/retail-erp/ (дата звернення 16.10.2025)

24. ERP плюс CRM: ключ до стабільного успіху продажів для постачальників проектних послуг URL: https://kumavision.com/uk/блог/Конкурентні-переваги-для-постачальників-проектних-послуг (дата звернення 17.10.2025)

25. Lapala, the human-centric automation platform URL: https://lapala.io/en/retail-sector (дата звернення 17.10.2025)

26. The Formalization of the Business Process Modeling Goals URL: https://www.researchgate.net/publication/309713985_The_Formalization_of_the_Business_Process_Modeling_Goals (дата звернення 17.10.2025)

27. Business Process Modelling in Production Logistics URL: https://www.diva-portal.org/smash/get/diva2%3A828083/FULLTEXT01.pdf (дата звернення 18.10.2025)

28. Supply chain process management: how to model end-to-end processes with BPMN URL: https://www.cardanit.com/blog/supply-chain-process-management (дата звернення 18.10.2025)

29. Managing business processes in organization URL: https://www.smart-it.com/2019/10/business-process-management (дата звернення 20.10.2025)

30. Structural analysis and optimization of IDEF0 functional business process model URL: https://www.researchgate.net/publication/329850819_ (дата звернення 20.10.2025)

31. Slimstock. Supply Chain KPIs: Which Ones To Improve & Measure. 2025. URL: https://www.slimstock.com/blog/supply-chain-kpis/ (дата звернення 21.10.2025)

32. Meegle. KPIs (Key Performance Indicators) – Supply Chain. 2024. URL: https://www.meegle.com/en_us/topics/supply-chain/kpis-key-performance-indicators (дата звернення 22.10.2025)

33. NetSuite. Inventory Management KPIs and Metrics. 2025. URL: https://www.netsuite.com/portal/resource/articles/inventory-management/inventory-management-kpis-metrics.shtml (дата звернення 23.10.2025)

34. Rackbeat. KPIs for Inventory Management: 10 Key Metrics You Should Track. 2025. URL: https://rackbeat.com/en/kpis-for-inventory-management-10-key-metrics-you-should-track/ (дата звернення 23.10.2025)

35. Exotec. 10 KPIs for Logistics: Measuring the Performance of Your Warehouse. 2024. URL: https://www.exotec.com/en-gb/insights/10-kpis-logistics-measuring-the-performance-of-your-warehouse/ (дата звернення 24.10.2025)

36. DeLone W. H., McLean E. R. The DeLone and McLean Model of Information Systems Success: A Ten-Year Update. Journal of Management Information Systems, 2003. URL: http://researchgate.net/publication/220591866_The_DeLone_and_McLean_Model_of_Information_Systems_Success_A_Ten-Year_Update (дата звернення 24.10.2025)

37. MDN Web Docs. Getting started with Angular: What is Angular? – «Angular is a framework and development platform, built on TypeScript, used for creating single-page web applications». URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Angular_getting_started (дата звернення 02.11.2025)

38. Angular.dev. What is Angular? – «Angular is a web framework that empowers developers to build fast, reliable applications that scale…» URL: https://angular.dev/overview (дата звернення 02.11.2025)

39. MDN Web Docs. Web application manifest – Progressive Web Apps. URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Manifest (дата звернення 02.11.2025)

40. web.dev. What are Progressive Web Apps? URL: https://web.dev/articles/what-are-pwas (дата звернення 02.11.2025)

41. web.dev. Progressive Web Apps – Explore. URL: https://web.dev/explore/progressive-web-apps (дата звернення 03.11.2025)

42. MDN Web Docs. Using Service Workers. URL: https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers (дата звернення 03.11.2025)

43. Firebase. Cloud Firestore documentation. URL: https://firebase.google.com/docs/firestore (дата звернення 03.11.2025)

44. Firebase. Choose a Database: Cloud Firestore or Realtime Database. URL: https://firebase.google.com/docs/database/rtdb-vs-firestore (дата звернення 03.11.2025)

45. Ramadan S. Firebase, Firebase Realtime Database, and Firestore. URL: https://medium.com/@ramadan123sayed/firebase-firebase-realtime-database-and-firestore-8b4fcddb1059 (дата звернення 04.11.2025)

46. MDN Web Docs; Progressive Web Apps URL: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps (дата звернення 04.11.2025)

