Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут комп'ютерних наук
та інформаційних технологій
(повне найменування інституту, назва факультету(відділення))

Кафедра комп'ютерних наук
(повна назва кафедри (предметної циклової комісії))

Магістерська кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему: «Адаптивна рекомендаційна система для продажу товарів на онлайн-платформах»

Виконав студент 6 курсу, групи КН-61
спеціальності:
122 „Комп'ютерні науки"
(шифр і назва напрямку підготовки спеціальності)

Василів Владислав Богданович
(прізвище, ім'я, по батькові)

Керівники: Сінкевич О.В., Борецька І.Б.
(прізвище, ініціали)

Рецензент: Сорочук
(прізвище, ініціали)

Львів-2025

Національний лісотехнічний університет України

(повне найменування вищого навчального закладу)

ННІ комп'ютерних наук та інформаційних технологій

Кафедра комп'ютерних наук

Рівень вищої освіти другий (магістерський)

Спеціальність 122 "Комп'ютерні науки"

ЗАТВЕРДЖУЮ:

Завідувачка кафедри КН

_____Борецька І.Б.

„10" грудня 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Василів Владислав Богданович

(прізвище, ім'я, по батькові)

1.Тема бакалаврської роботи: "Адаптивна рекомендаційна система для продажу товарів на онлайн-платформах", затверджені наказом вищого навчального закладу від "29" квітня 2025 року, № С-288.

2.Термін подання студентом проекту(роботи) 10 грудня 2025 р.

3. Вихідні дані до проекту (роботи) Створити програмне забезпечення яке дозволить отримувати рекомендацію щодо кількості товару який необхідно замовити на наступний місяць. Для розроблення програмного забезпечення використати мову програмування python.

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

Стан проблемної області

Інформаційне забезпечення

Математичне забезпечення

Програмне забезпечення

Розроблення стартап-проєкту

Висновки

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

Підготовка матеріалу до доповіді.

6. Дата видачі завдання 1 травня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Етапи бакалаврської роботи	Термін виконання	Відмітка про виконання
1.	Отримання звітів продаж	2-5 травня	виконано
2.	Обрання алгоритмів рекомендації	6-12 травня	виконано
3.	Обрання мов програмування	13-18 травня	виконано
4.	Реалізація алгоритму попередньої обробки даних	19-30 травня	виконано
5.	Попередня обробка даних	1-15 червня	виконано
6.	Агрегація даних	16 червня - 10 червня	виконано
7.	Розробка веб-інтерфейсу для візуалізації результатів	10 червня - 15 серпня	виконано
8.	Проведення тестування системи в умовах реальних даних	16 серпня - 15 жовтня	виконано
9.	Оформлення пояснювальної записки	16 жовтня - 5 грудня	виконано

Студент _____ (підпис) Василів В.Б. (прізвище та ініціали)

Керівники роботи _____ (підпис) Сінкевич О.В. (прізвище та ініціали)

_____ (підпис) Борецька І.Б. (прізвище та ініціали)

АНОТАЦІЯ

Дипломна робота містить 97 сторінок пояснювальної записки, 15 рисунків, 3 додаток, 29 джерел.

Основною метою роботи є розробка адаптивної рекомендаційної системи для дозамовлення товарів на основі щомісячних звітів продажів інтернет-магазину npshop.com.ua. Використання сучасних методів обробки часових рядів та адаптивних алгоритмів прогнозування дозволяє формувати рекомендації з урахуванням обсягу історичних даних для кожного товару. Розробка веб-інтерфейсу на основі Next.js та React з TypeScript забезпечує зручність відображення таблиці з назвами товарів і рекомендованими обсягами дозамовлення, а також графіків продажів. Для збереження та обробки даних застосовано PostgresSQL та PrismaORM.

Ключові слова: адаптивна рекомендаційна система, прогнозування продажів, Nest.js, Next.js, React, TypeScript, PostgresSQL, PrismaORM, python.

ABSTRACT

The thesis contains 97 pages of explanatory note, 15 figures, 3 appendices, 29 sources.

The main goal of the work is to develop an adaptive recommendation system for reordering goods based on monthly sales reports of the online store npshop.com.ua. The use of modern methods of time series processing and adaptive forecasting algorithms allows to generate recommendations taking into account the amount of historical data for each product. The development of a web interface based on Next.js and React with TypeScript provides the convenience of displaying a table with product names and recommended pre-order volumes, as well as sales charts. PostgresSQL and PrismaORM are used to store and process data.

Keywords: adaptive recommender system, sales forecasting, Nest.js, Next.js, React, TypeScript, PostgresSQL, PrismaORM, FastAPI, python.

ТЕХНІЧНЕ ЗАВДАННЯ

Розробити адаптивну рекомендаційну систему для інтернет-магазину npshop.com.ua, яка аналізує щомісячні звіти продажів товарів із різною довжиною часових рядів - деякі товари мають історію продажів кілька років, а деякі - лише рік чи менше. Система реалізує Prophet, SARIMA, LSTM моделі прогнозування дозамовлення, які повинні автоматично підібрати залежно від доступного обсягу даних для кожного товару після чого надається рекомендація, щодо кількості товару який необхідно замовити.

Розробити веб-застосунок, що надасть користувачу зручний інтерфейс з таблицею товарів та рекомендованою кількістю дозамовлення. Забезпечити можливість перегляду графіків продажів для кожного товару для візуального аналізу динаміки продажів.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ 7
ВСТУП 8
РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ 9
1.1 Опис проблемної області 9
1.2 Аналоги існуючих систем 11
1.3 Мови програмування 15
1.4 Фреймворки 19
1.5 Робота з стилями 21
1.6 Інструменти роботи з БД 22
1.7. Висновки до розділу 24
РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ 25
2.1 Система контролю версій 25
2.2 Віддалений репозиторій 26
2.3 Високопродуктивне логування 27
2.4 Структура БД 28
2.5. Висновки до розділу 31
РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ 32
3.1 Загальна математична модель Prophet 33
3.2 Загальна математична модель SARIMA 35
3.3 Загальна математична модель LSTM-мережі 39
3.4. Висновки до розділу 42
РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ 44
4.1 Середовище розробки 44
4.2 Система управління базами даних 45
4.3 Інструмент попередньої обробки звітних даних 47
4.4 Імпорт звітів до бази даних 54
4.5 Загальна схема функціонування системи та Адаптивний рекомендаційний алгоритм 56
4.6 Архітектура та реалізація API 64
4.7 Архітектура та структура користувацького інтерфейсу 66
4.8 Висновки до розділу 68
РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ 71
5.1 Опис ідеї проєкту 71
5.2 Аналіз технологічних можливостей реалізації ідей проєкту 72
5.3 Аналіз ринкових можливостей запуску стартап-проєкту 72
5.4 Розроблення ринкової стратегії проєкту 73

5.5 Маркетингова програма стартап-проєкту....74
5.5 Висновки до розділу....75
ВИСНОВКИ....76
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ....77
ДОДАТОК А - Вигляд логів PinoLogger....79
ДОДАТОК Б - Фрагменти програмного коду API....80
ДОДАТОК В - Фрагменти програмного коду UI....92

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ERP (Enterprise Resource Planning) – це програмна система, що об'єднує та автоматизує ключові бізнес-процеси компанії (фінанси, виробництво, логістика, продажі, управління персоналом тощо) в єдину інтегровану платформу.

CRM (Customer Relationship Management) – це стратегія та програмне забезпечення, що допомагають компаніям управляти всіма взаємодіями з існуючими та потенційними клієнтами.

БД – База даних.

Фронтенд (Frontend) – це частина веб-додатку, з якою безпосередньо взаємодіє користувач, включаючи інтерфейс та елементи керування.

Бекенд (Backend) – це складова програмного забезпечення або системи, яка забезпечує обробку даних, взаємодію з базою даних, зв'язок із фронтендом, реалізацію бізнес-логіки та інших внутрішніх процесів, що не видно користувачу.

ВСТУП

З кожним роком обсяги даних про продажі в інтернет-магазинах зростають, що створює потребу у впровадженні ефективних інструментів для аналізу та формування рекомендацій щодо товарних запасів. В умовах динамічного ринку важливо мати можливість оперативно приймати рішення щодо дозамовлення товарів, щоб уникнути як дефіциту, так і надлишку запасів, що впливає на загальну ефективність бізнесу.

Магістерська робота присвячена розробці програмного забезпечення для ФОП Блажівського Андрія Михайловича, власника інтернет-магазину npshop.com.ua. Для проведення дослідження замовником були надані помісячні звіти про продажі товарів, які стали вихідними даними для створення системи, що формує адаптивні рекомендації щодо дозамовлення продукції на основі аналізу історичних показників продажів.

Одним із ключових аспектів розв’язання цієї задачі є розробка адаптивної рекомендаційної системи, яка здатна аналізувати історичні дані продажів із різною тривалістю часових рядів. Система повинна автоматично підібрати найбільш доречні моделі формування рекомендацій залежно від обсягу та характеру наявних даних для кожного товару.

Об’єктом дослідження - це рекомендаційна система яка надає рекомендацію стосовно кількості товару який необхідно замовити на основі аналізу щомісячних звітів продажів інтернет-магазину npshop.com.ua.

Предметом дослідження є програмна реалізація алгоритмів адаптивного формування рекомендацій..

Метою роботи є створення та практична реалізація адаптивної рекомендаційної системи для формування обсягів дозамовлення товарів.

Практичне значення полягає у впровадженні інструменту, що допоможе власнику інтернет-магазину оптимізувати процес замовлення товарів, підвищити ефективність управління запасами та знизити витрати, пов’язані з надмірними або недостатніми запасами.

РОЗДІЛ 1. СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Опис проблемної області

В умовах стрімкого розвитку цифрової економіки, та активної цифровізації бізнес-процесів, все більшу увагу привертає діяльність малих, а особливо мікропідприємств, які працюють в сегменті продажу товарів. Ці суб'єкти господарювання відіграють важливу роль у забезпеченні сталого розвитку економіки будь-якої держави. Вони створюють нові робочі місця, наповнюють державні та місцеві бюджетів, а також задовольняють попит споживачів на широкий спектр товарів повсякденного вжитку.

Проте, незважаючи на свою значущість, такі підприємства досить часто зіштовхуються з численними труднощами, зумовленими обмеженістю доступних ресурсів - як фінансових, так і кадрових. Однією з ключових і водночас найбільш ресурсомістких проблем є ефективне управління товарним асортиментом, що з часом стає дедалі складнішим. Розширення асортименту товарів потребує регулярного аналізу даних щодо обсягів продажу, виявлення закономірностей попиту, прийняття обґрунтованих рішень щодо дозамовлення, а також ідентифікації нових, перспективних товарів. Це завдання є не просто складською чи логістичною операцією, а комплексною багаторівневою задачею, що охоплює елементи аналітики, прогнозування, оптимізації та стратегічного планування.

На початковому етапі діяльності, коли кількість товарів є відносно невеликою - кілька десятків або сотень найменувань - ці операції можуть виконуватись вручну і не створюють надмірного навантаження. Проте, щойно підприємство намагається розширити свою присутність на ринку і збільшує асортимент до тисячі або більше позицій, традиційні методи управління стають вкрай неефективними. У таких умовах ручна обробка даних вимагає значного часу, а прийняття управлінських рішень, що базуються виключно на особистому досвіді або інтуїції, може призводити до серйозних помилок і втрати прибутків.

Звісно, часткове зменшення навантаження можливе за рахунок функціонального розподілу обов'язків: окремі працівники можуть спеціалізуватися

на аналізі продажів та формуванні замовлень, інші - на дослідженні ринку та пошуку нових товарів. Але навіть за таких умов загальний обсяг інформації, що потребує постійного аналізу, не зменшується. Навпаки - через динаміку ринку, сезонні зміни, зміну поведінки споживачів, логістичні фактори та зовнішні економічні впливи, це завдання потребує постійної актуалізації та глибокої аналітичної роботи.

Окремо варто зосередити увагу на ще одній важливій проблемі: малі підприємства, як правило, рідко мають у своєму розпорядженні спеціалізоване програмне забезпечення або фахівців з аналітики даних. У більшості випадків їхні ресурси обмежуються базовими інструментами - зокрема, електронними таблицями, які, не можуть забезпечити необхідний рівень автоматизації, масштабованості та точності при розв'язанні складних задач з прогнозування попиту чи формування персоналізованих рекомендацій.

Саме тому для таких підприємств використання сторонніх сервісів аналітики або рекомендаційних платформ часто є непосильним - як з технічної, так і з фінансової точки зору. Більшість сучасних систем класу SaaS (software-as-a-service), які пропонують функціонал на основі штучного інтелекту та машинного навчання, працюють на умовах платної підписки. Причому вартість таких рішень стартує від 500, а в багатьох випадках - від 1000 доларів США на місяць. Для мікро- чи малого бізнесу, особливо в умовах обмеженого бюджету та нестабільного ринку, це є надзвичайно суттєвим фінансовим навантаженням, яке вони не можуть собі дозволити.

Крім того, навіть у разі наявності фінансової можливості придбати таку послугу, відсутність досвідченого персоналу, здатного правильно налаштувати, інтегрувати та підтримувати роботу таких сервісів, перетворює процес впровадження на складне і довготривале завдання з невизначеним результатом. Відтак виникає нагальна потреба у створенні адаптивних, доступних рішень, які не потребують глибоких технічних знань, є простими у використанні, легко інтегруються у наявні процеси і водночас дозволяють досягати суттєвих покращень в управлінні товарним асортиментом.

1.2 Аналоги існуючих систем

На сучасному етапі розвитку цифрових технологій спостерігається зростаючий інтерес до систем, що допомагають підприємствам ефективно управляти асортиментом товарів, прогнозувати попит та формувати персоналізовані пропозиції для споживачів. Ринок таких рішень активно розвивається, і сьогодні вже існує низка програмних продуктів та сервісів, які, так чи інакше, вирішують поставлені задачі.

Проте, попри зовнішню схожість функціональності, більшість із них орієнтовані насамперед на потреби середнього та великого бізнесу й переважно є системами типу ERP, CRM або їх комбінацією (ERP + CRM), що надають комплексний функціонал для рекомендацій у різних галузях, не лише в сфері продажів. Така універсальність часто супроводжується високою складністю впровадження та значними витратами, що робить подібні рішення малодоступними для мікро та малих підприємств.

Незважаючи на їхню обмежену придатність для малих підприємств, варто їх згадати:

1. **Zoho Inventory** - це хмарне програмне забезпечення для управління запасами, призначене для малого та середнього бізнесу (рис.1.1). Воно допомагає автоматизувати процеси контролю товарних запасів, замовлень, відвантажень і постачань. Основна мета Zoho Inventory - оптимізувати логістику, знизити ризик нестачі товару та підвищити ефективність операцій. Доступні детальні звіти щодо руху товарів, прибутковості, найбільш популярних товарів, ефективності постачальників. Це допомагає приймати обґрунтовані рішення для оптимізації запасів і продажів.

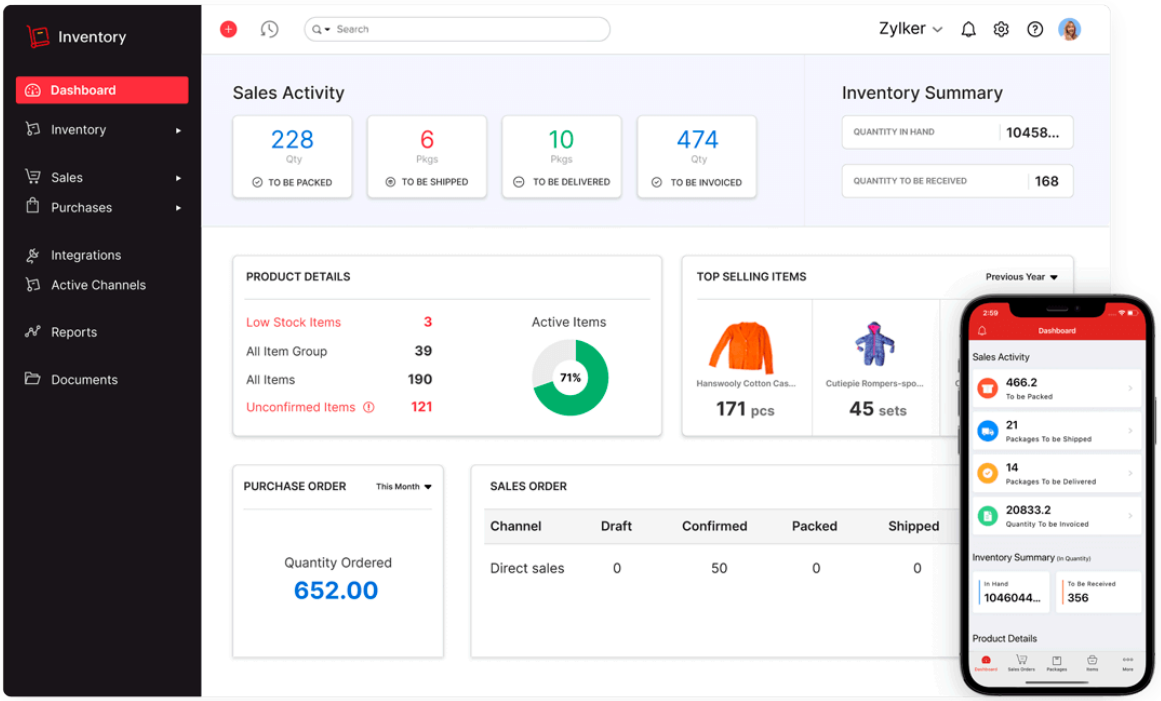


Рисунок 1.1 – Інтерфейс Zoho Inventory

2. **Odoo Inventory** - це комплексна ERP-система з відкритим кодом, що забезпечує управління бізнес-процесами компанії через інтегровані модулі (рис.1.2). Вона охоплює широкий спектр функціональностей, включаючи управління продажами, закупівлями, складом, виробництвом, бухгалтерією, CRM, проектами, людськими ресурсами, маркетингом та електронною комерцією.

Архітектура Odoo базується на модульному підході, що дозволяє підключати або відключати окремі функції залежно від потреб бізнесу. Система підтримує масштабованість, що робить її придатною як для малих підприємств, так і для великих корпорацій.

Основою платформи є веб-інтерфейс, доступний через браузер, що спрощує користування та адміністрування. Odoo побудований на Python з використанням фреймворку Odoo Framework, який забезпечує зручне створення та налаштування модулів, а також автоматичне оновлення бази даних.

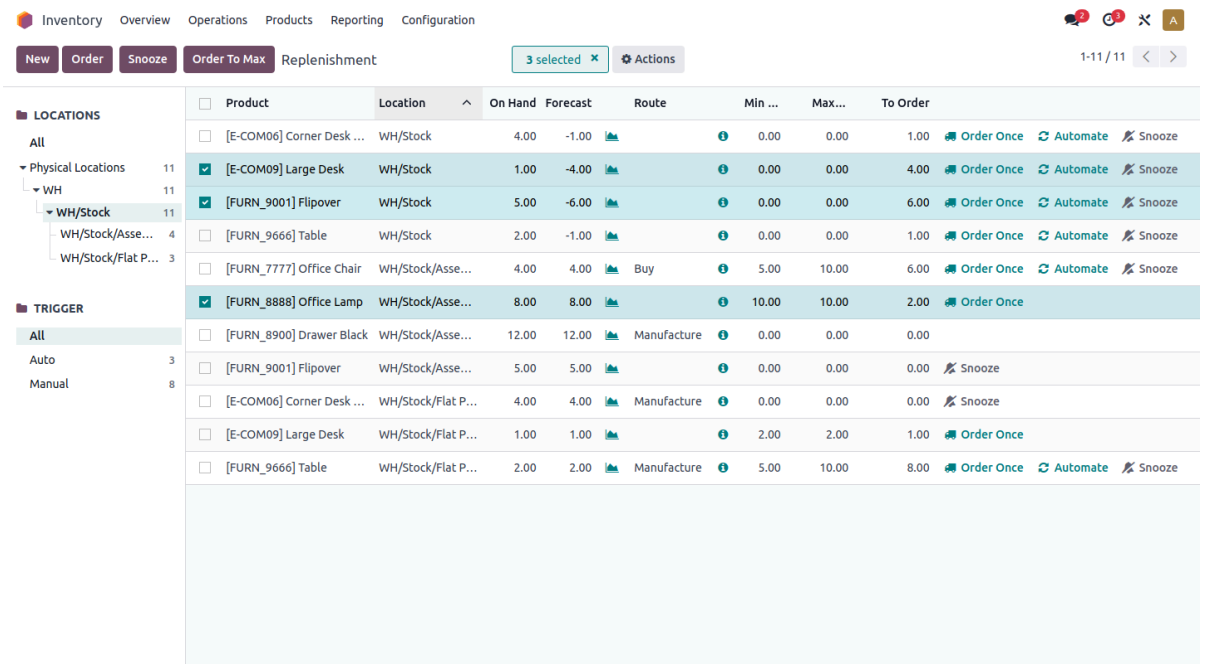


Рисунок 1.2 – Інтерфейс Odoo Inventory

3. **QuickBooks Commerce** - це хмарна платформа для управління інвентарем і замовленнями, призначена переважно для малого та середнього бізнесу, яка інтегрується з QuickBooks Online (рис.1.3). Вона дозволяє централізовано керувати товарами, продажами, закупівлями, каналами електронної комерції та клієнтами.

