ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
КАФЕДРА ІНФОРМАТИКИ І КОМП'ЮТЕРНИХ НАУК

Пояснювальна записка

кваліфікаційна робота бакалавра

на тему:

Розробка інтелектуальної системи управління товарними запасами на основі машинного навчання з NLP-інтерфейсом

Виконав: студент 4 курсу, групи 4КН
спеціальності 122 «Комп’ютерні науки»
Боярський Є.О.

Керівник: к.т.н., доцент Лур’є І.А.
Рецензент:
к.т.н., доцент Козел В.М.

Хмельницький 2026 р.

Факультет	**Інформаційних технологій та дизайну**
Кафедра	**Інформатики і комп'ютерних наук**
Рівень вищої освіти	**перший (бакалаврський) рівень**
Галузь підготовки	**12 «Інформаційні технології»**
Освітньо - професійна програма	**Комп'ютерні науки**
Спеціальність	**122 «Комп'ютерні науки»**

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри ІКН,

доцент

_____ С.В. Моісеєнко

«____» _____ 2026 року

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ СТУДЕНТА

Боярського Євгена Олександровича

(прізвище, ім’я, по батькові)

1. Тема роботи: «Розробка інтелектуальної системи управління товарними запасами на основі машинного навчання з NLP-інтерфейсом»

Керівник роботи Лур’є Ірина Анатоліївна, кандидат технічних наук, доцент,

(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ХНТУ від «26» січня 2026 року № 133-с.

2. Строк подання студентом роботи 10.06.2026р.

3. Вихідні дані до роботи Матеріали та результати, отримані під час проходження переддипломної практики, методичні вказівки, технічна література

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Вступ; 1. Теоретичний аналіз предметної області та постановка

завдання; 2. Обґрунтування вибору технологічного стеку та інструментарію; 3. Проєктування та розробка ядра системи; 4. Інтеграція методів штучного інтелекту; 5. Тестування працездатності та аналіз ефективності системи; Висновок

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

В роботі наведено:

Рисунки – 24

Таблиці – 9

Формули – 11

Лістинги програмного коду – 19

6. Дата видачі завдання 09.02.2026

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області, огляд існуючих систем та формулювання вимог	09.02.2026 – 01.03.2026	Виконано
2	Проєктування бази даних, архітектури додатку та вибір інструментальних засобів	02.03.2026 – 10.03.2026	Виконано
3	Розробка ядра системи обліку: база даних, шар доступу, графічний інтерфейс, пошук та багатомовність	11.03.2026 – 21.03.2026	Виконано
4	Реалізація основних бізнес-модулів: номенклатура, документообіг, інвентаризації, користувачі, аналітика	22.03.2026 – 10.04.2026	Виконано
5	Інтеграція методів штучного інтелекту: модуль прогнозування на TFT та NLP-асистент на основі хмарного API	11.04.2026 – 06.05.2026	Виконано
6	Тестування функціональності, обробка граничних ситуацій та оцінка продуктивності системи	07.05.2026 – 17.05.2026	Виконано
7	Оформлення та коригування звіту кваліфікаційної роботи, підготовка матеріалів до захисту.	18.05.2026 – 28.05.2026	Виконано

Студент _____ Боярський Є.О.

(підпис) (прізвище та ініціали)

Керівник роботи _____ Лур’є І.А.

(підпис) (прізвище та ініціали)

ЗМІСТ

АНОТАЦІЯ

Пояснювальна записка до кваліфікаційної роботи: 125 с., 24 рисунки, 9 таблиць, 11 формул, 19 лістингів програмного коду, 53 джерела.

У кваліфікаційній роботі розв'язано задачу автоматизації процесів складського обліку на підприємствах малого та середнього бізнесу із застосуванням методів машинного навчання та інтерфейсу взаємодії природною мовою. Розроблено автономну десктопну інформаційну систему управління товарними запасами, що поєднує облік номенклатури, складів і документообігу з інтерпретованим прогнозуванням попиту на основі архітектури Temporal Fusion Transformer та діалоговим NLP-асистентом на базі хмарної моделі Google Gemini, який перетворює природномовні запити у команди SQL.

Метою роботи є створення системи управління складом, яка поєднує базовий облік із прогнозуванням попиту та обробкою текстових і голосових запитів. Базу даних і шар доступу до даних з автоматичним резервним копіюванням реалізовано засобами СУБД SQLite, графічний інтерфейс – за допомогою бібліотеки CustomTkinter, а модуль прогнозування – на основі бібліотеки PyTorch Forecasting та архітектури Temporal Fusion Transformer, що забезпечує інтерпретоване мультигоризонтне прогнозування з інтервальними оцінками ризику. NLP-асистент генерує SQL-команди вибірки, приймає голосове введення та блокує запити на змінення чи видалення даних. Оптимальні значення довжини історичного вікна й горизонту прогнозу та ефективність архітектури TFT підтверджено експериментально на відкритому наборі даних Kaggle.

Ключові слова: управління складом, Temporal Fusion Transformer, прогнозування часових рядів, NLP, Gemini API, SQLite, CustomTkinter, PyTorch.

ABSTRACT

Explanatory note to the thesis: 125 pages, 24 figures, 9 tables, 10 formulas, 19 listings of source code, 53 sources.

This thesis addresses the task of automating warehouse accounting processes in small and medium-sized enterprises using machine learning methods and a natural language interface. An autonomous desktop information system for inventory management has been developed, combining the management of product ranges, warehouses and document flow with interpretive demand forecasting based on the Temporal Fusion Transformer architecture and a conversational NLP assistant based on the Google Gemini cloud model, which converts natural language queries into SQL commands.

The aim of the work is to create a warehouse management system that combines basic accounting with demand forecasting and the processing of text and voice queries. The database and data access layer with automatic backup are implemented using the SQLite DBMS, the graphical interface using the CustomTkinter library, and the forecasting module is based on the PyTorch Forecasting library and the Temporal Fusion Transformer architecture, which provides interpretable multi-horizon forecasting with interval risk estimates. The NLP assistant generates SQL query commands, accepts voice input and blocks requests to modify or delete data. Experiments to determine the optimal history length and the forecast horizon with near-zero bias were conducted on the Kaggle dataset to validate the effectiveness of the TFT architecture.

Keywords: inventory management, Temporal Fusion Transformer, time series forecasting, NLP, Gemini API, SQLite, CustomTkinter, PyTorch.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

Скорочення, термін, позначення	Пояснення
WMS	Warehouse Management System – система управління складом
ERP	Enterprise Resource Planning – планування ресурсів підприємства
SaaS	Software as a Service – програмне забезпечення як послуга
СУБД	Система управління базами даних
SKU	Stock Keeping Unit – складська одиниця обліку (артикул)
RBAC	Role-Based Access Control – керування доступом на основі ролей
NLP	Natural Language Processing – обробка природної мови
LLM	Large Language Model – велика мовна модель
API	Application Programming Interface – програмний інтерфейс застосунку
SDK	Software Development Kit – набір засобів розробки
TFT	Temporal Fusion Transformer – архітектура нейромережі для прогнозування часових рядів
LSTM	Long Short-Term Memory – довга

	короткострокова пам’ять
RNN	Recurrent Neural Network – рекурентна нейронна мережа
ARIMA	AutoRegressive Integrated Moving Average – авторегресійна інтегрована ковзна середня
VSN	Variable Selection Network – мережа відбору змінних
GRN	Gated Residual Network – вентильна залишкова мережа

ВСТУП

На сучасному етапі розвитку економіки управління товарними запасами трансформується з рутинної облікової функції у стратегічний інструмент управління підприємством. Як зазначають експерти компанії McKinsey, що надає послуги у сфері менеджмент-консалтингу, Майкл Біршан, Майкл Парк та Метт Стоун, тисячі щоденних рутинних рішень персоналу можуть кардинально збільшити обсяг готівки, необхідної для ведення бізнесу [1]. Динаміка ринку вимагає від бізнесу не лише точного обліку руху товарів, але й впровадження методів прогнозування. Сьогодні це один із ключових факторів, що визначає конкурентоспроможність підприємства. Це дозволяє вчасно реагувати на потреби ринку, мінімізувати витрати на зберігання неліквідної продукції та уникати дефіциту популярних товарів. На практиці ж у компаніях часто панує гіперконсерватизм – наприклад, коли менеджери замовляють додаткові запаси на місяці вперед заради безпеки, тим самим невиправдано зв'язуючи капітал [1].

Традиційні підходи до складського обліку, що базуються на ретроспективному аналізі або інтуїтивних рішеннях менеджерів, поступово втрачають свою ефективність. Дослідники Д. Яновський та М. Граф наголошують, що класичні математичні методи передбачають виключно лінійний розвиток подій, тоді як у реальності нелінійні фактори, наприклад, збої в роботі постачальників, суттєво знижують точність таких прогнозів [2]. В епоху великих даних на зміну їм приходять інтелектуальні інформаційні системи. Як зазначає колектив авторів на чолі з Т. Вакалюк, сучасна фінансова та торгова екосистема характеризується високою волатильністю, що створює нагальну потребу в алгоритмах машинного навчання для виявлення неочевидних закономірностей [3]. Інтеграція таких алгоритмів у процеси управління ланцюгами постачання дозволяє суттєво знизити ризики заморожування обігових коштів та здатна зменшити потребу в капіталі на 20–30 відсотків [1].

Особливої стратегічної ваги в сучасних реаліях набуває питання розвитку вітчизняних програмних продуктів. Створення та впровадження українського програмного забезпечення є питанням не лише технологічної незалежності, але й підтримки економічної стійкості держави. Вітчизняний малий та середній бізнес гостро потребує доступних, надійних та адаптованих до локальних умов інструментів, які б поєднували функціонал класичного обліку з технологіями штучного інтелекту. Застосування таких технологій також мінімізує залежність від людського фактору, суттєво знижуючи ризик операційних помилок [3; 4; 5; 6], та виступає гідною альтернативою вартісним іноземним системам.

Актуальність теми посилюється необхідністю подолання “цифрового бар'єру” між складними алгоритмами та кінцевим користувачем. Існуючі рішення часто вимагають глибоких технічних компетенцій або стикаються з проблемою недовіри до “чорної скриньки” складних нейромереж. Інтеграція технологій обробки природної мови дозволяє подолати цей бар’єр. Сучасні мовні моделі здатні аналізувати складні звіти, роз'яснювати значення показників та забезпечувати ефективну комунікацію в режимі діалогу [3].

Мета роботи полягає у проєктуванні та розробці інформаційної системи управління товарними запасами, яка забезпечуватиме автоматизацію облікових процесів, прогнозування попиту з використанням методів глибокого навчання та підтримку прийняття управлінських рішень через інтерфейс обробки запитів, як ШІ-асистента.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1) проаналізувати предметну область, наявні системи складського обліку та сучасні методи прогнозування часових рядів;
2) обґрунтувати вибір технологічного стеку (мова програмування, СУБД, бібліотеки машинного навчання та побудови інтерфейсу);
3) спроєктувати архітектуру системи, реляційну базу даних і шар доступу до даних з резервним копіюванням та синхронізацією;

4) реалізувати модулі обліку (номенклатура, документообіг, інвентаризація, користувачі, аналітика) з рольовим розмежуванням доступу;
5) розробити модуль прогнозування попиту на основі архітектури Temporal Fusion Transformer та експериментально визначити його оптимальні параметри;
6) інтегрувати NLP-асистента на базі хмарної мовної моделі для перетворення природномовних запитів у безпечні SQL-команди;
7) виконати тестування працездатності системи та оцінити ефективність інтелектуальних модулів.

Об’єктом дослідження є процеси автоматизованого управління номенклатурою на підприємствах малого та середнього бізнесу, що включають збір первинних даних, моніторинг поточних залишків та планування закупівель в умовах динамічної зміни ринкового попиту.

Предмет дослідження – методи побудови інформаційних систем, алгоритми глибокого навчання, зокрема архітектури, призначені для аналізу часових рядів для короткострокового прогнозування, а також методи організації взаємодії за допомогою чат-боту ШІ.

Методи дослідження. Методологічну основу роботи склав системний підхід до проєктування інформаційно-аналітичних систем. На етапі аналізу предметної області застосовувалися методи порівняння та узагальнення, що дозволило виявити недоліки існуючих систем управління складським обліком.

Для побудови архітектури системи використано методи об’єктно-орієнтованого проєктування та моделювання реляційних баз даних за допомогою СУБД SQLite та забезпечено таким чином модульність системи.

Вирішення задачі прогнозування попиту базується на методах математичної статистики та алгоритмах машинного навчання. Для оцінки та порівняння прогностичних моделей використано зважену абсолютну відсоткову похибку (WAPE), середню абсолютну похибку (MAE) та систематичне відхилення прогнозу (Bias) [2]. Зокрема, для аналізу часових

рядів використано нейромережеву модель Temporal Fusion Transformer, яка дозволяє враховувати як історичні дані, так і супутні фактори впливу.

Наукова новизна одержаних результатів полягає у розробці та комплексному дослідженні гібридної архітектури інтелектуальної інформаційної системи складського обліку, яка адаптує методи глибокого навчання та генеративного штучного інтелекту для потреб малого та середнього бізнесу без необхідності розгортання високовартісної серверної інфраструктури, зокрема:

- Вперше запропоновано підхід до застосування архітектури Temporal Fusion Transformer для мультигоризонтного прогнозування попиту на історичних даних роздрібних продажів, який не лише генерує інтервальні прогнози для оцінки ризиків, але й вирішує проблему “чорної скриньки” нейромереж шляхом візуалізації локальної важливості факторів та механізму агрегованої уваги безпосередньо в графічному інтерфейсі.
- Удосконалено метод людино-машинної взаємодії в системах обліку за рахунок інтеграції технології обробки природної мови на базі хмарної великої мовної моделі. Розроблено алгоритм трансляції неструктурованих природномовних та голосових запитів у валідні SQL-команди вибірки з використанням типізації через JSON-схеми, що повністю ізолює базу даних від модифікацій.
- Набув подальшого розвитку метод автономної гібридної синхронізації даних між стаціонарними та мобільними пристроями на основі нативної взаємодії схем вбудованих баз даних SQLite, що гарантує реляційну цілісність інформації без використання проміжних API-серверів.

Практичне значення одержаних результатів полягає у створенні програмної системи, яка дозволяє автоматизувати складські операції, підвищити точність планування закупівель завдяки адаптивним прогнозам та полегшити доступ персоналу до аналітичних даних. Реалізована архітектура

забезпечуватиме можливість синхронізації даних із зовнішніми пристроями, що, в свою чергу, гарантуватиме гнучкість та масштабованість системи.

Апробація результатів роботи. Основні наукові положення та практичні результати кваліфікаційної роботи оприлюднено й обговорено на III Міжнародній науково-практичній конференції «Синергія науки і бізнесу 2026» (м. Хмельницький, 22–24 квітня 2026 р.), що відбулася на базі Херсонського національного технічного університету. За матеріалами доповіді опубліковано тези:

Боярський Є. О., Лур'є І. А., Корніловська Н. В. Розробка інтелектуальної системи управління товарними запасами на основі машинного навчання з NLP-інтерфейсом. Синергія науки і бізнесу 2026 : матеріали III Міжнародної науково-практичної конференції (м. Хмельницький, 22–24 квітня 2026 р.). Хмельницький : Херсонський національний технічний університет, 2026.

РОЗДІЛ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Аналіз предметної області та огляд існуючих систем

Ринок систем управління складом WMS на сьогодні є досить насиченим, однак більшість рішень або призначені для великих логістичних центрів, або є застарілими бухгалтерськими системами, позбавленими можливостей інтелектуального аналізу [7; 8].

Для цілей даного аналізу існуючі програмні інструменти доцільно розділити на три основні групи:

1. Табличні процесори та універсальні редактори (MS Excel, Google Sheets).
2. Класичні ERP-системи (сімейство 1C/BAS, SAP Business One) [9; 10; 11].
3. Хмарні SaaS-рішення (Odoo, TradeGecko, UkrSklad) [12].

Табличні процесори є найбільш доступним інструментом для початку ведення обліку. Сучасні версії Excel мають потужні функціональні можливості такі, як Power Query та надбудови для аналізу даних. Їхня основна проблема полягає в “людському факторі”. Відсутність жорсткої структури бази даних часто призводить до проблем із цілісністю даних, дублікатів, помилок у формулах. Хоча Excel дозволяє виконувати прогнозування, наприклад за допомогою алгоритму експоненціального згладжування, цей процес складно автоматизувати для широкого асортименту товарів.

Монолітні ERP-системи – це професійні рішення, що охоплюють весь бухгалтерський цикл. Основною перешкодою для впровадження цих систем є їх надмірна складність і вартість використання [11]. Що стосується прогнозування, то базові конфігурації зазвичай обмежуються класичними статистичними методами.

Впровадження передових методів машинного навчання в подібні монолітні системи є технічним складним завданням і вимагає залучення програмістів для розробки зовнішніх модулів.

Сучасні SaaS-платформи, такі як Odoo, пропонують модульну архітектуру та веб-інтерфейс. Odoo, написана мовою Python, є архітектурно найближчим аналогом до розроблюваної системи, а крім того, є програмним забезпеченням з відкритим вихідним кодом, що теоретично дозволяє продовжувати її розвиток [12].

Однак головним недоліком моделі SaaS є повна залежність від інтернет-з'єднання, а також щомісячні абонентські платежі, які зростають пропорційно кількості користувачів. Для малих підприємств розробка власного легкого клієнта на мові Python може бути рентабельнішою, ніж спроба адаптувати логіку глобальної системи ERP до конкретних завдань прогнозування [7].
Порівняльна характеристика розглянутих аналогів наведена у табл. №1.1.

Таблиця 1.1

Порівняльний аналіз засобів автоматизації

Критерій порівняння	**MS Excel**	**BAS (ex-1C)**	**Odoo**	**Розроблювана система**
Тип платформи	Файл	Десктоп/ Сервер	Веб/Хмара	Гібридна (Android + Windows)
Складність впровадження	Низька	Дуже висока	Середня	Низька
Вартість	Ліцензія Office	Висока + підтримка	Безкоштовно /Підписка	Open Source
Прогнозування попиту	Лінійний тренд	Мінімакс. метод	Статистичні методи	Deep Learning
Інтерфейс взаємодії	Табличний	Форми/ Меню	Веб-форми	GUI + NLP (Чат-бот)
Робота офлайн	Так	Так	Ні (обмежено)	Так

Аналіз ринку програмного забезпечення виявив значний розрив у сегменті рішень, призначених для малих та середніх підприємств. Користувачам доводиться обирати між доступними за ціною електронними таблицями з обмеженою функціональністю та складними ERP системами, впровадження яких відрізняється надмірною складністю, високими витратами та закритою архітектурою.

Жодна з розглянутих альтернатив не пропонує інтегрованих, готових до використання інструментів прогнозування на основі глибокого навчання та не забезпечує обробки природної мови.

Це виправдовує розробку спеціалізованої системи, яка заповнить ринкову нішу. Пропоноване рішення поєднає мобільність збору даних, автономність офісного клієнта та потужність прогнозних моделей.

1.2. Обґрунтування мети, задач та функціональних вимог до системи

На основі проведеного аналізу предметної області та недоліків, виявлених у існуючих рішеннях, сформовано мету роботи, а саме – проєктування та розробка інтелектуальної інформаційної системи, яка автоматизує управління запасами та забезпечуватиме якісний прогноз попиту.

На відміну від традиційних бухгалтерських систем, які обробляють операції обліку, що вже відбулись, розроблюване ПЗ, має виступати в якості системи підтримки прийняття рішень, використовуючи при цьому алгоритми машинного навчання для управління закупівлями [13; 14].

Для досягнення поставленої мети система повинна відповідати таким функціональним вимогам:

1. Кількісний облік на різних складах з урахуванням обліку товарів.
2. Встановлення чіткого розмежування прав доступу до функцій системи для різних груп користувачів.
3. Автоматизація формування документів: замовлень, накладних та інвентаризацій.

4. Розрахунок прогнозів продажів на майбутній період за допомогою методів глибокого навчання, що враховують історичні тенденції та сезонні фактори.
5. Можливість отримувати довідкову інформацію за допомогою текстових та голосових запитів у діалоговому форматі.
6. Підтримка обміну інформацією між мобільним пристроєм для збору даних та стаціонарним клієнтом без необхідності постійного підключення до Інтернету.

Архітектурну декомпозицію розроблюваної системи наведено на рисунку 1.1.

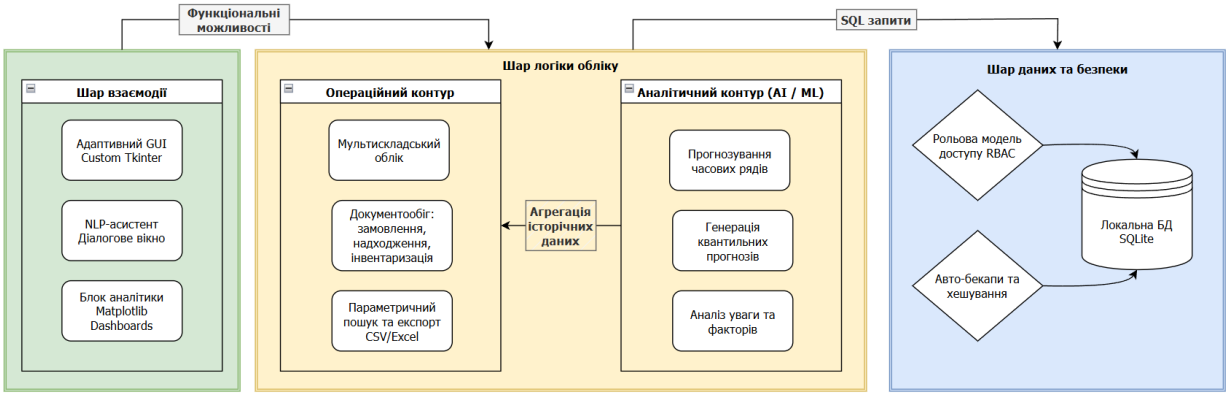

Рис. 1.1 – Архітектура рівнів розроблюваної системи

Для реалізації необхідних функцій потрібна розробка спеціальної програмної архітектури , яка поєднуватиме в собі високу обчислювальну потужність, необхідну для роботи нейронних мереж, з простотою впровадження та автономністю – двома аспектами, що мають ключове значення для розроблюваної системи.

1.3. Порівняльний аналіз архітектурних патернів для гібридних складських систем

Вибір архітектурної моделі є ключовим етапом процесу проектування, оскільки він визначає масштабованість, надійність та витрати на обслуговування майбутньої системи. Для повноцінного виконання поставлених вимог до проєкту було проаналізовано три основні підходи до проєктування корпоративних інформаційних систем.

У випадку класичної клієнт-серверної веб-архітектури логіка обліку та база даних розміщуються на віддаленому сервері, доступ до якого можна отримати через веб-браузер. Перевагами є кросплатформна сумісність та відсутність необхідності у встановленні програмного забезпечення на клієнтських комп’ютерах. До недоліків можна віднести повну залежність від стабільності інтернет-з'єднання та витрати на хостинг або хмарні ресурси, а також можливі затримки при передачі великих обсягів даних для аналізу.

Підхід монолітної архітектури при якому вся логіка, користувацький інтерфейс і база даних знаходяться на одному локальному комп'ютері забезпечували б високу продуктивність і незалежність від роботи мережі та відсутність витрат на сервери. Проте, маємо й недоліки – складнощі в організації роботи з даними, що надходять з різних пристроїв, наприклад, зі смартфону та стаціонарного комп'ютера.

Гібридна архітектура з локальною синхронізацією є найсучаснішим підходом, що поєднує переваги попередніх моделей. Система складається з автономних клієнтів, як мобільних так і стаціонарних, кожен з яких має власну локальну копію базу даних. Таким чином, система буде повністю автономною, а обмін даними відбуватиметься через синхронізацію файлу бази даних.

Враховуючи поставлені вимоги до системи, а саме: робота в умовах нестабільного зв'язку, необхідність важких обчислень для модуля прогнозування та мінімізація витрат для підприємства, найбільш доцільним є використання гібридної архітектури. Десктопний клієнт виступатиме

аналітичним центром, де виконуються ресурсомісткі задачі прогнозування, а мобільний додаток слугуватиме інструментом оперативного збору даних на складі. Така модель дозволятиме уникнути витрат на серверну інфраструктуру та забезпечуватиме максимальну швидкодію.

1.4. Аналіз моделей організації та зберігання даних

Розроблювана система оброблятиме великі обсяги структурованих даних: товари, облікові записи про запаси, а також історичні дані про продажі та закупівлі, які використовуватимуться як навчальні дані для нейронної мережі. Перед впровадженням бази даних було розглянуто три категорії систем управління базами даних.

Перша категорія включає в себе реляційні системи управління базами даних типу “клієнт-сервер”, – PostgreSQL та MySQL, які забезпечують високу надійність, доступ для безлічі користувачів і підтримку складних запитів [15]. Однак системи цієї категорії також вимагають розгортання та налаштування окремого сервера. Для підприємства це є зайвою перешкодою: необхідність встановлення сервера, налаштування портів та створення облікових записів користувачів. Крім того, структура даних PostgreSQL не є безпосередньо сумісною з локальною базою даних системи Android.

Друга категорія включає рішення NoSQL – розглянуто MongoDB, Redis. Це документально-орієнтовані бази даних, які дозволяють зберігати дані у форматі JSON без фіксованої схеми. Однак бухгалтерія вимагає суворої реляційної моделі, оскільки продукт не може існувати без категорії, так само як продаж – без продукту. Відсутність жорстких зв'язків, таких як зовнішні ключі в NoSQL, підвищує ризик порушення цілісності даних, що є неприпустимим.

Третя категорія представляє вбудовані системи управління базами даних. До цієї категорії належать бази даних, які не потребують окремого сервера, а безпосередньо інтегровані в програмне забезпечення у вигляді

бібліотеки. Вся база даних зберігається в одному файлі, що працює на різних платформах. Було розглянуто SQLite, який є нативним стандартом операційної системи Android [16]. Використання тієї ж системи управління базами даних на стаціонарному комп'ютері забезпечує повну сумісність форматів даних. Разом з тим використання такого роду систем СУБД не вимагає ні встановлення, ні адміністрування. Після проведення порівняльного аналізу було сформовано відповідну табл. №1.2 – класи СУБД.