47. Angular vs React vs Vue: Core Differences in 2025. URL: https://www.browserstack.com/guide/angular-vs-react-vs-vue (дата звернення 04.11.2025)

48. Comparative Analysis of Angular, React, and Vue.js in Single Page Application Development. URL: https://www.researchgate.net/publication/380291017_Comparative_Analysis_of_Angular_React_and_Vuejs_in_Single_Page_Application_Development (дата звернення 04.11.2025)

49. Frontend Framework Comparison: React, Vue, and Angular in SPA & PWA development. URL: https://mox.one/en/blog/315/frontend-framework-comparison-react-vue-and-angular-for-spa-and-pwa-development (дата звернення 04.11.2025)

50. Snipclip. SPA Frameworks – Which One is Best Suited for Your Project? URL: https://snipclip.com/en/blog/spa-Frameworks-which-one-is-best-suited-for-your-project (дата звернення 04.11.2025)

51. What is Angular? URL: https://angular.dev/overview (дата звернення 04.11.2025)

52. Why Angular? URL: https://angular.dev/essentials (дата звернення 04.11.2025)

53. Justin Barnes Consultancy. Angular Framework: Guide to Modern SPAs with TypeScript. URL: https://www.justinbarnesconsultancy.co.uk/Web/Page/angular (дата звернення 04.11.2025)

54. React Official Website. React – A JavaScript library for building user interfaces. URL: https://legacy.reactjs.org/ (дата звернення 04.11.2025)

55. PubNub. A Developer's Guide to the React Library. URL: https://www.pubnub.com/guides/react/ (дата звернення 04.11.2025)

56. What is the Virtual DOM in React? URL: https://www.freecodecamp.org/news/what-is-the-virtual-dom-in-react/ (дата звернення 04.11.2025)

57. Vue.js Official Website. Vue – The Progressive JavaScript Framework. URL: https://vuejs.org/ (дата звернення 04.11.2025)

58. MDN Web Docs. Introduction to Vue.js – JavaScript framework for building user interfaces. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/Frameworks_libraries/Vue_getting_started (дата звернення 04.11.2025)

59. JavaScript in Plain English. Vue.js: A Progressive JavaScript Framework for Building Interactive User Interfaces. URL: https://medium.com/@FullStackSoftwareDeveloper/vue-js-a-progressive-framework-for-building-user-interfaces-d8c66fa5ab1a (дата звернення 04.11.2025)

60. React vs Angular vs Vue Performance in 2025. URL: https://medium.com/@masoomdeveloper1615/angular-vs-react-vs-vue-who-actually-wins-in-2025-17226031c3f2 (дата звернення 04.11.2025)

61. Single Page Applications with Angular – Why Angular for SPAs? Hicron Software. URL: https://hicronsoftware.com/blog/single-page-applications-with-angular/ (дата звернення 04.11.2025)

ДОДАТКИ

Додаток А (обов'язковий)

Технічне завдання

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф.
_____Олег БІСІКАЛО
« 17 » жовтня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на магістерську кваліфікаційну роботу

«АВТОМАТИЗАЦІЯ ПРОЦЕСУ ПОСТАЧАННЯ ТА ПРОДАЖУ ПРОДУКЦІЇ В МАГАЗИНІ»

08-31.МКР.003.02.000 ТЗ

Керівник роботи
д.т.н., проф. каф. КСУ
Марія ЮХИМЧУК
« 16 » жовтня 2025 р.

Виконавець:
ст. гр. ЗАКІТР-24М
Вадим ВОЙТЕНКО
« 16 » жовтня 2025 р.

Вінниця ВНТУ 2025

1. Назва та галузь застосування

Магістерська кваліфікаційна робота: «Автоматизація процесу постачання та продажу продукції в магазині». Галузь застосування – інформаційні технології, автоматизація логістичних процесів на підприємствах.

2. Підстава для розробки

Розробку системи здійснювати на підставі наказу по університету № 313 від 24 вересня 2025 року та завдання до магістерської кваліфікаційної роботи, складеного та затвердженого кафедрою «Автоматизації та інтелектуальних інформаційних технологій»

3. Мета та призначення розробки

Метою магістерської кваліфікаційної роботи є наукове обґрунтування, проєктування та розробка прогресивного веб-застосунку (PWA) для автоматизації процесів постачання та продажу продукції в магазині на базі стеку Angular та Firebase/Firestore, а також оцінка впливу впровадженої системи на ефективність ключових бізнес-процесів.