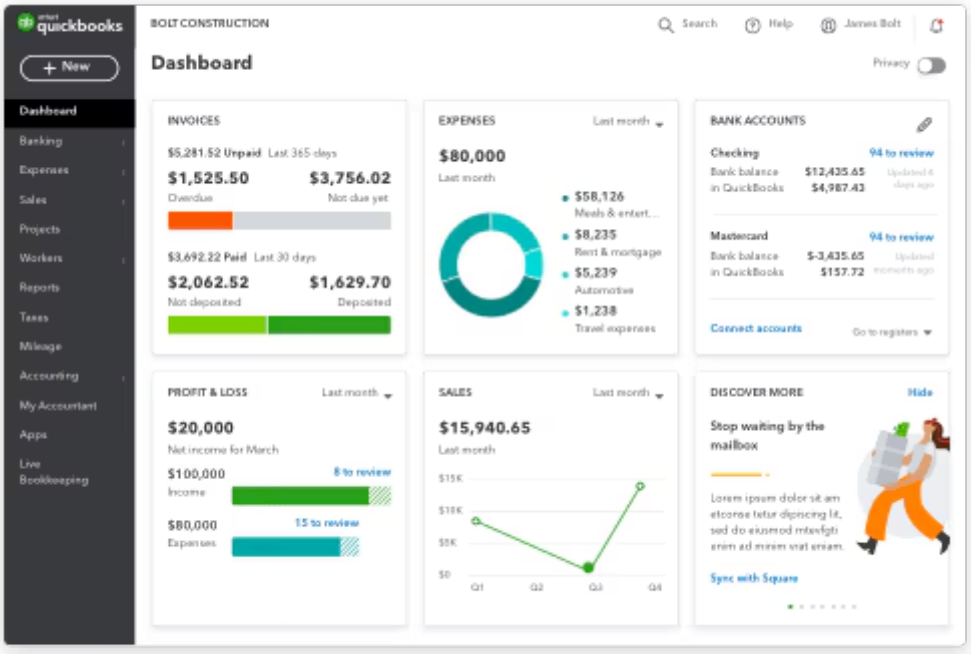


Рисунок 1.3 – Інтерфейс QuickBooks Commerce

4. **SAP Business One** - це ERP-система для малого та середнього бізнесу, розроблена компанією SAP (рис.1.4). Вона забезпечує централізоване управління основними бізнес-функціями, такими як фінанси, закупівлі, продажі, запаси, виробництво, CRM, аналітика та звітність. На відміну від великих SAP-рішень (як-от S/4HANA), Business One орієнтований на підприємства з обмеженими IT-ресурсами та меншою складністю процесів. Система пропонує комплексні можливості для звітності та аналітики, що дозволяє користувачам ефективно управляти даними та отримувати цінні інсайти. Вона включає вбудований генератор звітів для швидкого формування необхідної інформації, а також підтримує інтеграцію з SAP Crystal Reports для розширених можливостей професійного документообігу та візуалізації. Окрім цього, для глибокого аналізу та побудови інтерактивних дашбордів передбачена підтримка інтеграції з Power BI, що дозволяє перетворювати дані на дієві рішення та покращувати розуміння бізнес-процесів.

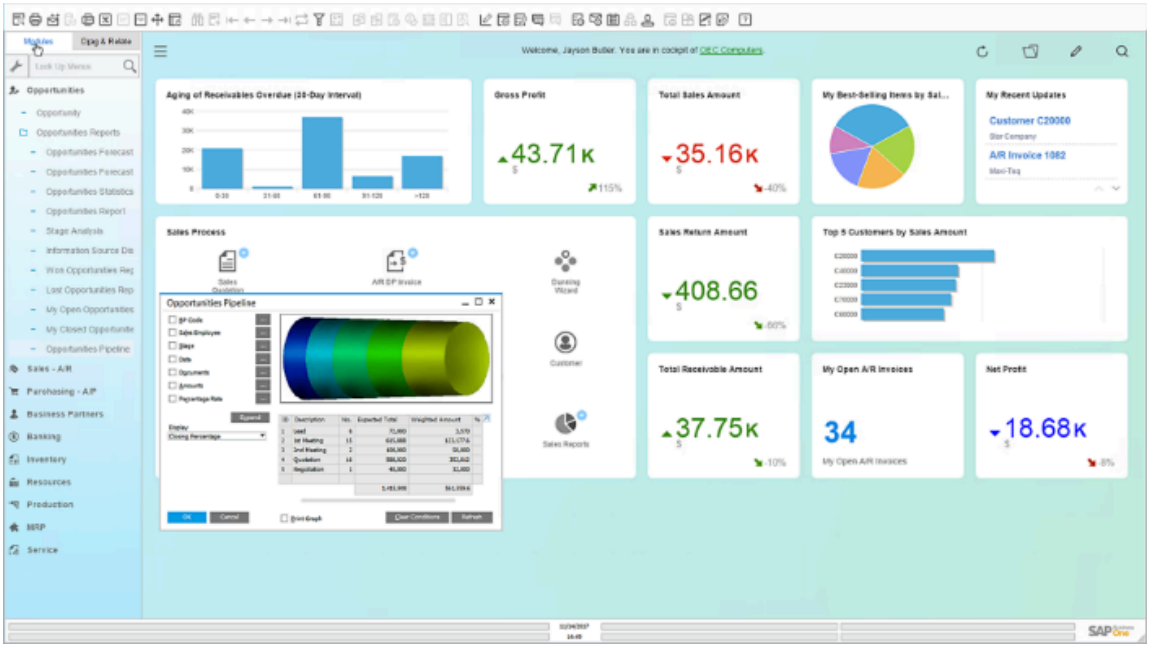


Рисунок 1.4 – Інтерфейс SAP Business One

5. **ERPNext** - це ERP-система з відкритим кодом, розроблена компанією Frappe Technologies на основі власного фреймворку Frappe Framework (рис.1.5). Вона охоплює широкий спектр бізнес-процесів і є прямим конкурентом таким

системам, як Odoo, SAP Business One та Microsoft Dynamics, але з орієнтацією на відкритість і простоту розгортання.

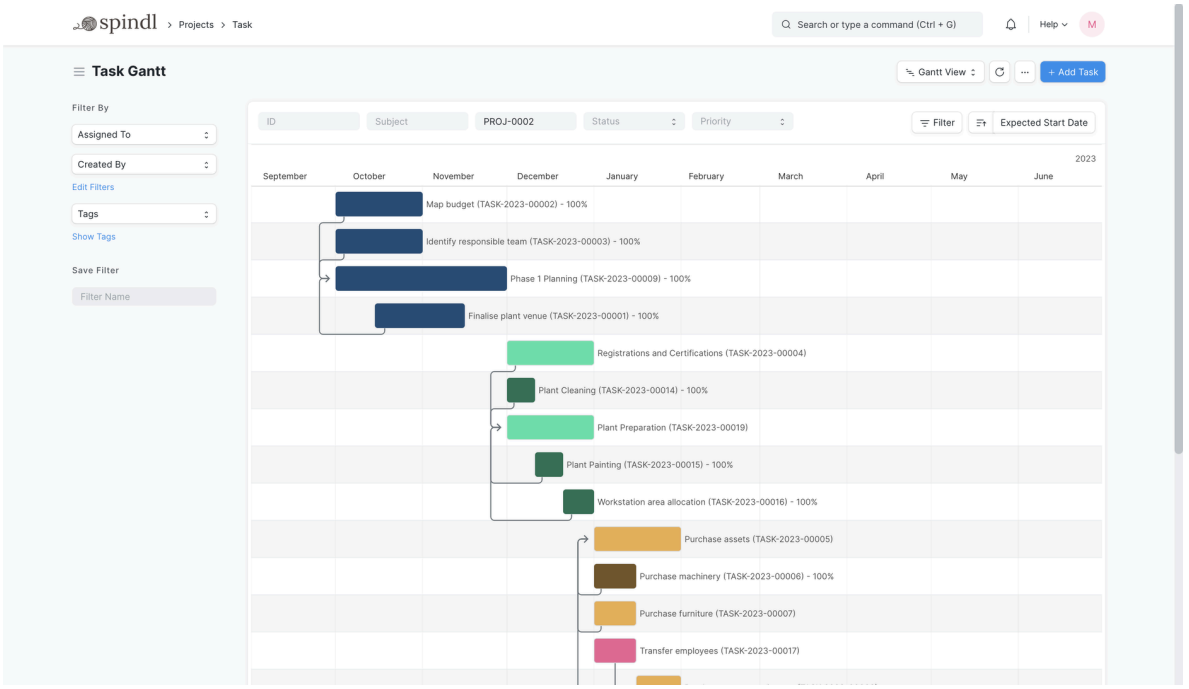


Рисунок 1.5 – Інтерфейс SAP Business One

1.3 Мови програмування

Python - це популярна, динамічно типізована та інтерпретована мова програмування, яка широко використовується в різноманітних галузях інформаційних технологій [1,2]. Вона поєднує в собі простоту синтаксису з потужними можливостями, що робить її доступною як для початківців, так і для досвідчених розробників.

Однією з ключових особливостей Python є динамічна типізація, яка означає, що тип змінної визначається автоматично під час виконання програми. Це дозволяє швидко писати код, хоча й вимагає уважності під час налагодження. Також мова вирізняється зрозумілим і лаконічним синтаксисом: код на Python часто нагадує псевдокод, що значно полегшує розуміння навіть складних програм.

Python підтримує кілька парадигм програмування, зокрема об'єктно-орієнтовану, процедурну та функціональну, що дає розробникам змогу обирати найбільш зручний стиль для конкретних завдань. Окрім того, мова має

широку екосистему бібліотек і фреймворків, які охоплюють практично всі сфери - від наукових обчислень до веб-розробки та штучного інтелекту.

Ще однією важливою перевагою є кросплатформеність: програми на Python можна запускати на різних операційних системах без суттєвих змін у коді, що значно полегшує їхнє розгортання і супровід.

Завдяки цим якостям Python зарекомендував себе як універсальний інструмент для вирішення широкого спектра прикладних завдань, а його популярність продовжує зростати завдяки активній спільноті розробників та постійному розвитку мови.

JavaScript - це високорівнева, інтерпретована мова програмування, яка відіграє ключову роль у розробці інтерактивних веб-додатків. Спочатку створена для додавання динаміки до веб-сторінок, з часом вона значно розширила свої можливості та міцно закріпилася не лише у фронтенді, а й у бекенд-розробці [3].

Ця мова не потребує попередньої компіляції: код виконується безпосередньо в браузері або у середовищі виконання, наприклад Node.js, що спрощує процес розробки і прискорює тестування [4]. Важливою рисою JavaScript є подієво-орієнтована модель, яка дозволяє створювати динамічні інтерфейси - особливо це цінно у веб-розробці, де взаємодія з користувачем має вирішальне значення [5].

Сучасні версії JavaScript (ES6 і новіші) забезпечують повноцінну підтримку об'єктно-орієнтованого програмування з класами, успадкуванням та інкапсуляцією [6]. Також мова має вбудовані механізми для роботи з асинхронними операціями, такими як Promises, async/await і подієвий цикл (event loop) [7]. Це дозволяє ефективно обробляти запити до сервера, таймери та інші фонові задачі.

Завдяки кросплатформеності і універсальності, що з’явилися з появою Node.js, JavaScript став не лише інструментом для створення фронтенду, а й потужним засобом для розробки серверної частини, відкриваючи шлях до повноцінних full-stack рішень [8].

Завдяки активній спільноті, широкій екосистемі бібліотек - таких як React, Angular, Vue - і популярним фреймворкам, як Nest та Next, JavaScript і надалі

залишається однією з найпопулярніших і найважливіших мов програмування у світі [9].

TypeScript повністю сумісний із JavaScript. Увесь валідний JS-код є також валідним TS-кодом. Це дозволяє легко інтегрувати існуючі бібліотеки та фреймворки, а також поступово впроваджувати типізацію в уже написаний код [10]. Завдяки підтримці сучасного синтаксису ESNext розробники можуть використовувати новітні можливості JavaScript навіть до того, як вони будуть підтримані всіма браузерами [11]. Після компіляції код транслюється у відповідну версію JavaScript - наприклад, ES5 для забезпечення сумісності зі старими браузерами [11].

Ще одна особливість - інтерфейси та структурне узгодження. Можна чітко описувати структури об’єктів, визначаючи обов’язкові й необов’язкові властивості, що значно покращує зрозумілість та підтримку коду [12]. Для організації великих кодових баз TypeScript надає підтримку модулів (ES Modules або CommonJS), а також просторів імен, що дозволяє логічно групувати код і уникати конфліктів імен [12].

Для прихильників об’єктно-орієнтованого програмування TypeScript пропонує повний набір можливостей: класи, абстрактні класи, модифікатори доступу, геттери, сеттери, дженерики - усе це робить його дуже зручним для розробників із бекграундом у C# або Java [13].

Розробка на TypeScript супроводжується активною екосистемою: плагіни, лінтери, типізовані оголошення бібліотек (через DefinitelyTyped), підтримка фреймворків на кшталт Angular або NestJS [14]. Також завдяки статичній типізації редактори підтримують розширені можливості рефакторингу, як-от безпечне перейменування змінних, витягування методів, переміщення файлів - усе це значно підвищує продуктивність роботи над великими проєктами [14].

Сьогодні TypeScript дедалі частіше використовують для створення складних фронтенд і бекенд-рішень. Він поєднує легкість JavaScript із суворою типовою перевіркою, що дозволяє попереджати про логічні помилки, полегшує масштабування, спрощує підтримку та робить код більш читабельним і

передбачуваним у довгостроковій перспективі [15].

React - це популярна JavaScript-бібліотека для створення інтерфейсів користувача, розроблена компанією Meta (раніше Facebook) [16]. Вона дозволяє будувати інтерактивні, динамічні та зручні у використанні веб-додатки. Основна ідея React полягає в компонентному підході, що значно спрощує розроблення масштабованих і зручних у підтримці застосунків.

В основі бібліотеки лежить створення окремих, повторно використовуваних компонентів. Кожен із них відповідає за свою частину інтерфейсу, може мати власний стан (state) і властивості (props), що сприяє модульності коду. React використовує JSX - розширення синтаксису JavaScript, яке дозволяє писати HTML-подібний код прямо в JavaScript. Це робить структуру додатку зрозумілішою, а логіку - зручнішою для реалізації [17].

Однією з головних переваг React є віртуальний DOM - проміжне представлення реального DOM, яке дозволяє оновлювати лише ті елементи інтерфейсу, що справді змінилися. Це забезпечує високу продуктивність додатків [18].

Управління станом в React може здійснюватися як локально, так і з використанням зовнішніх рішень - таких як Redux, MobX або ж через вбудовані інструменти React, зокрема хуки (useState, useReducer, useContext). Це відкриває широкі можливості для реалізації як простих, так і складних сценаріїв взаємодії з даними [19].

React ідеально підходить для створення односторінкових додатків (SPA), у яких контент змінюється без повного перезавантаження сторінки. Для реалізації маршрутизації в таких додатках зазвичай використовується бібліотека React Router.

Навколо React сформувалася велика екосистема - з інструментами та бібліотеками, такими як Create React App, Next.js, React Query, Material UI тощо. Це дає змогу швидко запускати проєкти та використовувати готові рішення для типових завдань.

Окрім клієнтського рендерингу, React також підтримує серверний рендеринг (SSR), особливо у зв’язці з фреймворком Next.js, що позитивно впливає на

продуктивність і SEO.

Завдяки активній спільноті, регулярним оновленням і наявності великої кількості навчальних матеріалів, React залишається одним із найпопулярніших інструментів для веб-розробки.

Поєднання React і TypeScript відкриває широкі можливості для підвищення надійності та якості коду ще на етапі розробки, що особливо важливо для великих проєктів. TypeScript дозволяє явно типізувати props та state, завдяки чому можна чітко визначити, які саме значення очікує компонент. Це знижує ризик помилок і робить код значно зрозумілішим.

Ще однією перевагою є покращена робота з редакторами коду: розробники отримують розгорнуті підказки, автодоповнення та попередження про помилки без необхідності запускати застосунок. Це значно пришвидшує процес розробки та спрощує налагодження.

Типізація також відіграє важливу роль у командній роботі, адже дозволяє створювати чіткий контракт між компонентами - кожен учасник команди бачить, які дані приймає або повертає компонент.

Завдяки своїй високій продуктивності та простоті використання, React залишається одним із найпопулярніших інструментів для фронтенд-розробки і часто обирається як основна технологія для створення сучасних веб-застосунків.

1.4 Фреймворки

Next.js - Next.js - це сучасний фреймворк для розробки React-додатків, створений компанією Vercel [20]. Він значно розширює можливості React, забезпечуючи вбудовану підтримку серверного рендеринга, генерації статичних сторінок, зручній маршрутизації, обробки API-запитів та інших критично важливих функцій для побудови повноцінних веб-застосунків.

Однією з ключових переваг Next.js є підтримка серверного рендеринга (SSR), завдяки якому HTML генерується на сервері перед тим, як потрапити до клієнта. Це суттєво покращує швидкість завантаження сторінок, загальну продуктивність і пошукову оптимізацію [21]. Крім того, фреймворк дозволяє використовувати

статичну генерацію сторінок (SSG) під час збірки проєкту - це ідеальний підхід для контенту, який змінюється рідко, наприклад, блоги чи документація.

Next.js також підтримує інкрементальну генерацію (ISR), яка поєднує переваги SSR і SSG: сторінки можуть оновлюватися у фоновому режимі після оновлення даних без повної перебудови проєкту. Створення нових сторінок реалізується через файлову маршрутизацію - достатньо додати новий файл у директорію pages, і він автоматично стане новим маршрутом, що значно спрощує навігацію та організацію коду [22].

Ще одна корисна можливість - API Routes. Вони дозволяють створювати серверні функції без необхідності в окремому бекенді, що особливо зручно для невеликих і середніх застосунків. Усе це доповнюється автоматичною оптимізацією продуктивності, зокрема розділенням коду, підтримкою зображень через next/image, кешуванням і lazy loading.

Next.js має повну підтримку TypeScript. Для початку роботи достатньо створити файли з розширенням .ts або .tsx - фреймворк автоматично налаштовує конфігурацію. Це дозволяє типізувати сторінки, компоненти, обробники API та глобальні дані, що сприяє якості коду та полегшує спільну розробку [23].

Завдяки своїй простоті використання та високій продуктивності, Next.js активно застосовується у створенні блогів, лендінгів, e-commerce платформ, адміністративних панелей і SaaS-рішень. Це один із найпотужніших інструментів для сучасної веб-розробки.

NestJS - це прогресивний фреймворк для створення серверних додатків на Node.js, який поєднує сучасні архітектурні підходи з повноцінною підтримкою TypeScript [24]. Завдяки модульності, інверсії залежностей та чіткому розділенню відповідальностей він чудово підходить для розробки масштабованих і підтримуваних бекенд-систем.

Фреймворк побудований на основі TypeScript, що дозволяє розробникам використовувати типізацію змінних, параметрів, DTO та інших елементів, підвищуючи надійність і читабельність коду. Структура застосунку в NestJS організована у вигляді модулів, де кожен модуль відповідає за конкретну частину

функціональності - наприклад, користувачі, товари або авторизація. Це дозволяє легко масштабувати проєкт і підтримувати його в чистоті [25].

Основна логіка застосунку ділиться між контролерами та сервісами: контролери обробляють HTTP-запити, а бізнес-логіка реалізується у відповідних сервісах. Такий підхід нагадує MVC-архітектуру, що є зрозумілим для більшості розробників. Особливістю NestJS також є вбудована система ін'єкції залежностей, яка спрощує керування об'єктами, їх життєвим циклом та взаємозв'язками, роблячи код більш тестованим [24].

Роутинг реалізується за допомогою зручних декораторів, таких як @Get() або @Post(), що значно пришвидшує створення REST API. Окрім цього, NestJS підтримує GraphQL, WebSocket, gRPC і побудову мікросервісів, що робить його універсальним інструментом для сучасної розробки [26].

Для взаємодії з базами даних NestJS легко інтегрується з такими ORM, як TypeORM, Prisma або Sequelize, що дозволяє ефективно працювати з даними в типізованому середовищі. Крім того, фреймворк забезпечує потужні механізми обробки запитів і відповідей через middleware, guards, interceptors та pipes - усе це дозволяє керувати поведінкою застосунку та відповідати високим стандартам якості коду [25].

1.5 Робота з стилями

Tailwind CSS - це сучасна утилітарна CSS-бібліотека, що надає набір готових класів для швидкої розробки інтерфейсів користувача. На відміну від традиційних фреймворків, вона дозволяє створювати унікальний дизайн без потреби писати кастомний CSS, комбінуючи вже наявні класи.

Один із ключових підходів Tailwind - утилітарний. Замість створення окремих CSS-класів для стилізації, розробник може використовувати класи на кшталт p-4, bg-blue-500, text-center, rounded-xl. Це значно прискорює розроблення та зменшує обсяг написаного коду. Ще одна перевага - повна кастомізація через файл tailwind.config.js, що дозволяє змінювати палітру кольорів, відступи, шрифти та інші властивості під конкретні потреби проєкту.

Tailwind чудово підходить для адаптивного дизайну. Завдяки breakpoint-класам (sm:, md:, lg:, xl:) можна швидко адаптувати інтерфейс під різні розміри екранів без зайвого CSS. Це робить його ідеальним інструментом для швидкого прототипування. Розробник може одразу побачити результат, не витрачаючи час на написання стилів з нуля.

Tailwind легко інтегрується з популярними JavaScript-фреймворками, такими як React, Next.js, Vue чи Angular. Особливо популярною є комбінація Tailwind + Next.js, яка дозволяє швидко створювати стильні та зручні UI. Для продакшн-білдів використовується механізм видалення невикористаних стилів, завдяки чому фінальний CSS-файл залишається легким та оптимізованим.

Окрім цього, Tailwind підтримує систему плагінів, що розширює його функціональність. Наприклад, можна додати підтримку анімацій, форм, типографіки тощо.

Використання Tailwind дає змогу зосередитися на логіці і компонуванні інтерфейсу, залишаючи в стороні рутинне написання стилів. Це особливо цінно при створенні кастомного дизайну або розробці MVP-версій веб застосунків.

1.6 Інструменти роботи з БД

Prisma - це сучасний ORM (Object-Relational Mapping) інструмент для Node.js та TypeScript, який забезпечує зручну, типізовану та безпечну роботу з базами даних. Він значно спрощує взаємодію з SQL-базами, надаючи зрозумілий API для операцій читання, запису, оновлення та видалення даних.

Однією з головних переваг Prisma є повна підтримка TypeScript: усі дії з базою даних супроводжуються автоматичною генерацією типів, що дозволяє уникнути багатьох помилок ще на етапі розробки. Робота з моделями стає не тільки безпечнішою, а й інтуїтивно зрозумілою.