Таблиця 1.2

Порівняльний аналіз класів СУБД

Критерій	**Клієнт-серверні (PostgreSQL)**	**NoSQL (MongoDB)**	**Вбудовані (SQLite)**
Архітектура	Окремий серверний процес	Різна (сервер/вбудована)	Бібліотека в додатку
Складність адміністрування	Середня	Середня	Відсутня
Підтримка SQL/зовнішніх ключів	Так	Обмежена/ні	Так
Сумісність з Android	Ні (потрібен сервер)	Частково	Так (нативна)

Виходячи з архітектурних вимог, як основний засіб зберігання даних обрано SQLite.

1.5. Теоретико-математичні основи методів прогнозування часових рядів

Прогнозування часових рядів є складною математичною задачею, важливість якої зумовлена необхідністю прийняття правильних рішень в

умовах невизначеності. Завдання полягає в тому, щоб на основі історичних спостережень створити адекватний прогноз щодо майбутнього розвитку даного явища. Відповідно до класифікації, прийнятої в літературі, формалізовані методи прогнозування часових рядів можна розділити на дві основні групи: статистичні моделі, в яких залежність формулюється у вигляді точного математичного рівняння, та структурні моделі, в яких залежність формулюється у вигляді певної структури та правил переходу, що складають основу алгоритмів машинного навчання [17; 2] . Усі ці методи мають одну спільну рису – вони використовують диференціальний оператор Δ^d для приведення ряду до стаціонарної форми (рис. 1.2).

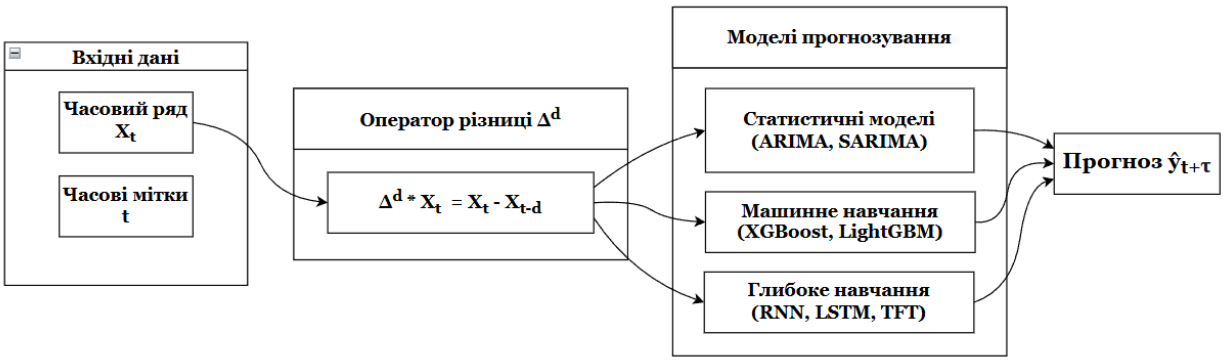

Рис. 1.2 – Спільна основа для моделей прогнозування часових рядів

Навіть найскладніші моделі починаються з простого кроку: часовий ряд стає “передбачуваним” шляхом віднімання значення попереднього дня або іншого зсуву від поточного значення. Це називається оператором різниці. Потім модель будує прогноз на основі такого перетвореного часового ряду.

До групи класичних статистичних методів належать адаптивні моделі експоненціального згладжування: моделі Брауна, Холта та Вінтерса, а також авторегресійні підходи, з яких найвідомішим є метод Бокса-Дженкінса, заснований на моделі ARIMA (Autoregressive Integrated Moving Average) [18; 19]. Підхід ARIMA об’єднує три елементи: авторегресію (AR) – прогноз, заснований на попередніх значеннях, ковзне середнє (MA) – прогноз,

заснований на попередніх похибках, та інтегрування (I) – приведення ряду до стаціонарної форми. Математична модель ARIMA (p, q, d) для нестаціонарного часового ряду визначається формулою 1.1:

$$\Delta^d X_t = c + \varepsilon_t + \sum_{i=1..p} \alpha_i \Delta^d X_{t-i} + \sum_{j=1..q} b_j \varepsilon_{t-j} \quad (1.1)$$

де Δ^d – оператор різниці порядку d; X_t – поточне значення ряду; ε_t – білий шум; c, α_i, b_j – параметри моделі; p та q – порядки авторегресії та ковзного середнього відповідно.

До переваг підходу можна віднести: високу математичну наочність, міцну теоретичну основу та ефективність при роботі з короткими часовими рядами, що мають явний лінійний тренд. Проте, головний недолік полягає в тому, що статистичні моделі є суто лінійними та локальними.

Процес ARIMA формує прогноз у кілька етапів: спочатку визначається тренд (на основі різниць), а потім використовуються попередні значення та похибки для прогнозування наступного значення. Процес представлений на рисунку 1.3. Модель добре працює у випадку лінійного тренду та невеликого шуму, однак зазнає труднощів із різкими стрибками.

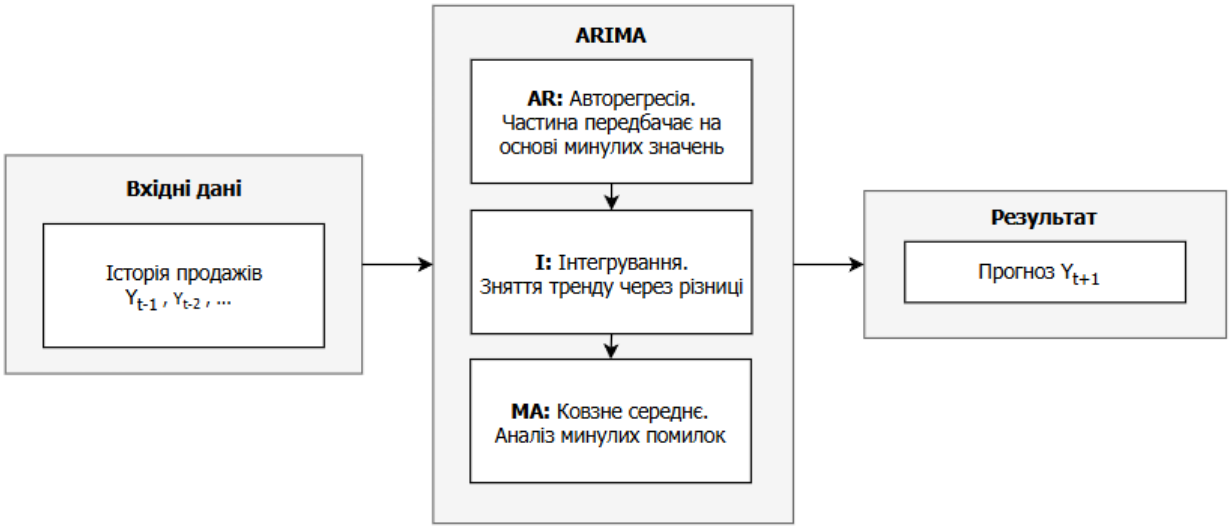

Рис. 1.3 – Принцип роботи ARIMA

Побудова прогнозу вимагає трудомісткого визначення параметрів для кожного часового ряду, що робить цей підхід обчислювально неефективним при роботі з широким спектром продуктів. Крім того, класична форма

рівняння не дозволяє безпосередньо моделювати вплив нелінійних зовнішніх факторів, таких як різкі коливання цін або маркетингові заходи.

Моделі градієнтного підсилення GBM [20] замінили класичні дерева прийняття рішень, які страждають від перенавчання та надмірної чутливості до вхідних даних. Суть цього підходу полягає в послідовній побудові ряду слабких моделей, причому кожне наступне дерево намагається мінімізувати функцію втрат на основі помилок попередніх моделей. У математичному вигляді модель адитивного підсилення наведена у формулі 1.2:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \qquad f_k \in F \tag{1.2}$$

де K – загальна кількість дерев в ансамблі, f_k – базова функція дерева з простору дерев F, x_i – вектор вхідних ознак.

Серед найпопулярніших сучасних реалізацій виділяються два основні алгоритми. Перший – XGBoost, що використовує додаткову L1- та L2-регуляризацію в цільовій функції для запобігання перенавченню. Забезпечує високу точність, але вимагає значних обсягів пам’яті та ретельного налаштування гіперпараметрів. Другий – LightGBM, який застосовує асиметричний алгоритм зростання дерева “Leaf-First”, який значно прискорює навчання на великих наборах роздрібних даних у порівнянні з поступовим зростанням XGBoost [21].

Однак дерева рішень за своєю природою не здатні до екстраполяції, тобто вони не можуть передбачати значення, що виходять за межі діапазону навчального набору даних. Крім того, ці алгоритми не мають вбудованого механізму, що враховує часові ряди.

Прогностичні здібності нейронних мереж безпосередньо випливають з їхньої здатності розпізнавати приховані зв'язки між вхідними та вихідними даними, а також з їхньої здатності до узагальнення. Серцем будь-якої штучної нейронної мережі є модель нейрона. З математичної точки зору нейрон являє

собою зважений підсумовуючий елемент, єдиний вихід якого визначається його входами та матрицею ваг відповідно до формули 1.3:

$$y = f\left(\sum_{i=1}^{n} \omega_i x_i + \omega_0 x_0\right) \quad (1.3)$$

де x_i та ω_i – відповідно сигнали на входах нейрона і ваги входів;
x_0 та ω_0 – додатковий вхід для ініціалізації нейрона, зміщення та його вага;
вираз під знаком функції називається індукованим локальним полем, а f(u) – нелінійною передавальною функцією.

Хоча класичні рекурентні нейронні мережі RNN та мережі з короткостроковою і довгостроковою пам’яттю LSTM є ефективними при моделюванні послідовностей, їхнім головним недоліком є проблема “чорного ящика”: принципова неможливість точно пояснити, які фактори вплинули на формування конкретного прогнозу [22; 23; 24].

Останнім досягненням у розвитку структурних моделей стала поява архітектур на основі трансформерів, зокрема Temporal Fusion Transformer [25; 26; 27; 28; 29]. На відміну від класичних мереж, TFT був спеціально розроблений для інтерпретованих прогнозів з декількома горизонтами і має низку унікальних архітектурних особливостей.

Temporal Fusion Transformer – це архітектура нейронної мережі, розроблена для прогнозування часових рядів на декількох горизонтах. Вона поєднує в собі переваги рекурентних мереж LSTM для обробки послідовностей, трансформерів, механізму уваги для виявлення довгострокових залежностей, а також інтерпретованих механізмів VSN – ваг уваги. Нижче розглянуто кожен елемент цієї архітектури більш детально.

Загальна структура моделі TFT представлена на рисунку 1.4. Вона складається з трьох основних етапів: поділу вхідних даних, обробки за допомогою мережі вибору змінних і блоків GRN, кодера/декодера LSTM, механізму уваги та квантильного виходу. Весь потік даних паралельно

доповнюється залишковими зв'язками та нормалізацією “Add & Norm”, які забезпечують стабільне навчання мережі.

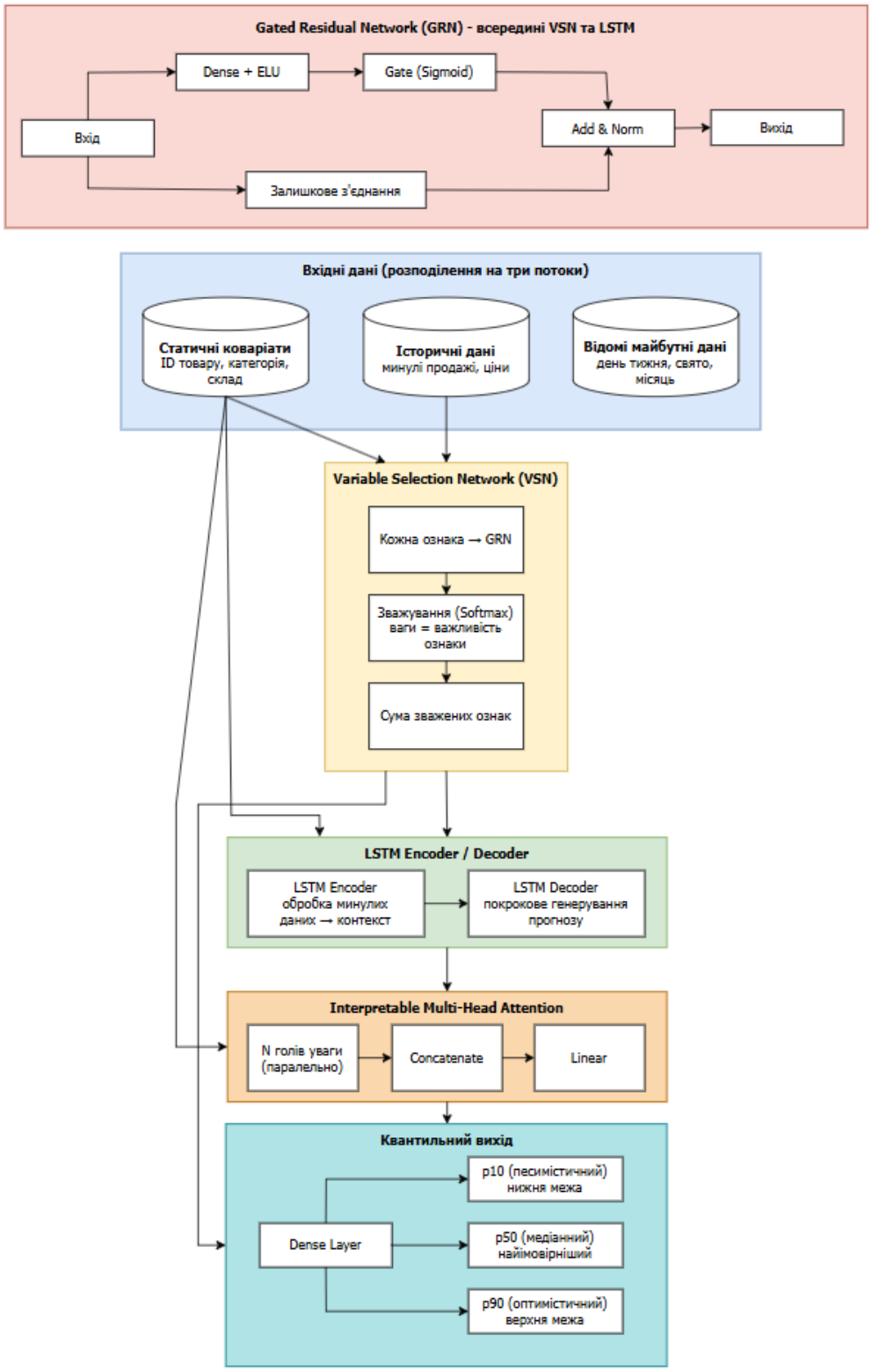

Рис. 1.4 – Загальна архітектура Temporal Fusion Transformer

Першим етапом є поділ усіх вхідних даних на три категорії:

- Статичні коваріати – параметри, які не змінюються з часом. Наприклад: ідентифікатор товару (SKU), категорія, місцезнаходження складу. Вони становлять глобальний контекст для всієї подальшої обробки.
- Вхідні дані, спостережувані в минулому – минулі значення цільових змінних, наприклад, закупівлі за попередні дні. Ці дані відомі лише до поточного моменту.
- Відомі майбутні вхідні дані – змінні, значення яких відомі заздалегідь для всього горизонту прогнозу: день тижня, номер місяця, свята, заплановані маркетингові кампанії тощо.

Ця особливість є ключовою перевагою TFT порівняно з класичними підходами ARIMA та LSTM, які не можуть явно використовувати інформацію щодо майбутнього.

Після поділу кожен потік даних проходить через мережу вибору змінних VSN. Її завдання – динамічно визначати, які вхідні характеристики на даний момент є найбільш важливими для прогнозу.

Принцип роботи мережі VSN:

1. Кожна вхідна змінна, наприклад, day_of_week, price, sales_lag_7 проходить через окремий блок GRN, Gated Residual Network, який визначає її характеристики.
2. Окремий підблок обчислює вектор ваг за допомогою функції Softmax, яка перетворює набір чисел у розподіл ймовірностей, при цьому сума всіх ваг = 1.
3. Змінні з дуже малими вагами, близькими до 0, фактично виключаються з подальшої обробки.
4. Зважені характеристики підсумовуються, і результат передається далі.

Після завершення навчання можна відобразити ваги, присвоєні кожній характеристиці. Якщо, наприклад, характеристика month_cos отримала найбільшу вагу для даного продукту, це означає, що модель спирається на річну сезонність. Адміністратор отримує чітке пояснення: “прогноз базується на оборотах за той самий місяць попереднього року”.

GRN є базовим елементом, що використовується в модулях VSN, LSTM й інших модулях TFT та виконує такі функції:

- Нелінійне перетворення – за допомогою функції ELU, Exponential Linear Unit.
- Гнучке керування глибиною – за допомогою сигмоїдальної функції, яка генерує коефіцієнт у діапазоні від 0 до 1. Цей коефіцієнт визначає частину обробленого сигналу, яка додається до вихідного сигналу.
- Залишкове з'єднання – вихідний вхідний сигнал додається до виходу функції. Завдяки цьому сигнал може обійти нелінійну обробку, коли вона не потрібна, що гарантує, що в разі простих залежностей модель не поступається за продуктивністю лінійній моделі.
- Нормалізація відбувається після додавання та стабілізує навчання глибокої мережі.

Для моделювання локальних і послідовних залежностей TFT використовує два рекурсивні шари LSTM:

- Енкодер LSTM обробляє історичні дані, наприклад, історію продажів. Він зчитує часовий ряд крок за кроком і створює контекстний вектор, тобто представлення всієї послідовності.
- Декодер LSTM генерує прогнози на майбутнє. Він також працює крок за кроком і використовує контекстний вектор, а також власну пам'ять для передбачення наступного значення.

Модель LSTM добре підходить для виявлення короткострокових трендів – від одного до декількох тижнів. Наприклад, вплив вчорашньої знижки на сьогоднішній оборот. Однак ця модель має недолік – при довгих послідовностях вона може забути старі події, наприклад, при річних циклах. Проте, цю проблему вирішує механізм уваги.

Варто звернути увагу на те, що TFT дозволяє деяким характеристикам обходити декодер LSTM. Якщо дана змінна вже є досить значущою і не потребує послідовної обробки, наприклад, статичні змінні, вона може

передаватися безпосередньо до наступних блоків через спеціальні обхідні з’єднання. Це прискорює обчислення та запобігає втраті інформації.

Механізм Interpretable Multi-Head Attention є ключовим елементом, що відрізняє TFT від класичних рекурентних мереж LSTM [30]. Механізм уваги дозволяє моделі повернутися до будь-якого моменту в минулому, незалежно від часової відстані. На відміну від LSTM, який з часом забуває про минулі події, увага зберігає доступ до всієї історії.

Принцип роботи уваги:

1. Для кожного часового кроку обчислюються три вектори: запит, ключ і значення.
2. Обчислюється матриця схожості між запитом поточного кроку та всіма ключами з попередніх кроків. Результатом цього є ваги уваги – числа, які визначають значимість кожного минулого дня для сьогоднішнього прогнозу.
3. Ваги нормалізуються за допомогою функції Softmax, сума яких рівна одиниці.
4. Зважені значення підсумовуються – результат роботи механізму уваги.

Термін multi-head означає, що кілька механізмів цього типу працюють паралельно. Кожен механізм може розпізнавати різні типи залежностей: один орієнтований на 7-денні цикли, другий – на 30-денні, а третій – на річні цикли. Потім результати всіх механізмів об’єднуються.

Після завершення навчання можна відобразити ваги, як показано на рисунку 4.3 у розділі 4.1.3. Якщо, наприклад, модель присвоює високі ваги на 15 листопада, 15 жовтня тощо у прогнозі на 15 грудня, це означає, що вона виявила місячну сезонність.

Спеціальні обхідні з'єднання в TFT дозволяють сигналу з попередніх блоків, наприклад, безпосередньо з VSN пропустити декодер LSTM і навіть механізм уваги та перейти безпосередньо до наступного етапу обробки Add & Norm. Це рішення має кілька важливих наслідків:

- Зниження обчислювальної складності, адже якщо дана характеристика вже добре оброблена, наприклад, статичний ідентифікатор продукту, немає необхідності пропускати її через мережу LSTM або механізм уваги.
- Непотрібні нелінійні перетворення пов'язані з ризиком того, що модель запам'ятає шум замість реальної моделі. Обхідні шляхи дозволяють моделі використовувати простий сигнал, коли складна обробка не потрібна.
- Коротші шляхи для градієнтів полегшують зворотне поширення помилки – прискорення навчання.

У коді ці обхідні шляхи реалізуються за допомогою залишкових з'єднань – шляхом додавання вхідного сигналу до виходу після блоку. Якщо блок не вносить ніякої корисної інформації, ваги можуть бути близькими до нуля, і сигнал тоді передається практично без змін.

Кінцевий шар Dense перетворює оброблені дані на три прогнози для кожного рівня горизонту прогнозування. Замість одного значення TFT генерує:

- p10 – песимістичний прогноз, нижня межа. Лише 10 % фактичних значень будуть нижчими за цю межу.
- p50 – медіанний прогноз, найбільш ймовірний, що фактичне значення буде вище або нижче цього значення, становить 50%.
- p90 – оптимістичний прогноз, верхня межа, що означає: тільки 10% фактичних значень будуть вище цієї межі.

Для навчання моделі генерації правильних квантилів використовується спеціальна функція квантильної втрати. Коли q = 0,5, тобто є медіаною, функція симетрична: штраф за заниження відповідає штрафу за завищення. Коли q = 0,9 оптимістичний, штраф за заниження у дев'ять разів вищий, ніж за завищення, адже модель боїться передбачити занадто низьке значення. Загальна функція втрат розраховується як сума помилок у всіх часових точках горизонту прогнозу та всіх квантилях, визначених відповідно до формули 1.4.

$$\mathcal{L} = \sum_{y_t \in \omega} \sum_{q \in Q} \sum_{\tau=1}^{\tau_{max}} \frac{\max\left(q * (y - \hat{y}), (1-q) * (\hat{y} - y)\right)}{M \tau_{max}} \qquad (1.4)$$

де ω – набір тренувальних даних (часових рядів), що складається з M елементів;

y та $\hat{y}$ – фактичні та спрогнозовані моделлю значення відповідно;

q – цільовий квантиль з множини досліджуваних квантилів Q.

τ_{max} – максимальний горизонт прогнозування.

Нормування похибки щодо розміру вибірки M та горизонту прогнозування τ_max забезпечує стабільність навчання моделі під час роботи з обширними та різноманітними каталогами продуктів. Такий підхід дозволяє нейронній мережі генерувати не лише очікуване середнє значення, а й повні довірчі інтервали. Це дає потужну математичну основу для оцінки ризику дефіциту або надлишку запасів.

Статичні дані включаються в модель не одноразово, але й на декількох рівнях. Вони впливають на:

- VSN – контекст вибору змінних.
- Кодер/декодер LSTM – початковий стан пам'яті (ініціалізація).
- Механізм уваги – налаштування за замовчуванням при розрахунку ваг.

Завдяки цьому модель може зрозуміти, що різні продукти можуть демонструвати різну динаміку, навіть якщо їхня історія продажів схожа. Наприклад, продукти з однієї й тієї ж категорії можуть демонструвати сезонні тенденції, але мати різні абсолютні обсяги продажів.

У порівняльній табл. 1.3 систематизовано переваги TFT у порівнянні з класичними методами.

Таблиця 1.3

Порівняльний аналіз методів прогнозування часових рядів

Характеристика	ARIMA	XGBoost / LightGBM	LSTM	TFT
Тип моделі	Статистична	Ансамблева	Рекурентна нейромережа	Трансформер + RNN
Інтерпретованість	Висока, прозорі коефіцієнти	Низька, “чорний ящик”	Низька	Висока, VSN, ваги уваги
Робота з різною довжиною історії	Фіксована	Фіксоване вікно	Змінна, теоретично	Змінна, енкодер/декодер
Врахування сезонності	Потребує ручного задання	Не враховує автоматично	Частково, затухання градієнта	Автоматично, через увагу
Екстраполяція	Так	Ні	Так	Так
Одна модель на всю номенклатуру	Ні, потрібна окрема для кожного товару	Частково, потрібна підготовка ознак	Так	Так
Довірчі інтервали	Ні	Ні	Ні	Так, квантилі: p10, p50, p90

Аналізуючи еволюцію методів прогнозування часових рядів, ми можемо стверджувати, що вона рухається від лінійних статистичних моделей до глобальних структурних алгоритмів глибокого навчання. Ці алгоритми

дозволяють автоматично обробляти складні нелінійні залежності та масиви даних, поєднуючи при цьому високу точність з інтерпретованістю.

1.6. Порівняльний аналіз підходів реалізації чат-боту ШІ

Додавання в систему діалогового інтерфейсу покликане зменшити бар'єр у використанні системи для співробітників та спростити доступ до даних. Замість того щоб створювати складні SQL-запити або налаштовувати фільтри, користувач зможе сформулювати запит природною мовою, наприклад, попросивши систему знайти товари, яких немає на складі, або скласти прогноз закупівель.

Для впровадження такого функціоналу маємо розглянути три основні архітектурні підходи: системи, засновані на правилах, локальні мовні моделі та хмарні API-інтерфейси.

Перший підхід полягає у використанні таких систем, які засновані на жорстких правилах. Логіка роботи таких чат-ботів описується за допомогою дерева рішень або пошуку за ключовими словами. Хоча цей метод гарантує повну передбачуваність, він має вкрай низьку гнучкість. Система не розуміє ні контексту, ні синонімів і не здатна робити жодних аналітичних висновків, що робить її непридатною для складних аналітичних запитів.

Ще одним рішенням є використання великих локальних мовних моделей безпосередньо на обладнанні користувача. Головною перевагою такого підходу є повна конфіденційність даних і можливість роботи в автономному режимі. Однак локальні моделі висувають високі вимоги до обладнання. Для того щоб модель працювала правильно і на достатньо високому рівні, потрібен потужний графічний процесор і великий обсяг відеопам'яті, а обидва ці компоненти часто відсутні в офісних комп'ютерах.