Розроблений програмний продукт призначений для комплексної підтримки процесів постачання та реалізації продукції в роздрібному магазині, зокрема: ведення номенклатури товарів, формування та оформлення замовлень, обробки й відстеження їх статусів різними категоріями користувачів (адміністратор, менеджер, касир, клієнт), а також накопичення даних у хмарному середовищі для подальшої аналітики та оптимізації роботи торговельного закладу.

4. Джерела розробки

1. Степанюк Т. М. Системи управління торговельними мережами: аналіз і автоматизація процесів. — Київ: КПІ ім. І. Сікорського, 2020. — 244 с. (дата звернення: 17.06.2025)
2. Novak O., Petrenko L. Integration of ERP and PWA Technologies in Modern Trade Systems. — IEEE Access, 2023, Vol. 11, pp. 48251–48263. (дата звернення:

17.06.2025)
3. Christopher M. Logistics and Supply Chain Management. – 5th ed. – Pearson, 2016 (дата звернення: 18.06.2025)
4. Войтенко В. М. Автоматизована система обліку складу підприємства. Матеріали X міжнародної науково-практичної конференції молодих учених, аспірантів і студентів "Автоматизація та комп'ютерно-інтегровані технології" (АКІТ-2024) - м.Київ: НТУУ «КПІ ім. Ігоря Сікорського» – 2024. – с. 95-96.

5. Показники призначення

Основні технічні вимоги та мінімальні системні вимоги до програмного продукту:

- Настільні ОС: Windows 10+, macOS 10.13+, сучасні дистрибутиви Linux (через сучасний браузер).
- Мобільні ОС: Android 8.0+, iOS 13+ (через встановлений PWA).
- Підтримувані браузери: Google Chrome, Microsoft Edge, Mozilla Firefox, Safari (з підтримкою JavaScript ES2018+, Service Worker, Web App Manifest).

Апаратне забезпечення:

- Процесор: 2 ядра з тактовою частотою від 1,6 ГГц
- Оперативна пам’ять: від 2 ГБ.
- Вільне місце на диску/флеш-пам’яті для встановлення PWA та кешу: не менше 100 МБ.
- Наявність доступу до мережі Інтернет (рекомендовано стабільне з’єднання, при цьому базовий функціонал доступний в офлайн-режимі завдяки PWA).

Серверна інфраструктура:

- Наявність налаштованого проєкту в Firebase з увімкненими сервісами Cloud Firestore, Firebase Authentication та Firebase Hosting.
- Доступ до інтернету для взаємодії клієнтського застосунку з Firebase API по HTTPS.

5.3 Вхідні дані

- Довідникові дані про продукцію: ідентифікатор, назва, категорія, опис, ціна, за потреби – зображення.
- Дані замовлення: обрані позиції (товар + кількість), загальна сума, дата та час створення замовлення.
- Дані користувачів: логін/пароль або інші облікові дані для автентифікації, роль користувача (адміністратор, менеджер, касир, клієнт).
- Контактні дані клієнта при оформленні замовлення: ім’я, прізвище, номер телефону, за потреби – додаткові параметри (коментар до замовлення тощо).
- Службова інформація: статус замовлення (нове, в обробці, виконане тощо), час зміни статусу.

5.4 Результати роботи програми

- Облік та зберігання замовлень у хмарній базі даних Cloud Firestore з розподілом на активні та завершені замовлення.
- Формування, редагування та відправка замовлень користувачем через веб-інтерфейс (PWA-застосунок).
- Перегляд списку поточних і виконаних замовлень з можливістю фільтрації та сортування (для адміністратора/менеджера та для клієнта).
- Зміна статусів замовлень та базові операції адміністрування (обробка черги замовлень, перенесення замовлень до архіву).
- Авторизація та аутентифікація користувачів із розмежуванням доступу до функціоналу відповідно до ролей.
- Накопичення історичних даних про замовлення для подальшої аналітики та використання в процесах планування постачання й оцінки ефективності роботи магазину.