Синтаксис Prisma базується на декларативному підході: структура бази описується у файлі schema.prisma, де задаються моделі, поля та зв’язки між ними. Це робить архітектуру даних прозорою та зручною для керування.

Завдяки Prisma Client розробники можуть реалізовувати CRUD-операції за

допомогою простих, читабельних конструкцій. А система міграцій дозволяє вносити зміни до бази даних у контрольований і безпечний спосіб. Наприклад, команда prisma migrate dev автоматично генерує SQL-код та застосовує відповідні зміни.

Prisma також відзначається високою продуктивністю та здатністю масштабуватися під сучасні вимоги - інструмент легко справляється з великими обсягами даних і складними зв’язками. Підтримка різноманітних СУБД, таких як PostgreSQL, MySQL, SQLite, SQL Server, MongoDB та CockroachDB, дозволяє використовувати Prisma у проєктах будь-якого рівня складності.

Крім того, Prisma легко інтегрується з популярними фреймворками, як-от Nest.js для побудови масштабованої серверної архітектури або Next.js для SSR-застосунків. Це дозволяє будувати взаємодію між логікою застосунку та шаром доступу до даних.

Загалом, Prisma ORM - це потужний інструмент, який ідеально підходить для розробників, що працюють з TypeScript. Його зручність, типізація, система міграцій та підтримка сучасних фреймворків сприяють швидкій і надійній розробці веб-застосунків.

SQL

SQL - це стандартизована мова запитів, яка призначена для створення, управління та взаємодії з реляційними базами даних. Вона є основним інструментом для обробки структурованої інформації в таких системах управління базами даних, як PostgreSQL, MySQL, SQLite, Microsoft SQL Server та інших.

За допомогою SQL можна маніпулювати даними, виконуючи базові операції, такі як додавання нових записів (INSERT), читання інформації (SELECT), оновлення існуючих даних (UPDATE) та їхнє видалення (DELETE). Крім того, SQL дає змогу створювати структуру бази даних: створювати таблиці, змінювати їхню структуру, додавати індекси, встановлювати ключі та зв’язки між таблицями.

Важливою функцією є управління доступом - SQL дозволяє керувати правами користувачів, що забезпечує безпеку даних у багатокористувацьких середовищах. Також мова підтримує роботу з транзакціями, що дає змогу об’єднувати кілька

операцій в одну логічну групу з гарантією, що або всі зміни будуть застосовані, або жодна - завдяки командам BEGIN, COMMIT і ROLLBACK.

Для аналізу даних SQL надає можливості агрегації та фільтрації. Можна виконувати обчислення над групами записів за допомогою GROUP BY і HAVING, використовувати різні функції, як-от COUNT(), AVG(), MAX() чи MIN(), а також відбирати дані за певними умовами за допомогою WHERE.

Окрім цього, SQL дозволяє з'єднувати таблиці, поєднуючи інформацію з кількох джерел. Це здійснюється через різні типи JOIN, такі як INNER JOIN, LEFT JOIN, RIGHT JOIN, що дає змогу формувати складні запити до взаємопов'язаних сутностей.

Переваги використання SQL полягають у його незалежності від конкретної мови програмування, адже SQL виступає універсальним інтерфейсом для роботи з базами даних. Завдяки системі запитів, він дозволяє отримувати складну інформацію з великих обсягів даних. Крім того, SQL підтримує транзакції, що забезпечує цілісність і надійність збережених даних.

1.7. Висновки до розділу

У результаті проведеного аналізу можна зробити висновок, що мікро- та малі підприємства, попри свою важливу роль в економіці, часто стикаються з серйозними труднощами в управлінні товарним асортиментом. Основною причиною цього є обмеженість ресурсів - як фінансових, так і технічних. Більшість існуючих рішень на ринку (ERP, CRM-системи тощо) виявляються занадто складними або дорогими для впровадження в умовах невеликого бізнесу. Крім того, для їхньої ефективної роботи потрібні спеціалісти, яких у малих компаніях зазвичай немає.

Аналіз існуючих аналогів показав, що хоча є багато цікавих продуктів, таких як Zoho Inventory, Odoo, QuickBooks Commerce чи SAP Business One, більшість із них орієнтовані на середній та великий бізнес. Тому постає потреба у створенні простого, доступного, але водночас функціонального рішення, яке дозволить малим підприємствам отримати інструмент рекомендацій щодо управління асортиментом без значних витрат та складнощів у використанні.

РОЗДІЛ 2. ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Система контролю версій

В ці роботі було використано **Git** - це розподілена система контролю версій, яка дозволила мені ефективно відстежувати зміни в коді та забезпечувати збереження історії змін. Вона стала стандартом де-факто в більшості сучасних IT-проєктів, особливо у сфері розробки програмного забезпечення. Саме тому я використав цю систему під час написання дипломної роботи.

Особливістю Git є його розподілена архітектура. На відміну від централізованих систем контролю версій, Git зберігає повну копію репозиторію на кожному комп'ютері розробника, що дає змогу працювати автономно, без постійного доступу до сервера. Завдяки цьому Git демонструє високу швидкість і ефективність: операції, такі як коміти, перемикання гілок чи об'єднання (merge), виконуються локально й дуже швидко.

Ще одна важлива перевага - підтримка гілкування. Git дозволяє створювати та об'єднувати гілки, що допомагає паралельно працювати над різними функціональностями без конфліктів. Це значно полегшує розроблення та впровадження нових функцій.

Відстеження змін у Git здійснюється через фіксацію кожної зміни у вигляді коміту з коментарем, автором, датою та унікальним хешем. Це дозволяє легко аналізувати історію проєкту і за потреби повертатися до попередніх версій коду.

Git тісно інтегрується з популярними хостинговими платформами, такими як GitHub, GitLab, Bitbucket, що дає змогу розміщувати репозиторії в хмарі, налаштовувати CI/CD, проводити рев'ю коду та багато іншого.

Крім того, Git характеризується високою конфігурованістю: можна кастомізувати робочі процеси, обираючи між різними підходами до розробки, наприклад Git Flow чи trunk-based development.

Таким чином, Git є невід'ємною частиною сучасного процесу розробки програмного забезпечення.

2.2 Віддалений репозиторій

GitHub - це популярна веб-платформа для хостингу репозиторіїв, яка використовує систему контролю версій Git. Вона надає розробникам можливість зберігати код у хмарі, працювати над проєктами спільно з іншими учасниками команди та автоматизувати процеси розробки. Сервіс активно застосовується як в особистих проєктах, так і у великих комерційних компаніях.

Однією з головних переваг GitHub є віддалене зберігання репозиторіїв - це дозволяє отримати доступ до коду з будь-якого пристрою, а також обирати, чи буде репозиторій публічним або приватним. Завдяки функції pull-запитів можна обговорювати запропоновані зміни з командою, що значно покращує якість коду і підтримує ефективну співпрацю.

GitHub також має зручну систему Issue та Projects, яка допомагає відстежувати баги, планувати завдання та організовувати роботу над проєктом у структурований спосіб. А з інтегрованою функцією GitHub Actions стає можливим налаштування повного CI/CD-процесу - автоматизація тестування, збірки та розгортання відбувається прямо в межах репозиторію, що значно спрощує підтримку стабільності проєкту.

Важливою складовою є підтримка Markdown-документації. README-файли можуть містити повну інформацію про проєкт, інструкції з встановлення, приклади використання тощо - це зручно для нових учасників або користувачів, які вперше знайомляться з репозиторієм.

Крім того, GitHub підтримує інтеграцію з багатьма зовнішніми сервісами, такими як VS Code, Slack, Discord, Heroku, Netlify, Vercel та іншими, що дозволяє розширити функціональність і налаштувати зручне середовище для розробки.

У підсумку, GitHub - це не просто платформа для зберігання коду. Це повноцінна екосистема, що охоплює весь життєвий цикл розробки програмного забезпечення. Інтуїтивно зрозумілий інтерфейс, глибока інтеграція з Git і потужні засоби командної роботи зробили її одним із найпопулярніших інструментів серед сучасних розробників.

2.3 Високопродуктивне логування

PinoLogger - це одна з найпродуктивніших бібліотек для ведення логування в середовищі JavaScript/Node.js, що забезпечує високу ефективність та мінімальні накладні витрати. Вона стала популярною завдяки своїй здатності забезпечувати високу продуктивність навіть в умовах інтенсивного навантаження. Однією з головних переваг PinoLogger є те, що він використовує асинхронне логування, яке значно зменшує час очікування при записуванні логів, що критично важливо для додатків з високими вимогами до швидкодії. Завдяки такій архітектурі бібліотека може працювати швидше за аналогічні інструменти, такі як Winston чи Bunyan.

Однією з ключових особливостей PinoLogger є мінімізація накладних витрат при виконанні логування. Бібліотека дозволяє знижувати вплив ведення журналів на загальну продуктивність системи завдяки використанню низькорівневих функцій запису, що значно зменшує затримки. PinoLogger спроектовано таким чином, щоб максимально ефективно працювати в режимі реального часу, зберігаючи високу швидкість обробки запитів і мінімізуючи вплив на продуктивність системи. Це дозволяє успішно використовувати бібліотеку в умовах високих навантажень, коли кожен мілісекунд важливий.

Окрім того, PinoLogger підтримує асинхронне записування логів, що забезпечує можливість паралельного виконання основної обробки запитів та логування, тим самим не блокуючи основний потік виконання програми. Така особливість забезпечує збереження високої продуктивності навіть у великих системах з численними запитами. Крім того, бібліотека підтримує потокове оброблення логів, що є важливим для додатків, що вимагають обробки даних у реальному часі.

PinoLogger також використовує структуровані логи у форматі JSON, що забезпечує зручність і ефективність обробки великих масивів даних. Завдяки цьому, можна легко здійснювати фільтрацію, агрегацію та аналіз логу, що надзвичайно важливо при інтеграціях з інструментами моніторингу, такими як Elasticsearch чи AWS CloudWatch. Структуровані логи дозволяють отримувати точну та надійну інформацію про стан системи, що критично важливо для відстеження помилок.

Ще однією важливою перевагою PinoLogger є його можливість легко налаштовувати різні рівні логування для різних середовищ, таких як розробка, тестування або продакшн. Це дозволяє отримувати потрібну інформацію в залежності від контексту без надмірного навантаження на систему, а також надає можливість зберігати логування максимально ефективно. Крім того, PinoLogger може бути інтегрований з іншими інструментами для збору та обробки журналів, що робить його універсальним для різноманітних сценаріїв.

Одним з ключових аспектів, який робить PinoLogger таким важливим для сучасних розподілених систем, є його масштабованість. Завдяки високій продуктивності, асинхронності та підтримці потоків, бібліотека може використовуватися в великих, розподілених середовищах без втрати продуктивності, що є критичним для систем, які обробляють величезні обсяги даних і мають високі вимоги до часу відповіді. PinoLogger здатний забезпечити ефективне логування навіть у складних умовах багатопотокових або мікросервісних архітектур.

Завдяки своїй простоті в налаштуванні та інтеграції з іншими інструментами, PinoLogger є потужним інструментом для створення надійних та продуктивних рішень для ведення журналів. Легкість у налаштуванні дозволяє розробникам швидко впроваджувати її в існуючі проекти, зберігаючи високу ефективність і забезпечуючи детальний моніторинг роботи системи.

В результаті, PinoLogger є незамінним інструментом для будь-якої сучасної інформаційної системи, де важлива висока продуктивність, масштабованість та можливість детального моніторингу.

2.4 Структура БД

Структура бази даних є важливим аспектом розробки адаптивної рекомендаційної системи, оскільки вона забезпечує ефективне збереження, обробку та доступ до даних про товари, їхні історії продажів та фінансові показники.

У даній системі використовується реляційна база даних, що включає три основні моделі: **Product**, **ProductHistory** та **ProductHistoryWithoutRelation** (рис 2.1).

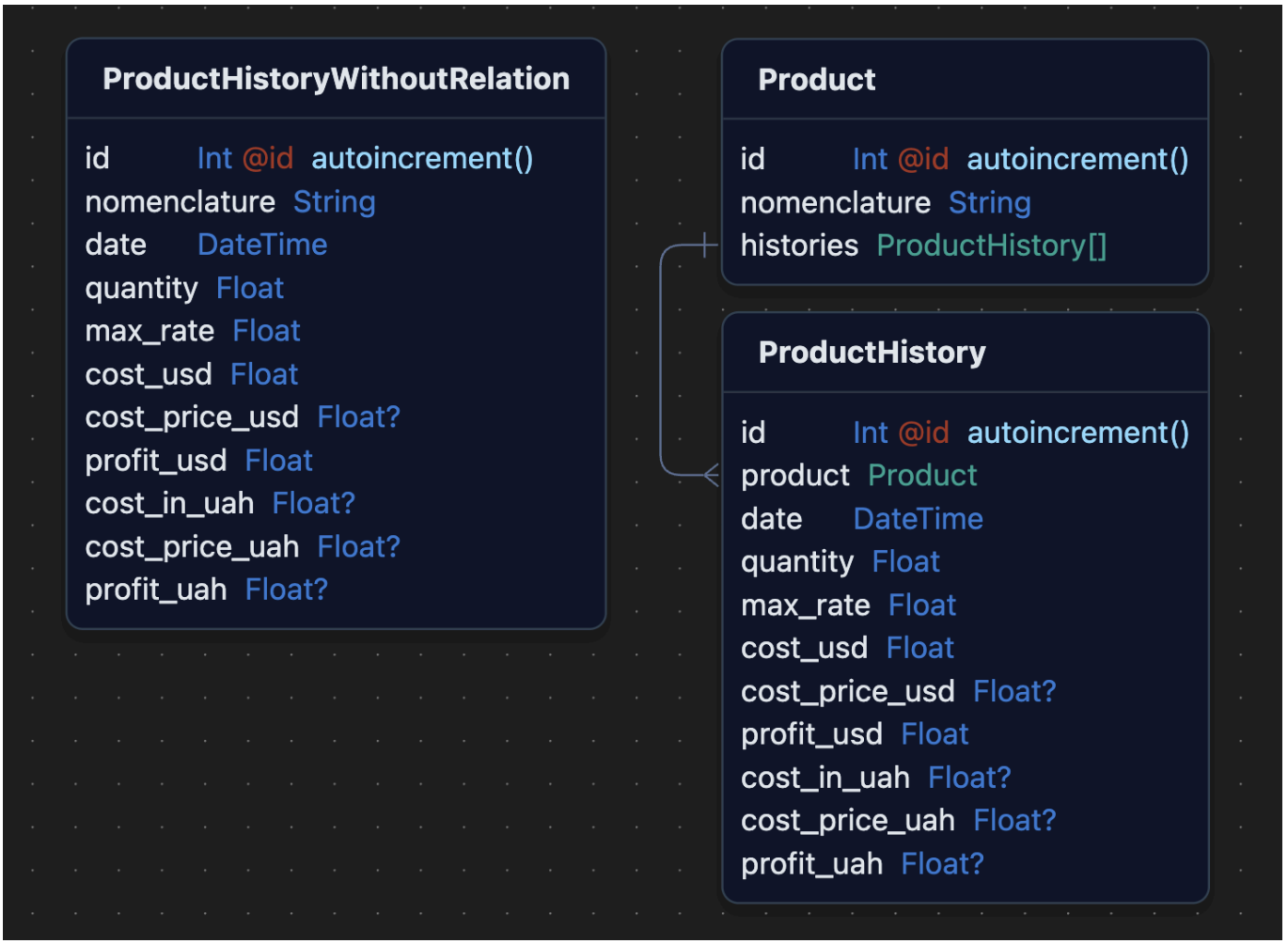


Рисунок 2.1 – Схема БД

Модель Product

Модель **Product** представляє інформацію про товари, які продаються в інтернет-магазині. Вона містить кілька важливих атрибутів:

- **id** - унікальний ідентифікатор товару, що є первинним ключем.
- **nomenclature** - унікальна назва товару, що слугує для ідентифікації товару в системі. Це поле є унікальним і використовується для подальших зв'язків з іншими таблицями.
- **histories** - зв'язок один до багатьох з таблицею **ProductHistory**, що зберігає історію продажів кожного товару, включаючи зафіксований курс валюти як в доларах так і в гривнях.

Ця модель є основою для збереження даних про кожен товар в системі, що дозволяє в подальшому проводити аналіз попиту, формувати рекомендації та здійснювати прогнозування.

Модель ProductHistory

Модель **ProductHistory** зберігає історичні дані про кожен товар, зокрема про його кількість, ціни, прибутки та інші фінансові показники за певні періоди часу. Важливі атрибути цієї моделі:

- **id** - унікальний ідентифікатор запису, що є первинним ключем.
- **productId** - зовнішній ключ, що вказує на зв'язок з моделлю **Product** через поле **id**.
- **date** - дата запису, що вказує на момент, коли товар був проданий або замовлений.
- **quantity** - кількість одиниць товару, що була продана чи отримана.
- **max_rate** - максимальна ставка обміну для товару на момент продажу.
- **cost_usd** - собівартість товару в доларах США.
- **cost_price_usd** - ціна закупівлі товару в доларах США (опціонально).
- **profit_usd** - прибуток від продажу товару в доларах США.
- **cost_in_uah** - собівартість товару в гривнях (опціонально).
- **cost_price_uah** - ціна закупівлі товару в гривнях (опціонально).
- **profit_uah** - прибуток від продажу товару в гривнях (опціонально).

Ця модель дозволяє вести детальний облік історії продажів кожного товару, що є важливим для подальшого прогнозування попиту і формування рекомендацій щодо замовлення.

Модель ProductHistoryWithoutRelation

Модель **ProductHistoryWithoutRelation** є подібною до моделі **ProductHistory**, але без зв'язку з іншими таблицями. Вона містить ті ж самі атрибути, однак тут

немає зв'язку з таблицею **Product**, що дозволяє зберігати історичні дані товарів без прив'язки до конкретних товарів у системі. Це може бути корисно, якщо необхідно працювати з зовнішніми джерелами даних, які не потребують інтеграції з іншими таблицями.

Основні атрибути цієї моделі:

- **id** - унікальний ідентифікатор запису.
- **nomenclature** - назва товару.
- **date** - дата продажу.
- **quantity** - кількість товару.
- **max_rate** - максимальна ставка обміну для товару на момент продажу.
- **cost_usd** - собівартість товару в доларах США.
- **cost_price_usd** - ціна закупівлі товару в доларах США (опціонально).
- **profit_usd** - прибуток від продажу товару в доларах США.
- **cost_in_uah** - собівартість товару в гривнях (опціонально).
- **cost_price_uah** - ціна закупівлі товару в гривнях (опціонально).
- **profit_uah** - прибуток від продажу товару в гривнях (опціонально).

Ця модель використовується для зберігання історичних даних без необхідності збереження зв'язків з іншими таблицями.

2.5. Висновки до розділу

В цьому розділі було розглянуто ключові інструменти, мови програмування та технології, які використовуються в рамках реалізації дипломної роботи. Зокрема, описано особливості таких мов програмування, як Python, C#, JavaScript і TypeScript, а також фреймворків і бібліотек, таких як React, Next.js, Nest.js та FastAPI. Окрему увагу приділено Tailwind CSS як сучасному інструменту стилізації інтерфейсів та Prisma ORM - ефективному засобу для взаємодії з базами даних. Також детально проаналізовано роль логування в розробці, зокрема використання бібліотеки PinoLogger для ефективного та продуктивного ведення журналів у системах, що

забезпечує високу швидкість обробки даних при мінімальних накладних витратах.

Особливу увагу було приділено використанню мови SQL як базового засобу для роботи з реляційними базами даних, що є основою для збереження та обробки даних у багатьох сучасних інформаційних системах. Показано, що комбінація згаданих технологій дозволяє створювати масштабовані, безпечні та продуктивні веб-застосунки з чіткою структурою коду і ефективним механізмом моніторингу через логування.

Таким чином, наведене інформаційне забезпечення демонструє доцільність вибору саме цих інструментів для реалізації поставлених у дипломній роботі завдань. Обраний стек технологій, включаючи PinoLogger для логування, сприяє розробці сучасного, надійного та зручного у користуванні програмного забезпечення, яке відповідає вимогам як розробників, так і кінцевих користувачів.

РОЗДІЛ 3. МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Загальна математична модель Prophet

У даній роботі для прогнозування часових рядів було використано бібліотеку **Prophet**, розроблену компанією Facebook(Meta). Prophet базується на адитивній моделі, яка враховує тренди, сезонність та вплив святкових днів або інших подій, що можуть впливати на динаміку показників.

Модель Prophet ґрунтується на припущенні, що часовий ряд можна описати як адитивну композицію таких компонентів:

$$y(t) = g(t) + s(t) + h(t) + \varepsilon_t \quad (3.1)$$

де:

$y(t)$ - значення досліджуваної змінної у момент часу t;

$g(t)$ - трендова функція (загальна тенденція зростання/спаду);

$s(t)$ - сезонна компонента;

$h(t)$ - вплив свят або інших зовнішніх подій;

ε_t - шум або випадкове відхилення (біла компонента шуму, $\varepsilon_t \sim N(0, \sigma^2)$);

N - стандартне відхилення шуму;

σ^2 - дисперсія шуму.

Моделювання тренду $g(t)$:

Prophet пропонує два основні типи тренду:

Лінійний тренд з точками зламу

$$g(t) = (k + a(t)^T\delta)t + (m + a(t)^T\gamma) \quad (3.2)$$

де:

k - початковий коефіцієнт нахилу (slope),

m - початкове зміщення (intercept),

$a(t)$ - вектор індикаторів для точок зламу (change points),

δ - вектор зміни нахилу в кожній точці зламу,

$\gamma_i = -s_j\delta_j$, де s_j момент виникнення j-ї точки зламу.

Цей підхід дозволяє моделі адаптуватися до змін тренду, наприклад, під час економічних криз або змін попиту.