Третій підхід полягає у використанні хмарних API штучного інтелекту. У цьому випадку запит обробляється на віддалених серверах, що дозволяє використовувати потужні моделі, які мають велике контекстне вікно і здатні

робити складні логічні висновки. Цей підхід не перенавантажує клієнтський комп'ютер та ефективно формує запити. З огляду на необхідність знайти баланс між якістю генерації та апаратними обмеженнями користувачів, було обрано саме цей підхід [31; 32; 33; 34].

1.7. Вимоги до захисту даних та криптографічного зберігання паролів

Розробка корпоративного програмного забезпечення для обробки комерційних даних нерозривно пов’язана з впровадженням певних заходів щодо захисту даних. Оскільки архітектура проєкту базується на локальній експлуатації з використанням файлової бази даних, потенційні загрози вже не пов'язані з типовими мережевими атаками, а скоріше з такими ризиками, як несанкціонований фізичний доступ до приміщень компанії або проблеми з цілісністю файлів.

У даному випадку модуль аутентифікації можна вважати найпростішим засобом безпеки. Користувач повинен мати можливість користуватися програмним забезпеченням виключно з використанням свого унікального ідентифікатора. Щоб захистити систему від спроб копіювання файлу бази даних і від несанкціонованого доступу, зберігання паролів користувачів у вигляді звичайного тексту є неприпустимим. Цю проблему можна вирішити шляхом застосування хешування паролів, такого як SHA-256 або bcrypt [35].

Другий рівень безпеки має передбачати ізоляцію прав доступу на основі рольового підходу. Система повинна містити чітко визначену ієрархію ролей та відповідних прав, щоб зменшити ймовірність атак зсередини. Так, наприклад, адміністративна роль отримає доступ до всіх функцій, пов’язаних з налаштуванням та управлінням персоналом; управлінська роль буде займатися звітуванням, прогнозуванням та ціноутворенням; роль комірника обмежуватиметься лише фіксацією операцій на складі і не матиме доступу до фінансових даних чи закупівельних цін.

І нарешті, нам потрібно вирішити питання цілісності та доступності даних. З огляду на те, що всі дані зберігатимуться в одному локальному файлі, ймовірність повної втрати даних через проблеми з апаратним забезпеченням, програмні помилки або будь-який людський фактор суттєво зросте. Тому нам потрібно забезпечити рішення для резервного копіювання, яке автоматично копіюватиме базу даних і дасть змогу відновити систему в разі будь-яких надзвичайних ситуацій.

Висновки до розділу 1

У першому розділі проаналізовано ринок систем складського обліку. З'ясувалося, що сучасним рішенням бракує вбудованого прогнозування та зручного інтерфейсу для спілкування з системою природною мовою.

На основі цього сформовано ключові вимоги до нашої системи: мультискладський облік, автоматизація документообігу, гнучкий розподіл ролей, гібридна офлайн-синхронізація між комп'ютером і смартфоном, а також інтеграція інтелектуальних модулів.

Для забезпечення швидкодії та повної автономності обрано гібридну архітектуру. Оптимальною базою даних стала SQLite – вона вбудовується прямо в додаток, не потребує адміністрування та нативно працює на Android, синхронізація пристроїв максимально проста [16].

Найкращим варіантом для прогнозування попиту стала архітектура Temporal Fusion Transformer (TFT). Вона використовує єдину модель для всіх товарів, будує прогнози для оцінки ризиків дефіциту і, що найважливіше, дозволяє зрозуміти логіку прогнозу завдяки механізмам уваги [25; 27].

Роль аналітичного NLP-асистента довірили хмарному API моделі Google Gemini, що звільняє комп'ютер користувача від важких обчислень, характерних для локальних нейромереж [31; 34]. За безпеку інформації відповідають: криптографічне хешування паролів SHA-256, рольова модель доступу та створення резервних копій бази даних [35].

РОЗДІЛ 2. ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЧНОГО СТЕКУ ТА ІНСТРУМЕНТАРІЮ

2.1. Обґрунтування вибору мови програмування

Вибір основної мови програмування є одним із ключових етапів розробки будь-якого програмного забезпечення, оскільки він безпосередньо впливає не лише на майбутню архітектуру системи, темпи розробки, масштабованість, а також на інструменти, доступні для інтеграції окремих модулів.

Оскільки система управління запасами, що розробляється, поєднуватиме бухгалтерську логіку зі складними математичними обчисленнями, а саме прогнозами на основі нейронних мереж та обробкою природної мови, мова програмування має бути універсальною.

Для розробки системи було обрано високорівневу та універсальну мову програмування Python [36]. Цей вибір обумовлений низкою архітектурних особливостей та переваг, які роблять Python стандартом у галузі інтелектуального аналізу даних та машинного навчання.

Python підтримує об'єктно-орієнтовану, імперативну та функціональну парадигми програмування [37]. Об'єктно-орієнтований підхід підходить для проектування графічного користувацького інтерфейсу та структурування бухгалтерської логіки системи, тоді як функціональний стиль є ефективнішим під час обробки даних.

Крім того, динамічна типізація значно прискорює процес створення та прототипування алгоритмів у порівнянні зі статично типізованими мовами, такими як C++ або Java [38].

Однією з основних причин вибору мови Python для цього проєкту є також її стандартна система управління пакетами pip, яка забезпечує доступ до сотень тисяч бібліотек. Реалізація архітектури Temporal Fusion Transformer на мовах C# або Java стала б надзвичайно ресурсомістким завданням, оскільки

в цих мовах відсутні ефективні нативні фреймворки для глибокого навчання.

Для мови Python існують оптимізовані інструменти – PyTorch, Pandas і NumPy, які використовують низькорівневі оптимізації на C/C++ [39; 40; 41]. Python є інтерпретованою мовою, що забезпечує її незалежність від апаратної платформи. Код, написаний на Python, може запускатися в операційних системах Windows, а також у системах macOS і Linux без необхідності компіляції або додаткових модифікацій.

2.2. Вибір інтегрованого середовища розробки

Ефективність написання, налагодження та супроводу програмного коду безпосередньо залежить від обраного інтегрованого середовища розробки. У рамках процесу проектування системи було проведено порівняння трьох основних альтернатив: спеціалізованого середовища PyCharm для професійного програмування на мові Python, інтерактивної платформи Jupyter Notebook для аналізу великих масивів даних та кросплатформного редактора Visual Studio Code.

Для реалізації проєкту було обрано Visual Studio Code від Microsoft [42]. VS Code позиціонується як легкий, але надзвичайно продуктивний і модульний редактор, заснований на фреймворку Electron. Вибір саме цього середовища обумовлений його архітектурою та розширеннями. На відміну від монолітних середовищ IDE, які під час запуску завантажують безліч непотрібних модулів, VS Code відразу після встановлення настільки легкий, наскільки це можливо. Усі функції, специфічні для проєкту, додаються шляхом встановлення відповідних розширень.

Розробка проєкту вимагає одночасного виконання складних математичних обчислень, зокрема навчання моделі Temporal Fusion Transformer, а також створення графічного інтерфейсу користувача. Професійні інтегровані середовища розробки, такі як PyCharm, базуються на віртуальній машині Java і самі споживають значний обсяг оперативної пам'яті.

VS Code значно легший завдяки оптимізованій архітектурі, що дозволяє безпосередньо вивільнити апаратні ресурси для прискорення навчання нейронних мереж та обробки даних.

Крім того, наявність розширень для перегляду локальних файлів баз даних, таких як SQLite Viewer, дозволяє візуально перевіряти структуру та вміст бази даних безпосередньо в середовищі розробки.

2.3. Аналіз та вибір системи управління базами даних

Розробка архітектури локальної системи вимагає послідовного підходу до організації рівня зберігання даних. Традиційні клієнт-серверні системи управління базами даних СУБД, такі як PostgreSQL або MySQL, є галузевим стандартом і забезпечують високу продуктивність при доступі великої кількості користувачів [15].

Однак специфіка створюваної системи вимагає автономної роботи на окремому стаціонарному комп'ютері. У таких умовах використання окремого серверного обладнання є зайвим. Оптимальним рішенням для подібних систем є використання вбудованих баз даних. Для реалізації проекту було обрано SQLite – це продуктивна реляційна система управління базами даних, реалізована у вигляді компактної бібліотеки та запускається безпосередньо в рамках самого процесу додатка [16].

Уся база даних, включаючи таблиці, індекси, тригери та представлення, зберігається в одному файлі. Саме ця особливість вирішує проблему синхронізації, оскільки пристрій збору даних на базі Android також спочатку використовує SQLite, обмін даними між комп’ютером і смартфоном здійснюється шляхом простої передачі двійкового файлу. Це усуває необхідність у створенні API-інтерфейсів для перетворення даних.

SQLite не вимагає встановлення фонових служб, налаштування мережевих портів або визначення прав доступу на рівні системи управління базами даних. Завдяки цьому готовий програмний продукт є портативним,

адже користувач може просто запустити програму і відразу приступити до роботи, що є значною перевагою, як для програмного забезпечення для малих підприємств.

Розробка системи значно спрощується, оскільки стандартна бібліотека мови Python містить вбудований модуль `sqlite3`. Це позбавляє від необхідності встановлювати та керувати драйверами для підключення до бази даних. Крім того, SQLite підтримує динамічну типізацію, що забезпечує більшу гнучкість при імпорті різнорідних даних, таких як прайс-листи у форматі CSV, з використанням бібліотеки Pandas. З огляду на широку підтримку запитів та архітектурну сумісність із мобільним клієнтом, SQLite є найбільш доцільним та ефективним інструментом для зберігання даних у цьому проєкті.

2.4. Обґрунтування засобів розробки графічного інтерфейсу

Оскільки кінцевими користувачами розроблюваної системи, будуть менеджери, комірники та логісти користувацький інтерфейс має бути інтуїтивно зрозумілим і сучасним, а також позбавленим зайвих технічних деталей. Для створення додатків мовою Python існує кілька стандартних фреймворків: PyQt/PySide, Kivy та вбудована бібліотека Tkinter [36].

Використання PyQt у даному випадку є зайвим, оскільки воно значно збільшує розмір виконуваного файлу, а Kivy більшою мірою орієнтований на мобільні пристрої. Стандартна бібліотека Tkinter, хоча і є інтегрованою, але користувацькі інтерфейси з її використанням виходять із застарілим візуальним стилем.

Тому було обрано сучасну бібліотеку CustomTkinter – це розширення класичного модуля Tkinter, повністю написане мовою Python. Бібліотека надає низку нестандартних віджетів, які відповідають більш сучасним стандартам. Ці віджети підтримують згладжування країв, закруглені кути, колірні градієнти та тіні [43; 44].

CustomTkinter також автоматично розпізнає налаштування масштабування операційної системи Windows і динамічно налаштовує розмір шрифтів та елементів.

Обрана бібліотека містить вбудовану функцію зміни теми. Вона може динамічно адаптуватися до глобальних системних налаштувань. Таким чином, саме за допомогою CustomTkinter спробуємо створити максимально привабливий і швидкий користувацький інтерфейс.

2.5. Інструментальні засоби попередньої обробки та візуалізації даних

Оскільки архітектура системи передбачає використання алгоритмів машинного навчання, сирі дані з реляційної бази даних SQLite не можуть передаватися безпосередньо до нейронної мережі. Ці дані необхідно очистити та агрегувати. Для вирішення цих завдань до проєкту було включено дві основні бібліотеки Python: Pandas та Matplotlib [45; 40].

Pandas базується на структурі даних DataFrame, яка дозволяє працювати з інформацією у вигляді двовимірних таблиць [46]. Перевагою бібліотеки Pandas при прогнозуванні часових рядів є наявність набору інструментів для роботи з часовими індексами. У нашій системі, ця бібліотека використовується для автоматичного перетворення потоку транзакцій у регулярний часовий ряд із фіксованою періодичністю. Якщо в деякі дні товари не продавалися, у часовому ряді виникають прогалини. Pandas дозволяє правильно заповнити їх нулями або усереднити значення за допомогою вбудованих методів.

Бібліотека Matplotlib була обрана для представлення результатів прогнозів та надання користувачеві аналітичного інструменту [45]. Це низькорівнева бібліотека, яка, однак, надзвичайно гнучка і дозволяє створювати двовимірні графіки. У рамках системи вона використовується для представлення графіків прогнозу, що генеруються нейронною мережею. В поєднанні з CustomTkinter ми маємо можливість подавати утворені графіки безспоредньо в інтерфейс програми.

2.6. Програмні інтерфейси генеративного штучного інтелекту

Розробка модуля NLP-помічника, здатного перетворювати запити користувачів, на коректні SQL-запити, а також генерувати коментарі, вимагає наявності надійного каналу зв’язку між нашою системою та мовною моделлю. Для забезпечення такої взаємодії було використано Google Gen AI SDK [31].

Хоча взаємодія з моделлю Gemini можлива за допомогою прямих HTTP-запитів, REST-API, використання спеціального SDK надасть деякі переваги. SDK містить логіку керування історією діалогу. Замість того, щоб вручну надсилати всі попередні повідомлення при кожному новому запиті, використовується об’єкт ChatSession. Це дозволяє користувачеві вести зв’язний діалог, у якому модель запам’ятовує попередній контекст.

Оскільки основним завданням цього модуля є отримання даних з таблиць, додаток має отримувати від моделі чіткі параметри фільтрації, а не довільний текст. Однією з найбільших проблем, пов’язаних із використанням моделей LLM, є ризик отримання відповіді в непередбачуваному форматі. В Google Gen AI SDK ця проблема вирішена за рахунок підтримки схем даних JSON [33]. Таким чином, нейронна мережа змушена аналізувати запит користувача і повертати виключно структурований словник, що містить параметри фільтрації. Такий підхід усуває ризик появи помилкового тексту у відповіді та дозволяє передати параметри в логіку додатка для фільтрації рядків таблиці.

Навіть під час обробки простих запитів на фільтрацію виклик хмарного API через Інтернет займає певний час, який може становити від кількох мілісекунд до кількох секунд. SDK підтримує асинхронні виклики, що власне запобігає блокуванню інтерфейсу додатка.

Таким чином, Google Gen AI SDK виступає в якості надійного проміжного програмного шару, який абстрагує складність мережевих

взаємодій і забезпечує перетворення запитів у програмні параметри для керування таблицями.

2.7. Обґрунтування вибору фреймворку глибокого навчання та бібліотеки прогнозування

Для впровадження модуля прогнозування попиту на основі архітектури Temporal Fusion Transformer необхідно було вибрати фреймворк, який підтримує побудову графів для обробки часових рядів змінної довжини, підтримує CUDA для прискорення навчання на графічному процесорі, а також пропонує готові механізми уваги та квантильної регресії. З доступних альтернатив TensorFlow і PyTorch – вибір припав на PyTorch [39; 47].

PyTorch створює граф динамічно і це дозволяє змінювати довжину послідовності під час навчання. Ця вимога є важливою бо у випадку історії продажів різних продуктів, можуть охоплюватись різні періоди. Крім того, PyTorch можна інтегрувати з CUDA, а це дозволяє перенести операції на графічний процесор, пришвидшуючи навчання.

Реалізація TFT вимагала б ручного написання складних механізмів уваги, залишкових мереж та квантильної функції втрат. З цієї причини було обрано бібліотеку PyTorch Forecasting, оскільки вона надає готові до використання, високоабстрактні функції для роботи з часовими рядами: класи для перетворення даних Pandas, а також всі необхідні компоненти: мережу вибору змінних, рекурентні мережі, LSTM-кодер/декодер, механізм уваги та квантильний вихід [48].

Усі дані витягуються з бази даних за допомогою SQL-запитів, перетворюються у формат, придатний для TimeSeriesDataSet, а після завершення навчання формуються графіки. Такий підхід дозволяє нам поєднати продуктивність TFT із гнучкістю PyTorch.

Висновки до розділу 2

У другому розділі обґрунтовано вибір інструментів, використаних для розробки системи. Основними критеріями вибору є: забезпечення автономності, продуктивності та безпеки, а також інтеграція сучасних методів машинного навчання та обробки природної мови.

Мова програмування Python була обрана завдяки екосистемі машинного навчання, підтримці об'єктно-орієнтованого та функціонального стилів програмування, а також наявності вбудованих модулів для роботи з SQLite та графічним інтерфейсом користувача. Обчислення ж виконуються за допомогою скомпільованих бібліотек C/C++.

Як середовище розробки було обрано Visual Studio Code завдяки архітектурі, модульності та простоті використання у великих проєктах. Обрана система управління базами даних SQLite зберігає всю базу даних в одному файлі, що працюватиме з додатками на більшості популярних платформ. Це дозволятиме здійснювати синхронізацію, наприклад, з мобільним клієнтом на системі Android, без використання окремого сервера. Для реалізації графічного інтерфейсу користувача використовувалася бібліотека CustomTkinter – сучасне розширення стандартного Tkinter, що пропонує сучасні віджети, підтримку тем та автоматичне масштабування.

Для підготовки даних для навчання, заповнення пропущених значень, агрегування та створення ознак було обрано Pandas через його розширені функції обробки часових рядів. Для візуалізації прогнозів використовувався Matplotlib, який легко поєднується в роботі з CustomTkinter. Для інтерфейсу з NLP-помічником використовуватиметься Gemini Cloud API, що виконуватиме конвертацію запитів у SQL.

Модуль прогнозування реалізовано з використанням PyTorch. Бібліотека пропонує готову до використання реалізацію архітектури Temporal Fusion Transformer.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ТА РОЗРОБКА ЯДРА СИСТЕМИ

3.1. Проєктування бази даних та шару доступу до даних

Розробка системи розпочалася з проєктування реляційної бази даних, у якій мають зберігатися всі необхідні відомості про: товари, склади, документи замовлень та надходжень, а також дані про інвентаризації та користувачів.

Структура бази даних в обраній СУБД SQLite складається з дев'яти таблиць, зв'язки між якими представлені на рисунку 3.1.

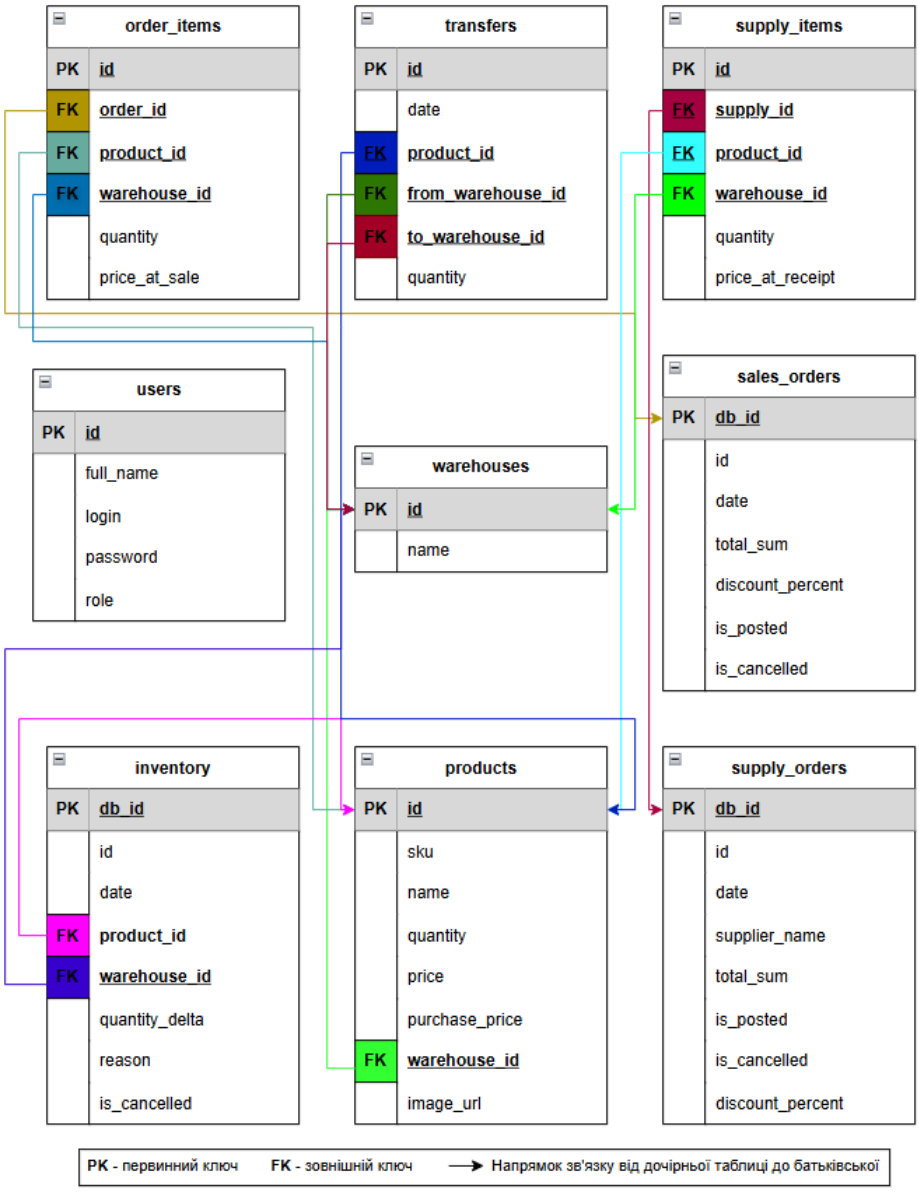

Рис. 3.1 – Структура бази даних проєкту

Детальний опис кожної таблиці, її призначення та ключові поля наведено в табл. 3.1.

Таблиця 3.1

Опис таблиць бази даних

Назва таблиці	Призначення	Первинний ключ (PK)	Зовнішні ключі (FK)	Додаткові обмеження
users	Облікові записи користувачів та їх ролі	id	–	Login UNIQUE
warehouses	Перелік складів	id	–	Name UNIQUE
products	Товари (номенклатура)	id	1. warehouse.id 1. warehouses.id	UNIGUE (sku, warehouse_id)
sales_orders	Заголовки замовлень на продаж	db_id	–	–
order_items	Позиції замовлень	id	1. order_id 1. sales_orders.db_id 2.product_id 2. products.id 3. warehouse_id 3. warehouses.id	–
supply_orders	Заголовки прибуткових накладних	db_id	–	–

supply_items	Позиції прибуткових накладних	id	1. supply_id 1. supply_orders.db_id 2. product_id 2. products.id 3. warehouse_id 3. warehouses.id	–
inventory	Історія змін залишків	db_id	1. product_id 1. products.id 2. warehouse_id 2. warehouses.id	Quantity_delta може бути від’ємним
transfers	Переміщення товарів між складами	id	1. product.id 1. products.id 2. from_warehouse_id 2. warehouses.id 3. to_warehouse_id 3. warehouses.id	–

Повний лістинг програмного коду з файлу database.py, що відповідає за створення всіх таблиць бази даних, наведений у Додатку А кваліфікаційної роботи.

Така структура дозволяє вести облік товарів на кожному складі окремо, оскільки один і той самий товарний артикул може зберігатися на різних складах. Разом з тим кількість товару зберігається на рівні пари “склад-товар”, що спрощує списання та надходження.

Структура бази даних формувалась також під впливом розробленого в рамках одного з навчальних проєктів – WMS-додатка для ОС Android, який створений за допомогою Kotlin та використовує СУБД SQLite. Саме він є прообразом нашої системи та вже має у собі частину функціоналу, аналогічного до розроблюваної системи. Відповідно для утворення цілісної системи, яка б працювала на різних пристроях та могла синхронізуватися

через обмін даними, велика увага приділяється побудові бази даних з ідентичною структурою та можливістю легкого експорту/імпорту [16].

Для забезпечення синхронізації з додатком на мобільних пристроях, модуль database.py ініціалізує базу даних SQLite, структура якої є ідентичною до схеми бази даних Room на ОС Android. Це дозволяє обмінюватися файлами формату .db без необхідності додаткової конвертації цих форматів або використання проміжних API-серверів.

Об’єднання даних з мобільного пристрою та головною базою даних на стаціонарному комп’ютері відбувається завдяки методу злиття баз даних smart_merge. Система використовує нативну можливість SQLite для об’єднання даних командою ATTACH_DATABASE, яка дозволяє підключити зовнішній файл як віртуальну схему Android і виконувати базові SQL-запити безпосередньо на рівні рушія СУБД. Поданий лістинг 3.1 відображає перенесення користувачів між базами.

Лістинг 3.1

Синхронізація користувачів при об’єднанні бази даних

```
pc_cursor.execute("ATTACH DATABASE ? AS android",
(android_db_path,))
pc_cursor.execute(""" INSERT OR IGNORE INTO users
(full_name, login, password, role)
SELECT full_name, login, password, role FROM
android.users """)
```

Алгоритм виконує перенесення номенклатури, документів та інвентаризацій. При перенесенні рядків замовлень скрипт автоматично виконує тригери перерахунку залишків, що подано на лістингу 3.2 нижче.

Лістинг 3.2

Перерахунок залишків номенклатури при об’єднанні бази даних

```
if post and not canc:
pc_cursor.execute("UPDATE products SET quantity =
quantity - ? WHERE id = ?", (qty, pid))
```

Така логіка забезпечує максимальну швидкодію синхронізації та гарантує, що головний комп'ютер адміністратора завжди містить зведену та актуальну інформацію з усіх мобільних терміналів підприємства. Повний алгоритм злиття баз даних подано в Додатку Б.

Для створення всіх таблиць і перевірки їх наявності при першому запуску застосунку створено функцію init_db(). Вона послідовно виконує створення структур для кожної сутності, а потім, якщо, наприклад, таблиця users порожня, додає трьох базових користувачів: адміністратора, менеджера та комірника. Паролі зберігаються у вигляді хешів SHA-256. Функція hash_password(password) реалізує це хешування [35]. Нижче наведено лістинг 3.3 додавання початкових користувачів:

Лістинг 3.3

Ініціалізація базових облікових записів

```
if cursor.fetchone()[0] == 0:
    users = [
        ('Головний Адмін', 'admin',
hash_password('admin'), 'admin'),
        ('Менеджер Продаж', 'manager',
hash_password('1234'), 'manager'),
        ('Комірник Складу', 'worker',
hash_password('1234'), 'storekeeper')
    ]
    cursor.executemany("INSERT INTO users ...", users)
```

Для уникнення дублювання коду підключення до бази даних створено функцію get_db_connection(), яка повертає з'єднання з рядками, доступними за назвами стовпців. Це спрощує читання результатів запитів в інших модулях, наприклад, row['name'] замість row[1].