6. Стадії розробки

1. Розділ 1 «Аналіз предметної області» має бути виконаний до 01.10.2025 р.
2. Розділ 2 «Математичне та інформаційне забезпечення» має бути виконаний до 20.10.2025 р.
3. Розділ 3 «Накопичення історичних даних про замовлення для подальшої аналітики та використання в процесах планування постачання й оцінки

ефективності роботи магазину» має бути виконаний до 15.11.2025 р.

4. Розділ 4 «Накопичення історичних даних про замовлення для подальшої аналітики та використання в процесах планування постачання й оцінки ефективності роботи магазину.» має бути виконаний до 10.12.2025 р.

7. Порядок контролю та приймання

1. Рубіжний контроль провести до 14.11.2025.
2. Попередній захист магістерської кваліфікаційної роботи провести до 02.12.2025.
3. Захист магістерської кваліфікаційної роботи провести до 19.12.2025.

Додаток Б (обов’язковий)

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

АВТОМАТИЗАЦІЯ ПРОЦЕСУ ПОСТАЧАННЯ ТА ПРОДАЖУ ПРОДУКЦІЇ В МАГАЗИНІ

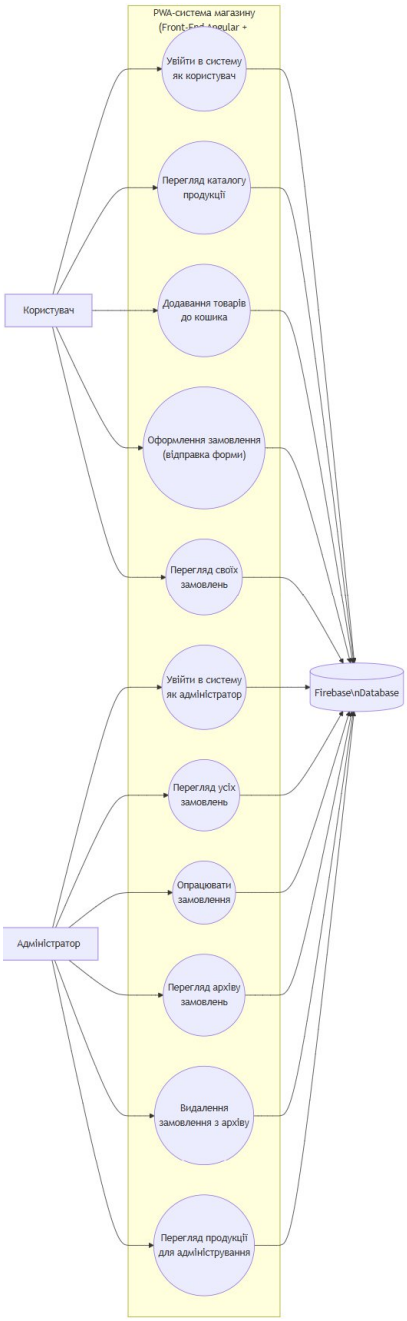

Рисунок Б.1 – Use Case UML-діаграма варіантів використання

Рисунок Б.2 – UML-діаграма діяльності

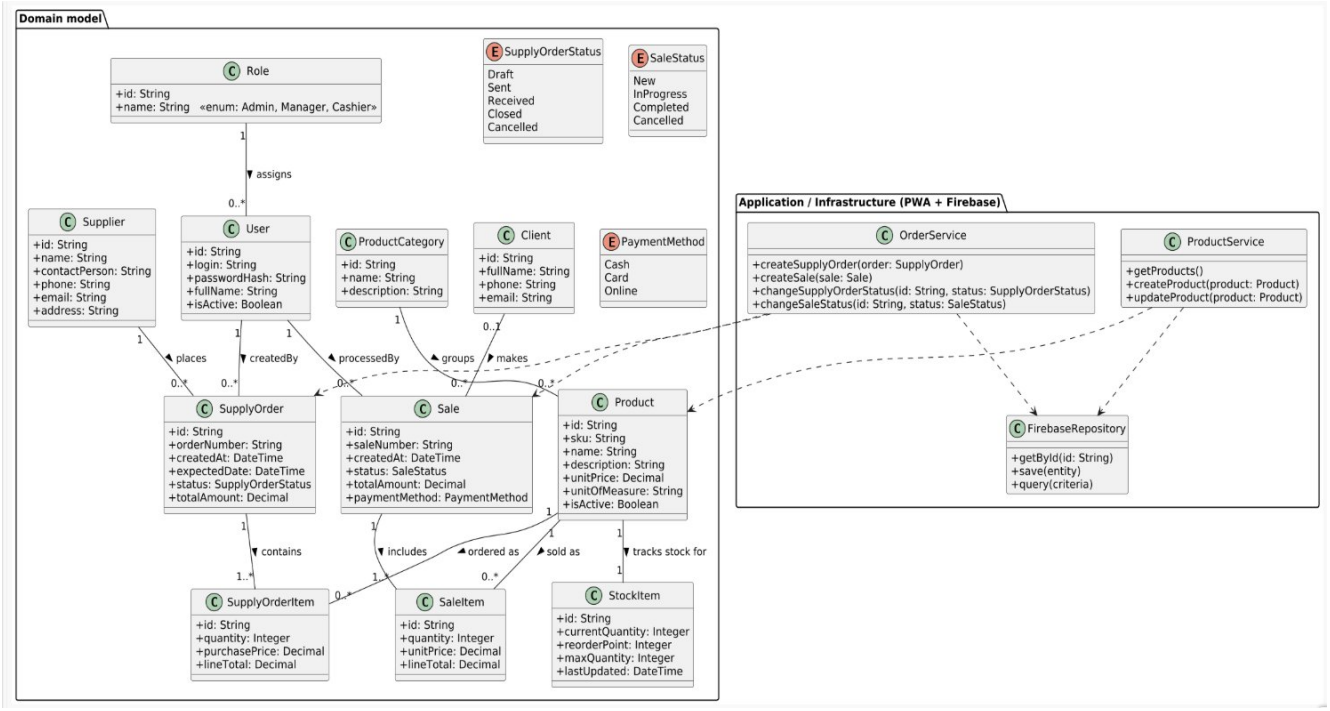

Рисунок Б.3 – UML-діаграма класів

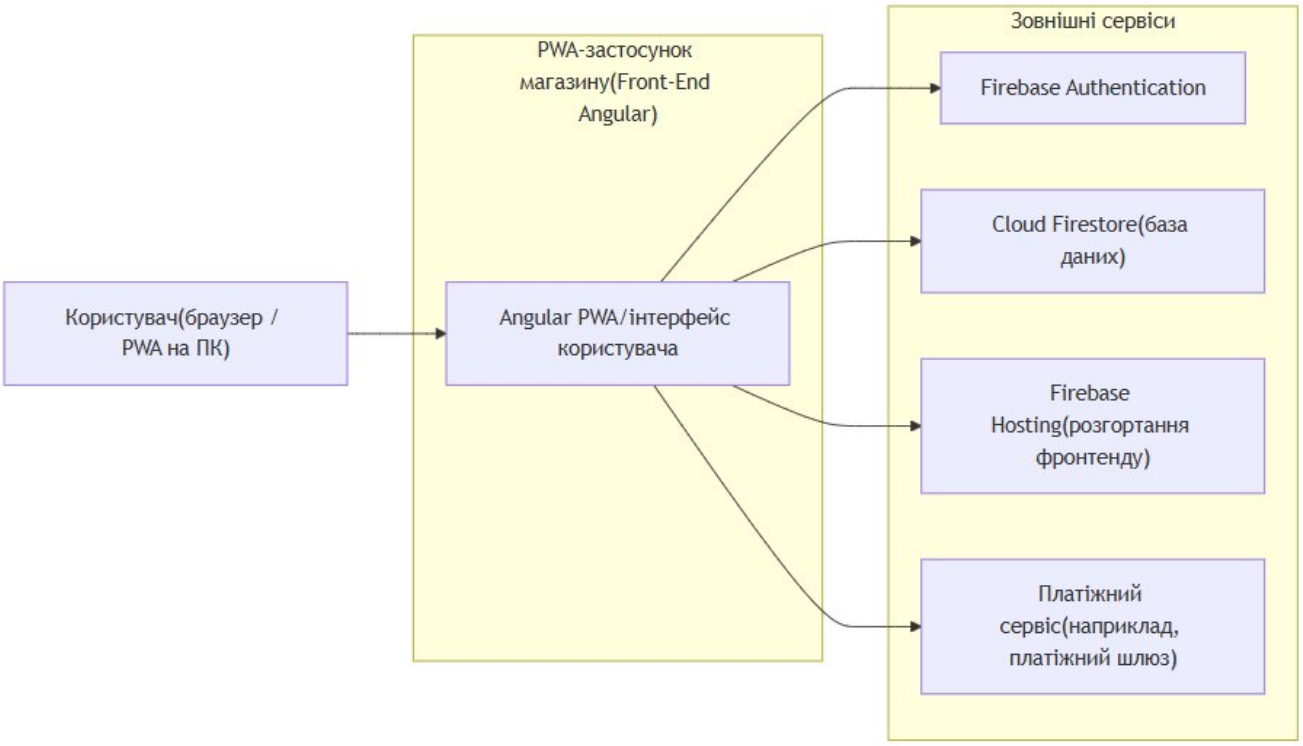

Рисунок Б.4 - UML-діаграма компонентів

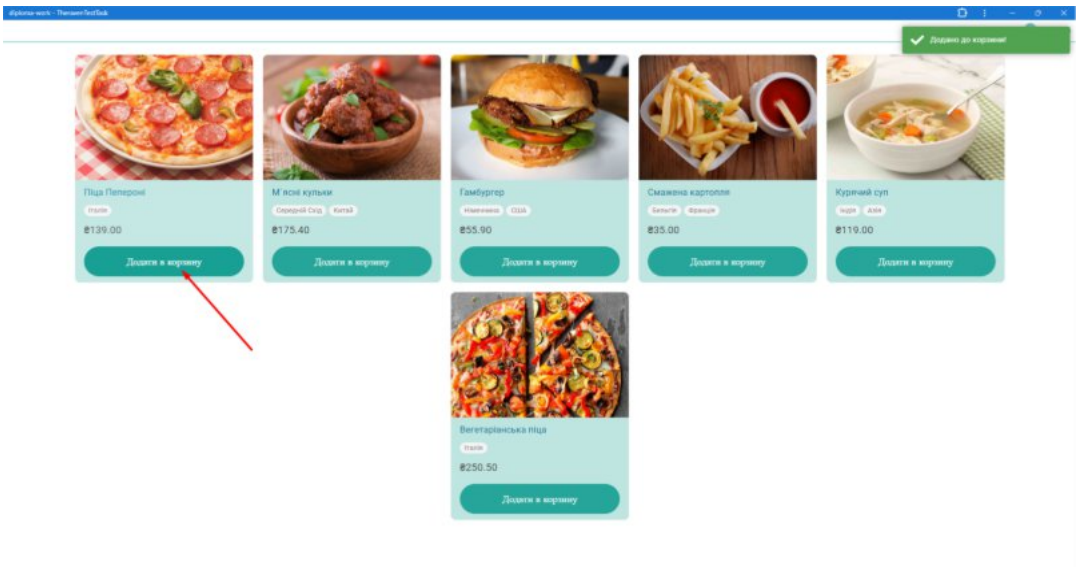

Рисунок Б.5 - Скріншот інтерфейсу ролі “Користувач”