Логістичний тренд (для обмеженого зростання)

$$g(t) = \frac{C}{1+e^{-k(t-m))}} \quad (3.3)$$

де:

C - граничне значення (cap), до якого прямує ряд;

k, m - як у лінійному випадку.

Сезонність $s(t)$

Сезонність моделюється за допомогою рядів Фур'є:

$$s(t) = \sum_{n=1}^{N}\left[a_n cos(\frac{2\pi nt}{P}) + b_n sin(\frac{2\pi nt}{P})\right] \quad (3.4)$$

де:

P - період сезонності (наприклад, 365.25 днів для річної);

N - порядок Фур'є-ряду (визначає рівень деталізації);

a_n, b_n - коефіцієнти, які оцінюються при навчанні моделі.

Зазвичай використовується регуляризація (L2-норма) для запобігання перенавчанню:

$$\lambda \sum_{n=1}^{N} (a_n^2 + b_n^2) \quad (3.5)$$

Компонента свят $h(t)$

Ця компонента моделює вплив подій/свят:

$$h(t) = \sum_j \theta_j Z_j(t) \quad (3.6)$$

де:

$Z_j(t)$ - індикатор наявності свята j у момент часу t (1 - якщо є, 0 - інакше);

θ_j - ефект свята j, який також визначається при навчанні моделі.

Шумова компонента ε_t

Залишкова похибка моделі зазвичай вважається нормально розподіленою:

$$\varepsilon_t \sim N(0, \sigma^2) \tag{3.7}$$

Але в Prophet також можна використовувати метод бутстрепінгу для побудови довірчих інтервалів без припущення нормальності.

Навчання моделі

Оцінка параметрів $k,\ m,\ \delta,\ a_n,\ b_n,\ \theta_j$ здійснюється шляхом оптимізації функції втрат, що враховує:

- помилку прогнозу;
- регуляризацію (особливо для сезонних і святкових компонентів);
- апріорні розподіли (може бути використано байєсівський підхід із MCMC через Stan, або оптимізація максимуму апріорного правдоподібного значення).

Прогнозування

Прогноз формується як:

$$\widehat{y}(t) = \widehat{g}(t) + \widehat{s}(t) + \widehat{h}(t) \tag{3.8}$$

Крім того, Prophet будує довірчі інтервали для прогнозу, зазвичай з рівнем довіри 80% або 95%, які враховують невизначеність у оцінці компонентів.

3.2 Загальна математична модель SARIMA

У даній роботі також було використано модель **SARIMA** (*Autoregressive Integrated Moving Average*), яка є одним із найпоширеніших статистичних методів аналізу та прогнозування часових рядів. Ця модель поєднує три ключові компоненти.

Авторегресійна частина (AR) описує залежність поточного значення ряду від його попередніх значень. Інтегруюча складова (I) використовується для перетворення нестаціонарного ряду на стаціонарний шляхом диференціювання.

Нарешті, компонент ковзного середнього (MA) згладжує випадкові коливання, враховуючи попередні помилки прогнозування.

Завдяки такій комбінації SARIMA здатна ефективно працювати з різними типами часових рядів, навіть якщо вони містять тренди або сезонні коливання .

Модель позначається як SARIMA(p, d, q),

де:

p - порядок авторегресії;

d - ступінь диференціювання;

q - порядок згладжування (ковзного середнього).

Формалізація моделі SARIMA. Розглянемо детальніше математичну постановку.

Авторегресійна модель порядку p має вигляд:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \varepsilon_t \quad (3.9)$$

де:

X_t - значення часового ряду в момент часу ttt,

ϕ_p - коефіцієнти авторегресії,

ε_t - випадкова похибка з нормальним розподілом.

Модель ковзного середнього (MA). Модель MA порядку q записується як:

$$X_t = \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} \quad (3.10)$$

де θ_q - параметри моделі ковзного середнього.

Інтегрування (I). Інтегрування відповідає за приведення часового ряду до стаціонарного вигляду шляхом диференціювання. Перше диференціювання:

$$Y_t = X_t - X_{t-1} = \nabla X_t \quad (3.11)$$

друге диференціювання:

$$\nabla^2 X_t = \nabla(\nabla X_t) = X_t - 2X_{t-1} + X_{t-2} \quad (3.12)$$

Таким чином, для будь-якого $d \in N, \nabla^d X_t$ означає, що ряд був диференційований d разів.

Повна модель SARIMA(p, d, q). Після застосування d-разового диференціювання до нестаціонарного часового ряду X_t, отримуємо стаціонарний ряд $Y_t = \nabla^d X_t$, для якого будується модель:

$$Y_t = \phi_1 Y_{t-1} + \cdots + \phi_p Y_{t-p} + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q} \quad (3.13)$$

Або у вигляді з використанням операторів зсуву (backshift operator B):

$$\phi(B)\nabla^d X_t = \theta(B)\varepsilon_t \quad (3.14)$$

де:

$\phi(B) = 1 - \phi_1 B - \cdots - \phi_p B^p$,

$\theta(B) = 1 - \theta_1 B - \cdots - \theta_p B^p$,

$BX_t = X_{t-1}$,

$\nabla = 1 - B$.

Ідентифікація параметрів моделі. Перед побудовою моделі SARIMA важливо правильно визначити параметри p, d, та q, оскільки від них залежить якість прогнозу. Зазвичай для цього використовуються кілька статистичних підходів.

Порядок q(для частини Moving Average) оцінюється за допомогою автокореляційної функції (ACF), яка показує, наскільки значення ряду корелюють із власними попередніми значеннями. Порядок p(для авторегресійної частини) визначається за частковою автокореляційною функцією (PACF), що дозволяє зрозуміти залежність між поточним значенням та попередніми, з виключенням

проміжних впливів. Ступінь інтегрування d, який вказує, скільки разів потрібно диференціювати ряд для досягнення стаціонарності, зазвичай визначається за допомогою тестів на стаціонарність, зокрема популярного ADF-тесту (Augmented Dickey-Fuller).

Оцінка параметрів. Параметри ϕ_i та θ_j зазвичай оцінюються методом максимального правдоподібності або методом найменших квадратів. Максимізація функції правдоподібності базується на мінімізації залишкової дисперсії:

$$L = -\frac{n}{2} log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{t=1}^{1} \varepsilon_t^2 \tag{3.15}$$

σ^2 - оцінка дисперсії похибки.

Прогнозування. Після побудови моделі SARIMA її можна використовувати для прогнозування майбутніх значень ряду. Прогноз на h-кроків уперед обчислюється рекурсивно, використовуючи отримані параметри та попередні значення ряду:

$$X_{t+h} = f(X_t, X_{t-1}, \cdots) \tag{3.16}$$

де f - функція, визначена структурою моделі.

Оцінка якості моделі. Для оцінки якості моделі використовуються такі метрики:
Середньоквадратична похибка (MSE - Mean Squared Error):

$$MSE = \frac{1}{n}\sum_{i=1}^{n} (X_i - \widehat{X}_i)^2 \tag{3.17}$$

Середня абсолютна похибка (MAE - Mean Absolute Error):

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |X_i - \widehat{X}_i| \tag{3.18}$$

Інформаційний критерій Акайке (AIC - Akaike Information Criterion):

$$AIC = 2k - 2ln(L) \quad (3.19)$$

Отож, модель SARIMA є потужним інструментом для моделювання часових рядів, які демонструють автокореляцію, тренди, та мають нестаціонарний характер. Вона базується на строгому математичному апараті, що дозволяє обґрунтовано підійти до побудови та аналізу прогнозів. Математичне забезпечення включає як класичні статистичні методи, так і сучасні алгоритми оптимізації для добору параметрів моделі.

3.3 Загальна математична модель LSTM-мережі

У даній роботі також було використано модель довгострокової короткочасної пам'яті (LSTM, від англ. Long Short-Term Memory) - це різновид рекурентних нейронних мереж (RNN), спеціально розроблений для моделювання довготривалих залежностей у часових рядах або інших послідовностях даних. LSTM-мережі здатні зберігати інформацію протягом тривалого періоду часу завдяки своїй унікальній архітектурі, яка включає так звані комірки пам'яті та керуючі гейти.

Стандартні RNN, хоча і добре підходять для обробки послідовностей, часто страждають від проблеми затухаючого градієнта (vanishing gradient), що унеможливлює ефективне навчання на довгих послідовностях. Архітектура LSTM подолала ці недоліки завдяки введенню спеціальних механізмів контролю потоку інформації.

Архітектура комірки LSTM. Одна LSTM-комірка на кожному часовому кроці t виконує набір обчислень для оновлення внутрішнього стану пам'яті C_t і прихованого стану h_t. Основними компонентами є три гейти:

- **Вентиль забування** (*forget gate*) f_t
- **Вхідний вентиль** (*input gate*) i_i
- **Вихідний вентиль** (*output gate*) o_t

Крім того, формується *кандидат на нове значення пам'яті* $\widehat{C}_t$, який оновлює стан пам'яті. Розглянемо формули, які описують функціонування однієї LSTM-комірки:

Вентиль забування

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.20}$$

Цей вентиль визначає, яку частину попереднього стану пам'яті C_{t-1} потрібно зберегти.

Вхідний вентиль:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.21}$$

$$\widehat{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.22}$$

Ці два рівняння відповідають за оновлення пам'яті. i_t визначає, які значення змінюються, а $\widehat{C}_t$ - нові кандидатні значення.

Оновлення стану пам'яті:

$$C_t = f_t \odot C_{t-1} + i_t \odot \widehat{C}_t \tag{3.23}$$

Стан пам'яті комбінується з урахуванням старого значення та нового кандидата.

Вихідний вентиль:

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o \tag{3.24}$$

$$h_t = o_t \odot tanh(C_t) \tag{3.25}$$

Ці рівняння відповідають за обчислення нового прихованого стану, який передається до наступної комірки або на вихід моделі де:

x_t - вхідний вектор у момент часу t,

h_{t-1} - прихований стан попереднього кроку,

σ - сигмоїдна функція активації,

$tanh$ - гіперболічний тангенс,

$\odot$ - покомпонентне множення.

Переваги LSTM перед звичайними RNN. LSTM має кілька важливих переваг порівняно з традиційними RNN. По-перше, завдяки вбудованому механізму пам'яті ці моделі здатні зберігати інформацію протягом десятків, сотень або навіть тисяч часових кроків. Це особливо важливо для задач, де потрібен довготривалий контекст. По-друге, наявність трьох вентилів дозволяє ефективно управляти потоком інформації - контролювати, що зберігати, що оновлювати, а що передавати на вихід. І нарешті, завдяки стабільному внутрішньому стану пам'яті, LSTM значно знижує ризик виникнення проблем із затухаючими або вибуховими градієнтами, що робить процес навчання набагато стабільнішим і ефективнішим.

Математична модель навчання LSTM. Процес навчання LSTM здійснюється з використанням методу зворотного поширення помилки через час (*Backpropagation Through Time*, BPTT). Функція втрат (наприклад, середньоквадратична помилка або крос-ентропія) диференціюється за параметрами ваг:

$$\frac{\partial L}{\partial W} = \sum_t \frac{\partial L}{\partial h_t} \cdot \frac{\partial h_t}{\partial W} \tag{3.26}$$

Тренування включає оновлення ваг згідно методу градієнтного спуску:

$$W \leftarrow W - \eta \cdot \frac{\partial L}{\partial W} \tag{3.27}$$

де:

η - швидкість навчання,

L - функція втрат,

W - матриці ваг (усі W_f, W_i, W_C, W_o одночасно).

Застосування LSTM до часових рядів. LSTM-мережі застосовуються для прогнозування/класифікації часових рядів, що в свою чергу надає можливість робити відповідні рекомендації спираючись на отримані результати. Вхідними даними є вектори ознак, впорядковані у часі. Вихідними - відповідні значення, які потрібно передбачити. Таким чином, математичне забезпечення на основі архітектури LSTM дозволяє реалізувати ефективну модель, здатну до навчання з часових залежностей та прогнозування майбутніх значень з високою точністю. Детальна структура комірки та обчислювальні формули забезпечують основу для реалізації алгоритму та його адаптації під конкретні завдання дипломної роботи дослідження.

3.4. Висновки до розділу

У цьому розділі було розглянуто математичне забезпечення трьох популярних підходів до прогнозування часових рядів: Prophet, SARIMA та LSTM.

Prophet - це адитивна модель, що поєднує тренд, сезонність та вплив свят. Вона є зручним інструментом для користувачів без глибоких знань у сфері статистики. Завдяки вбудованій обробці точок зламу та використанню рядів Фур'є для моделювання сезонності, Prophet добре підходить для бізнес-застосунків із чіткими календарними структурами.

SARIMA базується на класичній статистичній моделі авторегресії, інтегрування та ковзного середнього. Вона потребує стаціонарності ряду та вимагає ретельного підбору параметрів. Незважаючи на свою простоту, SARIMA демонструє хорошу ефективність на лінійних і не надто складних даних.

LSTM (Long Short-Term Memory) - це різновид рекурентних нейронних мереж, здатний враховувати довгострокові залежності в послідовностях. На відміну від Prophet і SARIMA, LSTM не потребує стаціонарності ряду й здатна моделювати нелінійні, складні взаємозв’язки між елементами часового ряду. Проте для її успішного застосування потрібні великі обсяги даних, потужні ресурси та ретельна настройка архітектури мережі.

Таким чином, кожен з розглянутих методів має свої переваги та обмеження. Вибір оптимального підходу залежить від специфіки задачі, доступних даних, необхідної точності та обчислювальних можливостей.

РОЗДІЛ 4. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Середовище розробки

Для розробки програмного забезпечення в рамках даної дипломної роботи було обрано текстовий редактор Visual Studio Code (рис.4.1). Цей інструмент є одним із найпопулярніших серед розробників завдяки своїй легкості, функціональності та широкій підтримці різних мов програмування. Visual Studio Code надає зручний та інтуїтивно зрозумілий інтерфейс, що значно полегшило процес написання коду.

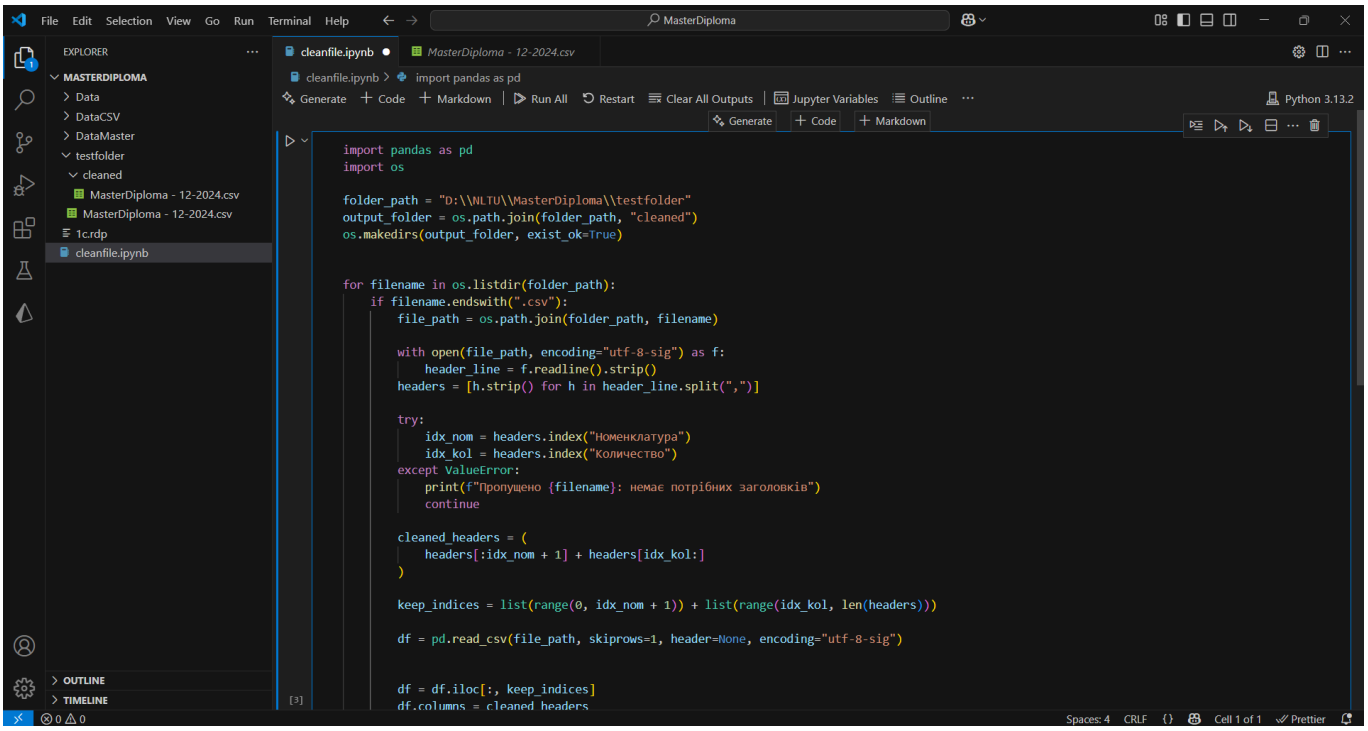


Рисунок 4.1 – Інтерфейс розробленої програми в середовищі Visual Studio Code

Важливою особливістю цього середовища є наявність інтелектуального автодоповнення коду(підказок), що дозволяє не лише пришвидшити розроблення, але й знизити ймовірність помилок, пов’язаних із синтаксисом або використанням API. Крім того, Visual Studio Code підтримує широкий спектр розширень і плагінів, які допомагають адаптувати його під специфічні потреби проєкту - від налагодження та форматування коду до інтеграції з системами контролю версій.

Особливою перевагою є тісна інтеграція з Git - однією з найпоширеніших систем контролю версій, що дозволило керувати історією змін, виконувати коміти,

зливання гілок та інші операції безпосередньо в середовищі розробки. Вбудовані можливості для налагодження коду дали змогу швидко виявляти та виправляти помилки, що є критично важливим у процесі створення якісного та стабільного програмного забезпечення.

Ще однією важливою перевагою Visual Studio Code є його кросплатформеність - він однаково добре працює на операційних системах Windows, macOS та Linux, що робить його універсальним вибором для розробників які в роботі використовують різні операційні системи.

Загалом, Visual Studio Code забезпечує високий рівень продуктивності та є зручним інструментом розробки, що дозволяє реалізовувати поставлені завдання в межах даної дипломної роботи.

4.2 Система управління базами даних

Для зберігання та обробки даних у рамках дипломної роботи була використана реляційна система управління базами даних PostgreSQL. Це потужна, надійна і відкрита СУБД, яка широко застосовується у розробці як невеликих, так і масштабних програмних рішень.

Вибір PostgreSQL обґрунтований її багатофункціональністю, високою продуктивністю, стійкістю до навантажень та що особливо важливо це простота у використанні. Ця система підтримує складні запити, транзакції з гарантією цілісності даних, а також розширені типи даних.

Однією з ключових переваг PostgreSQL є її підтримка стандарту SQL разом із численними розширеннями, які дозволяють ефективно працювати з геопросторовими даними, JSON-документами, масивами та іншими складними структурами. Завдяки цьому розробники можуть використовувати універсальні та спеціалізовані інструменти для обробки інформації без необхідності залучення додаткових систем.

Крім того, PostgreSQL відзначається високим рівнем безпеки та можливістю налаштування прав доступу, що дозволяє захистити дані від несанкціонованого використання. Вбудовані механізми резервного копіювання та відновлення сприяють

підтримці стабільності роботи системи, адже під час тестування нових змін, можливість швидко та легко зробити копію бази даних, є значною перевагою.

Інтеграція PostgreSQL із середовищем розробки Visual Studio Code та іншими інструментами дозволила організувати зручний та ефективний робочий процес, що забезпечує швидкий доступ до даних і їхнє надійне збереження.

Таким чином, використання PostgreSQL у дипломній роботі забезпечило надійність, масштабованість та високу продуктивність системи управління даними.

Для управління базою даних та зручної роботи з SQL-запитами під час розробки проєкту використовувався графічний клієнт Beekeeper Studio (рис.4.2). Це сучасний, кросплатформений інструмент з відкритим вихідним кодом, який надає розробникам можливість ефективно взаємодіяти з різними системами управління базами даних, зокрема з PostgreSQL.

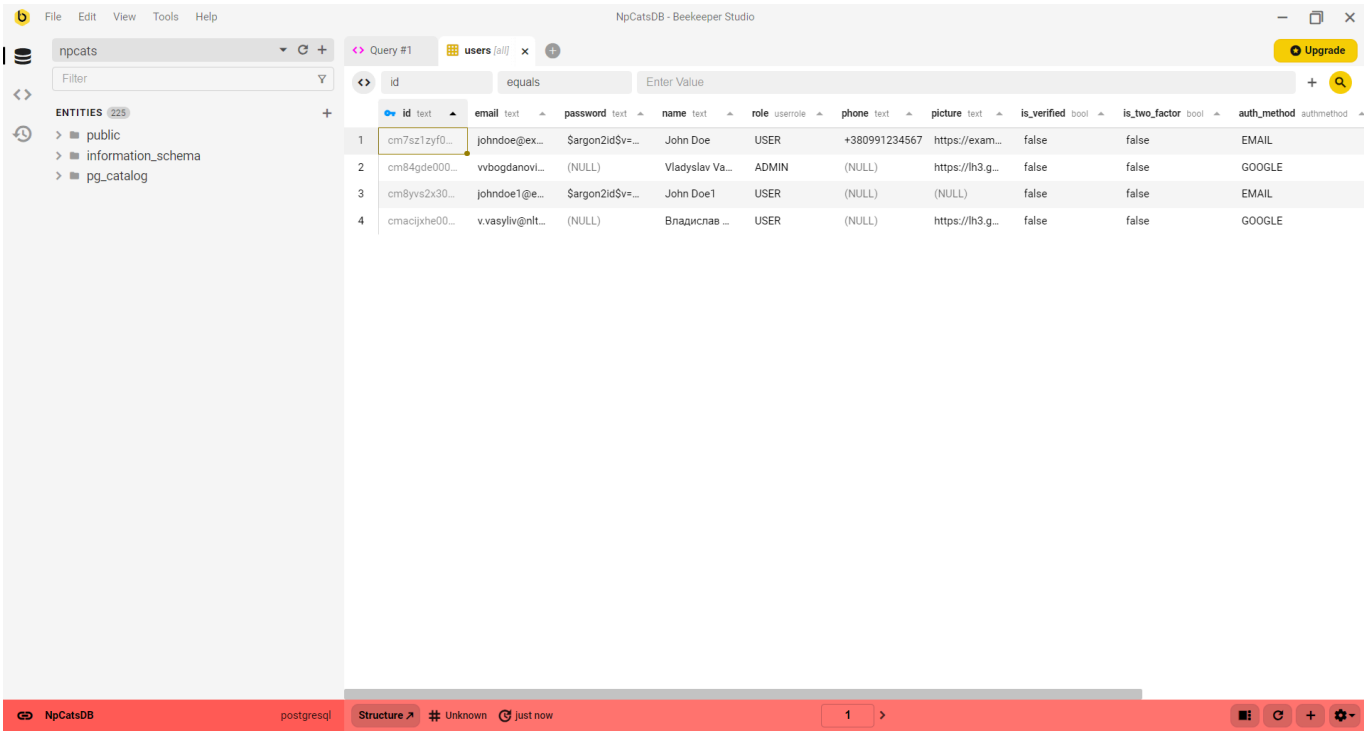


Рисунок 4.2 – Перегляд таблиць БД в Beekeeper Studio

Beekeeper Studio вирізняється простим і зрозумілим інтерфейсом, який дозволяє швидко створювати, виконувати і редагувати SQL-запити, а також переглядати результати їх виконання у зручному табличному форматі. Такий підхід значно спрощує процес тестування та налагодження роботи з базою даних.