Шар доступу до даних реалізовано у вигляді окремого модуля database.py, який надає всім іншим компонентам системи функції для читання та запису даних. Наприклад, для отримання списку всіх замовлень використовується функція, що наведена у лістингу 3.4.

Лістинг 3.4

Отримання списку замовлень з бази даних

```
def get_all_orders():
    conn = get_db_connection()
    try:
        return conn.execute("SELECT db_id, id, date,
total_sum, is_posted, is_cancelled FROM sales_orders
ORDER BY date DESC").fetchall()
    finally:
        conn.close()
```

Використання блоку try...finally, в лістингу вище, гарантує закриття з'єднання навіть у разі помилки. Такий підхід застосовано в усіх функціях. Крім простих запитів, модуль містить складніші операції, зокрема із об'єднанням декількох таблиць.

Для маніпуляцій з базою даних було створено також функції очищення документів – clear_operational_data() та повного скидання даних , крім таблиці користувачів – factory_reset(). Вони використовуються в модулі налаштувань і перед виконанням обов'язково створюють резервну копію БД.

За роботу підсистеми резервного копіювання відповідає функція create_backup(), яка копіює файл wareflow.db у папку backups проєкту з

часовою міткою, після чого викликає іншу функцію – cleanup_old_backups(), яка залишає лише останні 30 копій. Лістинг 3.5 демонструє реалізацію автоматичного бекапу.

Лістинг 3.5

Функція автоматичного зберігання даних

```
def start_auto_backup(interval_hours=24,
callback=None):
    def backup_worker():
        while True:
            time.sleep(interval_hours * 3600)
            success, result = create_backup()
            if callback:
                callback(success, result)
    thread = threading.Thread(target=backup_worker,
daemon=True)
    thread.start()
    return thread
```

Також в системі можлива автоматизація процесу – запускається окремий потік через, який регулярно створює новий бекап. Таким чином, навіть якщо користувач забуде про ручне резервне копіювання, система самостійно зберігає дані та мінімізує ризик їх втрати при збоях.

Окрім простих операцій обліку, модуль database.py реалізує кілька механізмів, які забезпечують цілісність даних. Наприклад, наявна функція delete_warehouse(warehouse_id), яка перед видаленням складу перевіряє, чи є на ньому товари. Якщо кількість товарів більша за нуль, то видалення блокується з відповідним повідомленням.

У SQLite також автоматично створюється службова таблиця sqlite_sequence, яка зберігає поточне значення лічильника для кожного поля з

AUTOINCREMENT. Після видалення всіх записів з таблиці лічильник не скидається автоматично. Щоб номери документів або інших сутностей починалися з одиниці після очищення, функції адміністрування виконують DELETE FROM sqlite_sequence WHERE name='table_name'. Ця особливість запобігає некоректній нумерації нових записів.

Більшість функцій, що змінюють дані – додавання, оновлення, видалення, повертають кортеж, тобто успішність, повідомлення. Це власне дозволяє інтерфейсу показувати конкретну причину помилки.

3.2. Розробка ядра графічного інтерфейсу та спільних компонентів

Після створення шару доступу до даних маємо спроєктувати графічний інтерфейс користувача. Оскільки система мала працювати в середовищі Windows, а також мати сучасний вигляд із підтримкою світлої та темної тем, для реалізації було обрано бібліотеку CustomTkinter [43]. На відміну від класичного Tkinter, customtkinter дозволяє створювати інтерфейс, який візуально наближений до сучасних веб-застосунків.

3.2.1. Організація навігації та процедура авторизації

Робота з програмою починається з вікна авторизації, реалізованого в класі LoginApp у файлі main.py. Це вікно має фіксований розмір та центрується на екрані.

Інтерфейс містить поля для введення логіну та пароля, кнопку входу та підтримує відправку форми натисканням клавіші вводу. Після отримання облікових даних пароль хешується і виконується запит до таблиці users.

Якщо знайдено відповідний запис, вікно авторизації закривається та створюється головне вікно програми DashboardApp, якому передаються ім'я та роль користувача. У разі невдачі з’являється модальне вікно з повідомленням про помилку (рис. 3.2).

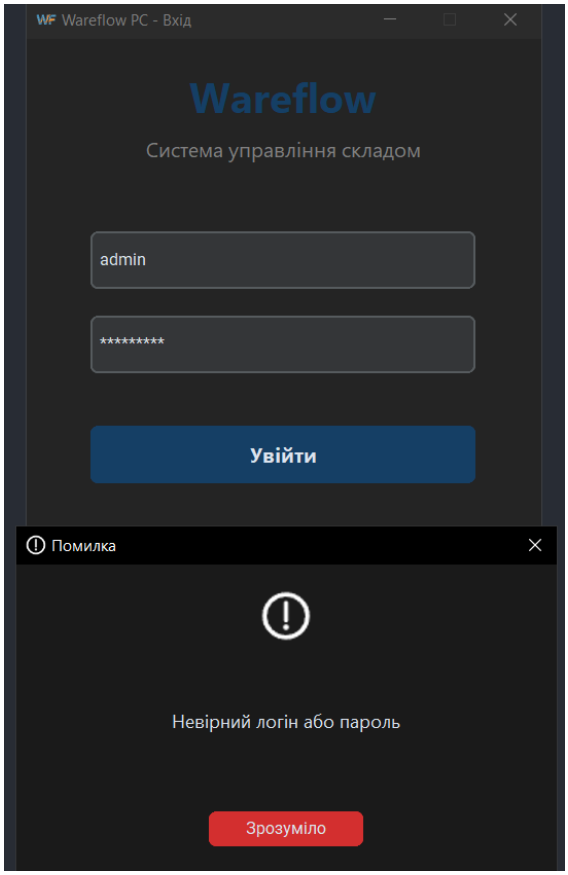

Рис. 3.2 – Вікно авторизації та помилки вводу

Головне вікно DashboardApp є контейнером, що складається з двох основних областей: бічної панелі ліворуч та робочої області, що займає майже увесь вільний простір. Ліва панель має фіксовану ширину пікселів, акцентний темно-синій фон “#153F65” та містить навігаційні кнопки для доступу до всіх модулів: “Номенклатура”, “Замовлення”, “Надходження”, “Інвентаризація”, “Користувачі”, “Налаштування”, “Прогноз”, “AI Асистент” та кнопку виходу (рис. 3.3). Кожна кнопка при натисканні змінює колір фону, а в робочій області відображається обраний модуль.

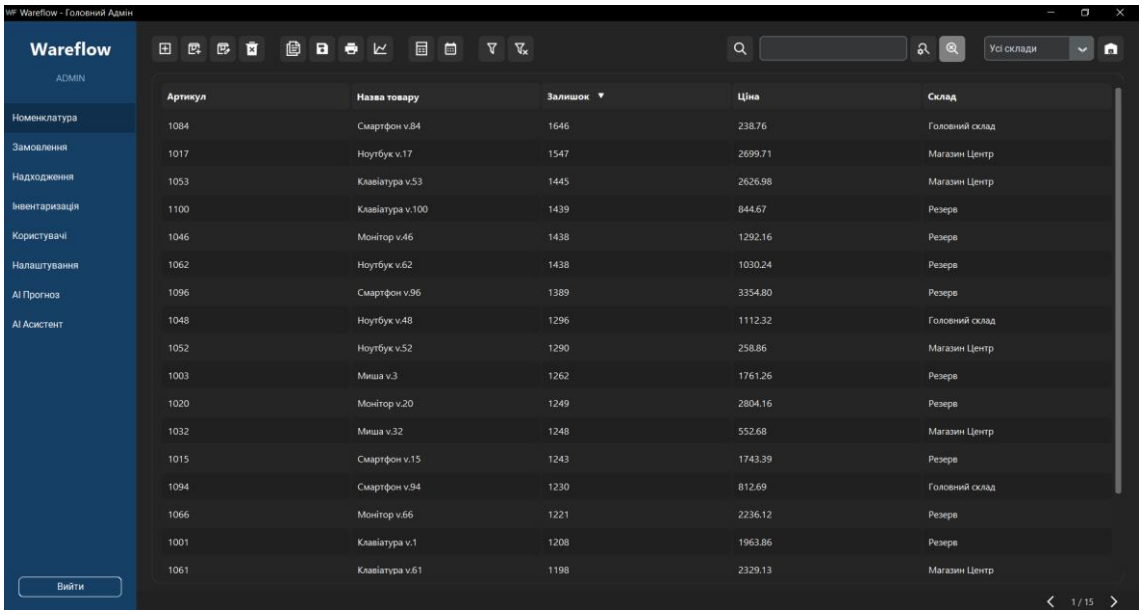

Рис. 3.3 – Головне вікно розроблюваної системи

У конструкторі DashboardApp відбувається ініціалізація всіх модулів, наприклад, WarehouseModule, OrdersModule, InventoryModule тощо. Кожен з модулів є окремим класом, успадкованим від базового BaseModule. Базовий клас надає стандартні методи для очищення області та створення панелі інструментів. Далі викликається метод setup_sidebar() для побудови навігаційного меню, а потім активується модуль “Номенклатура” за замовчуванням. Вікно за замовчуванням завжди розгортається на весь екран автоматично.

Для підтримки перекладу елементів інтерфейсу різними мовами всі текстові рядки отримуються через функцію _() з модуля translations.py. При зміні мови в налаштуваннях викликається метод update_sidebar_language(), який оновлює текст кнопок бічної панелі. Кожен активний модуль перезавантажує свої дані, використовуючи вже нові переклади без перезапуску системи.

3.2.2. Універсальні елементи інтерфейсу: адаптивна таблиця, модальні вікна та підказки

Для уникнення дублювання коду та забезпечення єдиного стилю було створено допоміжний модуль core.py. Він містить набір констант, функцій та класів, які використовуються у всіх інших модулях програми. У модулі визначено глобальні константи кольорів для акцентних елементів, наведення, а також кольори для чергування рядків таблиці, виділення клітинки тощо.

Для використання власних іконок створено функцію get_icon(icon_name, size=(20,20)), яка автоматично завантажує зображення з папки icons залежно від поточної теми. Якщо для темної теми існує файл з суфіксом _dark.png, він використовується, інакше підставляється світла версія – _light.png.

Найскладнішим компонентом core.py є клас CTable. Він реалізує повноцінний табличний віджет із такими можливостями:

- Сортування даних за будь-якою колонкою з підтримкою різних типів числа, дати, рядки.
- Рядки з чергуванням кольорів фону для кращої читабельності.
- Виділення поточного рядка та комірки при кліку.
- Обробка подвійного кліку для виклику додаткових дій, наприклад, відкриття картки товару.

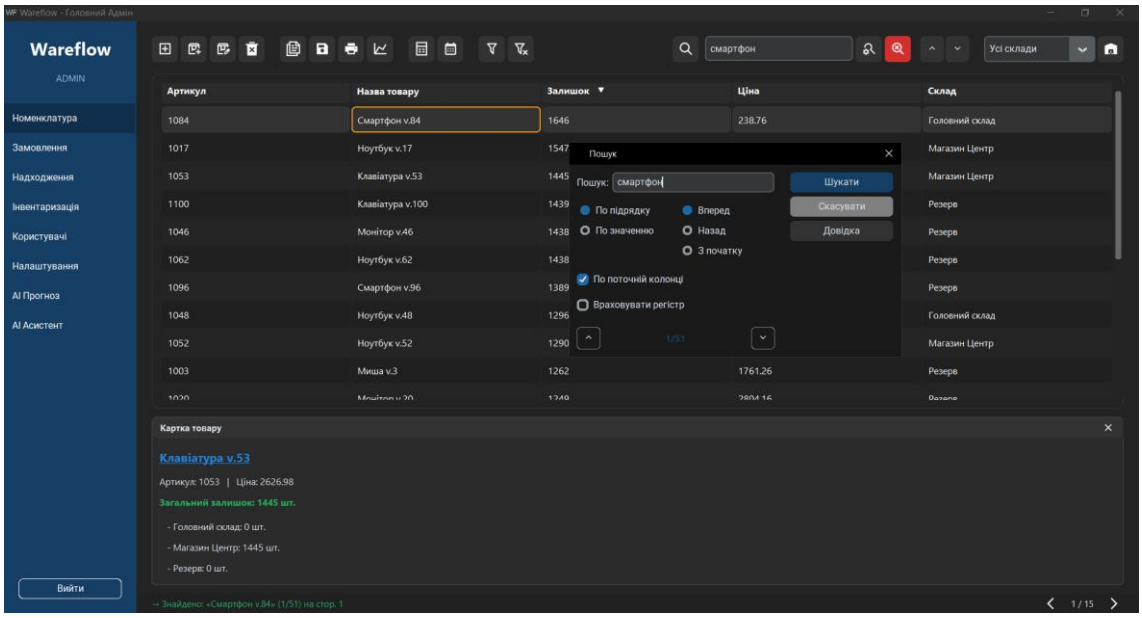

Рис. 3.4 – Приклад роботи таблиці CTable: виділення рядка та комірки, сортування за колонкою.

3.2.3. Реалізація розширеного пошуку з підтримкою пагінації

Для забезпечення швидкого та гнучкого пошуку даних у будь-якій таблиці системи було розроблено модуль search_engine.py. Він складається з двох частин: вікна розширеного пошуку в інтерфейсі системи та класу SearchLogicMixin, який додає відповідні методи до всіх модулів.

Алгоритм роботи:

1. Користувач вводить текст у поле швидкого пошуку або відкриває розширене вікно.
2. SearchLogicMixin викликає get_search_data(), який повертає список рядків у вигляді кортежів з урахуванням поточного сортування та фільтрації.
3. Виконується сканування для кожного рядка, перевіряються всі колонки або лише одна, якщо активовано опцію “Тільки в поточній колонці” на збіг за обраним режимом.

4. Знайдені позиції зберігаються у вигляді кортежів, що зберігає номер сторінки, локальний індекс рядка, індекс колонки та відображуване значення.
5. Користувач переміщується між знайденими елементами за допомогою кнопок “Наступний”/”Попередній”; система за необхідності перемикає сторінку та підсвічує відповідний рядок і комірку.

Клас AdvancedSearchWindow (рис. 3.5) надає додаткові налаштування: вибір режиму збігу, напрямок пошуку тощо. Вікно також містить лічильник знайдених елементів.

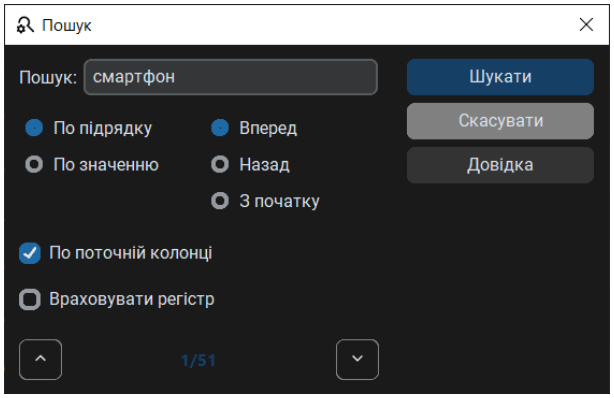

Рис. 3.5 – Вікно розширеного пошуку з налаштуваннями

Для додавання пошуку до будь-якого модуля, нам достатньо було успадкувати SearchLogicMixin поряд з BaseModule, викликати init_search_state() у конструкторі, реалізувати метод get_search_data() та прив'язати кнопки панелі інструментів до методів.

3.2.4. Підтримка багатомовності та динамічне перемикання локалізації

Оскільки до системи виставлено вимогу підтримки різних мов, було реалізовано translations.py, що містить словники для української та англійської мов, які охоплюють близько 400 ключів – від назв кнопок до повідомлень про помилки та підказок. У лістингах 3.6 – 3.7 наведено функцію _(key), яка

повертає переклад поточного рядка відповідно до глобальної змінної current_lang та приклад використання.

Лістинг 3.6

Функція перекладу рядка _(key)

```
def _(key): return LANGUAGES.get(current_lang,
LANGUAGES["Українська"]).get(key, f"{key}")
```

Лістинг 3.7

Приклад використання функції в коді з модуля main.py

```
self.btn_logout = ctk.CTkButton(...,
text=_("btn_logout"), ...)
```

Вибір мови зберігається у файлі config.json, тому після перезапуску програми мова інтерфейсу залишається незмінною. Функція set_language(lang_name) оновлює поточну мову та записує нове значення в конфігураційний файл. При зміні мови в налаштуваннях викликається ця функція, після чого перезавантажуються всі активні модулі.

3.3. Реалізація модуля управління номенклатурою та складами

Модуль warehouse.py є одним з ключових компонентів системи, оскільки реалізує управління номенклатурою та складами. Він забезпечує повний цикл роботи з товарами: додавання, редагування, клонування, видалення, фільтрацію за складом, сортування, пагінацію, пошук, експорт даних у формати Excel/CSV, друк, а також коригування залишків. Основні методи класу WarehouseModule наведено в табл. 3.2.

Таблиця 3.2

Основні методи класу WarehouseModule

Метод / Компонент	Призначення
show()	Побудова інтерфейсу, панелі інструментів, таблиці, пагінації
load_data()	Завантаження товарів з БД з урахуванням фільтрації, сортування, пагінації
open_product_dialog()	Діалог додавання/редагування товару, фото, залишків по складах
dialog_adjust_stock()	Коригування залишку з записом в inventory
dialog_transfer_stock()	Переміщення між складами з записом у transfers
filter_by_selected_cell()	Фільтрація таблиці за виділеною клітинкою
save_table_to_file()	Експорт у Excel/CSV
print_table_contents()	Друк HTML-версії таблиці

При виклику методу show() створюється адаптивна панель інструментів, набір кнопок якої залежить від ролі користувача. Як адміністратор так і комірник отримує повний доступ, а менеджер має лише права перегляду та експорту.

Основна область містить таблицю CTable, у якій підтримується сортування за будь-якою колонкою. Пагінація у таблиці реалізована через запити до бази даних з використанням LIMIT та OFFSET. Фільтрація за складом виконується через випадаючий список, який з'являється на панелі інструментів. При зміні складу таблиця оновлюється автоматично.

Діалог додавання/редагування товару open_product_dialog() створює модальне вікно. Для нового товару автоматично генерується наступний доступний артикул на основі максимального числового значення серед існуючих. При клонуванні артикул генерується заново, а назва отримує суфікс “(копія)”. Лістинг 3.8 демонструє збереження нового товару. Детальніший код методу open_product_dialog() наведено в Додатку В.

Лістинг 3.8

Збереження даних номенклатурної одиниці

```
def save_action():
    conn.execute("INSERT INTO products (sku, name,
quantity, price, purchase_price, warehouse_id) VALUES
(?, ?, 0, ?, 0, ?)",
                 (new_sku, name, price,
wh_dict[e_wh.get()]))
    conn.commit()
```

Коригування залишків дозволяє змінити кількість товару на вибраному складі. При збереженні обчислюється різниця delta = new_qty – old_qty, оновлюється поле quantity в таблиці products та додається запис до таблиці inventory з причиною. Переміщення виконується аналогічно: зменшується кількість на вихідному складі, збільшується на цільовому або створюється новий запис товару, якщо його там ще немає, і фіксується запис у таблиці transfers.

При кліку на рядок таблиці з'являється панель з детальною інформацією про вибраний товар. Вона містить назву товару, оформлену як гіперпосилання для відкриття картки редагування, артикул, ціну, загальну кількість на всіх складах, список залишків по кожному складу та фотографію, якщо вона завантажена.

Модуль підтримує експорт поточної таблиці з урахуванням фільтрів та пошуку у формати Excel через бібліотеку openpyxl та CSV з роздільником “;” та кодуванням UTF-8 [49]. Друк реалізовано через генерацію HTML-сторінки, що містить таблицю, її відкриття в тимчасовому файлі та автоматичний виклик системного діалогу друку.

3.3.1. Журнал обліку змін складських залишків.

Інтерфейс перегляду історії змін залишків товарів реалізовано в модулі inventory.py. На відміну від модуля номенклатури, тут відсутні кнопки додавання, редагування або видалення, оскільки інвентаризації є історичними даними, що формуються автоматично при ручному коригуванні залишків. Таким чином, модуль виконує виключно функцію звітування та аудиту.

Інтерфейс модуля складається з адаптивної панелі інструментів, кнопки збереження, друку, калькулятора, календаря, фільтрації та пошуку й таблиці з даними. Дані отримуються через функцію database.get_inventory_history(), яка виконує об'єднання таблиць inventory, products та warehouses і повертає вже відсортований за датою набір записів.. Перед відображенням (лістинг 3.9) кожен запис обробляється: timestamp дати перетворюється у формат дд.мм.рррр, величина зміни отримує знак “+” для додатних значень, встановлюється статус як “Активний” або “Скасовано”, а причина “Ручна інвентаризація ПК” замінюється на переклад з модуля translations.py.

Лістинг 3.9

Фрагмент автоматичного перекладу розділу системи

```
for r in page_rows: dt =
    datetime.datetime.fromtimestamp(r['date']/1000).str
ftime('%d.%m.%Y')
    delta = f"+{r['quantity_delta']}" if
r['quantity_delta'] > 0 else f"{r['quantity_delta']}"
    status = _("inv_status_cancelled") if
r.get('is_cancelled', 0) else _("inv_status_active")
    reason_text = _("inv_manual_reason") if r['reason']
== "Ручна інвентаризація ПК" else r['reason']
    self.table.insert(row_id=r['id'], values=(dt,
r['name'], r['warehouse_name'], delta, reason_text,
status))
```

Фільтрація за виділеною клітинкою та пошук працюють аналогічно до модуля номенклатури, використовуючи методи з SearchLogicMixin. Метод get_search_data() повертає відсортований список рядків у тому ж форматі, що й таблиця, що дозволяє пошуку працювати з усім набором даних, незалежно від поточної сторінки. Експорт та друк реалізовано ідентично до інших модулів. Таким чином, модуль inventory.py забезпечує прозорість усіх змін залишків.

3.3.2. Автоматизація складського документообігу

Модуль documents.py реалізує підсистему складського документообігу, що включає два типи документів: вихідні замовлення та прибуткові накладні. Класи OrdersModule та ReceiptsModule успадковують спільний базовий клас DocumentModule, який, у свою чергу, успадковує BaseModule та SearchLogicMixin. Таким чином уникаємо дублювання коду, оскільки обидва

модулі мають ідентичну логіку відображення списку документів, пагінації, сортування, фільтрації, пошуку, експорту, друку, тощо.

Доступ до кнопок залежить від ролі: адміністратори та менеджери мають повний доступ до замовлень; адміністратори та комірники – до прибуткових накладних. Інші ролі мають лише права перегляду, копіювання та експорту.

Для роботи з документами створюється універсальне вікно методом open_document_window. Цей метод створює модальне вікно розміром, яке динамічно змінює свої елементи залежно від типу документа та режиму. Вікно складається з кількох логічних блоків:

1. Блок налаштувань документа – відображає статус, поле знижки, а для прибуткових накладних – поля постачальника та складу. Якщо документ вже проведений або скасований, всі поля стають недоступними для редагування.
2. Блок додавання позицій – містить кнопку вибору товару, поле для відображення обраного товару, поля для введення кількості та ціни. Для замовлень ціна продажу підставляється автоматично, для прибуткових – роздрібна ціна показується для інформації, а закупівельну ціну вводить користувач.
3. Таблиця кошика – відображає додані позиції. Після кожної зміни перераховується загальна сума з урахуванням знижки.
4. Кнопки дій – збереження, як чернетки, проведення, скасування проведення, видалення.

Для замовлень відображаються лише товари з додатнім залишком, а також склад, кількість та роздрібна ціна. Для накладних товари групуються за артикулами, без прив'язки до складу, оскільки при оприбуткуванні склад обирається окремо. Подвійний клік по рядку вибирає товар і закриває вікно.

При проведенні замовлення система перевіряє наявність достатньої кількості товару на кожному складі. Якщо перевірка успішна, оновлюються залишки товарів, документі встановлюється прапорець is_posted=1. При

проведенні прибуткової накладної для кожної позиції перевіряється, чи існує товар з таким SKU на обраному складі. Якщо існує – збільшується його кількість та оновлюється закупівельна ціна; якщо ні – створюється новий запис у таблиці products з початковою кількістю, роздрібною ціною з довідника та закупівельною ціною введеною користувачем. У разі помилки транзакція відкочується. Скасування проведеного документа повертає залишки: для замовлень – додає, для прибуткових – віднімає і встановлює is_cancelled=1.

Номери документів id у таблицях sales_orders та supply_orders генеруються за схемою, що прив'язує документи до користувача. До ідентифікатора користувача додається зміщення, множення на 10 000 000, а потім визначається наступний вільний номер у цьому діапазоні. Це дозволяє кожному користувачеві мати свою серію номерів, що полегшує пошук та фільтрацію. Методи збереження та друку аналогічні іншим модулям. Таким чином, модуль documents.py забезпечує повноцінний документообіг із підтримкою чернеток, проведенням, скасуванням та контролем залишків.

3.4. Управління обліковими записами та розмежування прав доступу

Безпека та розмежування прав доступу є критично важливими аспектами будь-якої корпоративної системи обліку. Для управління обліковими записами працівників, їхніми ролями та автентифікаційними даними було розроблено окремий модуль – users.py. Доступ до нього мають лише користувачі з роллю admin – розділ “Користувачі” відображається на бічній панелі головного вікна лише за умови self.role == 'admin'. Це перший рівень розмежування доступу, реалізований ще на рівні побудови інтерфейсу.

3.4.1. Рольова модель доступу та обмеження функціоналу

У системі реалізовано концепцію управління доступом на основі ролей Role-Based Access Control, RBAC. Структура бази даних передбачає наявність атрибута role у таблиці users, який визначає глобальний рівень привілеїв користувача. Система передбачає три ролі: admin, manager та storekeeper. Кожна роль має різний набір доступних дій у межах кожного модуля. Наприклад, у модулі замовлень набір кнопок панелі інструментів формується динамічно, як показано в лістингу 3.10, залежно від ролі.