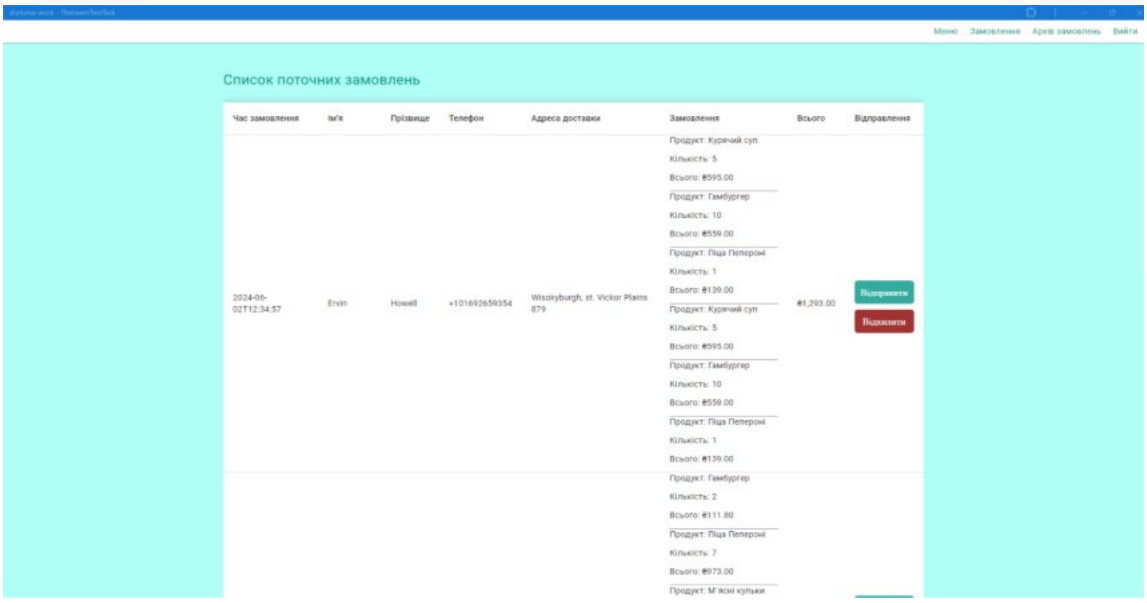

Рисунок Б.6 - Скріншот інтерфейсу ролі “Адміністратор”

Додаток В (обов’язковий)

Лістинг основних функцій програми

```
import { IOrder } from '@shared/models/IOrder';
import { Injectable } from '@angular/core';
import {
  Firestore,
  collection,
  deleteDoc,
  doc,
  getDocs,
  orderBy,
  query,
  setDoc,
} from '@angular/fire/firestore';
import { Observable, Subject, from, tap } from 'rxjs';

@Injectable({
  providedIn: 'root',
})
export class OrderService {
  constructor(private firestore: Firestore) {}

  getOrders(): Observable<IOrder[]> {
    const orderRef = collection(this.firestore, 'orders');
    const orderedQuery = query(orderRef, orderBy('sendTime'));

    return from(
      getDocs(orderedQuery).then((querySnapshot) => {
```

```
      const data: IOrder[] = [];
      querySnapshot.forEach((doc) => {
        data.push(doc.data() as IOrder);
        console.log(doc.id);
      });
      return data;
    })
  );
}

getCompletedOrders(): Observable<IOrder[]> {
  const completedOrderRef = collection(this.firestore, 'sent_orders');
  const completedQuery = query(completedOrderRef, orderBy('sendTime'));

  return from(
    getDocs(completedQuery).then((querySnapshot) => {
      const data: IOrder[] = [];
      querySnapshot.forEach((doc) => {
        data.push(doc.data() as IOrder);
      });
      return data;
    })
  );
}

async addOrder(orderData: IOrder) {
  const orderRef = doc(this.firestore, 'orders', orderData.id);
  await setDoc(orderRef, orderData);
}
```

```
  async addCompletedOrder(orderData: IOrder) {
    const orderRef = doc(this.firestore, 'sent_orders', orderData.id);
    await setDoc(orderRef, orderData);
  }

  async deleteOrder(collectionName: string, orderData: IOrder) {
    const orderRef = collection(this.firestore, collectionName);
    const orderDeleteDoc = doc(orderRef, orderData.id);
    await deleteDoc(orderDeleteDoc);
  }

  async deleteCompletedOrder(collectionName: string, orderData: IOrder) {
    const completedOrderRef = collection(this.firestore, collectionName);
    const completedDeleteDoc = doc(completedOrderRef, orderData.id);
    await deleteDoc(completedDeleteDoc);
  }
}
```

Додаток Г (обов'язковий) Протокол перевірки кваліфікаційної роботи

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Автоматизація процесу постачання та продажу продукції в магазині»

Тип роботи: магістерська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра АІІТ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism (КП1) 0,40 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

- Запозичення, виявлені у роботі, оформлені коректно і не містять ознак академічного плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту.
- У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.
- У роботі виявлено ознаки академічного плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень. Робота до захисту не приймається.

Експертна комісія:

Бісікало О.В., зав. каф. АІІТ (підпис)
(прізвище, ініціали, посада)

Овчинников К.В., доц. каф. АІІТ (підпис)
(прізвище, ініціали, посада)

Особа, відповідальна за перевірку (підпис) Маслій Р.В.
(прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник (підпис) Юхимчук М.С., професор. каф. КСУ
(прізвище, ініціали, посада)

Студент (підпис) Войтенко В.М.
(прізвище, ініціали)