Інструмент підтримує управління кількома з’єднаннями одночасно, що особливо корисно під час розробки складних проєктів з декількома середовищами (розробка, тестування, продакшн). Beekeeper Studio також надає можливість збереження запитів, що дозволяє повторно використовувати їх без необхідності написання з нуля.

Особливо корисною є підтримка вкладок і пошуку по таблицях, що прискорює навігацію у великих базах даних. Крім того, вбудовані інструменти для імпорту та експорту даних допомагають організувати ефективний обмін інформацією між системами.

Завдяки своїй простоті та функціональності Beekeeper Studio є зручним інструментом для розробників у межах даної дипломної роботи, забезпечуючи комфортну і продуктивну роботу з базою даних.

4.3 Інструмент попередньої обробки звітних даних

У процесі роботи над дипломною роботою, виникла потреба у попередній обробці звітних файлів, що генеруються у форматі Excel. Ці звіти містять велику кількість службової інформації, яка не має аналітичної цінності та заважає подальшому аналізу. Крім того, формат числових значень у таких звітах часто не відповідає вимогам, прийнятим у системах обробки даних, що унеможливлює їхнє пряме використання в програмному забезпеченні проєкту.

Для вирішення цього завдання було обрано Google Sheets - хмарний табличний редактор, який надає інструментарій для ручного редагування та підготовки даних. Google Sheets забезпечує простоту у використанні, доступ з будь-якого пристрою, а також автоматичне збереження змін, що мінімізує ризик втрати даних.

Основне застосування Google Sheets у рамках проєкту полягає в очищенні звітів від зайвої службової інформації, яка могла з’являтися у випадкових місцях таблиці внаслідок некоректної роботи стороннього генератора звітів. Завдяки інтерфейсу навігації по великому обсягу даних цей процес був значно спрощений.

Крім того, Google Sheets дозволяє легко переформатувати числові значення у стандартний вигляд з фіксованою кількістю знаків після коми (наприклад, 0.00), що є

критично важливим для забезпечення коректного зчитування даних при подальшому експорті до формату CSV. Саме у форматі CSV очищені таблиці використовувалися у подальшій роботі програмного модуля, оскільки цей формат є зручним для імпорту, обробки та аналізу.

Таким чином, Google Sheets виконував роль проміжного етапу - середовища для підготовки, очищення та стандартизації даних перед їх подальшим автоматизованим аналізом. Його використання дозволило суттєво підвищити якість вхідних даних та забезпечити стабільну роботу системи. Після очищення буде сформовано набір даних, приклад якого наведено на рисунку 4.3.

Проте після експорту даних у формат CSV виникає проблема з наявністю чотирьох порожніх колонок. Це зумовлено тим, що, як показано на рисунку 4.3, колонки від A до E включно в оригінальному файлі є об'єднаними.

A1:E1 | fx Номенклатура

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Номенклатура					Кількість	Курс (максимальний за період)	Вартість	Собівартість	Прибуток	Вартість в нац валюті	Собівартість в нац валюті	Прибуток в нац валюті
2	(НЗ) Вулична ретро Smart гірлянда UKC 20 ламп 20 метрів Tuya + пульт (7920)					1.00	41.54	65.69	40.80	24.89	2728.76	1694.83	1033.93
3	USB Wi-Fi мережний адаптер PIX-LINK Wi-Fi 802.11n LV-UW 10-2DB (2367)					2.00	41.47	6.69	4.08	2.61	274.90	167.65	107.26
4	Абсорбер (демпфер) підсилювача бампера переднього (піна) Tesla Model 3 (1104669-00-F)					1.00	41.49	13.09	10.92	2.17	543.10	453.07	90.03
5	Автоматичний перемикач введення резерву (реле) EARU EAATS-F-CG-63A 2P (ABP/ATS) 1-фаза					1.00	40.96	27.38	18.77	8.61	1121.48	768.82	352.67
6	Автоматичний перемикач введення резерву (реле) EARU EAATS-F-CG-63A 4P (ABP/ATS) 3-фази					3.00	41.55	98.54	69.33	29.21	4091.05	2878.35	1212.70
7	Автоматичний перемикач введення резерву (реле) TOMZN TOQ5-125/4P 125A 4P (ABP) 3-фази					1.00	41.38	32.59	17.82	14.77	1348.57	737.39	611.18
8	Автоматичний перемикач введення резерву (реле) TOMZN TOQ5-63/2 63A 2P (ABP) 1-фаза					5.00	41.47	128.00	80.90	47.10	5273.06	3332.76	1940.31
9	Автоматичний перемикач введення резерву (реле) TOMZN TOQ7-63/2P 63A 2P (ABP) 1-фаза					2.00	41.46	58.34	32.40	25.94	2406.82	1336.66	1070.15
10	Автоматичний перемикач введення резерву (реле) TOMZN TOQ7-63/4P 63A 4P (ABP) 3-фази					1.00	41.11	32.59	17.93	14.66	1339.77	737.10	602.67
11	Автоматичний перемикач введення резерву EARU EAATS-125 125A 4P (ABP) 3-фази					1.00	40.95	34.21	22.07	12.14	1400.90	903.77	497.13
12	Автоматичний перемикач введення резерву TOMZN TOQ5-63/4P 63A 4P (ABP) 3-фази					14.00	41.38	406.67	249.76	156.91	16743.36	10282.98	6460.38
13	Автомобільна USB зарядка від прикурювача 12v HZ HC-1 LCD з екраном Black (3101)					2.00	41.46	5.85	2.86	2.99	241.08	117.86	123.22
14	Автомобільна підніжка на петлю дверей для доступу на дах авто					3.00	41.54	14.84	8.10	6.74	613.52	334.77	278.75
15	Автомобільний магнітний тримач для телефону Hoco CA61 для панелі приладів (96394)					1.00	40.96	3.06	2.95	0.11	125.34	120.83	4.51
16	Автомобільний монітор (екран) UKC TFT LCD 4,3" для двох камер (1309)					2.00	41.54	24.72	14.28	10.44	1024.94	592.05	432.89
17	Автомобільний монітор (екран) для камери заднього виду (підтримує 2 камери) TFT 7" (8321)					1.00	41.50	24.09	14.28	9.81	999.74	592.62	407.12

Рисунок 4.3 – Очищений набір даних

У результаті цього експортується кілька зайвих порожніх стовпців, які ускладнюють подальшу обробку даних(рис 4.4).

Цю проблему можна вирішити двома способами. Перший спосіб - це ручне скасування об'єднання клітинок та видалення порожніх колонок у кожному файлі. Другий, більш ефективний, полягає в автоматизації процесу: файли експортуються у поточному вигляді, після чого спеціальний скрипт, написаний мовою Python, відкриває кожен файл (загальна кількість яких перевищує 50) та видаляє з них порожні колонки.

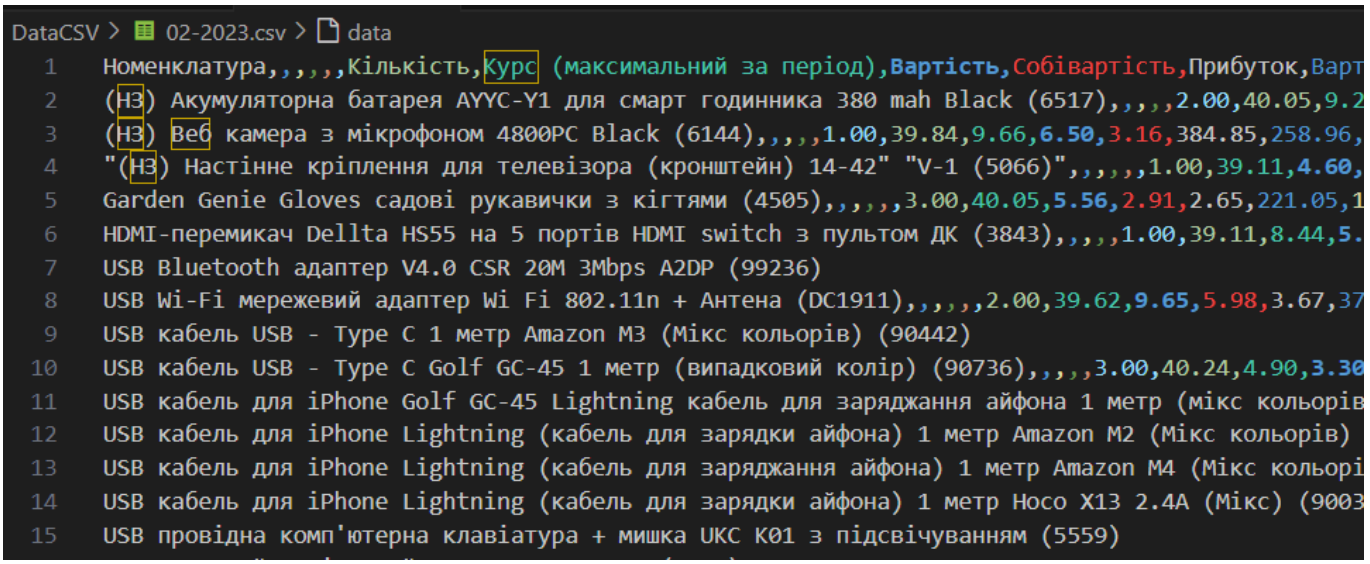


Рисунок 4.4 – Звіт в форматі CSV після експорту

Також варто врахувати, що деякі товари містять у своїй назві символ коми. У форматі CSV такі значення автоматично беруться в лапки, щоб уникнути порушення структури файлу під час імпорту. Водночас інші поля, які не містять розділових символів, залишаються без лапок. У результаті цього виникає неоднорідність у форматуванні, що може ускладнити подальшу обробку даних.

З метою забезпечення коректної інтерпретації кожного було реалізовано механізм, який під час обробки файлу автоматично враховує наявність лапок або, за потреби, огортає всі текстові значення у лапки (рис 4.5), забезпечуючи єдиний формат представлення даних.

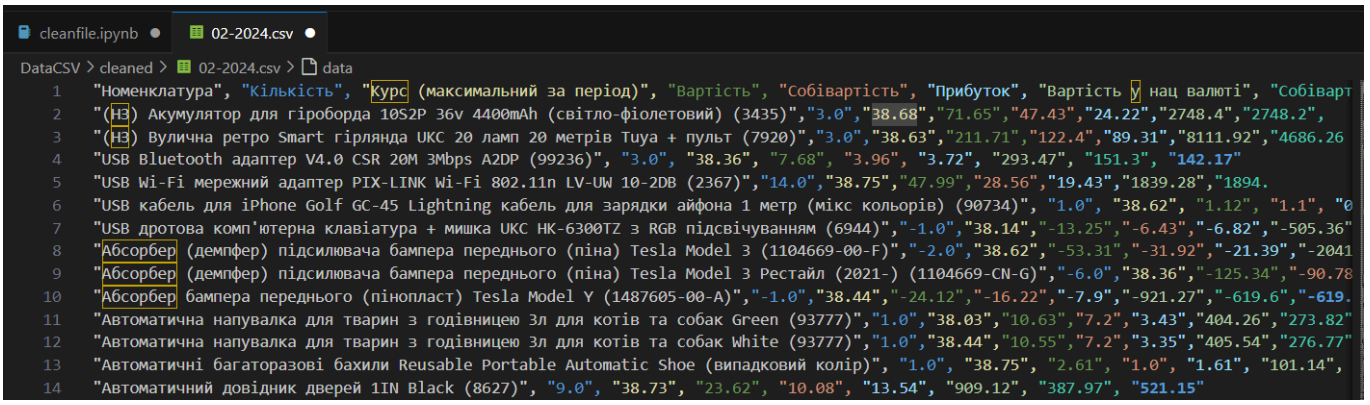


Рисунок 4.5 – Звіт в форматі CSV після експорту

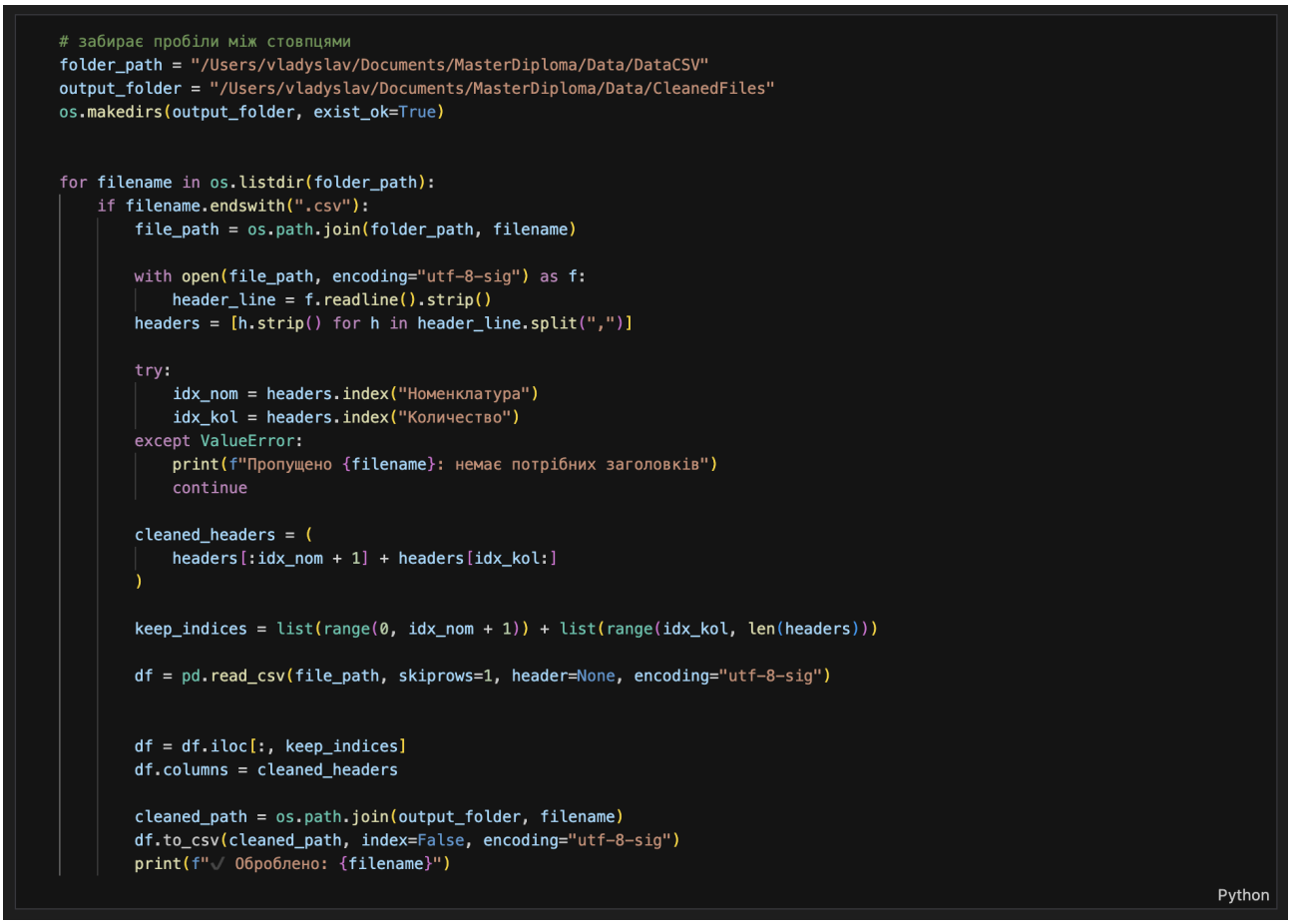
Ще один із найбільш важливих етапів - це вибір стовпців для подальшого збереження. В даному випадку зчитуються лише ті стовпці, які містять інформацію

після очищення. Це дозволяє зберегти тільки релевантні дані і значно зменшити обсяг інформації, яка буде використовуватись у подальшому. Завдяки цьому кроку обробка даних стає більш ефективною, оскільки ми працюємо лише з необхідною інформацією.Очищення даних є одним із найважливіших етапів у роботі з великими обсягами даних. Це процес, що включає в себе виявлення та усунення помилок у наборах даних, коригування некоректних значень, видалення зайвих пробілів та інших непотрібних елементів, а також перетворення даних у такий вигляд, який буде придатний для подальшого аналізу або обробки. Під час реалізації даного етапу важливо забезпечити максимальну точність і ефективність, адже навіть незначні помилки у даних можуть значно вплинути на кінцеві результати.

У рамках дипломної роботи очищення даних стало необхідним кроком для підготовки інформації, що буде використовуватись для аналізу. Однією з типових задач, з якими стикається кожен аналітик даних, є необхідність очищення великих наборів даних від непотрібних символів, пробілів або невірних значень, що можуть з'являтися в результаті некоректних введень або помилок при експорті з інших систем.

У наведеному коді на рисунку 4.6 реалізовано процес очищення CSV-файлів, що зберігаються у папці folder_path. Код починає свою роботу з визначення шляху до директорії, де зберігаються вихідні файли, а також папки, куди будуть збережені очищені дані - output_folder. Цей крок є важливим, оскільки дозволяє організувати збереження результатів обробки в окрему папку, щоб не змішувати вихідні дані з очищеними.

Далі відбувається перебір всіх файлів у зазначеній директорії. Для кожного файлу з розширенням .csv виконується відкриття і обробка його вмісту. Першим кроком є зчитування заголовків стовпців із першого рядка файлу. Тут важливо забезпечити коректне зчитування даних з урахуванням можливих зайвих пробілів, що можуть бути присутніми між значеннями в заголовках. Цей крок сприяє нормалізації заголовків, що є важливим для подальшої обробки.



```python
# забирає пробіли між стовпцями
folder_path = "/Users/vladyslav/Documents/MasterDiploma/Data/DataCSV"
output_folder = "/Users/vladyslav/Documents/MasterDiploma/Data/CleanedFiles"
os.makedirs(output_folder, exist_ok=True)


for filename in os.listdir(folder_path):
    if filename.endswith(".csv"):
        file_path = os.path.join(folder_path, filename)

        with open(file_path, encoding="utf-8-sig") as f:
            header_line = f.readline().strip()
        headers = [h.strip() for h in header_line.split(",")]

        try:
            idx_nom = headers.index("Номенклатура")
            idx_kol = headers.index("Количество")
        except ValueError:
            print(f"Пропущено {filename}: немає потрібних заголовків")
            continue

        cleaned_headers = (
            headers[:idx_nom + 1] + headers[idx_kol:]
        )

        keep_indices = list(range(0, idx_nom + 1)) + list(range(idx_kol, len(headers)))

        df = pd.read_csv(file_path, skiprows=1, header=None, encoding="utf-8-sig")


        df = df.iloc[:, keep_indices]
        df.columns = cleaned_headers

        cleaned_path = os.path.join(output_folder, filename)
        df.to_csv(cleaned_path, index=False, encoding="utf-8-sig")
        print(f"✓ Оброблено: {filename}")
```

Python

Рисунок 4.6 – Фрагмент алгоритму очищення даних

Наступний етап - пошук необхідних стовпців, які повинні бути присутніми в кожному файлі. В даному випадку шукаються стовпці з назвами "Номенклатура" та "Кількість". Якщо такі стовпці не знаходяться, код виводить повідомлення про помилку і переходить до наступного файлу. Така перевірка є важливою, оскільки вона дозволяє переконатись, що всі необхідні дані присутні, і в разі їх відсутності уникнути помилок в подальшій обробці.

Коли необхідні стовпці знайдені, код формує нові заголовки для даних, що залишаються після видалення непотрібних стовпців між "Номенклатура" та "Кількість". Таким чином, ми очищаємо набір даних від зайвих стовпців, що не несуть корисної інформації для подальшого аналізу.

Після виконання всіх етапів очищення, результат зберігається у новому файлі в директорії output_folder, що дозволяє зберігати чисті дані окремо від початкових. Таким чином, у результаті отримуємо очищений файл, який готовий для подальшого аналізу або використання в інших системах.

Цей код є простим, але потужним інструментом для автоматизації процесу очищення даних. Він дозволяє швидко та ефективно обробляти великі обсяги CSV-файлів, забезпечуючи їх нормалізацію та підготовку до подальшого аналізу. Використання цього підходу дозволяє значно зекономити час і зусилля, які зазвичай витрачаються на ручне очищення даних, і забезпечує високу якість та точність результатів.