Лістинг 3.10

Фрагмент формування панелі інструментів, залежно від ролі користувача у розділі замовлень

```
my_permissions = ["copy", "save", "print", "calc",
"calendar", "filter", "filter_remove", "analytics"]
if self.app.role in ['admin', 'manager']:
    my_permissions.extend(["add", "clone", "edit",
"delete"])
```

Для менеджера надаються додаткові права – він має дозвіл на створення, видалення, редагування та клонування замовлень. Комірник, в свою чергу, лише перегляд, експорт та аналітику. В розділі надходжень, у лістингу 3.11, спостерігаємо протилежну ситуацію.

Лістинг 3.11

Фрагмент формування панелі інструментів, залежно від ролі користувача у розділі надходжень

```
my_permissions = ["copy", "save", "print", "calc",
"calendar", "filter", "filter_remove", "analytics"]
if self.app.role in ['admin', 'storekeeper']:
    my_permissions.extend(["add", "clone", "edit",
"delete"])
```

Комірник може додавати, клонувати, редагувати та видаляти товари, а менеджер лише переглядати. Адміністратор отримує всі можливості, включаючи управління складами.

Загальна схема налаштувань прав доступу має вигляд, показаний на рисунку 3.6. Вона ілюструє, як система визначає роль користувача після автентифікації, формує відповідний набір кнопок на панелі інструментів, а потім при спробі виконати дію перевіряє, чи дозволена вона для цієї ролі. Це дозволяє приховати або зробити неактивними кнопки, що не відповідають правам, ще на етапі побудови інтерфейсу.

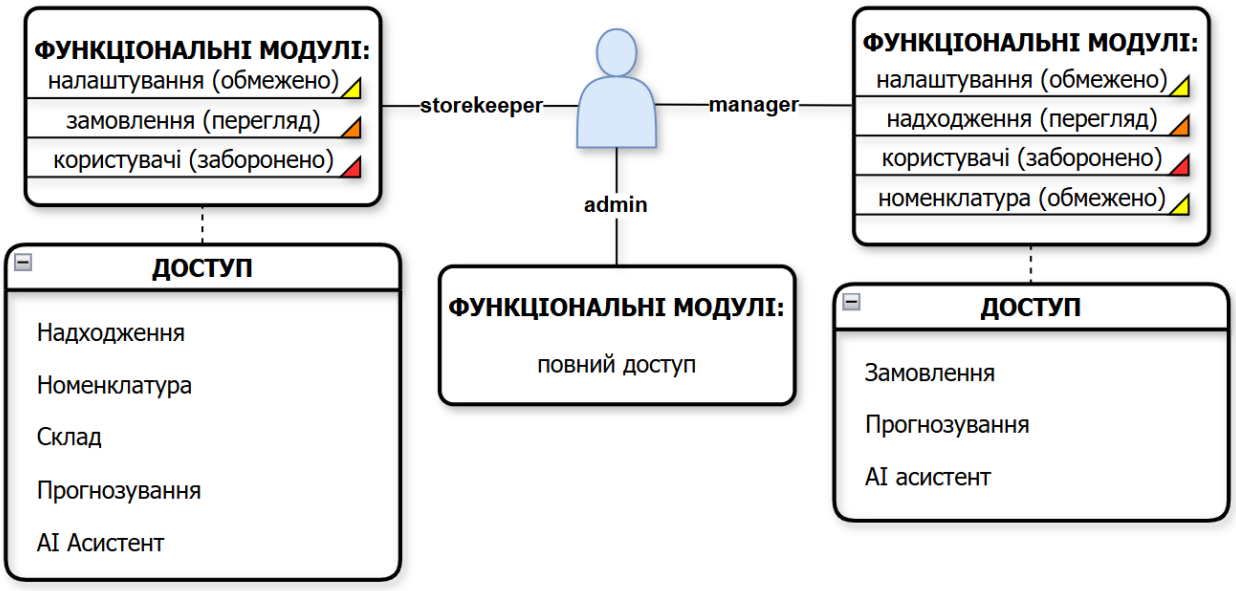

Рис. 3.6 – Архітектура рольової моделі доступу (RBAC) системи

При ініціалізації будь-якого модуля викликається базовий метод build_adaptive_toolbar, якому передається масив дозволених операцій. Якщо поточна роль користувача не передбачає виконання певної дії, наприклад, менеджер не має доступу до видалення прибуткових накладних, відповідна кнопка на панелі інструментів не генерується.

3.4.2. Криптографічний захист автентифікаційних даних

Згідно з сучасними стандартами інформаційної безпеки, зберігання паролів користувачів у відкритому вигляді є неприпустимим. У розробленій системі реалізовано механізм одностороннього криптографічного хешування автентифікаційних даних за алгоритмом SHA-256 [35]. Під час створення або оновлення облікового запису вони хешуються за алгоритмом SHA-256 за допомогою функції, поданої на лістингу 3.12, hash_password(password) з модуля database.py:

Лістинг 3.12

Функція хешування паролів за допомогою модуля hashlib

```
def hash_password(password):
    return hashlib.sha256(password.encode('utf-
8')).hexdigest()
```

При автентифікації введений пароль знову хешується і порівнюється збереженим значенням. Це унеможливлює відновлення оригінального пароля навіть у разі компрометації файлу бази даних. Лістинг 3.13 демонструє створення нового користувача з хешуванням пароля.

Лістинг 3.13

Функція створення нового користувача

```
def add_user(full_name, login, password, role):
    hashed_pwd = hash_password(password)
    conn.execute("INSERT INTO users (full_name, login, password, role)
VALUES (?, ?, ?, ?)",
                 (full_name, login, hashed_pwd, role))
```

Процес обробки пароля під час авторизації та створення користувача проілюстровано на рисунку 3.7.

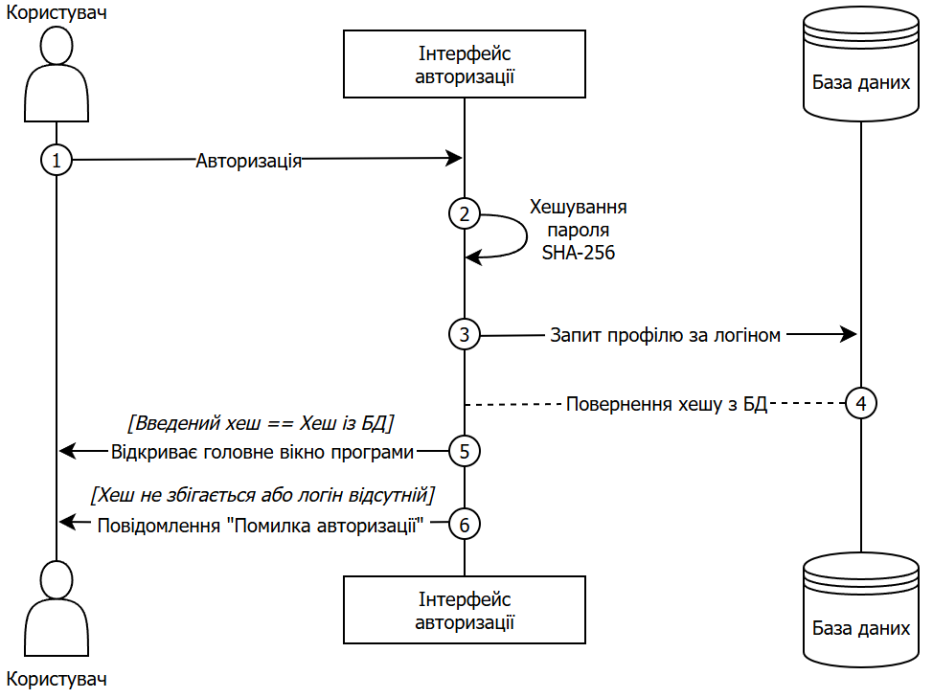

Рис. 3.7 – Схема процесу криптографічного захисту паролів та авторизації

Модуль надає інструменти для управління профілями, що здійснюють валідацію вхідних даних:

- При створенні нового користувача перевіряється унікальність логіна на рівні БД. У разі конфлікту транзакція відхиляється.

- При модифікації профілю поле пароля залишається порожнім. Якщо воно не заповнюється новим значенням, SQL-запит UPDATE ігнорує його, зберігаючи попередній хеш.

Таким чином, у модулі users.py реалізовано повноцінне керування обліковими записами та розмежування прав доступу.

3.5. Розробка аналітичного модуля та візуалізація статистики

Аналітичний модуль analytics.py являє собою універсальне вікно для побудови графіків і звітів на основі даних системи. Він розроблений як окремий клас AnalyticsWindow, успадкований від CPopupWindow. Модуль викликається з панелі інструментів з передачею параметра report_type – “warehouse”, “orders”, “receipts”. Залежно від цього параметра динамічно змінюється набір доступних видів аналізу, джерело даних та заголовок вікна.

Було визначено три типи звітів, кожен з яких має власні SQL-запити. Для складського звіту – “Найбільші залишки”, “Дефіцит товарів”, “Вартість складів”. Для звіту по замовленнях – “Популярні товари”, “Динаміка продажів”, “Порівняння товарів”. Для звіту по надходженнях – “Найбільш закуповувані товари”, “Постачальники”, “Динаміка закупівель”, “Порівняння товарів”. Для кожного виду аналізу було створено параметризований SQL-запит з урахуванням часового діапазону для звітів, що залежать від часу та обмеження LIMIT для Top N. У лістингу 3.14 наведено приклад параметризованого SQL-запиту для популярних товарів.

Лістинг 3.14

Запит до бази даних по номенклатурним одиницям з найвищим попитом

```
data = conn.execute("""
    SELECT p.name, SUM(oi.quantity) as s_qty
    FROM order_items oi
    JOIN products p ON oi.product_id = p.id
    JOIN sales_orders so ON oi.order_id = so.db_id
    WHERE so.is_posted = 1 AND so.is_cancelled = 0
        AND so.date >= ? AND so.date <= ?
    GROUP BY p.name ORDER BY s_qty DESC LIMIT ?
""", (start_ms, end_ms, top_n)).fetchall()
```

Для фіксованих періодів – 7, 30, 90 днів початкова дата обчислюється відносно максимальної дати в таблиці, а не поточної дати – це дозволяє коректно аналізувати історичні дані навіть якщо останній запис не є сьогоднішнім. Для довільного періоду виконано парсинг рядків формату ДД.ММ.РРРР з перевіркою коректності; у разі помилки виводиться діалогове вікно.

Вікно містить панель керування та область візуалізації. На панелі розташовані: випадаючий список періоду, кнопка вибору виду аналізу, випадаючий список типу графіка (авто, стовпчикова, горизонтальна, кругова, лінійна), кнопка вибору основного кольору та кнопка скидання кольору.

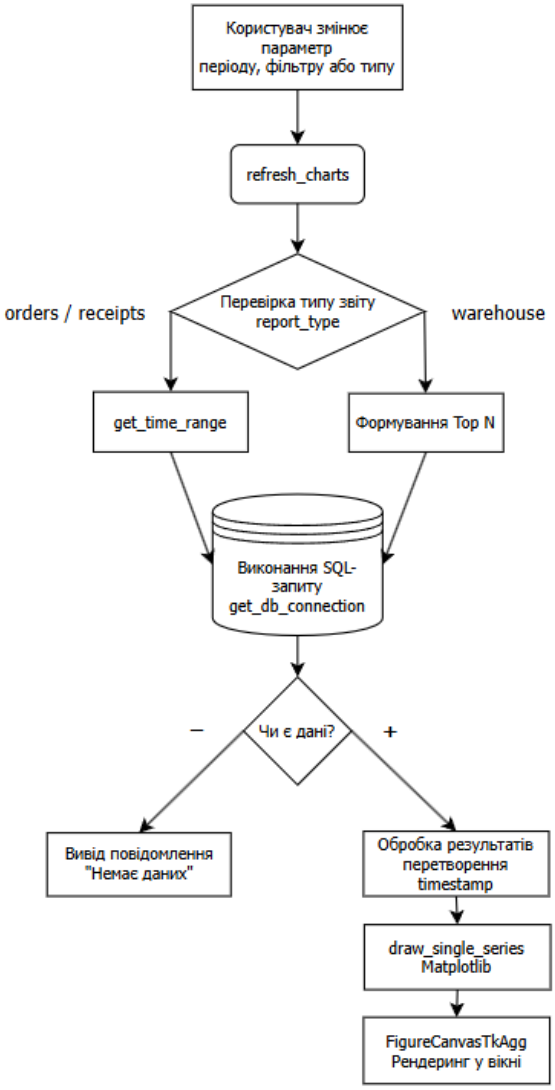

Рис. 3.8 – Схема побудови графіків

Динамічні елементи – кнопка “Обрати товари” для порівняння та панель Top N – з’являються лише для відповідних видів аналізу.

При активізації режиму порівняння стає доступною кнопка, яка відкриває модальне вікно open_product_selector(). У цьому вікні реалізовано пошук та список з чекбоксами для кожного товару. Після підтвердження вибору список ідентифікаторів обраних товарів зберігається в self.selected_product_ids, і графік оновлюється.

Метод draw_single_series() відповідає за візуалізацію. Він приймає списки x і y, тип графіка за замовчуванням та формат підписів. Логіка автоматичного вибору типу: для часових рядів обирається лінійна діаграма,

для категоріальних даних з малою кількістю елементів ≤5 – кругова, для більшої – горизонтальна стовпчикова. Кругова діаграма при кількості категорій >10 об’єднує решту в сектор “Інше”. Стовпчасті та горизонтальні діаграми отримують підписи значень. Лінійні діаграми – маркери, якщо точок ≤ 60 та заливку під кривою. Колір графіка можна змінювати – він передається як параметр у виклики, а також використовується для генерації відтінків у круговій діаграмі.

У результаті розроблено гнучкий аналітичний інструмент, який дозволяє користувачеві будувати різноманітні графіки без необхідності писати SQL-запити вручну.

3.6. Допоміжні утиліти для роботи з обчисленнями та датами

У процесі роботи з системою, особливо під час проведення інвентаризацій, розрахунку знижок у замовленнях або планування дат надходжень, у користувачів регулярно виникає потреба у виконанні базових арифметичних обчислень та роботі з датами. Щоб уникнути необхідності перемикатися між сторонніми програмами операційної системи, до системи було розроблено власні допоміжні утиліти.

Доступ до цих інструментів реалізовано через відповідні кнопки на панелі інструментів, що робить їх доступними з будь-якого основного модуля системи, таких як номенклатура, замовлення, находження чи користувачі. Обидві утиліти реалізовані як незалежні класи, успадковані від базового CPopupWindow та відповідно мають модальну поведінку.

3.6.1. Модуль програмного калькулятора

Клас CalculatorWindow реалізує функціонал класичного калькулятора. Інтерфейс побудовано з використанням сіткового компонування, додано кнопки базових математичних операцій та функціональні клавіші: очищення,

видалення останнього символу, обчислення. Реалізовано можливість швидкого введення даних без використання миші, в класі додано прив'язку клавіатурних подій до головного вікна утиліти. Обчислення математичних виразів виконується за допомогою вбудованої функції eval() із попередньою підстановкою символів. Для безпеки використовується словник safe_math_dict, який містить лише дозволені математичні функції та константи й унеможливлює виконання шкідливого коду. Таким чином обчислення виконуються із попередньою валідацією або використанням блоку try...except. Це гарантує, що спроба ділення на нуль або введення некоректного синтаксису не призведе до аварійного завершення всієї програми, а лише виведе повідомлення про помилку.

3.6.2. Календар для вибору дат з копіюванням у буфер обміну

Для зручного планування складських операцій реалізовано клас CalendarWindow. Логіка генерації календарної сітки базується на стандартних бібліотеках Python – calendar та datetime. При ініціалізації поточний рік і місяць визначаються за системною датою. Інтерфейс календаря складається з двох основних частин: панель навігації, яка містить назву поточного місяця та року, а також кнопки-стрілки для перегортання місяців уперед та назад. Кожен день представлений кнопкою CTkButton.

Перший рядок містить скорочені назви днів тижня (понеділок – неділя). Дні, що належать до попереднього або наступного місяця, візуально відображаються сірим кольором, і хоча кнопки залишаються активними, їхній текст затемнюється, а команда копіювання дати виконується, але такі дні не виділяються як вихідні. Поточний системний день автоматично виділяється акцентним кольором. При натисканні на будь-яку дату вона форматується за допомогою date_obj.strftime(“%d.%m.%Y”), копіюється в буфер обміну, а заголовок вікна тимчасово, на 1.5 секунди, змінюється на підтвердження “Скопійовано: дата”, після чого повертається до початкового заголовка.

Висновки до розділу 3

У третьому розділі здійснено програмну реалізацію базової системи складського обліку. Розроблено реляційну базу даних SQLite з дев'яти таблиць. Структура БД є ідентичною до схеми мобільного клієнта, що дозволяє обмінюватися файлами .db без додаткової конвертації. Метод smart_merge з використанням команди ATTACH DATABASE забезпечує автоматичне злиття даних.

Модуль database.py реалізує шар доступу до даних, надаючи CRUD-операції, функції адміністрування та підсистему резервного копіювання. Паролі користувачів зберігаються у вигляді хешів SHA-256, що унеможливлює їх відновлення навіть при компрометації БД.

Графічний інтерфейс побудовано на бібліотеці CustomTkinter. Головне вікно DashboardApp містить бічну панель для навігації між модулями та робочу область. Клас BaseModule уніфікує створення адаптивних панелей інструментів, заголовків та таблиць. Механізм розширеного пошуку SearchLogicMixin підтримує пошук по всьому набору даних. Завдяки модулю translations.py реалізовано динамічне перемикання мовних налаштувань.

Основні модулі обліку забезпечують повний цикл роботи: warehouse.py – управління номенклатурою та складами; documents.py – створення накладних; inventory.py – перегляд історії інвентаризацій; users.py – адміністрування користувачів. Модуль analytics.py надає інтерактивні графіки та звіти з використанням matplotlib, підтримкою фільтрації за періодом, типом графіка та вибором товарів для порівняння [45].

РОЗДІЛ 4. ІНТЕГРАЦІЯ МЕТОДІВ ШТУЧНОГО ІНТЕЛЕКТУ

4.1. Інтеграція машинного навчання для прогнозування попиту

Головною особливістю розроблюваної системи є модуль прогнозування майбутніх продажів на основі архітектури Temporal Fusion Transformer (TFT). Для об'єктивного оцінювання здатності обраної моделі виявляти сталі закономірності, наприклад, сезонність та локальні тренди необхідний репрезентативний масив історичних даних. З метою валідації алгоритму на етапі розробки було використано відкритий датасет змагання Kaggle “Store Item Demand Forecasting Challenge”, що містить п'ять років щоденних продажів 50 товарів у 10 магазинах, загалом 913 000 записів [50]. Набір даних є репрезентативним для задач роздрібної торгівлі, адже містить яскраво виражену сезонність і різноманітні патерни попиту.

4.1.1. Підготовка даних та конструювання ознак для навчання нейромережі

Датасет “Store Item Demand Forecasting Challenge” містить файл train.csv із такими атрибутами: date – дата у форматі YYYY-MM-DD, store – ідентифікатор магазину, item – ідентифікатор товару та sales – кількість проданих одиниць. Для проведення експериментів було сформовано вибірку, що включає дані одного магазину та десяти товарів за весь п'ятирічний період спостережень.

Такий обсяг даних, понад 18 250 записів, є статистично достатнім для навчання моделей прогнозування часових рядів. Він повністю зберігає різноманітність динаміки попиту для різних категорій товарів, але при цьому дозволяє уникнути надмірних витрат обчислювальних ресурсів під час багаторазових експериментів із гіперпараметрами. Усі експерименти

проводилися на цьому фіксованому наборі, що гарантує рівні умови при порівнянні моделей.

Попередня обробка даних включала такі етапи:

1. Фільтрація для виокремлення цільового кластера – умова store == 1 та item <= 10 за допомогою бібліотеки pandas [40].
2. Оскільки дані вже мають денну роздільність, для кожної пари “дата-товар” кількість проданих одиниць підсумовувалася для гарантії відсутності дублікатів та цілісності ряду.
3. Створення суцільного часового ряду для кожного товару. Відсутні значення sales у календарі замінювалися нулем.
4. Додано лінійний індекс часу time_idx, порядковий номер дня та циклічні ознаки. Щоб уникнути розривів на межі циклів, наприклад, між неділею та понеділком, використано тригонометричні перетворення:

- Для днів тижня:

$$day\_sin = \sin\left(2\pi * \frac{day}{7}\right) \quad (4.1)$$

$$day\_cos = cos(2\pi * \frac{day}{7}) \quad (4.2)$$

- Для місяців:

$$month_{sin} = \sin\left(2\pi * \frac{month}{12}\right) \quad (4.3)$$

$$month_{cos} = \cos\left(2\pi * \frac{month}{12}\right) \quad (4.4)$$

Приклад отриманих даних після виконання всіх кроків наведено в табл. 4.1.

Таблиця 4.1

Приклад даних після попередньої обробки

date	sku	name	quantity	tim_idx	day_sin	day_cos	month_sin	month_cos
2013-01-01	KGL-ITEM-001	Kaggle Product #1	13.0	0	0.781831	0.62349	0.5	0.866025
2013-01-01	KGL-ITEM-002	Kaggle Product #2	33.0	0	0.781831	0.62349	0.5	0.866025
2013-01-01	KGL-ITEM-003	Kaggle Product #3	15.0	0	0.781831	0.62349	0.5	0.866025
2013-01-01	KGL-ITEM-004	Kaggle Product #4	10.0	0	0.781831	0.62349	0.5	0.866025
2013-01-01	KGL-ITEM-005	Kaggle Product #5	11.0	0	0.781831	0.62349	0.5	0.866025

Така підготовка даних забезпечує формування об'єкта TimeSeriesDataSet, необхідного для навчання Temporal Fusion Transformer.

4.1.2. Методологія експериментального дослідження та метрики якості прогнозу

Для оцінки здатності моделі до узагальнення застосовано хронологічне розбиття даних. Усі спостереження до дня T – max_prediction_length використовуються для навчання, а останні max_prediction_length днів – для тестування. Такий підхід відтворює реальну ситуацію, коли модель будує

прогноз на основі минулих даних, не знаючи майбутніх значень. Випадкове розбиття 80/20 не застосовується, оскільки воно порушує часову структуру рядів і призводить до використання майбутньої інформації під час навчання, що дає нереалістично оптимістичну оцінку якості.

Оскільки горизонт планування на складі становить 14 днів, тестова вибірка була обмежена останніми 14 днями історичного ряду. Хоча у відсотковому відношенні це становить невелику частину загального масиву даних, такий підхід є методологічно коректним: він точно імітує реальні умови експлуатації системи, де модель прогнозує короткостроковий попит, спираючись на всю доступну історію.

Для визначення оптимальної довжини історії max_encoder_length та підтвердження наявності сезонних закономірностей у даних було проведено автокореляційний аналіз. На рисунку 4.1 наведено графік автокореляційної функції для щоденних продажів усереднених по всіх товарах обраного магазину із глибиною огляду 365 днів (рис. 4.1).

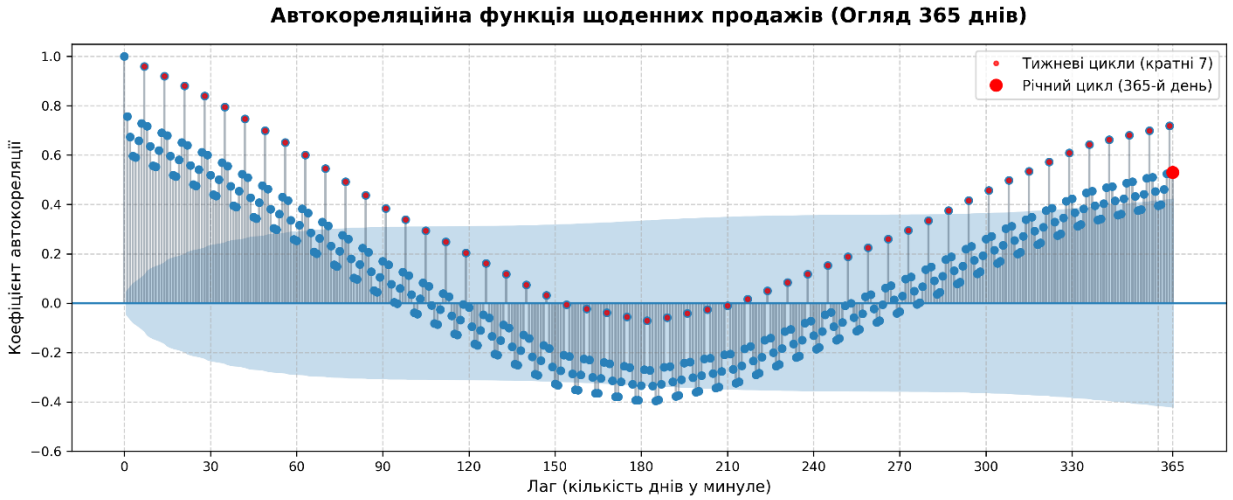

Рис. 4.1 – Автокореляційна функція щоденних продажів

Графік згенеровано за допомогою бібліотеки statsmodels. Він демонструє суперпозицію двох часових закономірностей: жорсткої тижневої циклічності – піки на лагах, кратних 7 та глобальної річної сезонності –

поступове відновлення кореляції до 365-го дня. З графіка видно, що навіть на глибині 60 днів коефіцієнт автокореляції утримується на статистично значущому рівні, близько 0.6. Це свідчить про наявність сильної довгострокової пам'яті у часовому ряді [51]. Відповідно маємо необхідність використання достатньо широкого історичного вікна max_encoder_length для моделі.

Базова конфігурація моделі TFT відповідає параметрам, рекомендованим у документації бібліотеки pytorch_forecasting [48]:

- max_encoder_length = 30 – довжина історії 30 днів;
- max_prediction_length = 14 – горизонт прогнозу 14 днів;
- hidden_size = 64, attention_head_size = 4, dropout = 0.15;
- функція втрат – QuantileLoss, що дозволяє отримувати інтервальні прогнози (10%, 50%, 90% квантилі).

Програмна реалізація навчання моделі – метод run_tft_pipeline(), наведена в Додатку Г. Як базову модель для порівняння обрано просту ковзну середню за 7 днів. Цей метод є класичним бенчмарком у задачах прогнозування часових рядів, адже він не потребує навчання, легко реалізується і слугує нижньою межею якості, яку має перевершити складна модель. Якщо TFT не зможе суттєво покращити SMA, її практичне застосування буде невиправданим.

Для кількісного порівняння якості прогнозів використано три показники, розрахунок яких вбудовано в програмний код. Результати усереднюються за всіма товарами:

- WAPE (Weighted Absolute Percentage Error) – зважена абсолютна відсоткова помилка, стійка до нульових значень.

$$WAPE = \left(\frac{\sum_t |y_t - \hat{y}_t|}{\sum_t |y_t|}\right) * 100\% \qquad (4.5)$$

де y_t – фактичне значення продажів, $\hat{y}_t$ – прогнозоване значення, $|.|$ – абсолютна величина, $\sum$ – сумування по всіх спостереженнях тестової

вибірки. У коді реалізовано усереднення по кожному часовому ряду окремо.

- MAE (Mean Absolute Error) – середня абсолютна помилка в одиницях товару на день:

$$MAE = \frac{1}{n}\sum|y_t - \hat{y}_t| \quad (4.6)$$

де n – кількість спостережень у тестовій вибірці. Ця метрика дозволяє оцінити величину помилки в натуральному масштабі.

- Bias – середнє систематичне відхилення прогнозу від факту:

$$Bias = \frac{1}{n}\sum(y_t - \hat{y}_t), \quad (4.7)$$

додатне значення вказує на схильність до надлишку, від'ємне – до дефіциту. Значення, близьке до нуля, свідчить про збалансованість.

Для статистичної значущості результати усереднюються за всіма товарами, код обчислює mean для WAPE, MAE, Bias по групах sku [17; 2]. Кожен експеримент повторюється тричі для виключення випадкових коливань, у звіт наводяться середні значення.

План експериментів передбачає:

1. Базовий експеримент – порівняння TFT та SMA при стандартних налаштуваннях, результати наведено в табл. 4.2.
2. Дослідження впливу довжини історії – зміна max_encoder_length (14, 30, 60, 90, 180 днів) при фіксованому горизонті 14 днів, табл. 4.3.
3. Дослідження впливу горизонту прогнозу – зміна max_prediction_length (7, 14, 30 днів) при фіксованій історії 60 днів, табл. 4.4).

Таблиця 4.2

Порівняння TFT та SMA (базовий експеримент)

Модель	WAPE, %	MAE, од./день	Bias
TFT	16.40	5.35	+0.06
SMA	20.87	6.8	+1.5

Результати базового експерименту підтверджують доцільність використання глибокого навчання: архітектура TFT перевершує статистичний підхід більш ніж на 4 відсоткові пункти за метрикою WAPE, формуючи при цьому збалансований прогноз, адже Bias наближений до нуля.

Таблиця 4.3

Вплив довжини історії на WAPE (при горизонті на 14 днів)

Max_encoder_length	WAPE (TFT), %	MAE, од/день	Bias
14	16.17	5.31	-0.61
30	17.50	5.80	+0.91
60	15.69	5.18	-0.09
90	17.42	5.64	+1.02
180	19.88	6.52	+0.16

Аналіз отриманих результатів демонструє нелінійну залежність, U-подібну криву помилки точності прогнозу від обсягу вхідної історії. Збільшення вікна спостереження до 60 днів дозволило моделі охопити необхідний контекст, що підтверджується високим коефіцієнтом на графіку ACF і мінімізувати показник WAPE до 15.69% та MAE до 5.18 од./день.

Однак подальше розширення історії до 90 і 180 днів призвело до стрімкої деградації якості прогнозу, WAPE зріс до 19.88%. Це явище пояснюється ефектом розсіяння уваги та перенавчанням на інформаційному шумі. Тобто модель починає надавати надмірну вагу нерелевантним подіям піврічної давності, ігноруючи поточний локальний мікро-тренд. Таким чином, експериментально доведено, що оптимальною довжиною історичного вікна є 60 днів.

На заключному етапі експериментів було оцінено вплив горизонту планування max_prediction_length на точність передбачень. Для забезпечення чистоти експерименту вікно спостереження max_encoder_length було

зафіксовано на експериментально підтвердженому оптимальному рівні – 60 днів.

Таблиця 4.4

Вплив горизонту прогнозу на метрики точності (при історії 60 днів)

Max_prediction_length	WAPE (TFT), %	WAPE (SMA), %	MAE, од/день	Bias
7	17.03	20.51	5.55	-1.50
14	17.97	20.87	5.92	0
30	15.74	37.68	5.27	-1.40

Аналіз даних табл. 4.4 виявляє нестандартну, але логічну для ритейлу динаміку. У класичному прогнозуванні збільшення горизонту зазвичай призводить до експоненціального накопичення помилки. Однак у даному випадку метрики абсолютної похибки WAPE 15.74% та MAE 5.27 виявилися найкращими саме на 30-денному горизонті, де перевага TFT над статистичною базовою моделлю SMA досягла 58,2%. Такий результат пояснюється тим, що на довшому проміжку часу локальні щоденні флуктуації та шум взаємокомпенсуються, дозволяючи алгоритму точніше відтворювати глобальний місячний макро-тренд.

Водночас, результати свідчать, що прогнози на 7 та 30 днів мають виражену схильність до заниження попиту, Bias = -1.50 та -1.40 відповідно. У реальних умовах таке систематичне заниження є небажаним, оскільки призводить до дефіциту товару та втраченого прибутку.

Горизонт у 14 днів, незважаючи на дещо вищі показники абсолютної похибки WAPE 17.97%, демонструє ідеальну збалансованість, Bias наближений до нуля. Це означає, що модель рівномірно розподіляє помилки в обидва боки, мінімізуючи як ризики надмірного затоварення, так і дефіциту.

4.1.3. Інтерпретація результатів: важливість факторів та механізм уваги

Однією з головних проблем впровадження нейромереж у корпоративні системи є їхня непрозорість – проблема “чорної скриньки”. Користувачі, тобто менеджери та аналітики часто не довіряють прогнозам, якщо не розуміють логіки їх формування. Архітектура Temporal Fusion Transformer вирішує цю проблему, надаючи вбудовані механізми інтерпретації прийнятих рішень [25; 30]. Це досягається завдяки вбудованим механізмам оцінки важливості факторів та візуалізації уваги. У рамках даного дослідження ці механізми було використано для реалізації програмного модуля, що вирішує проблему поглибленого аналізу. Він дозволяє візуалізувати внутрішню логіку для кожного окремого товару. Інтерпретація базується на двох ключових аспектах: локальній важливості факторів та динаміці механізму уваги.

Локальна важливість факторів демонструє, які саме вхідні змінні: числові значення минулих продажів, циклічні ознаки дня тижня, місяця або лінійний індекс часу здійснили найбільший вплив на формування прогнозу для конкретного товару в поточний момент часу. Алгоритм аналізує ваги змінних в енкодері мережі та нормалізує їх у відсотковому співвідношенні.

Як видно з рисунка 4.2, система дозволяє кількісно оцінити внесок кожної ознаки. Наприклад, висока вага змінних day_sin та day_cos свідчить про те, що для даного товару визначальним фактором був би специфічний день тижня. В даному випадку домінування quantity вказує на залежність від різких стрибків або падінь обсягів продажу в недавньому минулому. Це дає змогу аналітику зрозуміти природу попиту на конкретну номенклатуру без додаткових статистичних досліджень.

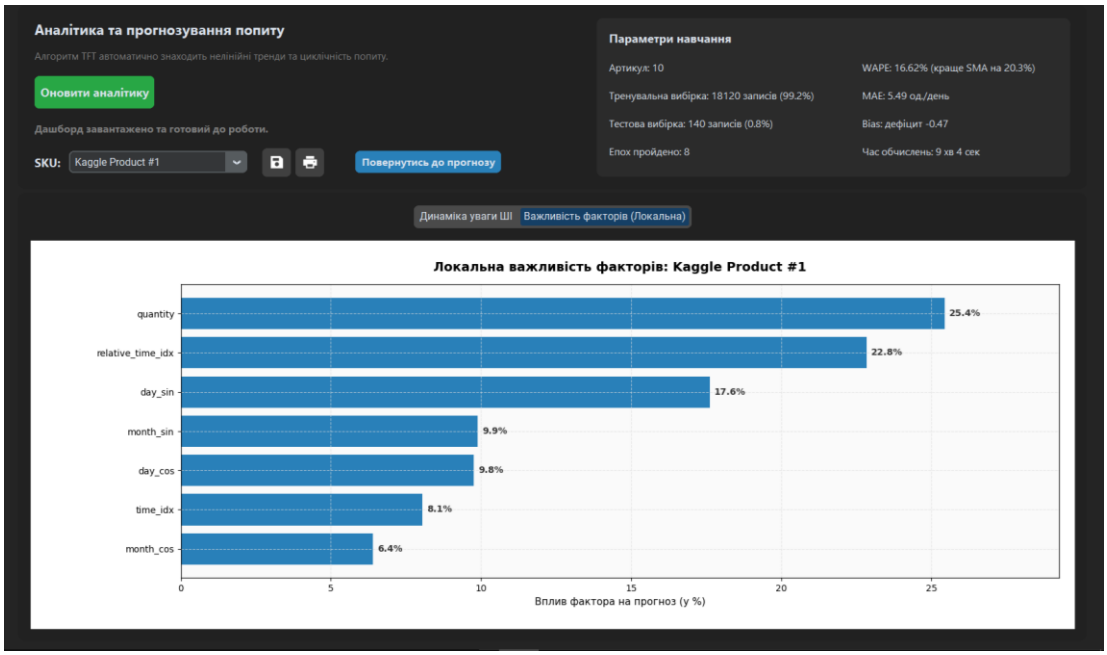

Рис. 4.2 – Локальна важливість факторів для конкретного товару

На відміну від рекурентних мереж LSTM, які стискають усю історію в один прихований стан, TFT використовує механізм Multi-Head Attention, що дозволяє моделі динамічно озиратися назад і обирати найрелевантніші дні з вікна історії, у нашому випадку – з останніх 60 днів [27]. На рисунку 4.3 показано графік агрегованої історичної уваги для того самого товару.

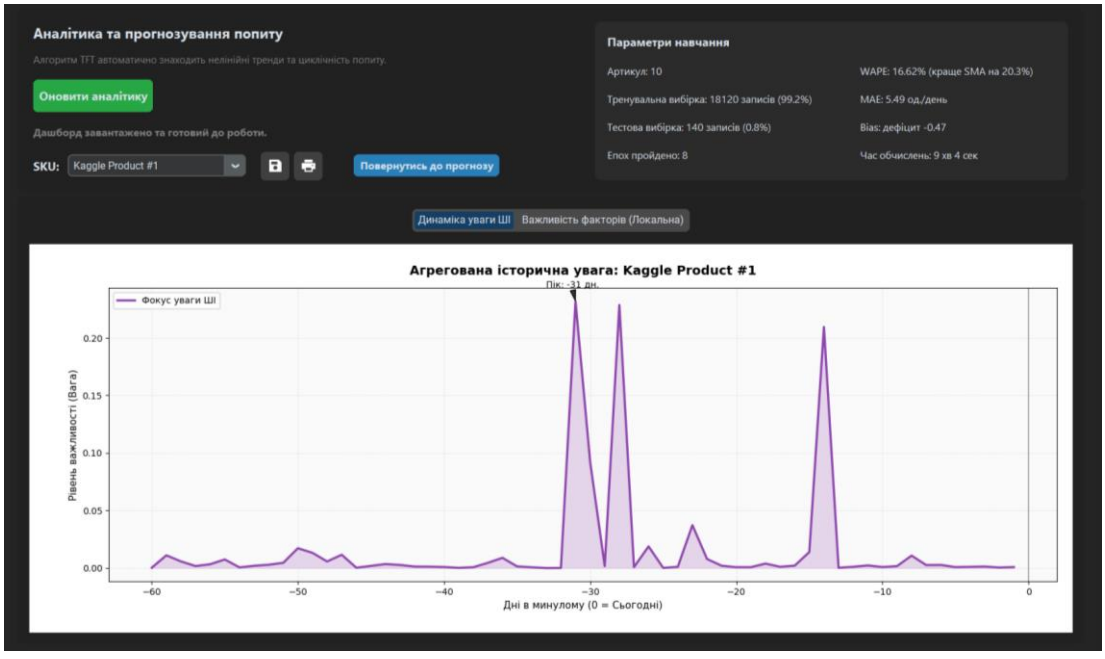

Рис. 4.3 – Агрегована історична увага

Графік уваги візуалізує розподіл ваг механізму вздовж осі часу в минуле. Пікові значення на цьому графіку вказують на дні, інформація з яких виявилася найкориснішою для побудови прогнозу. Наявність циклічних піків уваги, наприклад, кожні 7 днів підтверджує, що нейромережа самостійно, без жорстко заданих правил, виявила тижневу сезонність і звертає увагу на аналогічні дні в минулому. Якщо увага фокусується виключно на останніх кількох днях, це свідчить про наявність короткострокового мікро-тренду, який перекриває вплив сезонності.

Отримані результати важливості факторів та уваги мають безпосереднє практичне значення. По-перше, адміністратор складу може переконатися, що модель дійсно спирається на об'єктивні закономірності, наприклад, тижневі та місячні цикли, а не на випадкові кореляції. По-друге, якщо зміниться структура попиту, наприклад, сезонність стане слабшою, зміниться й графік важливості факторів – це може слугувати сигналом для перенавчання моделі. По-третє, пояснюваність прогнозів підвищує довіру до системи з боку користувачів.

4.2. Розробка NLP-асистента на основі хмарного великої мовної моделі

Для забезпечення зручного доступу до даних системи було розроблено модуль nlp_assistant.py, який реалізує функціонал чат-бота з інтеграцією великої мовної моделі Gemini. Асистент перетворює запити користувача природною мовою, українською або англійською, на виконувані SQL-запити SELECT до бази даних та відображає результати у вигляді текстової таблиці. Завдяки цьому користувач може отримувати відповідь на такі питання як: “Скільки товарів продано за останній тиждень?”, “Покажи залишки конкретного товару” без участі програміста.

Модуль складається з трьох основних частин:

- графічний інтерфейс чату – бульбашки діалогу, поле вводу, кнопки;
- голосове введення, на основі бібліотеки speech_recognition

- механізм генерації SQL через API Gemini.

Генерація SQL-запитів через Gemini відбувається за моделлю gemini-2.5-flash через API [31]. Користувацький запит надсилається до моделі разом із промптом, що містить опис схеми бази даних, таблиці products, sales_orders, order_items та жорсткі правила: генерувати тільки SELECT-запити для запобігання зміні даних; відповідати виключно у форматі JSON з полями “message”, тобто коментарем мовою користувача та “sql”. Лістинг 4.1 демонструє структуру промпту та обробку відповіді.

Лістинг 4.1

Промпт для обробки запитів користувача до Gemini

```
"""You are an NLP gateway for the Wareflow inventory management system.
Convert natural language queries into valid SQLite SELECT queries.
DATABASE SCHEMA:
- `products` (id, sku, name, quantity, price, purchase_price, warehouse_id)
- `sales_orders` (db_id, date, is_posted, is_cancelled)
- `order_items` (id, order_id, product_id, quantity)
RULES:
1. ONLY SELECT queries.
2. MUST be strictly JSON.
3. "message" field in {target_lang}.
RESPONSE FORMAT:
{{"message": "commentary", "sql": "SELECT..."}}
USER QUERY: "{user_text}"""
```

Після отримання SQL-запиту він виконується через стандартне з'єднання SQLite, тобто функцію database.get_db_connection(). Результат форматується у вигляді текстової таблиці: перший рядок – заголовки колонок у верхньому регістрі, другий – роздільник, далі рядки з даними, розділені символами. Цей текст виводиться в чат як окрема бульбашка з типом

is_sql_result. Якщо згенерований запит не є SELECT, а, наприклад, UPDATE або DELETE, він блокується.

Додатково за допомогою бібліотеки speech_recognition активується кнопка мікрофона [52]. При натисканні запускається _listen_thread(), який використовує sr.Recognizer() для захоплення аудіо з мікрофона. Далі виконує розпізнавання через Google Speech Recognition. Мова визначається за поточною мовою інтерфейсу. Якщо розпізнавання не вдалося через тайм-аут, нерозбірливий голос, помилку мережі – виводиться відповідне повідомлення.

Таким чином, NLP-асистент значно знижує поріг входу для роботи з системою, дозволяючи отримувати аналітику без знання SQL та без необхідності відкривати окремі модулі або сортувати дані.

Висновки до розділу 4

У четвертому розділі описано процес інтеграції двох ключових модулів: системи прогнозування попиту на основі Temporal Fusion Transformer та NLP-асистента з використанням великої мовної моделі Gemini.

Для TFT реалізовано повний цикл підготовки даних: агрегація продажів, заповнення пропусків, генерування циклічних ознак day_sin/cos, month_sin/cos, навчання моделі на відкритому датасеті Kaggle. Експериментально підтверджено, що TFT перевершує базову модель ковзної середньої SMA. Визначено оптимальну довжину історії. Аналіз впливу горизонту прогнозу показав, що 14 днів забезпечує найкращий баланс між точністю та нульовим зміщенням. Механізми інтерпретації, важливість факторів та агрегована історична увага, дозволяють візуалізувати, які ознаки та історичні періоди впливають на прогноз, долаючи проблему “чорної скриньки”.

NLP-асистент на базі Gemini забезпечує перетворення природномовних запитів у SQL-запити SELECT з обов’язковим дотриманням формату JSON,

виконання їх у базі даних та повернення результатів у вигляді текстових таблиць. Історія діалогу зберігається в межах сесії та може бути очищена.

Обидва модулі додано в графічний інтерфейс. Таким чином, розроблювана система отримала інструменти підтримки прийняття рішень на основі методів глибокого навчання та ШІ-асистента.

РОЗДІЛ 5. ТЕСТУВАННЯ ПРАЦЕЗДАТНОСТІ ТА АНАЛІЗ ЕФЕКТИВНОСТІ СИСТЕМИ

Завершальним етапом розробки системи є її комплексне тестування. Оскільки система призначена для управління складом та прийняття управлінських рішень, важливою є перевірка її надійності, стійкості до некоректних дій користувача та точності обробки даних.

5.1. Перевірка коректності роботи базових облікових операцій та обробка помилок

Перевірка логіки обліку системи проводилася на основі методології тестування “чорного ящика”. Головною метою цього етапу була валідація алгоритмів обробки даних та перевірка реакції системи на нестандартні або помилкові дії оператора.

Найбільш вразливим місцем будь-якої облікової системи є проведення видаткових документів. Під час тестування було зімітовано спробу проведення замовлення на продаж, де кількість товару в кошику перевищувала фактичний залишок на обраному складі. Система успішно перехопила цю дію: алгоритм заблокував транзакцію та вивів модальне вікно (рис. 5.1). Це доводить, що виникнення від'ємних залишків у базі даних унеможливлено.

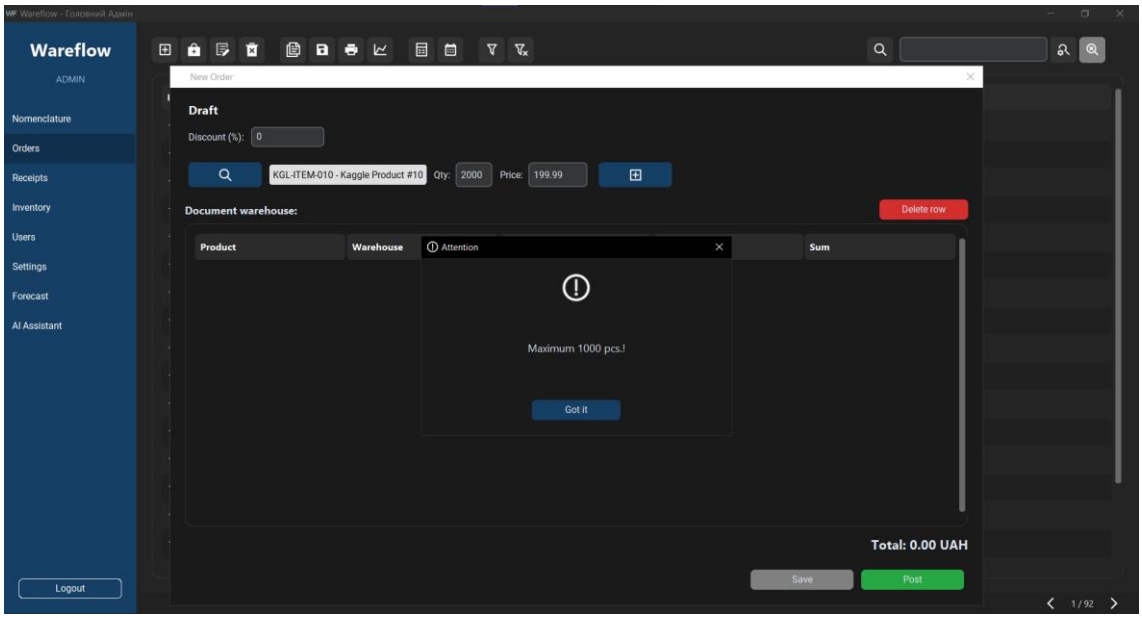

Рис. 5.1 – Вікно помилки системи при перевищенні доступності товарів

Найчастішою помилкою користувачів є спроба зберегти форму з незаповненими даними. Для перевірки захисних механізмів було зімітовано спробу створення нового користувача без введення пароля, а також спробу створення товару без вказівки назви. У всіх випадках система також вивела відповідне модальне вікно.

Додатково перевірено такі сценарії:

- спроба видалити склад, на якому є товари – операція блокується з поясненням, що потрібно спочатку перемістити або видалити товари.
- спроба додати користувача з логіном, що вже існує – повертає відповідне вікно попередження.
- при проведенні документа замовлення або надходження всі зміни зменшення/збільшення залишків та запис у order_items/supply_items виконуються в одній транзакції. Якщо будь-яка частина операції завершується помилкою, наприклад, недостатньо товару, жодні зміни не фіксуються.
- ручне створення бекапу через модуль налаштувань успішно копіює файл wareflow.db у папку backups із часовою міткою, а старі копії – понад 30 автоматично видаляються.

- автоматичний бекап тестувався з інтервалом 1 хвилина – коректно запускає потік і створює копії.
- операція очищення історії операцій видаляє всі документи та інвентаризації, але залишає довідники, товари, склади, а кількість товарів обнуляється.
- повне скидання додатково видаляє всі товари та склади, повертаючи БД до стану, як після першого запуску. Перед кожною з цих операцій автоматично створюється резервна копія, що дозволяє відновити дані в разі помилки.

Синхронізація баз даних додатка на операційних системах Android та Windows відбувається ідентично з обох сторін. Для тестування на мобільному додатку було створено новий тестовий товар та завантажено нову змінену базу даних до системи на ПК (рис. 5.2). Синхронізація відбулась за декілька секунд, проте додаток на Android імпортує дані помітно довше.

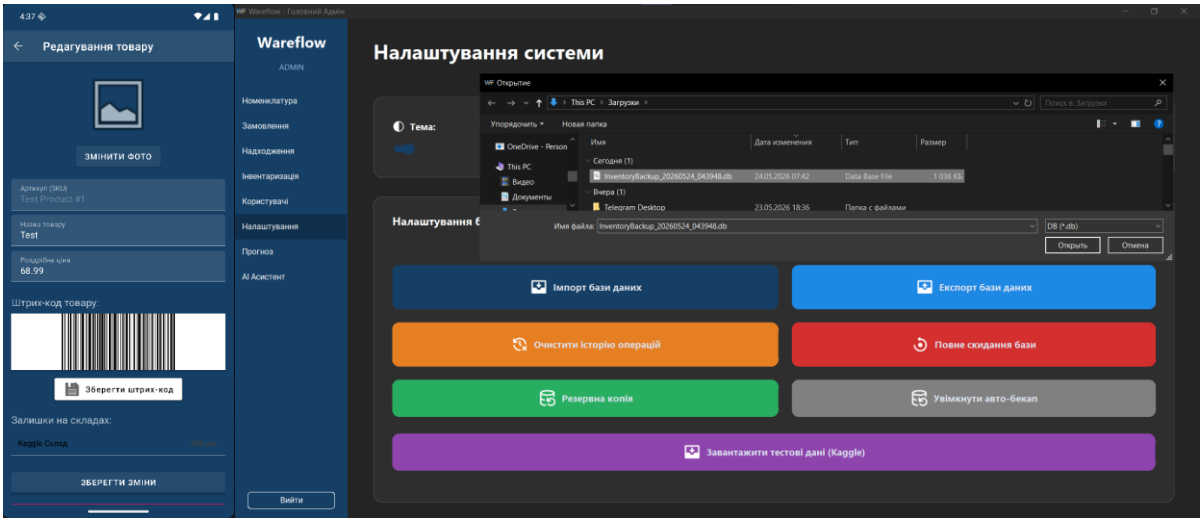

Рис. 5.2 – Завантаження експортованої бази даних з Android-додатку

На рисунку 5.3 показано головне вікно програми після входу адміністратора. Бічна панель має фіксовану ширину, робоча область займає решту простору. Вікно автоматично розгортається на весь екран.