Наведений код на рисунку 4.7 є повним описом процесу обробки CSV-файлів, де кожне значення в стовпцях огортається в лапки для забезпечення правильного формату при збереженні даних.

```python
# огортає в лапки кожне значення
input_dir = "/Users/vladyslav/Documents/MasterDiploma/Data/CleanedFiles"
output_dir = "/Users/vladyslav/Documents/MasterDiploma/Data/QuotedCleanedFiles"
os.makedirs(output_dir, exist_ok=True)

for filename in os.listdir(input_dir):
    if not filename.endswith('.csv'):
        continue

    file_path = os.path.join(input_dir, filename)
    try:
        df = pd.read_csv(file_path)

        # Очищуємо пробіли і зайві лапки у Номенклатура
        if 'Номенклатура' in df.columns:
            df['Номенклатура'] = (
                df['Номенклатура']
                .astype(str)
                .str.strip()
                .str.strip('"')
            )

        # Видаляємо порожні колонки між Номенклатура і Кількість
        cols = df.columns.tolist()
        if 'Номенклатура' in cols and 'Кількість' in cols:
            i1 = cols.index('Номенклатура')
            i2 = cols.index('Кількість')
            if i2 - i1 > 1:
                df.drop(columns=cols[i1+1:i2], inplace=True)

        # Зберігаємо з обгорткою ВСІХ значень у лапки
        output_path = os.path.join(output_dir, filename)
        df.to_csv(output_path, index=False, quoting=csv.QUOTE_ALL)

        print(f"✅ {filename} → обгорнуто в лапки та збережено.")
    except Exception as e:
        print(f"❌ Помилка в {filename}: {e}")
```

Python

Рисунок 4.7 – Фрагмент алгоритму форматування даних

Така операція важлива для того, щоб уникнути проблем із зчитуванням та інтерпретацією даних у майбутньому, особливо в тих випадках, коли значення можуть містити роздільники, такі як коми або інші символи.

Першим кроком у коді є визначення шляхів до вхідної та вихідної директорій, де зберігаються відповідно очищені та оброблені файли. Далі код здійснює перебір всіх CSV-файлів у вхідній директорії input_dir, що дозволяє автоматизувати процес обробки кількох файлів за один раз. Якщо файл не має розширення .csv, він пропускається, що забезпечує ефективну обробку лише необхідних даних.

Для кожного файлу код намагається зчитати його вміст за допомогою бібліотеки Pandas, що є потужним інструментом для роботи з даними у форматі таблиць. Одним із важливих етапів є очищення стовпця "Номенклатура" від зайвих пробілів та непотрібних лапок, що можуть з’явитися під час обробки або імпорту даних. Для цього значення в цьому стовпці перетворюються на рядки, потім із кожного значення видаляються зайві пробіли на початку та в кінці, а також лапки, що можуть бути присутніми в текстових значеннях.

Наступним кроком є перевірка наявності стовпців "Номенклатура" та "Кількість", після чого код визначає індекси цих стовпців у списку заголовків. Якщо між ними є зайві колонки, які не містять корисної інформації, вони видаляються з даних. Це важливий етап очищення, оскільки в багатьох випадках непотрібні стовпці можуть з’являтися в процесі збирання або експорту даних, і вони не повинні бути враховані при подальшій обробці.

Основною операцією цього коду є обгортання всіх значень у таблиці в лапки. Це досягається завдяки використанню параметра quoting=csv.QUOTE_ALL при збереженні даних у новому файлі. Такий підхід гарантує, що кожне значення в таблиці буде обгорнуте в лапки, що дозволяє уникнути проблем, коли значення містять роздільники або інші спеціальні символи. Збереження даних у такому форматі підвищує сумісність файлів з іншими системами та інструментами, що можуть бути використані для аналізу або імпорту даних.

Після виконання всіх операцій з очищення та обробки даних, файл зберігається в новій директорії output_dir. Це дозволяє забезпечити організованість і структурування даних, щоб результати обробки не змішувалися з вихідними файлами. Кожен файл, що був успішно оброблений, отримує повідомлення про успішне завершення операції, що дозволяє користувачу відстежувати хід обробки.

В результаті цього процесу отримуються файли, в яких кожне значення обгорнуте в лапки, що дозволяє забезпечити належний формат даних для подальшого використання. Очищення даних таким чином робить їх більш структурованими, зручними для зберігання та обміну між різними системами. Це є важливим етапом підготовки даних для подальших етапів обробки, таких як аналіз, візуалізація чи інтеграція з іншими системами.

4.4 Імпорт звітів до бази даних

Імпорт звітів продажів до бази даних є важливою складовою частиною процесу формування адаптивної рекомендаційної системи. Оскільки дані, необхідні для прогнозування та рекомендацій, зберігаються у вигляді CSV-файлів, важливо організувати ефективний процес імпорту, щоб забезпечити точність, повноту і швидкість обробки інформації.

У даній роботі для імпорту даних з файлів у базу даних використовується Python, а саме бібліотеки Pandas для обробки даних та Prisma ORM для взаємодії з базою даних. Усі дані про товари та їхні продажі збираються в окремі таблиці бази даних, що дозволяє ефективно працювати з великими обсягами інформації і зберігати дані в структурованому вигляді.

Для початку імпорту необхідно підготувати середовище розробки, завантаживши всі необхідні бібліотеки. Зокрема, використовуються Pandas для обробки CSV-файлів, що містять звіти продажів. Ця бібліотека дозволяє зручно читати, очищати і перетворювати дані. Prisma ORM забезпечує зручну роботу з реляційною базою даних, автоматизуючи збереження даних у відповідні таблиці. Nest AsyncIO дає можливість асинхронно працювати з базою даних, що знижує час обробки запитів і підвищує ефективність роботи.

Після завантаження необхідних бібліотек, наступним етапом є читання CSV-файлів із папки, де зберігаються звіти продажів. Кожен файл має ім’я у форматі MM-YYYY, що дозволяє зручно визначити місяць та рік, до яких відносяться дані. Завдяки використанню модуля calendar можна визначити останній день місяця для коректного встановлення дати.

Далі дані з CSV-файлів імпортуються в об’єкт DataFrame за допомогою Pandas. Кожен стовпчик у CSV-файлі переіменовується, щоб забезпечити відповідність назві полів у базі даних. Це важливий крок для забезпечення узгодженості між даними та структурою бази даних.

Оскільки звіти можуть містити нечислові значення або бути неповними (наприклад, порожні поля або значення, які не можна перетворити в числа), для коректного імпорту використовується функція to_float_default(). Ця функція перевіряє значення в кожному полі та перетворює його на тип float, якщо це можливо, або на значення за замовчуванням (наприклад, 0.0), якщо дані відсутні або некоректні. Такий підхід дозволяє уникнути помилок при імпорті і забезпечує коректність збереження даних у базі.

Для кожного звіту дані з DataFrame записуються в таблицю ProductHistoryWithoutRelation. Згідно з вимогами, кожен запис включає nomenclature - найменування товару, date - дата (остання дата місяця, визначена на початку), quantity - кількість товару, проданого в даному періоді, max_rate - максимальний курс обміну для цього товару, cost_usd, cost_price_usd - вартість і собівартість товару в доларах США, profit_usd - прибуток від продажу товару, cost_in_uah, cost_price_uah - вартість і собівартість товару в національній валюті, profit_uah - прибуток у національній валюті.

Використовуючи Prisma ORM, дані з кожного рядка DataFrame записуються до відповідних полів таблиці за допомогою асинхронного методу create(). Це дозволяє одночасно обробляти кілька записів, зменшуючи час на їх обробку та збереження в базі даних.

Після завершення процесу імпорту даних до бази даних, з’єднання з базою закривається за допомогою методу disconnect(). Це важливий крок для очищення ресурсів і запобігання витокам пам'яті.

Автоматизація процесу імпорту даних дозволяє зменшити людський фактор, прискорити обробку великих обсягів даних та знизити кількість помилок. Використання асинхронних запитів та обробки даних дозволяє масштабувати систему для роботи з більшими обсягами даних без суттєвого зниження

продуктивності. Завдяки перевірці та перетворенню даних у відповідні формати перед їх записом до бази даних, забезпечується висока точність і коректність збережених даних.

Імпорт звітів продажів до бази даних є важливим етапом у створенні адаптивної рекомендаційної системи. Використання Python, Pandas та Prisma ORM дозволяє автоматизувати обробку великих обсягів даних, забезпечуючи точність, ефективність та швидкість імпорту. Це в свою чергу дозволяє системі ефективно генерувати рекомендації на основі актуальних даних та підвищувати точність прогнозів щодо товарних запасів.

4.5 Загальна схема функціонування системи та Адаптивний рекомендаційний алгоритм

Функціонування адаптивної рекомендаційної системи базується на низці етапів, які включають обробку вхідних звітів, очищення та збереження даних, вибір відповідного алгоритму прогнозування, а також формування остаточних рекомендацій. Загальну логіку роботи системи подано на рисунку 4.8.

На початковому етапі в систему додається звіт продажності у форматі CSV. Ці файли попередньо обробляються з метою усунення зайвої та службової інформації, привести структури таблиць до єдиного формату, приведення чисел до єдиного формату та забезпечення консистентності текстових полів (наприклад, обгортання назв товарів, що містять кому, у лапки). Далі виконується агрегація даних: об'єднання повторюваних записів, обрахунок сумарних показників за обраними періодами тощо.

Після агрегації очищені дані зберігаються в базу даних, з якої згодом витягуються для подальшої обробки. Наступним кроком є визначення обсягу історичних даних для кожного товару. Залежно від цього обсягу обирається один із трьох алгоритмів прогнозування:

Якщо історія продажів для товару становить менше 6 місяців, система застосовує алгоритм Prophet, який ефективно працює з короткими часовими рядами та добре адаптується до трендів і сезонності.

Якщо історія продажів більша за 6 місяців, але менша за 2 роки, використовується модель SARIMA, що є класичним підходом до обробки стаціонарних часових рядів.

Якщо доступна історія понад 2 роки, для прогнозування обирається глибока нейронна мережа LSTM, яка дозволяє виявляти складні нелінійні залежності в довгих часових рядах.

Результати прогнозу використовуються для формування рекомендацій щодо обсягів дозамовлення кожного товару. Після завершення розрахунків формується фінальна таблиця з рекомендаціями, яка відображається в інтерфейсі системи та може бути експортована у зручному для користувача форматі.

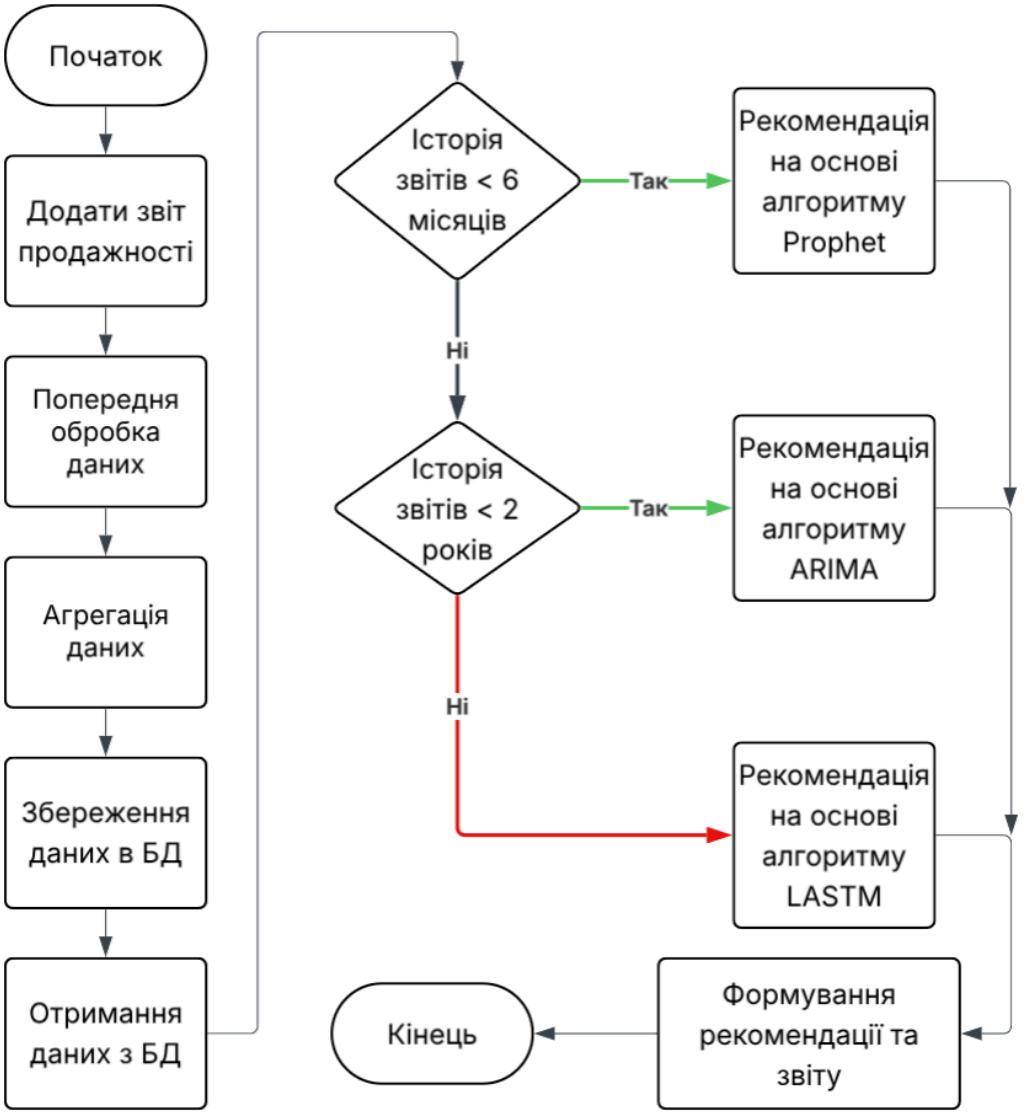


Рисунок 4.8 – схема функціонування системи

Адаптивний рекомендаційний алгоритм виступає фундаментальною складовою інтегрованої системи прогнозування обсягів реалізації продукції та формування обґрунтованих рекомендацій щодо оптимальних параметрів дозамовлення товарних позицій. Його призначення виходить за межі традиційних підходів до аналітики попиту, адже ключовою дослідницькою та прикладною метою алгоритму є досягнення максимальної точності, стабільності прогнозів у динамічному середовищі сучасного ринку.

Зокрема, алгоритм побудований таким чином, щоб враховувати цілу низку критично важливих факторів: індивідуальні особливості кожного товару, довжину та якість історичних даних про продажі, сезонні цикли, характерні для конкретних груп продуктів, а також стохастичні коливання попиту, зумовлені як внутрішніми (маркетингові активності, зміни в асортименті), так і зовнішніми чинниками (економічні коливання, поведінкові патерни споживачів, форс-мажорні обставини).

Особливістю розробленого підходу є його адаптивність, що проявляється у здатності алгоритму автоматично добирати найбільш релевантну математико-статистичну або машинно-навчальну модель прогнозування залежно від обсягу та структури доступних вхідних даних. Таким чином, у випадках обмеженої кількості спостережень застосовуються моделі, здатні коректно працювати за умов недостатньої інформації, тоді як для повних і довгострокових рядів використовуються більш складні алгоритмічні конструкції, орієнтовані на виявлення глибинних закономірностей. Це надає системі універсального характеру, дозволяючи їй ефективно функціонувати в умовах високої варіативності та різнорідності інформаційного середовища.

З огляду на викладене, адаптивний рекомендаційний алгоритм можна розглядати як методологічну основу побудови сучасних інтелектуальних систем підтримки прийняття управлінських рішень у сфері логістики, закупівель та управління товарними запасами, що забезпечує підвищення економічної ефективності та зниження ризиків, пов’язаних із не точними прогнозами.

Реалізація адаптивного алгоритму прогнозування здійснена на основі інтеграції трьох ключових методологій часових рядів та машинного навчання:

Prophet, SSARIMA та LSTM-мережі. Такий підхід зумовлений необхідністю врахування різних сценаріїв інформаційного забезпечення та специфіки даних, що уможливлює підвищення точності прогнозів у контексті неоднорідності товарних груп. Вибір відповідного методу здійснюється на основі довжини та структурних характеристик історичних даних про продажі, що є принциповим у забезпеченні адаптивності системи.

Зокрема, Prophet, розроблений у лабораторіях компанії Meta (Facebook), застосовується у випадках, коли часовий ряд має обмежену кількість спостережень і характеризується високою чутливістю до екзогенних факторів. Його модельна архітектура дозволяє коректно враховувати трендові компоненти, сезонні коливання та ефекти зовнішніх регресорів, серед яких вирішальне значення мають валютні курси, собівартість виробництва та динаміка маржинального прибутку. Це забезпечує адекватність прогнозів навіть за умов неповної чи нерегулярної інформації.

SSARIMA (Seasonal AutoRegressive Integrated Moving Average), у свою чергу, орієнтована на аналіз часових рядів середньої довжини, де спостерігаються чітко виражені сезонні коливання та регулярні цикли. Використання SSARIMA особливо виправдане у випадках, коли дані мають чітку календарну структуру (наприклад, місячні показники реалізації), що дозволяє ефективно моделювати повторювані закономірності та відокремлювати трендову і стохастичну компоненти. Таким чином, SSARIMA забезпечує збалансоване поєднання аналітичної строгості при обробці детермінованих і випадкових факторів.

Для аналізу часових рядів із великою кількістю історичних спостережень використовується глибинна архітектура LSTM (Long Short-Term Memory), що належить до класу рекурентних нейронних мереж. Завдяки своїй здатності зберігати та актуалізувати інформацію на довгих часових горизонтах, LSTM є особливо ефективною при виявленні прихованих нелінійних залежностей та складних взаємозв’язків між показниками обсягів продажів і фінансово-економічними детермінантами. Застосування цього методу дозволяє системі враховувати не лише безпосередню динаміку попиту, а й кумулятивний вплив широкого спектра факторів.

Таким чином, комбіноване використання Prophet, SSARIMA та LSTM формує методологічне підґрунтя адаптивного алгоритму прогнозування, де кожен метод виконує свою функціональну роль залежно від структури даних. Така багаторівнева архітектура забезпечує універсальність і масштабованість розробленої системи, підвищуючи її ефективність як у короткостроковому, так і в довгостроковому горизонті планування.

Однією з визначальних характеристик адаптивності розробленого алгоритму виступає наявність етапу попередньої валідації даних перед запуском безпосередньої процедури прогнозування. Такий підхід передбачає перевірку актуальності та змістовності часових рядів, зокрема, здійснюється оцінка останніх трьох місяців продажів товарної позиції. У випадку, якщо у цей період фіксується нульовий обсяг реалізації (тобто quantity = 0), алгоритм свідомо відмовляється від побудови прогнозної моделі. Подібне рішення ґрунтується на припущенні, що відсутність продажів упродовж зазначеного проміжку часу є достовірним маркером нульового або майже нульового попиту в поточних ринкових умовах.

Впровадження такого фільтра виконує подвійну функцію. По-перше, він дозволяє уникати створення прогнозів, що не мають практичного значення для управлінських рішень, знижуючи ризик формування хибних рекомендацій. По-друге, механізм перевірки сприяє раціоналізації використання обчислювальних ресурсів, запобігаючи витратам часу та енергії на моделювання ситуацій, де відсутній предмет для прогнозування. У масштабах великих товарних баз та систем з інтенсивними обчисленнями це має стратегічне значення, адже оптимізація продуктивності безпосередньо впливає на швидкість та ефективність усієї системи.

Додатково результати емпіричного аналізу історичних даних показали, що навіть у випадках із вираженою сезонністю, більшість товарів зберігають хоча б мінімальні обсяги продажів, які не знижуються нижче умовного порогового рівня (приблизно десяти одиниць). Таким чином, повна відсутність продажів протягом кількох місяців інтерпретується не як прояв сезонного спаду, а радше як сигнал структурної зміни попиту, наприклад, зникнення актуальності товару для споживачів чи припинення його ринкового обігу.

Отже, описана процедура попередньої перевірки даних формує додатковий рівень інтелектуальної адаптивності алгоритму, адже вона демонструє здатність системи не лише коригувати вибір моделей залежно від обсягу історії, але й приймати мета-рішення щодо доцільності самого процесу прогнозування. Це підкреслює комплексність розробленого підходу, що поєднує алгоритмічну строгість із прагматичною економією ресурсів.

У реалізації алгоритму на основі Prophet передбачено використання розширеної специфікації моделі шляхом інтеграції екзогенних регресорів, що відображають вплив ключових економічних та фінансових детермінант на динаміку попиту. До таких регресорів, зокрема, належать максимальні значення валютних курсів, показники собівартості виробництва, а також дані щодо прибутковості як у національній, так і в іноземній валюті. Залучення цих змінних до структури моделі дозволяє не лише враховувати сезонні та трендові компоненти часових рядів, але й відображати складні взаємозв’язки між макроекономічними коливаннями та поведінкою споживачів на мікрорівні.

Прогнозування здійснюється з горизонтом в один календарний місяць, що відповідає практичним потребам більшості бізнес-процесів управління товарними запасами та планування закупівель. При цьому для формування прогнозів використовуються актуальні значення екзогенних змінних, що забезпечує релевантність та своєчасність отриманих рекомендацій. Такий підхід дозволяє зменшити ризик хибних оцінок, які можуть виникати у випадках ігнорування зовнішніх факторів, та водночас підвищує аналітичну цінність моделі, оскільки вона здатна інтегрувати як внутрішні дані (історія продажів), так і зовнішні індикатори ринкового середовища.

Таким чином, застосування Prophet із включенням екзогенних регресорів формує багатофакторну адаптивну систему прогнозування, що демонструє здатність до побудови більш реалістичних сценаріїв розвитку попиту. Це робить рекомендації, отримані на основі алгоритму, не лише статистично обґрунтованими, але й економічно доцільними, забезпечуючи прийняття управлінських рішень із вищим рівнем точності та стратегічної релевантності.