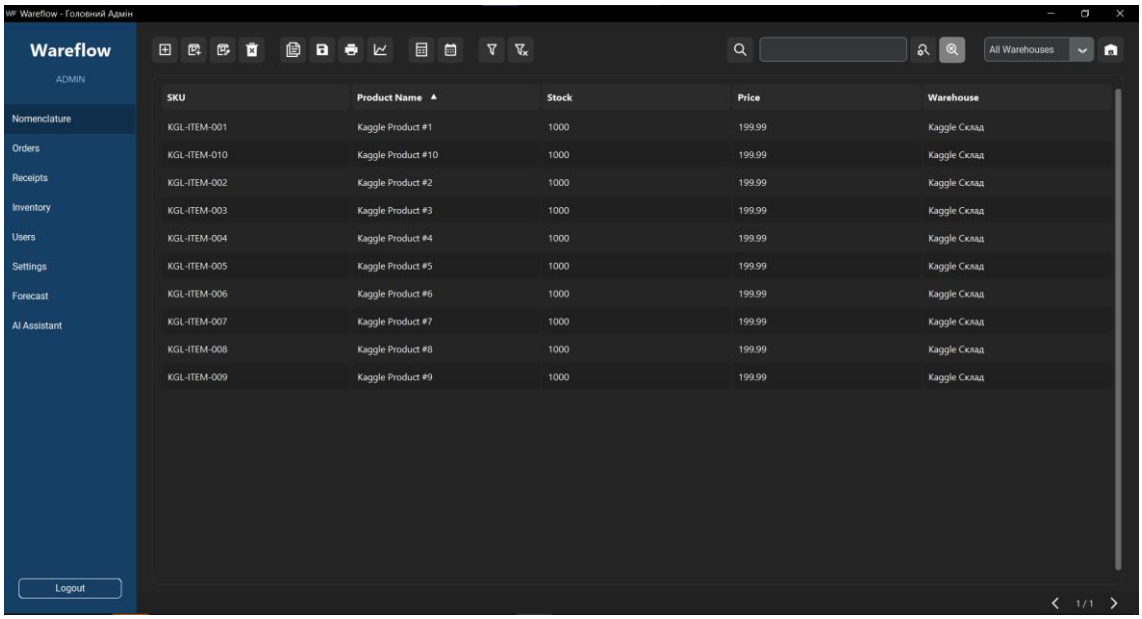

Рис. 5.3 – Головне вікно системи Wareflow

Перемикання між світлою та темною темами (рис. 5.4) відбувається миттєво без перезапуску – змінюються кольори всіх віджетів, іконки автоматично підвантажуються з папки icons з суфіксами _light.png або _dark.png. При зміні мови, всі тексти, кнопки, заголовки, повідомлення оновлюються. Вибір мови зберігається у файлі config.json, після перезапуску програма відкривається з обраною мовою та темою.

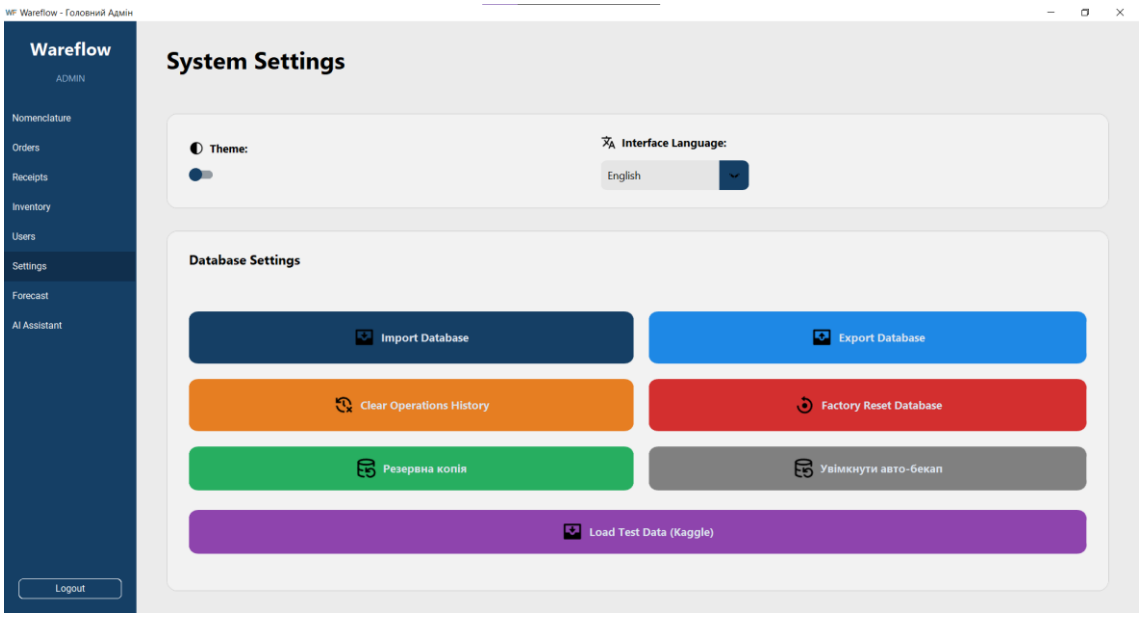

Рис. 5.4 – Розділ налаштувань системи з увімкненою світлою темою

У модулі номенклатури швидкий пошук за артикулом або назвою показує збіги в реальному часі. Розширений пошук (рис. 5.5) дозволяє обрати режим збігу, напрямок, врахувати регістр тощо. Знайдені елементи підсвічуються, а панель статусу відображає лічильник, наприклад, “Знайдено: “Товар А” (3/12)”. Пошук працює по всіх сторінках таблиці.

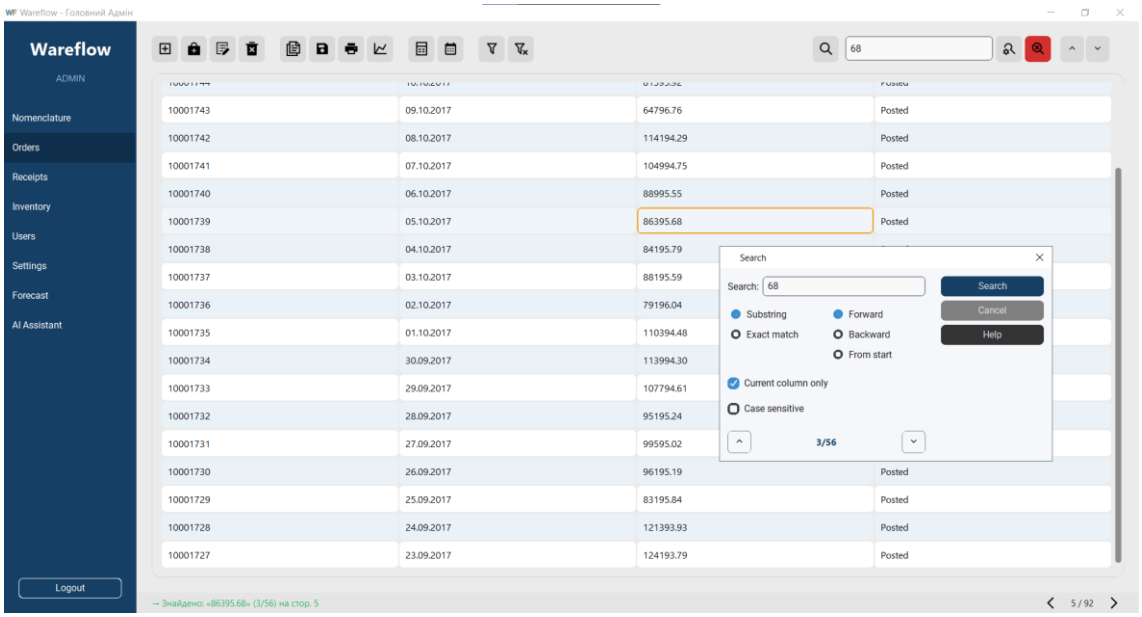

Рис. 5.5 – Вікно розширеного пошуку з налаштуваннями

У модулі аналітики перевірено побудову графіків для всіх трьох типів звітів: склади, замовлення, надходження. Зміна періоду коректно фільтрує дані. Вибір типу графіка – візуалізація оновлюється. Наявний автоматичний вибір: для часових рядів – лінійна, для категорій ≤5 – кругова працює згідно з логікою. Зміна кольору графіка через діалог colorchooser застосовується до всіх типів діаграм, для кругової генеруються відтінки (рис. 5.6).

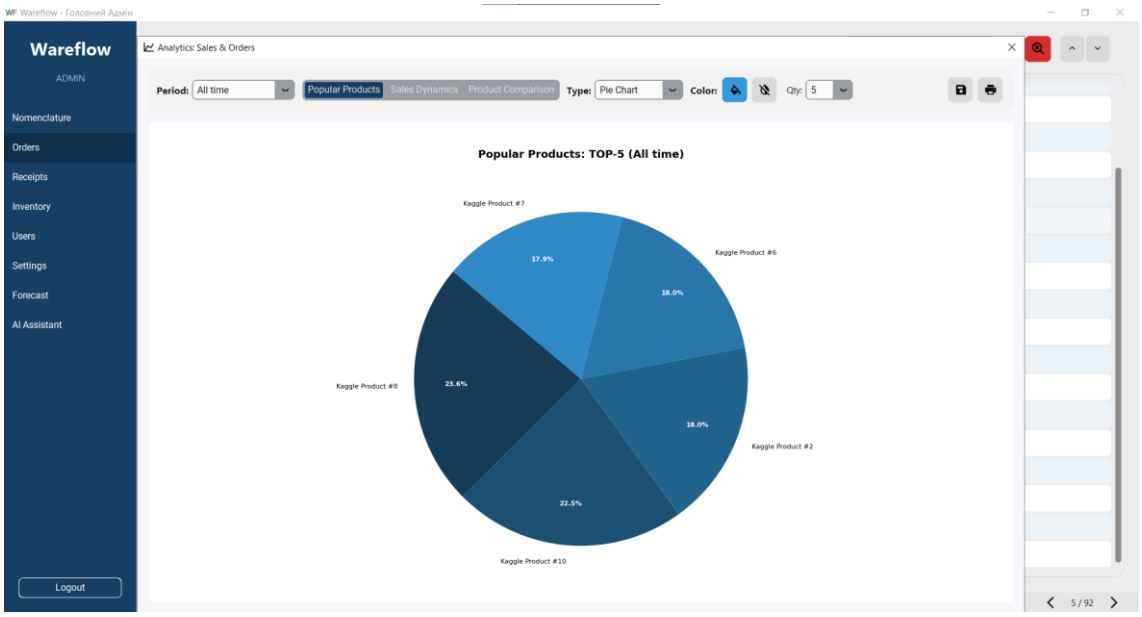

Рис. 5.6 – Модуль аналітики зі звітом кругової діаграми найвищого попиту

Наявний також режим порівняння товарів: вибір кількох товарів через вікно з пошуком та побудова суміщеного графіку – лінійного або стовпчикового, легенда якого відображає назви товарів. Доступний експорт графіка у різних форматах, зокрема PNG, JPG, PDF та друк через генерацію HTML – функціонує без помилок.

5.2. Оцінка роботи інтелектуальних модулів: прогнозування, діалогового асистента та аналітики

Модуль прогнозування стабільно запускає навчання на даних Kaggle. Прогрес-бар оновлюється в реальному часі, статусна панель показує поточний етап. Після завершення метрики WAPE, MAE, Bias відображаються в правій панелі (рис. 5.7). При перемиканні в режим “Як думає ШІ” коректно візуалізуються важливість факторів та увага. Продуктивність: навчання на всіх даних займає близько 2-5 хвилин на CPU; завдяки виконанню в окремому потоці інтерфейс не блокується.

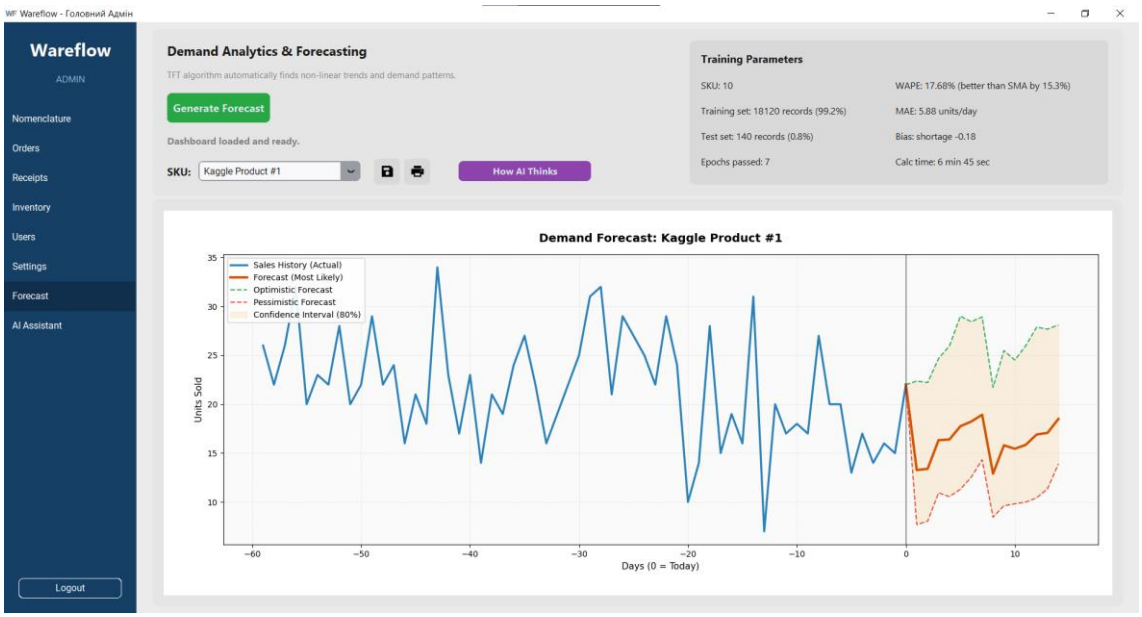

Рис. 5.7 – Вікно прогнозування TFT з графіком та панеллю метрик

Асистент коректно перетворює природномовні запити на SQL. Наприклад, на запит “5 товарів з найвищим попитом за увесь час” генерується SELECT p.name, SUM(oi.quantity) as s_qty ... ORDER BY s_qty DESC LIMIT 5, результат виводиться у вигляді текстової таблиці. Спроба виконати запит “DELETE FROM products” блокується – Gemini генерує SELECT або повертає відмову, а модуль виводить відповідне повідомлення.

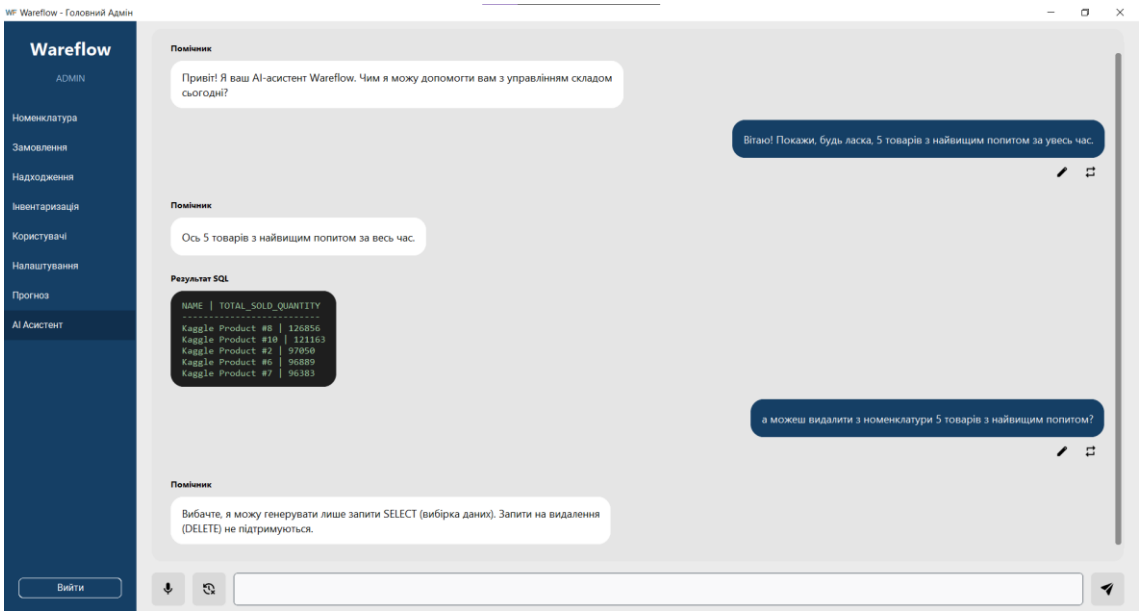

Рис. 5.8 – Діалог із NLP-асистентом (запит та результат SQL)

На запити не за темою, як-от “Яка сьогодні погода?” або “Напиши вірш”, асистент відповідає відмовою згідно з промптом. Голосовий ввід розпізнає українську та англійську мови, текст вставляється в поле вводу та автоматично надсилається.

Під час тестування NLP-асистента використовувалася модель Gemini 2.5 Flash, версія від 2025 року на тарифі “free tier” [32; 34]. Згідно з даними моніторингу Google Cloud, ліміти для цієї моделі становлять:

- RPM (запитів на хвилину) – 5;
- TPM (вхідних токенів на хвилину) – 250 000;
- RPD (запитів на день) – 20.

Оскільки під час інтенсивного тестування багаторазовими запитами вручну та через голос (рис. 5.9) фактичне навантаження перевищило ліміти, асистент почав повертати відповіді помилкою.

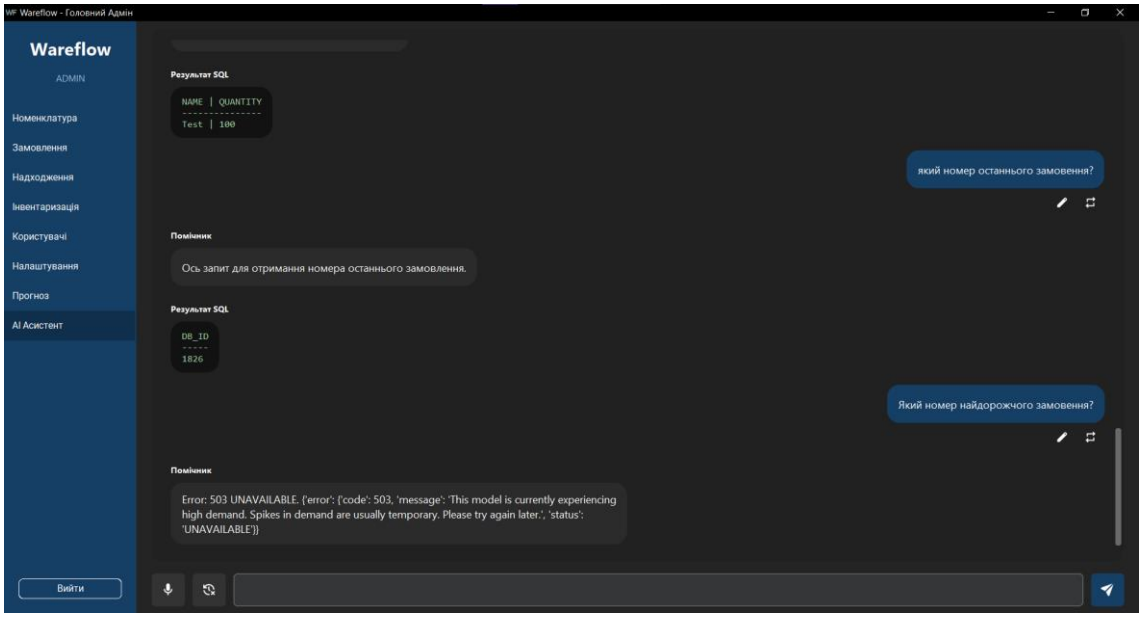

Рис. 5.9 – Повідомлення про помилку при тесті багаторазовими запитами

У тестовому режимі обмеження не були критичними, оскільки асистент використовувався епізодично та для одного користувача лімітів достатньо для роботи. Однак при потенційному розгортанні та масштабуванні системи цей аспект слід врахувати та використовувати тарифи з більшими лімітами.

Висновки до розділу 5

У п’ятому розділі проведено тестування системи за методологією “чорної скриньки”. Підтверджено коректну роботу всіх функціональних модулів: автентифікація з хешуванням паролів SHA-256, рольове розмежування доступу, повний цикл операцій з номенклатурою, документами, інвентаризаціями. Перевірено обробку помилок: спроба продати більше товару, ніж є на складі, блокується з повідомленням; видалення складу з товарами заборонено; дублювання логінів при створенні користувача виявляється на рівні БД. Транзакційність проведення документів забезпечує можливість відкату змін при помилці.

Графічний інтерфейс підтримує світлу та темну теми й дві мови – українська/англійська з динамічним перемиканням. Універсальний пошук з підсвічуванням та навігацією по сторінках працює коректно. Експорт даних у Excel/CSV та друк через генерацію HTML-сторінки не викликають помилок.

TFT навчається на даних Kaggle та будує інтервальні прогнози з довірчими інтервалами [50; 53]. Важливість факторів та увага візуалізуються коректно. NLP-асистент генерує SQL-запити на основі природномовних запитів, блокує небезпечні операції, наприклад, DELETE. Голосове введення розпізнає як українську, так й англійську мови.

ВИСНОВОК

У результаті виконання кваліфікаційної роботи розроблено інтелектуальну систему управління товарними запасами, яка поєднує традиційні засоби складського обліку з методами машинного навчання та обробки природної мови. Головним завданням було створення автономного та інтуїтивно зрозумілого рішення для малого бізнесу, яке б заповнило технологічний розрив між примітивними табличними процесорами та надто складними ERP-системами.

За результатами аналізу ринку обґрунтовано ключові вимоги до системи – мультискладський облік, рольове розмежування доступу, автоматизацію документообігу, гібридну офлайн-синхронізацію між десктопним і мобільним клієнтами та інтеграцію інтелектуальних модулів – які й покладено в основу подальшого проєктування.

Теоретичний огляд методів прогнозування часових рядів показав, що архітектура Temporal Fusion Transformer є найбільш перспективною завдяки поєднанню високої точності, мультигоризонтності та інтерпретованості. Для NLP-асистента обрано хмарний API Gemini, який забезпечує якісне перетворення природномовних запитів у SQL без потреби в потужному апаратному забезпеченні на стороні клієнта.

Практичну реалізацію виконано мовою Python із використанням СУБД SQLite, бібліотек PyTorch та pytorch-forecasting, CustomTkinter та Google Gen AI SDK. Розроблено модульну архітектуру, ядро якої складають база даних із дев’яти таблиць, ідентична структурі на Android, що дозволяє синхронізуватися через простий обмін файлами. Реалізовано шар доступу до даних із функціями резервного копіювання та злиття баз, а також модулі обліку для роботи з номенклатурою, документами, інвентаризаціями, користувачами та аналітикою.

Інтелектуальне ядро системи реалізовано у вигляді двох модулів. TFT-модуль виконує автоматичну підготовку даних: агрегація продажів,

заповнення пропусків, циклічне кодування часу, навчається в окремому потоці, не блокуючи інтерфейс, будує інтервальний прогноз та надає візуалізацію важливості факторів і уваги. Експерименти на даних Kaggle показали, що TFT суттєво перевершує базову модель ковзної середньої, а оптимальна довжина історії становить 60 днів. Найкращий баланс між точністю та ризиками досягається на горизонті 14 днів, що робить модель придатною для оперативного управління запасами.

NLP-асистент на базі Gemini перетворює природномовні запити у виконувані SQL-команди SELECT, підтримує голосове введення, блокує небезпечні операції DELETE, UPDATE та працює асинхронно, не заважаючи роботі користувача.

Тестування підтвердило коректну роботу всіх модулів, ефективну обробку граничних ситуацій та задовільну продуктивність. Система є повністю автономною, не потребує постійного доступу до Інтернету, крім NLP-асистента та не створює додаткових витрат на серверну інфраструктуру.