SSARIMA-модель (Seasonal AutoRegressive Integrated Moving Average) у межах реалізованої системи прогнозування виступає ключовим інструментом для обробки часових рядів, що характеризуються вираженою сезонністю та повторюваними закономірностями. Завдяки своїй математичній структурі модель забезпечує коректне врахування щомісячних коливань обсягів продажів, дозволяючи одночасно моделювати як трендову, так і сезонну компоненти ряду. Це створює підґрунтя для формування прогнозів, які відображають не лише загальну динаміку розвитку попиту, але й його циклічні особливості, пов’язані з календарними чи поведінковими факторами.

Застосування SSARIMA є доцільним лише за умови наявності достатнього масиву спостережень - принаймні 12 місяців історичних даних. Такий поріг пояснюється необхідністю виявлення і статистичного підтвердження сезонних закономірностей, без чого модель втрачає свою аналітичну перевагу. Використання повного річного циклу дозволяє враховувати як внутрішньорічні тренди, так і характерні періодичні коливання, що забезпечує високу достовірність і стійкість прогнозних оцінок.

Окрему увагу приділено роботі з неповними часовими рядами. У випадках наявності пропусків у даних реалізовано механізми прямої та зворотної заповнюваності (forward/backward filling), які дозволяють відновити цілісність інформаційного ряду. Це забезпечує стабільність функціонування моделі, знижує ймовірність спотворення результатів через відсутні значення та підвищує загальну надійність прогнозів. Таким чином, SSARIMA інтегрується у систему як метод, що поєднує математичну строгість з практичною адаптивністю, гарантуючи адекватність результатів навіть у випадках неповної інформаційної бази.

LSTM-мережі (Long Short-Term Memory) інтегруються в систему прогнозування для роботи з великими масивами даних, де наявні складні та багаторівневі нелінійні залежності між показниками. Використання цього методу є обґрунтованим у випадках, коли обсяг історичних спостережень перевищує кілька десятків місяців, що забезпечує достатню інформаційну базу для навчання моделі. На відміну від класичних статистичних підходів, LSTM здатні зберігати та

опрацьовувати інформацію у довгострокових часових інтервалах, завдяки чому вони ефективно виявляють приховані патерни у поведінці попиту.

Перед безпосереднім етапом тренування здійснюється ретельна передобробка даних, яка включає масштабування змінних (normalization/standardization) та формування вхідного масиву у форматі тривимірної структури [samples, timesteps, features] [samples, timesteps, features] [samples, timesteps ,features]. Такий формат відповідає архітектурним вимогам LSTM і забезпечує можливість врахування як часових послідовностей, так і багатофакторних залежностей. Завдяки цьому модель отримує адекватне уявлення про динаміку процесу продажів у контексті кількох взаємопов’язаних характеристик.

Застосування LSTM дозволяє моделювати складні взаємозв’язки між ключовими параметрами: обсягами реалізації, рівнем цін, собівартістю та маржинальним прибутком. Це забезпечує значно вищу адаптивність у порівнянні зі статистичними моделями, оскільки нейронна мережа здатна враховувати нелінійність впливу, лагові ефекти та взаємодію змінних. У результаті формуються прогнози з високим рівнем точності, які відображають не лише прямі закономірності, але й приховані нелінійні структури у даних.

Таким чином, LSTM-мережі виступають потужним інструментом глибинного навчання в архітектурі адаптивного рекомендаційного алгоритму, забезпечуючи можливість побудови прогностичних моделей, релевантних для низьковимірних і динамічно змінних систем. Їх застосування сприяє суттєвому підвищенню достовірності прогнозів та ефективності управлінських рішень у сфері логістики та стратегічного планування.

Таким чином, запропонований підхід формує цілісну концепцію прогнозування, що вирізняється високим рівнем адаптивності. Його особливість полягає не лише у здатності автоматично здійснювати вибір найбільш релевантної моделі прогнозування з-поміж доступних методів, але й у реалізації механізму мета-рівневої оцінки доцільності самого запуску процесу прогнозування для кожної окремої товарної позиції, що враховує специфіку історичних даних, характер динаміки продажів і наявність або відсутність актуального попиту.

Поєднання алгоритмічної адаптивності (динамічний вибір моделі залежно від довжини та структури часових рядів) із контекстуальною доцільністю (перевірка релевантності прогнозування на основі останніх даних) робить систему не лише методологічно стійкою, але й економічно виправданою. Вона дозволяє досягти ресурсозберігаючого ефекту, оскільки уникає непотрібних обчислень, і водночас забезпечує високу точність та практичну значущість отриманих результатів.

У підсумку, описана система функціонує як інтелектуальний інструмент підтримки управлінських рішень, орієнтований на оптимізацію процесів дозамовлення товарів. Вона поєднує статистичну строгость, глибинне машинне навчання та прагматичний підхід до використання ресурсів, що забезпечує комплексну ефективність у динамічних ринкових умовах. Завдяки цьому запропонований алгоритм може розглядатися як перспективна основа для розвитку сучасних систем прогнозної аналітики у сфері логістики, торгівлі та стратегічного планування.

4.6 Архітектура та реалізація API

Розробка серверного API здійснювалася на основі фреймворку NestJS, який забезпечує модульну, масштабовану та легко підтримувану архітектуру. NestJS поєднує сучасні підходи до розробки серверних застосунків із підтримкою TypeScript, що дозволяє забезпечити сувору типізацію, контроль за структурами даних та безпечну взаємодію між компонентами. Архітектура проекту побудована за принципом модульності, що дозволяє логічно розділяти функціональні блоки та підтримувати високу масштабованість системи.

Проєкт має чітку структуру директорій, де src містить основні модулі та логіку застосунку. Головний модуль app.module.ts відповідає за інтеграцію всіх підмодулів та ініціалізацію сервісів. Контролери, такі як app.controller.ts, реалізують маршрутизацію та обробку HTTP-запитів, забезпечуючи доступ клієнтської частини до функціоналу API. Сервіси, зокрема app.service.ts, містять бізнес-логіку і виконують операції з базою даних через Prisma ORM, що дозволяє абстрагувати безпосередню роботу з SQL-запитами.

Для забезпечення безпеки та автентифікації користувачів використовується модуль auth, який включає auth.controller.ts, auth.service.ts, а також схеми DTO та стратегії автентифікації. Цей модуль відповідає за генерацію та перевірку JWT-токенів, управління правами доступу та забезпечує захищений доступ до ресурсів системи.

Модуль forecast реалізує ключовий функціонал прогнозування продажів та генерації рекомендацій. Контролер forecast.controller.ts приймає запити від клієнта, обробляє параметри та передає дані до сервісу forecast.service.ts, де відбувається виклик адаптивного алгоритму прогнозування, інтеграція з моделями Prophet, SSARIMA та LSTM, а також підготовка результатів у форматі, придатному для відправки на фронтенд. Модуль використовує DTO для строгої типізації вхідних і вихідних даних, що забезпечує коректність і передбачуваність обробки запитів.

Модуль search відповідає за пошукову функціональність у системі. Контролер search.controller.ts обробляє запити користувача на пошук товарів або звітів, використовуючи сервіс search.service.ts, який взаємодіє з базою даних через Prisma та реалізує логіку фільтрації, сортування та пошуку за ключовими параметрами.

Модуль services містить допоміжні сервіси, такі як aws-s3.service.ts, що забезпечує інтеграцію з хмарним сховищем для завантаження та зберігання файлів, наприклад, CSV-звітів. Модуль configs містить конфігураційні файли, зокрема getJwtConfig.ts, які визначають налаштування JWT та інші параметри безпеки.

Важливим компонентом архітектури є Prisma ORM, що реалізований у вигляді окремого сервісу prisma.service.ts та відповідних типів у директорії prisma. Prisma забезпечує безпечну та ефективну роботу з реляційною базою даних PostgreSQL, автоматично генерує типи для таблиць, моделей та їхніх зв’язків. Міграції бази даних зберігаються у prisma/migrations, що дозволяє відслідковувати зміни у схемі та підтримувати синхронізацію структури бази з кодом.

Додатково проект має тестову структуру у директорії test, яка включає E2E-тести для перевірки роботи основних функціональних компонентів API. Це забезпечує високу надійність системи та дозволяє впевнено впроваджувати зміни у коді без ризику порушення існуючого функціоналу.

Таким чином, реалізоване API забезпечує повну інтеграцію серверної логіки з фронтендом, безпечну роботу з даними, простоту при масштабуванні та можливість ефективного використання адаптивного алгоритму прогнозування. Модульна структура NestJS дозволяє розширювати систему у майбутньому, додавати нові функціональні блоки та інтегрувати зовнішні сервіси без порушення цілісності архітектури.

4.7 Архітектура та структура користувацького інтерфейсу

Користувацький інтерфейс розроблений на основі сучасного стеку Next.js та React із застосуванням TypeScript, що забезпечує строгість типізації, підвищену передбачуваність поведінки компонентів та полегшує підтримку великого коду. Архітектура UI побудована за принципом модульності, де окремі функціональні блоки виділені у незалежні компоненти та логічні директорії, що забезпечує легку масштабованість та повторне використання елементів.

Верхній рівень структури проекту містить основні конфігураційні файли, такі як package.json, next.config.ts, postcss.config.mjs та глобальні стилі. Папка public зберігає статичні ресурси, включаючи іконки та графічні файли, що використовуються у інтерфейсі.

Директорія src є центральною у структурі UI і включає кілька логічних блоків. Піддиректорія app відповідає за маршрутизацію та глобальні налаштування сторінок, включаючи layout, глобальні стилі та провайдери стану.

Окремі маршрути, такі як login і register, реалізують функціонал аутентифікації та реєстрації користувачів, забезпечуючи безпечний доступ до системи.

Всі компоненти інтерфейсу зберігаються у директорії components, де вони структуровані за функціональними категоріями. Піддиректорія core містить базові елементи, які використовуються повторно по всьому додатку. Dialogs відповідають за модальні вікна та попереджувальні повідомлення, тоді як forms забезпечують реалізацію інтерактивних форм для введення даних. Icons зберігає усі графічні іконки, а loaders відповідають за анімації завантаження та індикатори очікування. Додатково під директорія ui містить універсальні UI-компоненти.

Для взаємодії з серверною частиною застосовано директорію api, що містить налаштування клієнта API (apiClient.ts) та окремі папки для запитів та мутацій (queries і mutation), які забезпечують централізовану роботу з бекендом. Це дозволяє стандартизувати обробку HTTP-запитів та полегшує інтеграцію нових сервісів.

Директорія hooks реалізує кастомні React-хуки, такі як useForecast.ts, useRegisterForm.ts та useSearch.ts, що забезпечують повторне використання логіки взаємодії з даними, станом форм та пошуковими запитами.

Для управління станом застосовується store, де зберігаються глобальні дані користувача, такі як автентифікація та права доступу (useAuthStore.ts). Піддиректорія providers містить провайдери контексту, включаючи AuthProvider.tsx та PrivateRoute.tsx, які забезпечують захищений доступ до компонентів і маршрутів залежно від ролі користувача.

Директорія services включає сервіси для взаємодії з бекендом: categoryService.ts обробляє категорії товарів, forecastService відповідає за отримання прогнозів від адаптивного алгоритму, а searchService реалізує пошук та фільтрацію товарів. У lib/utils.ts містяться допоміжні функції для обробки даних та форматування.

Файли у types визначають строгі TypeScript-типи для категорій, прогнозів, продуктів, ролей та користувачів, що забезпечує цілісність даних і запобігає помилкам під час розробки.

Інтерфейс системи, відображений на рисунку 4.9, демонструє зручне та інтуїтивне представлення даних. Таблиці з назвами товарів і рекомендованими обсягами дозамовлення дозволяють швидко оцінити потребу у запасах, а інтерактивні графіки продажів забезпечують наочне відображення історичних даних та тенденцій. Модальні вікна та форми дозволяють користувачам коректно вводити дані та отримувати рекомендації без необхідності додаткового навчання, що підвищує ефективність використання системи.

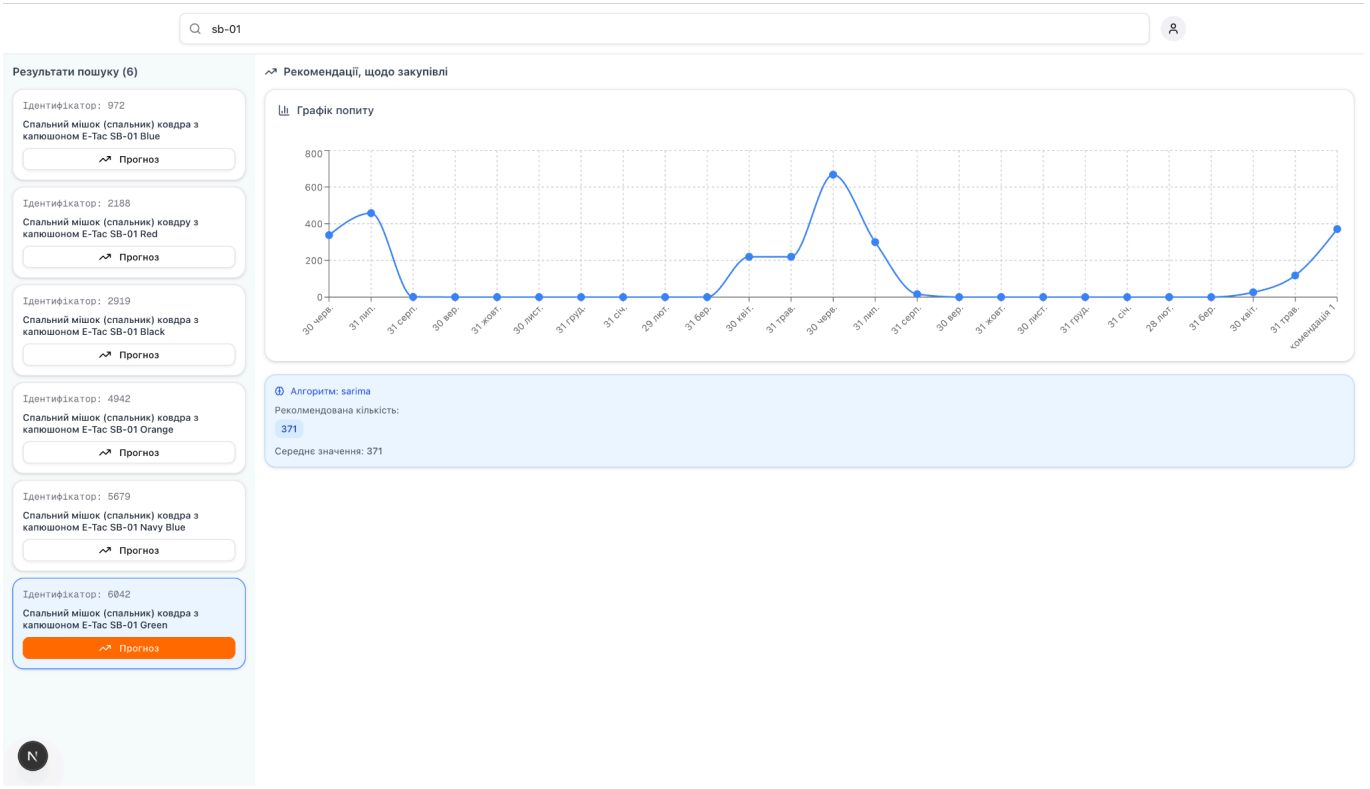


Рисунок 4.9 – Користувацький інтерфейс

Таким чином, реалізована структура UI забезпечує сучасний, масштабований інтерфейс користувача, що інтегрується із серверною частиною через стандартизовані API, підтримує адаптивне відображення даних і сприяє прийняттю оперативних рішень на основі прогнозів.

4.8 Висновки до розділу

У розділі 4 було зосереджено увагу на практичній реалізації системи, яка поєднує всі попередньо розглянуті теоретичні та методологічні аспекти у єдиний комплекс. Детальний опис програмного забезпечення дав можливість продемонструвати, як обрані технології, інструменти та архітектурні рішення трансформуються у робочий інструмент, здатний забезпечити стабільну й передбачувану роботу в умовах реальної експлуатації.

Насамперед було показано важливість вибору адекватного технічного середовища, яке дозволяє легко керувати залежностями та масштабувати програмний продукт. Використання сучасних інструментів розробки дало змогу не

лише зручно організувати процес створення системи, а й забезпечити відтворюваність, прозорість та контрольованість усіх етапів її життєвого циклу.

Особливу увагу приділено роботі з даними, адже саме вони є базисом для формування будь-яких рекомендацій. Було показано, що ефективна система зберігання та обробки інформації здатна гарантувати узгодженість і цілісність даних навіть у випадках великого навантаження. При цьому підготовка та попередня обробка вхідних звітів стала критичним етапом, без якого було б неможливо забезпечити коректність подальшого прогнозування. Очищення, нормалізація та приведення даних до єдиного формату утворили основу для стабільного функціонування алгоритмів.

Логічним продовженням стало впровадження інструментів для перенесення оброблених даних у сховище, що забезпечує їхнє збереження та доступність для аналітичних модулів. Сформована структура бази даних продемонструвала оптимальний баланс між швидкодією та надійністю, а також створила передумови для масштабованості системи в майбутньому.

Ключове значення мало розроблення адаптивного алгоритму рекомендацій, який здатний ураховувати різноманітні фактори: довжину історії продажів, сезонні коливання, стохастичні відхилення та інші чинники, що впливають на попит. Завдяки цьому система набуває можливості формувати більш точні й обґрунтовані пропозиції щодо дозамовлення товарів, а її результати стають практично корисними для прийняття управлінських рішень.

Важливим компонентом є також програмний інтерфейс, який виступає сполучною ланкою між даними, алгоритмічним ядром і кінцевим користувачем. Завдяки чітко організованій архітектурі вдалося досягти узгодженості між внутрішньою логікою системи та зовнішнім середовищем її використання. Це забезпечує прозорість взаємодії, простоту інтеграції та можливість подальшого розширення функціоналу без кардинальних змін у базових структурах.

Не менш значущим є створений користувацький інтерфейс, який дозволяє у наочній та доступній формі відображати результати аналізу. Наявність інтерактивних таблиць і графіків перетворює сухі числові показники на зрозумілі й візуально

привабливі інструменти підтримки рішень. Це робить систему не лише технічно досконалою, а й реально зручною для практичного використання в електронній комерції.

Узагальнюючи результати, можна стверджувати, що обрані архітектурні та технологічні рішення дозволили створити цілісний та перспективний програмний продукт. Система поєднує в собі стабільність, точність прогнозування та зручність використання, а також відкриває можливості для подальшого вдосконалення й адаптації під нові вимоги ринку.

РОЗДІЛ 5. РОЗРОБЛЕННЯ СТАРТАП-ПРОЄКТУ

5.1 Опис ідеї проєкту

Ідея стартап-проєкту полягає у виведенні на ринок програмного продукту - «Адаптивної системи прогнозування та управління товарними запасами для e-commerce». Це веб-застосунок, який використовує методи машинного навчання (Prophet, SARIMA, LSTM) для автоматичного аналізу історії продажів та надання рекомендацій щодо дозамовлення товарів. Ключова особливість системи - адаптивність, тобто здатність автоматично обирати найточніший алгоритм прогнозування залежно від обсягу наявних історичних даних для кожного конкретного товару, починаючи від нових позицій і закінчуючи тими, що продаються роками. Потенційними напрямами застосування продукту є інтернет-магазини малого та середнього розміру (B2C) із широким асортиментом (від 500 SKU), роздрібні торговельні мережі, які потребують автоматизації закупівель, а також дистриб'юторські компанії, яким необхідно ефективно планувати складські запаси.

Впровадження системи надає користувачеві низку вигод, насамперед фінансову економію шляхом зменшення замороженого капіталу в неліквідних товарах та уникнення втрати прибутку через відсутність популярних позицій (out-of-stock). Крім того, забезпечується економія часу завдяки автоматизації рутинних розрахунків, які раніше доводилося виконувати вручну в Excel. Важливою перевагою є зниження порогу входження: на відміну від складних ERP-систем, користувач отримує простий інтерфейс із конкретними рекомендаціями щодо обсягів замовлення, що не вимагає наявності в штаті професійного аналітика даних.

Як було зазначено в аналізі ринку, існуючі аналоги, такі як SAP, Odoo або Zoho Inventory, є надто дорогими та складними для малого бізнесу. Головна відмінність пропонованої системи від них полягає у фокусуванні виключно на задачах прогнозування та рекомендацій без перевантаження користувача зайвим функціоналом. Застосування унікального адаптивного підходу до вибору математичної моделі забезпечує високу точність розрахунків навіть для товарів із короткою історією продажів.

5.2 Аналіз технологічних можливостей реалізації ідей проєкту

Для технічної реалізації стартап-проєкту було обрано стек, який поєднує високу продуктивність, масштабованість та доступність. Серверна частина (Backend) побудована на базі NestJS (Node.js), що забезпечує модульну архітектуру та швидку обробку запитів, тоді як Data Science ядро реалізовано мовою Python із використанням спеціалізованих бібліотек, таких як prophet, statsmodels для SARIMA та tensorflow/keras для LSTM. Для створення швидкого та зручного клієнтського інтерфейсу використовується зв'язка Next.js та React, а надійне зберігання структурованих даних про продажі забезпечує база даних PostgreSQL у поєднанні з Prisma ORM. Розгортання системи також можливе на серверах клієнтів або у хмарі, що однозначно передбачає використання Docker для контейнеризації.

Аналіз доступності технологій засвідчує, що всі обрані інструменти належать до відкритого програмного забезпечення (Open Source). Необхідні бібліотеки для реалізації методів Prophet, SARIMA та LSTM є вільно доступними та мають активну підтримку спільноти, тому розробляти їх з нуля не потрібно. Водночас, у рамках дипломної роботи було створено логіку - оркестратор, який автоматично перемикається між цими алгоритмами та проводить попередню обробку даних.

Використання відкритого ПЗ дозволяє мінімізувати собівартість розробки завдяки відсутності витрат на дорогі ліцензії. При цьому реалізована архітектура дозволяє легко масштабувати систему, додавати нові модулі або інтегрувати її з іншими CRM-системами.