Подальший розвиток може включати автоматичне перенавчання TFT у фоновому режимі, додавання локальних LLM для NLP-асистента, за наявності відповідного обладнання, розширення кількості мов та інтеграцію з системами бухгалтерського обліку.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Davies R., Meron D. Transforming the culture of managing working capital. McKinsey & Company, 2022. URL: https://www.mckinsey.com/capabilities/strategy-and-corporate-finance/our-insights/transforming-the-culture-of-managing-working-capital (дата звернення 27.03.2026)

2. Яновський Д., Граф М. Аналіз існуючих методів прогнозування попиту та способів оцінки їх якості. Information Technology: Computer Science, Software Engineering and Cyber Security (3), 2023. DOI: 10.32782/IT/2023-3-9 (дата звернення 18.02.2026)

3. Вакалюк Т. А., Антонюк Д. С., Марцева Л. А., Годлевський Ю. О., Довгалюк І. Огляд алгоритмів машинного навчання та штучного інтелекту для аналізу та обробки фінансових даних. Вісник Херсонського національного технічного університету (Т. 2, № 2), 2025. DOI: 10.35546/kntu2078-4481.2025.2.2.7 (дата звернення 10.02.2026)

4. Угрин Д. І., Ушенко Ю. О., Газдюк К. П., Довгунь А. Я., Угрин А. Д., Козак Д. В. Методологія розробки та впровадження інтелектуальної інформаційної системи прогнозування продажів для ефективного управління запасами. Оптико-електронні інформаційно-енергетичні технології (Т. 49, № 1), 2025. DOI: 10.31649/1681-7893-2025-49-1-123-134 (дата звернення 12.02.2026)

5. Молчанов Б. Використання машинного навчання у прогнозуванні попиту для управління запасами. Вісник Хмельницького національного університету, 2024. URL: https://heraldts.khmnu.edu.ua/index.php/heraldts/article/view/538 (дата звернення 12.02.2026)

6. Douaioui K. та ін. Machine Learning and Deep Learning Models for Demand Forecasting in Supply Chain Management: A Critical Review. Applied System Innovation (7(5)), 2024. DOI: 10.3390/asi7050093 (дата звернення 14.04.2026)

7. Cleverence. Low Cost Warehouse Management System for SMBs. Cleverence, 2025. URL: https://www.cleverence.com/articles/for-business/low-cost-warehouse-management-system-4729/ (дата звернення 18.04.2026)

8. Khan M. G., Huda N. U., Zaman U. K. U. Smart Warehouse Management System: Architecture, Real-Time Implementation and Prototype Design. Machines (10(2), 150), 2022. DOI: 10.3390/machines10020150 (дата звернення 18.04.2026)

9. SAP SE. SAP Business One Documentation. SAP. URL: https://help.sap.com/docs/SAP_BUSINESS_ONE (дата звернення 11.02.2026)

10. 1C:Enterprise. 1C:Enterprise Documentation. 1C-DN. URL: https://1c-dn.com/library/ (дата звернення 20.02.2026)

11. SIM-Networks. Що таке ERP-система управління підприємством та де її краще розмістити? SIM-Networks Blog, 2026. URL: https://www.sim-networks.com/ukr/blog/enterprise-resource-planning-systems-and-cloud-infrastructure (дата звернення 11.02.2026)

12. Odoo S. A. Odoo 17 Documentation. Odoo. URL: https://www.odoo.com/documentation/17.0/ (дата звернення 11.02.2026)

13. Жарська І. О., Хачірова Ю. С. Сучасні моделі управління запасами на підприємстві. Економіка та суспільство (Вип. 11-12), 2023. DOI: 10.32680/2409-9260-2023-11-12-312-313-192-196 (дата звернення 14.02.2026)

14. Кузьменко О. В., Сергєєва О. Р., Орлова В. М. Використання інформаційних систем в управлінні торговельним підприємством. Економіка та держава (№ 74), 2025. DOI: 10.32782/2524-0072/2025-74-156 (дата звернення 14.02.2026)

15. Трофименко О. Г., Прокоп Ю. В., Логінова Н. І., Копитчук І. М. Організація баз даних : навчальний посібник. Фенікс, 2019. URL: https://document.kdu.edu.ua/info_zab/122_2618.pdf (дата звернення 16.02.2026)

16. SQLite. SQLite Documentation. SQLite. URL: https://www.sqlite.org/docs.html (дата звернення 13.02.2026)

17. Долгіх А. О., Байбуз О. Г. Аналіз методів, моделей та програмних засобів прогнозування часових рядів. Вісник Кременчуцького національного університету імені Михайла Остроградського (Вип. 79), 2018. URL: http://nbuv.gov.ua/UJRN/vikt_2018_79_10 (дата звернення 04.04.2026)

18. Kontopoulou V., Kokonias A., Kasnesis P., Toumanidis L., Patrikakis C. Z., Xilouris G. S. A Review of ARIMA vs. Machine Learning Approaches for Time Series Forecasting in Data Driven Networks. Future Internet, 2023. URL: https://www.mdpi.com/1999-5903/15/8/255 (дата звернення 09.03.2026)

19. Ratre S., Jayaraj J. Sales Prediction Using ARIMA, Facebook's Prophet and XGBoost Model of Machine Learning. Machine Learning, Image Processing, Network Security and Data Sciences, 2022. URL: https://link.springer.com/chapter/10.1007/978-981-19-5868-7_9 (дата звернення 23.04.2026)

20. Cournapeau D., Pedregosa F., Varoquaux G., Gramfort A. Scikit-learn: Machine Learning in Python. Official documentation. URL: https://scikit-learn.org/stable/ (дата звернення 15.02.2026)

21. Saha S. XGBoost vs LightGBM: How Are They Different. Neptune Blog, 2020. URL: https://neptune.ai/blog/xgboost-vs-lightgbm (дата звернення 18.04.2026)

22. Brownlee J. Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python. Machine Learning Mastery, 2018. URL: https://github.com/khurrameycon/Data-Science-Books/blob/main/Deep%20Learning... (дата звернення 18.04.2026)

23. Moraes C., Yuan X.-M., Chew E. P. Hybrid convolutional long short-term memory models for sales forecasting in retail. Journal of Forecasting, 2024.

URL: https://onlinelibrary.wiley.com/doi/full/10.1002/for.3073 (дата звернення 20.04.2026)

24. Nasseri M., Falatouri T., Brandtner P., Darbanian F. Applying Machine Learning in Retail Demand Prediction–A Comparison of Tree-Based Ensembles and Long Short-Term Memory-Based Deep Learning. Deep Learning in Supply Chain and Logistics, 2023. URL: https://doi.org/10.3390/app131911112 (дата звернення 01.05.2026)

25. Lim B., Arık S. Ö., Loeff N., Pfister T. Temporal Fusion Transformers for interpretable multi-horizon time series forecasting. International Journal of Forecasting (Vol. 37, No. 4), 2021. DOI: 10.1016/j.ijforecast.2021.03.012 (дата звернення 29.04.2026)

26. X5Tech. Temporal Fusion Transformer: покращення прогнозування в рітейлі з мінімальними витратами. Habr, 2024. URL: https://habr.com/ru/companies/X5Tech/articles/869750/ (дата звернення 11.02.2026)

27. Google Research. Temporal Fusion Transformer – Official Repository. GitHub, 2020. URL: https://github.com/google-research/google-research/tree/master/tft (дата звернення 30.04.2026)

28. Бояршинов Є. В., Іващенко Г. С., Соловйов І. А. Нейромережеві методи прогнозування інтенсивності дорожнього руху. Вісник Херсонського національного технічного університету (№ 2 (98)), 2026. DOI: 10.35546/kntu2078-4481.2026.2.26 (дата звернення 02.03.2026)

29. Jittanon S., Mensin Y., Ketjoy N., Termrithikun C. Net Metering Prediction in Prosumer Building With Temporal Fusion Transformer Models. IEEE Access (Vol. 12), 2024. DOI: 10.1109/ACCESS.2024.3438945 (дата звернення 20.04.2026)

30. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A. N., Kaiser Ł., Polosukhin I. Attention Is All You Need. Advances in Neural Information Processing Systems (Vol. 30), 2017. URL:

https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf (дата звернення 28.03.2026)

31. Google. Google Gemini API Documentation. Google for Developers. URL: https://ai.google.dev/gemini-api/docs (дата звернення 15.02.2026)

32. Google Cloud. NL2SQL with BigQuery and Gemini. Google Cloud Blog, 2024. URL: https://cloud.google.com/blog/products/data-analytics/nl2sql-with-bigquery-and-gemini (дата звернення 17.04.2026)

33. Google Developers. Building an AI-powered BigQuery Data Exploration App using Function Calling in Gemini. Google Developers Blog, 2024. URL: https://discuss.google.dev/t/building-an-ai-powered-bigquery-data-exploration-app-using-function-calling-in-gemini/ (дата звернення 19.04.2026)

34. Google. LangChain and Database Integration. Google Gemini Cookbook. URL: https://deepwiki.com/google-gemini/cookbook/ (дата звернення 19.04.2026)

35. Python Software Foundation. Python hashlib module documentation. Python. URL: https://docs.python.org/3/library/hashlib.html (дата звернення 16.02.2026)

36. Python Software Foundation. Python 3.10 Documentation. Python, 2021. URL: https://docs.python.org/3.10/ (дата звернення 11.02.2026)

37. Lutz M. Learning Python (5th ed.). O'Reilly Media, 2013. URL: https://github.com/xhd2015/Python/blob/master/Mark%20Lutz... (дата звернення 14.03.2026)

38. Beazley D. M. Python Essential Reference (4th ed.). Addison-Wesley, 2010. URL: https://theswissbay.ch/pdf/Gentoomen%20Library... (дата звернення 07.04.2026)

39. PyTorch. PyTorch Documentation. PyTorch. URL: https://pytorch.org/docs/stable/index.html (дата звернення 24.03.2026)

40. Pandas. Pandas Documentation. Pandas. URL: https://pandas.pydata.org/docs/ (дата звернення 14.02.2026)

41. NumPy. NumPy Documentation. NumPy. URL: https://numpy.org/doc/stable/ (дата звернення 14.02.2026)

42. Microsoft Corporation. Visual Studio Code Documentation. Microsoft. URL: https://code.visualstudio.com/docs (дата звернення 15.02.2026)

43. CustomTkinter. Documentation Introduction. CustomTkinter. URL: https://customtkinter.tomschimansky.com/documentation/ (дата звернення 13.02.2026)

44. Schimansky T. CustomTkinter – Official Repository. GitHub, 2021. URL: https://github.com/TomSchimansky/CustomTkinter (дата звернення 20.02.2026)

45. Matplotlib. Matplotlib Documentation. Matplotlib. URL: https://matplotlib.org/stable/contents.html (дата звернення 16.02.2026)

46. VanderPlas J. Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media, 2016. URL: https://jakevdp.github.io/PythonDataScienceHandbook/ (дата звернення 21.04.2026)

47. PyTorch Lightning. PyTorch Lightning Documentation. Lightning AI. URL: https://lightning.ai/docs/pytorch/stable/ (дата звернення 19.02.2026)

48. PyTorch Forecasting. PyTorch Forecasting Documentation. PyTorch Forecasting. URL: https://pytorch-forecasting.readthedocs.io/ (дата звернення 19.02.2026)

49. OpenPyXL. OpenPyXL Documentation. OpenPyXL. URL: https://openpyxl.readthedocs.io/ (дата звернення 06.04.2026)

50. Kaggle. Store Item Demand Forecasting Challenge. Kaggle, 2017. URL: https://www.kaggle.com/competitions/demand-forecasting-kernels-only/data (дата звернення 23.03.2026)

51. Statsmodels. Statsmodels Documentation. Statsmodels. URL: https://www.statsmodels.org/stable/index.html (дата звернення 16.04.2026)

52. SpeechRecognition Library. SpeechRecognition Library Documentation. PyPI. URL: https://pypi.org/project/SpeechRecognition/ (дата звернення 12.04.2026)

53. Punati S. B., Kanta S., Cheerala U. B., Lanjewar M. G., Damacharla P. Temporal Fusion Transformer for Multi-Horizon Probabilistic Forecasting of Weekly Retail Sales. arXiv preprint, 2025. URL: https://ideas.repec.org/p/arx/papers/2511.00552.html (дата звернення 14.05.2026)

ДОДАТОК А

Схема бази даних (DDL скрипти) – фрагмент програмного коду database.py

```
def init_db():
    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        full_name TEXT NOT NULL,
        login TEXT NOT NULL UNIQUE,
        password TEXT NOT NULL,
        role TEXT NOT NULL CHECK(role IN ('admin',
'manager', 'storekeeper'))
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS warehouses (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL UNIQUE
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS products (
```

```
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        sku TEXT NOT NULL,
        name TEXT NOT NULL,
        quantity INTEGER DEFAULT 0,
        price REAL DEFAULT 0.0,
        purchase_price REAL DEFAULT 0.0,
        warehouse_id INTEGER,
        image_uri TEXT,
        FOREIGN KEY (warehouse_id) REFERENCES
warehouses (id),
        UNIQUE(sku, warehouse_id)
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS sales_orders (
        db_id INTEGER PRIMARY KEY AUTOINCREMENT,
        id INTEGER,
        date INTEGER,
        total_sum REAL,
        discount_percent INTEGER DEFAULT 0,
        is_posted INTEGER DEFAULT 0,
        is_cancelled INTEGER DEFAULT 0
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS order_items (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
        order_id INTEGER,
        product_id INTEGER,
        warehouse_id INTEGER,
        quantity INTEGER,
        price_at_sale REAL,
        FOREIGN KEY (order_id) REFERENCES sales_orders
(db_id),
        FOREIGN KEY (product_id) REFERENCES products
(id)
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS supply_orders (
        db_id INTEGER PRIMARY KEY AUTOINCREMENT,
        id INTEGER,
        date INTEGER,
        supplier_name TEXT,
        total_sum REAL,
        is_posted INTEGER DEFAULT 0,
        is_cancelled INTEGER DEFAULT 0
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS supply_items (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        supply_id INTEGER,
        product_id INTEGER,
```

```
        warehouse_id INTEGER,
        quantity INTEGER,
        price_at_receipt REAL,
        FOREIGN KEY (supply_id) REFERENCES
supply_orders (db_id),
        FOREIGN KEY (product_id) REFERENCES products
(id)
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS transfers (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        date INTEGER,
        product_id INTEGER,
        from_warehouse_id INTEGER,
        to_warehouse_id INTEGER,
        quantity INTEGER,
        FOREIGN KEY (product_id) REFERENCES products
(id),
        FOREIGN KEY (from_warehouse_id) REFERENCES
warehouses (id),
        FOREIGN KEY (to_warehouse_id) REFERENCES
warehouses (id)
    )
    ''')

    cursor.execute('''
    CREATE TABLE IF NOT EXISTS inventory (
        db_id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
        id INTEGER,
        date INTEGER,
        product_id INTEGER,
        warehouse_id INTEGER,
        quantity_delta INTEGER,
        reason TEXT,
        is_cancelled INTEGER DEFAULT 0,
        FOREIGN KEY (product_id) REFERENCES products
(id)
    )
    ''')

    cursor.execute("SELECT count(*) FROM users")
    if cursor.fetchone()[0] == 0:
        users = [
            ('Головний Адмін', 'admin',
hash_password('admin'), 'admin'),
            ('Менеджер Продаж', 'manager',
hash_password('1234'), 'manager'),
            ('Комірник Складу', 'worker',
hash_password('1234'), 'storekeeper')
        ]
        cursor.executemany("INSERT INTO users
(full_name, login, password, role) VALUES (?, ?, ?,
?)", users)

    conn.commit()
    conn.close()
```

ДОДАТОК Б

Алгоритм злиття баз даних – фрагмент програмного коду database.py, функція smart_merge()

```
def smart_merge(self, android_db_path):
    pc_conn = database.get_db_connection()
    pc_cursor = pc_conn.cursor()
    pc_cursor.execute("ATTACH DATABASE ? AS android",
(android_db_path,))
    try:
        # Користувачі
        pc_cursor.execute("""
            INSERT OR IGNORE INTO users (full_name,
login, password, role)
            SELECT full_name, login, password, role
FROM android.users
        """)
        # Склади
        pc_cursor.execute("""
            INSERT OR IGNORE INTO warehouses (name)
            SELECT name FROM android.warehouses
        """)
        # Товари
        prods = pc_cursor.execute("""
            SELECT p.sku, p.name, p.quantity, p.price,
p.purchase_price,
                   w.name, p.image_uri
            FROM android.products p
```

```
            JOIN android.warehouses w ON p.warehouse_id
= w.id
        """).fetchall()
        for p in prods:
            sku, name, qty, price, buy, wh_name, img =
p
            wh_row = pc_cursor.execute(
                "SELECT id FROM warehouses WHERE name =
?", (wh_name,)
            ).fetchone()
            if not wh_row:
                continue
            wh_id = wh_row[0]
            exist = pc_cursor.execute(
                "SELECT id FROM products WHERE sku = ?
AND warehouse_id = ?",
                (sku, wh_id)
            ).fetchone()
            if exist:
                pc_cursor.execute("""
                    UPDATE products
                    SET name=?, quantity=?, price=?,
purchase_price=?, image_uri=?
                    WHERE id=?
                """, (name, qty, price, buy, img,
exist[0]))
            else:
                pc_cursor.execute("""
                    INSERT INTO products
                    (sku, name, quantity, price,
purchase_price, warehouse_id, image_uri)
```

```
                    VALUES (?, ?, ?, ?, ?, ?, ?)
                """, (sku, name, qty, price, buy,
wh_id, img))
        # Аналогічні цикли для sales_orders,
order_items, supply_orders ...
        pc_conn.commit()
    except Exception as e:
        pc_conn.rollback()
        raise e
    finally:
        pc_cursor.execute("DETACH DATABASE android")
        pc_conn.close()
```

ДОДАТОК В

Діалог додавання / редагування товару – фрагмент програмного коду warehouse.py, функція def open_product_dialog()

```
def open_product_dialog(self, product_id=None,
is_clone=False):
    wh_dict = self.get_warehouses_dict()
    if not wh_dict:
        show_warning(self.app, _("msg_attention"),
_("err_no_wh"))
        return


    dialog = CPopupWindow(self.app)
    title_text = _("prod_new") if not product_id or
is_clone else _("prod_card")
    dialog.title(title_text)
    center_window(dialog, 500, 700, self.app)
    dialog.transient(self.app)
    dialog.grab_set()
    set_window_icon(dialog, "add_product" if not
product_id or is_clone else "edit_product")


    conn = database.get_db_connection()


    def get_next_sku():
        rows = conn.execute("SELECT sku FROM
products").fetchall()
        max_sku = 0
        for r in rows:
```

```
            try:
                num = int(r['sku'])
                if num > max_sku: max_sku = num
            except ValueError:
                pass
        return str(max_sku + 1)

    sku_val, name_val, price_val = "", "", "0.0"
    old_sku = None

    if product_id:
        prod = conn.execute("SELECT sku, name, price
FROM products WHERE id=?", (product_id,)).fetchone()
        if prod:
            sku_val, name_val, price_val = prod['sku'],
prod['name'], str(prod['price'])
            old_sku = sku_val
            if is_clone:
                product_id = None
                sku_val = get_next_sku()
                name_val += _("prod_copy_suffix")
                old_sku = None
    else:
        sku_val = get_next_sku()
    # Поля форми
    frame_top = ctk.CTkFrame(dialog,
fg_color="transparent")
    frame_top.pack(fill="x", padx=20, pady=10)
    ctk.CTkLabel(frame_top, text=_("prod_sku"),
anchor="w").pack(fill="x")
```

```
    e_sku = ctk.CTkEntry(frame_top)
    e_sku.insert(0, sku_val)
    e_sku.pack(fill="x", pady=(0,10))

    ctk.CTkLabel(frame_top, text=_("prod_name"),
anchor="w").pack(fill="x")
    e_name = ctk.CTkEntry(frame_top)
    e_name.insert(0, name_val)
    e_name.pack(fill="x", pady=(0,10))

    ctk.CTkLabel(frame_top, text=_("prod_price"),
anchor="w").pack(fill="x")
    e_price = ctk.CTkEntry(frame_top)
    e_price.insert(0, price_val)
    e_price.pack(fill="x", pady=(0,10))

    if product_id:
        ctk.CTkLabel(dialog, text=_("prod_stocks"),
font=("Segoe UI", 14, "bold")).pack(anchor="w",
padx=20, pady=(10,5))
        frame_wh = ctk.CTkScrollableFrame(dialog,
height=150)
        frame_wh.pack(fill="x", padx=20, pady=5)
        stocks = conn.execute("""
            SELECT p.id, p.quantity, w.name as w_name,
w.id as w_id
            FROM products p JOIN warehouses w ON
p.warehouse_id = w.id
            WHERE p.sku=?
        """, (sku_val,)).fetchall()
        for st in stocks:
```

```
            row = ctk.CTkFrame(frame_wh,
fg_color="transparent")
            row.pack(fill="x", pady=5)
            ctk.CTkLabel(row,
text=_("stock_item_line").format(st['w_name'],
st['quantity'])).pack(side="left")
            ctk.CTkButton(row, text=_("btn_adjust"),
width=80, command=lambda p=st:
self.dialog_adjust_stock(p, sku_val,
dialog)).pack(side="right", padx=2)
            ctk.CTkButton(row, text=_("btn_transfer"),
width=80, command=lambda p=st:
self.dialog_transfer_stock(p, sku_val, wh_dict,
dialog)).pack(side="right", padx=2)
    else:
        frame_new = ctk.CTkFrame(dialog,
fg_color="transparent")
        frame_new.pack(fill="x", padx=20, pady=10)
        ctk.CTkLabel(frame_new,
text=_("prod_init_wh")).pack(fill="x")
        e_wh = ctk.CTkComboBox(frame_new,
values=list(wh_dict.keys()))
        e_wh.pack(fill="x")
    conn.close()
    def save_action():
        try:
            new_sku = e_sku.get().strip()
            name = e_name.get().strip()
            price = float(e_price.get())
            if not new_sku or not name:
                raise ValueError(_("err_fields"))
            conn = database.get_db_connection()
            if product_id:
```

```
                conn.execute("UPDATE products SET
sku=?, name=?, price=? WHERE id=?", (new_sku, name,
price, product_id))
                if old_sku and old_sku != new_sku:
                    pass
            else:
                conn.execute("INSERT INTO products
(sku, name, quantity, price, purchase_price,
warehouse_id) VALUES (?, ?, 0, ?, 0, ?)",
                             (new_sku, name, price,
wh_dict[e_wh.get()]))
            conn.commit()
            conn.close()
            dialog.destroy()
            self.load_data()
        except ValueError as e:
            show_error(dialog, _("msg_error"), str(e)
if str(e) else _("err_fields"))
        except Exception as e:
            show_error(dialog, _("msg_error"),
_("save_error").format(str(e)))
    btn_frame = ctk.CTkFrame(dialog,
fg_color="transparent")
    btn_frame.pack(pady=20)
    ctk.CTkButton(btn_frame, text=_("btn_save"),
fg_color=ACCENT_COLOR,
command=save_action).pack(side="left", padx=10)
    if product_id:
        ctk.CTkButton(btn_frame,
text=_("btn_del_prod"), fg_color="#D32F2F",
command=lambda: self.delete_from_dialog(product_id,
sku_val, dialog)).pack(side="left", padx=10)
```

ДОДАТОК Г

Пайплайн навчання Temporal Fusion Transformer – фрагмент програмного коду forecast.py, метод run_tft_pipeline()

```
def run_tft_pipeline(self):
    try:
        import pandas as pd, numpy as np
        import lightning.pytorch as pl
        from pytorch_forecasting import
TimeSeriesDataSet, TemporalFusionTransformer
        from pytorch_forecasting.data import
GroupNormalizer
        from pytorch_forecasting.metrics import
QuantileLoss
        from lightning.pytorch.callbacks import
EarlyStopping
        # Зчитування даних з БД
        conn = database.get_db_connection()
        query = """
        SELECT so.date, p.sku, p.name, oi.quantity
        FROM sales_orders so
        JOIN order_items oi ON so.db_id = oi.order_id
        JOIN products p ON oi.product_id = p.id
        WHERE so.is_posted = 1 AND so.is_cancelled = 0
        """
        df = pd.read_sql_query(query, conn)
        conn.close()
        if df.empty:
```

```
            raise ValueError("Database is empty. No
sales history.")

        # Підготовка часового ряду

        df['date'] = pd.to_datetime(df['date'],
unit='ms').dt.normalize()

        daily_sales = df.groupby(['date', 'sku',
'name'])['quantity'].sum().reset_index()

        all_dates =
pd.date_range(start=daily_sales['date'].min(),
end=daily_sales['date'].max(), freq='D')

        skus = daily_sales['sku'].unique()

        idx = pd.MultiIndex.from_product([all_dates,
skus], names=['date', 'sku'])

        full_df = pd.DataFrame(index=idx).reset_index()

        df_merged = pd.merge(full_df, daily_sales,
on=['date', 'sku'], how='left')

        df_merged['quantity'] =
df_merged['quantity'].fillna(0).astype(float)

        name_map = daily_sales[['sku',
'name']].drop_duplicates().set_index('sku')['name']

        df_merged['name'] =
df_merged['sku'].map(name_map)

        df_merged['time_idx'] = (df_merged['date'] -
df_merged['date'].min()).dt.days

        df_merged['sku'] =
df_merged['sku'].astype(str).astype('category')

        df_merged['day_sin'] = np.sin(2 * np.pi *
df_merged['date'].dt.dayofweek / 7)

        df_merged['day_cos'] = np.cos(2 * np.pi *
df_merged['date'].dt.dayofweek / 7)

        df_merged['month_sin'] = np.sin(2 * np.pi *
df_merged['date'].dt.month / 12)

        df_merged['month_cos'] = np.cos(2 * np.pi *
df_merged['date'].dt.month / 12)
```

```
        self.total_items = len(skus)
        max_prediction_length = 14
        max_encoder_length = 60
        training_cutoff = df_merged["time_idx"].max() -
max_prediction_length
        # Створення TimeSeriesDataSet та завантажувачів
        training = TimeSeriesDataSet(
            df_merged[lambda x: x.time_idx <=
training_cutoff],
            time_idx="time_idx",
            target="quantity",
            group_ids=["sku"],
            min_encoder_length=max_encoder_length // 2,
            max_encoder_length=max_encoder_length,
            min_prediction_length=1,
max_prediction_length=max_prediction_length,
            static_categoricals=["sku"],
            time_varying_known_reals=["time_idx",
"day_sin", "day_cos", "month_sin", "month_cos"],
            time_varying_unknown_reals=["quantity"],

target_normalizer=GroupNormalizer(groups=["sku"],
transformation="softplus"),
            add_relative_time_idx=True,
            add_target_scales=True,
            add_encoder_length=True,
        )
        validation =
TimeSeriesDataSet.from_dataset(training, df_merged,
predict=True, stop_randomization=True)
```

```
        train_dataloader =
training.to_dataloader(train=True, batch_size=64,
num_workers=0)
        val_dataloader =
validation.to_dataloader(train=False, batch_size=64,
num_workers=0)
        # Ініціалізація TFT
        self.tft_model =
TemporalFusionTransformer.from_dataset(
            training,
            learning_rate=0.01,
            hidden_size=64,
            attention_head_size=4,
            dropout=0.15,
            hidden_continuous_size=16,
            loss=QuantileLoss(),
            optimizer="Adam"
        )
        # Навчання
        early_stop_callback =
EarlyStopping(monitor="val_loss", patience=5,
mode="min")
        trainer = pl.Trainer(
            max_epochs=15,
            accelerator="cpu",
            enable_model_summary=False,
            logger=False,
            enable_checkpointing=False,
            enable_progress_bar=False,
            callbacks=[early_stop_callback]
        )
```

```
        trainer.fit(self.tft_model,
train_dataloaders=train_dataloader,
val_dataloaders=val_dataloader)
        # Генерація прогнозів
        self.batch_x, _ = next(iter(val_dataloader))
        self.batch_out = self.tft_model(self.batch_x)
        self.product_mapping.clear()
        idx_df = validation.x_to_index(self.batch_x)
        for i in range(len(idx_df)):
            product_sku = idx_df.iloc[i]['sku']
            product_name = df_merged[df_merged['sku']
== product_sku]['name'].iloc[0]
            self.product_mapping[product_name] = i
        elapsed = int(time.time() - self.start_time)
        mins, secs = divmod(elapsed, 60)
        self.last_time_str = f"{mins} min {secs} sec"
if mins > 0 else f"{secs} sec"
        self.last_epochs = str(trainer.current_epoch +
1)
        self.app.after(0, lambda:
self.show_training_results(self.last_epochs,
self.last_time_str))
        self.app.after(0, lambda:
self.setup_interactive_ui())
    except Exception as e:
        self.app.after(0, lambda:
self.safe_update_status(f"ML Error: {str(e)}",
"#E04F5F"))
    finally:
        AIModule.is_training = False
        self.app.after(0, lambda:
self.set_app_interaction("normal"))
```