5.3 Аналіз ринкових можливостей запуску стартап-проєкту

Аналіз ринкового середовища свідчить про стабільне зростання сфери електронної комерції в Україні, де значна частка належить малим підприємцям, які поступово переходять від інтуїтивного ведення торгівлі до системного підходу. Розвитку цього напрямку сприяють цифровізація бізнесу, посилення конкуренції, що змушує підприємців оптимізувати витрати. Водночас впровадження нових рішень стримується такими факторами, як низька технічна грамотність частини власників малого бізнесу та вкорінена звичка вести облік у простих Excel-таблицях.

В рамках аналізу попиту основними потенційними клієнтами визначено власників інтернет-магазинів з асортиментом від 500 до 10 000 позицій. Їх можна розподілити на дві категорії: малі магазини (500–2000 товарів) та середній бізнес (2000–10 000+ товарів). Перша група потребує максимально простого та недорогого інструменту, який надаватиме чіткі відповіді на питання «що купити», дозволяючи уникнути заморожування коштів. Для другої групи пріоритетними є точність прогнозів, врахування сезонності, надійність аналітики та можливість інтеграції з наявними CRM-системами.

Щодо конкурентного середовища, то на ринку вже присутні великі гравці, такі як SAP, Odoo та Zoho, що пропонують комплексні ERP-рішення. Проте для цільового сегмента їхніми слабкими сторонами є висока ціна, складність впровадження, яка часто потребує залучення інтеграторів, та надлишковий функціонал. Натомість сильні сторони пропонованого проєкту полягають у вузькій спеціалізації на рекомендаціях закупівель, адаптивності алгоритмів до роботи з неповними даними, доступній ціні та українській локалізації. Головними викликами для проєкту залишаються відсутність впізнаваного бренду та необхідність подолання бар'єру недовіри користувачів до алгоритмів штучного інтелекту.

5.4 Розроблення ринкової стратегії проєкту

Враховуючи специфіку продукту, доцільно обрати стратегію концентрованого маркетингу, що передбачає роботу у вузькій ніші - малому та середньому e-commerce бізнесі в Україні з перспективою подальшого масштабування на країни Східної Європи. Це підприємства, які вже використовують базові системи обліку, такі як Excel або прості CRM, проте продовжують стикатися з проблемою неефективного управління товарними залишками. Основу цільової аудиторії складають власники інтернет-магазинів, що ведуть діяльність як на власних платформах, так і на популярних маркетплейсах, наприклад Prom.ua чи Rozetka. Також ключовими користувачами є категорійні менеджери невеликих мереж, які змушені витрачати значний час - часто понад 10 годин на тиждень - на ручне формування замовлень постачальникам.

Важливим елементом стратегії є позиціювання продукту не просто як чергового програмного інструменту, а як доступного «віртуального аналітика», що працює в режимі 24/7. На відміну від складних ERP-систем, які часто відлякують малий бізнес своєю громіздкістю та потребою в дорогому впровадженні, розроблена система акцентує увагу на простоті використання та прозорості алгоритмів. Це дозволяє сформувати у споживача довіру до автоматичних рекомендацій та подолати психологічний бар'єр переходу від інтуїтивного управління до прийняття рішень на основі даних.

Для успішного виходу на ринок та швидкого залучення користувачів обрано стратегію, яка базується на моделі монетизації через доступну щомісячну підписку. Встановлення ціни, що є значно нижчою за вартість корпоративних рішень конкурентів, дозволяє мінімізувати фінансові ризики для клієнтів та забезпечує швидку окупність інвестицій завдяки оптимізації складських запасів вже у перший місяць використання. Просування продукту планується здійснювати шляхом поєднання контент-маркетингу, спрямованого на підвищення обізнаності підприємців щодо методів уникнення втрат прибутку, та партнерських програм із розробниками інтернет-магазинів, що забезпечить прямий доступ до цільової аудиторії в момент виникнення у неї потреби в автоматизації.

5.5 Маркетингова програма стартап-проєкту

Концепція товару передбачає реалізацію продукту у вигляді веб-сервісу (SaaS). Користувач отримує можливість завантажити історію продажів у форматах csv чи xls або підключити свою CRM через API, після чого система генерує дашборд із графіками попиту та готовий звіт «Рекомендації до закупівлі» з конкретними цифрами. Важливою особливістю є прозорість роботи сервісу, адже він надає пояснення до кожної рекомендації, обґрунтовуючи, чому саме таку кількість товару необхідно замовити.

У стратегії ціноутворення, зважаючи на високу вартість рішень конкурентів, обрано підхід проникнення на ринок із доступною ціною. Модель монетизації базується на щомісячній підписці орієнтовною вартістю 30–50 доларів, що в 10–20

разів дешевше за Enterprise-аналоги. Це робить продукт доступним навіть для мікробізнесу та забезпечує клієнту швидку окупність інвестицій за рахунок оптимізації складу вже протягом першого місяця використання.

Щодо стратегії збуту та просування, то продажі здійснюватимуться як напряму через власний сайт, так і через партнерські угоди з розробниками інтернет-магазинів. Просуванню сприятимуть контент-маркетинг із матеріалами про уникнення фінансових втрат на складських залишках, а також реклама в соціальних мережах для власників бізнесу. Для демонстрації ефективності алгоритмів на реальних даних клієнтам пропонується безкоштовний пробний період тривалістю 14 днів.

5.5 Висновки до розділу

У п’ятому розділі розроблено концепцію стартап-проєкту на базі створеної адаптивної рекомендаційної системи. Встановлено, що цей проєкт має потенціал комерціалізації завдяки чітко визначеній цільовій аудиторії - малому та середньому бізнесу в сфері e-commerce, а також наявній проблемі неефективного управління запасами, яку великі ERP-системи часто ігнорують через свою високу вартість. Технологічний аналіз підтвердив можливість реалізації та масштабування системи силами невеликої команди розробників завдяки використанню сучасного відкритого стеку технологій, таких як Python, NestJS та React.

Обрана стратегія концентрованого маркетингу та доступна модель ціноутворення у вигляді SaaS-підписки дозволять проєкту успішно конкурувати з дорожчими аналогами, пропонуючи клієнтам швидкий та вимірюваний економічний ефект. Проєкт є доцільним для подальшого розвитку, оскільки він вже успішно пройшов етап впровадження у реальному бізнесі на прикладі npshop.com.ua та на практиці довів свою працездатність.

ВИСНОВКИ

У результаті виконання дипломної роботи було досягнуто поставленої мети - розроблено адаптивну рекомендаційну систему для прогнозування та формування обсягів дозамовлення товарів в інтернет-магазині. У процесі дослідження та реалізації завдання вдалося комплексно поєднати теоретичні засади аналізу часових рядів, методи машинного навчання, алгоритми прогнозування та сучасні засоби програмної інженерії.

Математичне забезпечення системи стало основою для реалізації процесу прогнозування. Було досліджено та реалізовано декілька підходів - від класичних моделей часових рядів Prophet і SARIMA до нейронних мереж типу LSTM.

Адаптивність системи полягає у виборі методу прогнозування залежно від обсягу та якості історичних даних, що забезпечує підвищену точність і надійність результатів. Такий підхід дозволяє системі враховувати різноманітні бізнес-сценарії та автоматично підібрати найбільш адекватне алгоритмічне рішення.

Особливу увагу приділено етапу попередньої обробки даних, який значно впливає на достовірність прогнозів. Розроблена архітектура системи забезпечує наскрізний процес обробки даних - від збору та очищення до візуалізації результатів у зручному для користувача інтерфейсі. Реалізоване API гарантує структуровану взаємодію між компонентами системи, а графічний інтерфейс забезпечує інтуїтивний доступ до рекомендацій у вигляді таблиць та графіків.

Завдяки поєднанню алгоритмів прогнозування з технологіями сучасної веб-розробки створено систему, яка має високий практичний потенціал і може бути ефективно застосована в умовах малого та середнього бізнесу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Why Python is called Dynamically Typed? – GeeksforGeeks. URL: https://www.geeksforgeeks.org/why-python-is-called-dynamically-typed/ (дата звернення: 02.06.2025).
2. Python Features – GeeksforGeeks. URL: https://www.geeksforgeeks.org/python-features/ (дата звернення: 02.06.2025).
3. Flanagan D. JavaScript: The Definitive Guide. 7th ed. O'Reilly Media, 2020. 720 p.
4. ECMAScript 2015 Language Specification (ES6). URL: https://www.ecma-international.org/ecma-262/6.0/ (дата звернення: 02.06.2025).
5. MDN Web Docs. Event-driven programming in JavaScript. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Event_loop (дата звернення: 02.06.2025).
6. MDN Web Docs. Classes in JavaScript. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes (дата звернення: 03.06.2025).
7. MDN Web Docs. Asynchronous JavaScript: Promises, async/await. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous (дата звернення: 03.06.2025).
8. Tilkov S., Vinoski S. Node.js: Using JavaScript to Build High-Performance Network Programs. IEEE Internet Computing, 2010. Vol. 14, No. 6, pp. 80-83.
9. React – A JavaScript library for building user interfaces. URL: https://reactjs.org/ (дата звернення: 03.06.2025).
10. TypeScript Documentation. URL: https://www.typescriptlang.org/docs/ (дата звернення: 03.06.2025).
11. ECMAScript® 2023 Language Specification. URL: https://tc39.es/ecma262/ (дата звернення: 03.06.2025).
12. Microsoft Docs. TypeScript Handbook: Interfaces and Modules. URL: https://www.typescriptlang.org/docs/handbook/interfaces.html (дата звернення: 03.06.2025).
13. K. Rauschmayer. Exploring TypeScript. 1st ed. Manning Publications, 2021. 350 p.

14. DefinitelyTyped – High-quality TypeScript type definitions. URL: https://definitelytyped.org/ (дата звернення: 03.06.2025).
15. Angular Documentation. URL: https://angular.io/ (дата звернення: 04.06.2025).
16. React documentation. URL: https://reactjs.org/docs/getting-started.html (дата звернення: 04.06.2025).
17. JSX Introduction. URL: https://reactjs.org/docs/introducing-jsx.html (дата звернення: 04.06.2025).
18. Virtual DOM and React Rendering. URL: https://reactjs.org/docs/faq-internals.html (дата звернення: 04.06.2025).
19. State Management in React. URL: https://reactjs.org/docs/state-and-lifecycle.html (дата звернення: 04.06.2025).
20. R. Clark, Fullstack React with TypeScript and Next.js, O'Reilly Media, 2023. 20 p.
21. M. Banks, Server-Side Rendering with Next.js, Apress, 2022. 63 p.
22. J. Smith, Modern Web Development with Next.js, Manning Publications, 2024. 25p.
23. K. Lee, TypeScript in Modern React Projects, Packt Publishing, 2023. 87p.
24. J. R. Bakar, NestJS - The Complete Guide to Building Efficient Server-side Applications, Packt Publishing, 2023, 15-60p.
25. S. Patel, Mastering Backend Development with NestJS, Manning Publications, 2024, 30-80p.
26. A. Fernandez, Modern Node.js Frameworks: NestJS and Beyond, O'Reilly Media, 2023, 20-40p.
27. Sebastián Ramírez, FastAPI Documentation, 2020. 13-15p.
28. M. Richardson, Python Web Development with FastAPI, 2022. 39-41p.
29. A. Smith, API Security in Modern Frameworks, 2023. 65-69p.

ДОДАТОК А - Вигляд логів PinoLogger

```
LoggerModule.forRoot({
    pinoHttp: {
        level: process.env.NODE_ENV !== 'production' ? 'debug' : 'info',
        transport:
            process.env.NODE_ENV !== 'production'
                ? {
                        target: 'pino-pretty',
                        options: {
                            colorize: true,
                            singleLine: false,
                            translateTime: 'HH:MM:ss'
                        }
                    }
                : undefined
    }
})
```

Конфігурація логгера

```
    }
    responseTime: 6
[Nest] 85802  - 09/21/2025, 4:09:49 PM     LOG [ForecastService] Використовуємо алгоритм: forecast_sarima для 24 записів
[Nest] 85802  - 09/21/2025, 4:09:51 PM     LOG [ForecastService] Python скрипт завершився з кодом 0
[Nest] 85802  - 09/21/2025, 4:09:51 PM     LOG [ForecastService] Stderr: /Users/vladyslav/Documents/MasterDiploma/Algorithms/For
dArray/Index representation will drop timezone information.
  df["date"] = pd.to_datetime(df["date"]).dt.to_period("M").dt.to_timestamp()
/Users/vladyslav/.pyenv/versions/3.11.13/lib/python3.11/site-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: To
 seasonal ARMA. All parameters except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
/Users/vladyslav/.pyenv/versions/3.11.13/lib/python3.11/site-packages/statsmodels/base/model.py:607: ConvergenceWarning: Maximum
_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
/Users/vladyslav/Documents/MasterDiploma/Algorithms/Forecast/forecast.py:162: FutureWarning: Series.__getitem__ treating keys as
 keys will always be treated as labels (consistent with DataFrame behavior). To access a value by position, use `ser.iloc[pos]`
  forecast_value = result[0] if hasattr(result, '__getitem__') else result

[Nest] 85802  - 09/21/2025, 4:09:51 PM     LOG [ForecastService] Stdout: {"algorithm": "sarima", "forecast": [370.598666523174]}

[Nest] 85802  - 09/21/2025, 4:09:51 PM     LOG [ForecastService] Алгоритм: forecast_sarima
[Nest] 85802  - 09/21/2025, 4:09:51 PM     LOG [ForecastService] Кількість записів: 24
[Nest] 85802  - 09/21/2025, 4:09:51 PM     LOG [ForecastService] Прогноз успішно виконано за допомогою forecast_sarima
[13:09:51] INFO (85802): request completed
    req: {
      "id": 8,
      "method": "GET",
      "url": "/api/forecast/6042",
      "query": {},
      "params": {
        "path": [
          "forecast",
          "6042"
        ]
      },
      "headers": {
        "host": "localhost:5070",
        "connection": "keep-alive",
        "sec-ch-ua-platform": "\"macOS\"",
        "user-agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 S
        "accept": "application/json, text/plain, */*",
        "sec-ch-ua": "\"Chromium\";v=\"140\", \"Not=A?Brand\";v=\"24\", \"Google Chrome\";v=\"140\"",
        "sec-ch-ua-mobile": "?0",
        "origin": "http://localhost:3070",
                                                                                    ⌘K to generate a command
```

Логи

ДОДАТОК Б - Фрагменти програмного коду API

```
import pandas as pd
import os
import csv

input_dir = "D:\\NLTU\\MasterDiploma\\DataCSV"
output_dir = os.path.join(folder_path, "cleaned")
os.makedirs(output_dir, exist_ok=True)

for filename in os.listdir(input_dir):
    if not filename.endswith('.csv'):
        continue

    file_path = os.path.join(input_dir, filename)
    try:
        df = pd.read_csv(file_path)

        # Очищуємо пробіли і зайві лапки у Номенклатура
        if 'Номенклатура' in df.columns:
            df['Номенклатура'] = (
                df['Номенклатура']
                .astype(str)
                .str.strip()
                .str.strip('"')
            )

        # Видаляємо порожні колонки між Номенклатура і Кількість
        cols = df.columns.tolist()
        if 'Номенклатура' in cols and 'Кількість' in cols:
            i1 = cols.index('Номенклатура')
            i2 = cols.index('Кількість')
            if i2 - i1 > 1:
                df.drop(columns=cols[i1+1:i2], inplace=True)

        # Зберігаємо з обгорткою ВСІХ значень у лапки
        output_path = os.path.join(output_dir, filename)
        df.to_csv(output_path, index=False, quoting=csv.QUOTE_ALL)

        print(f" {filename} → обгорнуто в лапки та збережено.")
    except Exception as e:
        print(f" Помилка в {filename}: {e}")

df.rename(columns={'datetime': 'Date'}, inplace=True)
df.sort_values('Date', inplace=True)
df.set_index('Date', inplace=True)

ts = df['Close'].dropna().values[-43800:]
ts_scaled = MinMaxScaler().fit_transform(ts.reshape(-1, 1)).flatten()

window_size = 250
window_step = 20

l = []
time_idx = []

rp_full = RecurrencePlot(ts_scaled, dim=1, tau=1, threshold_std=0.1)

for start in range(0, len(ts_scaled) - window_size + 1, window_step):
    window = ts_scaled[start:start + window_size]

    ts_obj = TimeSeries(window, embedding_dimension=1, time_delay=1)
```

```
    settings = Settings(
        ts_obj,
        analysis_type=Classic,
        neighbourhood=FixedRadius(0.1),
        similarity_measure=EuclideanMetric,
        theiler_corrector=1
    )

    computation = RQAComputation.create(settings, verbose=False)
    res = computation.run()

    rr.append(res.recurrence_rate)
    det.append(res.determinism)
    l.append(res.average_diagonal_line)
    entr.append(res.entropy_diagonal_lines)
    lam.append(res.laminarity)
    tt.append(res.trapping_time)

    rp_win = RecurrencePlot(window, dim=1, tau=1, threshold_std=0.1)

    mean_t1 = np.mean(rp_win.mean_recurrence_time())
    t1.append(mean_t1 if not np.isnan(mean_t1) else 0)

    try:
        mean_t2 = np.mean(rp_win.second_order_recurrence_time())
        t2.append(mean_t2 if not np.isnan(mean_t2) else 0)
    except AttributeError:
        rec_mat = rp_win.recurrence_matrix()
        second_returns = []
        for row in range(rec_mat.shape[0]):
            points = np.where(rec_mat[row, :])[0]
            if len(points) >= 2:
                diffs = points[1:] - points[:-1]
                second_returns.extend(diffs)
        mean_t2 = np.mean(second_returns) if second_returns else 0
        t2.append(mean_t2)

    time_idx.append(start + window_size // 2)
df.rename(columns={'datetime': 'Date'}, inplace=True)
df.sort_values('Date', inplace=True)
df.set_index('Date', inplace=True)

ts = df['Close'].dropna().values[-43800:]
ts_scaled = MinMaxScaler().fit_transform(ts.reshape(-1, 1)).flatten()

window_size = 250
window_step = 20


rp_full = RecurrencePlot(ts_scaled, dim=1, tau=1, threshold_std=0.1)

for start in range(0, len(ts_scaled) - window_size + 1, window_step):
    window = ts_scaled[start:start + window_size]

    ts_obj = TimeSeries(window, embedding_dimension=1, time_delay=1)

    settings = Settings(
        ts_obj,
        analysis_type=Classic,
        neighbourhood=FixedRadius(0.1),
        similarity_measure=EuclideanMetric,
        theiler_corrector=1
    )
```

```
    computation = RQAComputation.create(settings, verbose=False)
    res = computation.run()

    rr.append(res.recurrence_rate)
    det.append(res.determinism)
    l.append(res.average_diagonal_line)
    entr.append(res.entropy_diagonal_lines)
    lam.append(res.laminarity)
    tt.append(res.trapping_time)

    rp_win = RecurrencePlot(window, dim=1, tau=1, threshold_std=0.1)

    mean_t1 = np.mean(rp_win.mean_recurrence_time())
    t1.append(mean_t1 if not np.isnan(mean_t1) else 0)

    try:
        mean_t2 = np.mean(rp_win.second_order_recurrence_time())
        t2.append(mean_t2 if not np.isnan(mean_t2) else 0)
    except AttributeError:
        rec_mat = rp_win.recurrence_matrix()
        second_returns = []
        for row in range(rec_mat.shape[0]):
            points = np.where(rec_mat[row, :])[0]
            if len(points) >= 2:
                diffs = points[1:] - points[:-1]
                second_returns.extend(diffs)
        mean_t2 = np.mean(second_returns) if second_returns else 0
        t2.append(mean_t2)

    time_idx.append(start + window_size // 2)
df.rename(columns={'datetime': 'Date'}, inplace=True)
df.sort_values('Date', inplace=True)
df.set_index('Date', inplace=True)

ts = df['Close'].dropna().values[-43800:]
ts_scaled = MinMaxScaler().fit_transform(ts.reshape(-1, 1)).flatten()


rp_full = RecurrencePlot(ts_scaled, dim=1, tau=1, threshold_std=0.1)

for start in range(0, len(ts_scaled) - window_size + 1, window_step):
    window = ts_scaled[start:start + window_size]

    ts_obj = TimeSeries(window, embedding_dimension=1, time_delay=1)

    settings = Settings(
        ts_obj,
        analysis_type=Classic,
        neighbourhood=FixedRadius(0.1),
        similarity_measure=EuclideanMetric,
        theiler_corrector=1
    )

    computation = RQAComputation.create(settings, verbose=False)
    res = computation.run()

    rr.append(res.recurrence_rate)
    det.append(res.determinism)
    l.append(res.average_diagonal_line)
    entr.append(res.entropy_diagonal_lines)
    lam.append(res.laminarity)
    tt.append(res.trapping_time)
```

```
   rp_win = RecurrencePlot(window, dim=1, tau=1, threshold_std=0.1)

   mean_t1 = np.mean(rp_win.mean_recurrence_time())
   t1.append(mean_t1 if not np.isnan(mean_t1) else 0)

   try:
      mean_t2 = np.mean(rp_win.second_order_recurrence_time())
      t2.append(mean_t2 if not np.isnan(mean_t2) else 0)
   except AttributeError:
      rec_mat = rp_win.recurrence_matrix()
      second_returns = []
      for row in range(rec_mat.shape[0]):
         points = np.where(rec_mat[row, :])[0]
         if len(points) >= 2:
            diffs = points[1:] - points[:-1]
            second_returns.extend(diffs)
      mean_t2 = np.mean(second_returns) if second_returns else 0
      t2.append(mean_t2)

   time_idx.append(start + window_size // 2)
```

```
import { Module } from '@nestjs/common'
import { AppController } from './app.controller'
import { AppService } from './app.service'
import { AuthModule } from './auth/auth.module'
import { ConfigModule } from '@nestjs/config'
import { LoggerModule } from 'nestjs-pino'
import { SearchModule } from './search/search.module'
import { ForecastModule } from './forecast/forecast.module'

@Module({
	imports: [
		ConfigModule.forRoot({
			isGlobal: true
		}),
		AuthModule,
		SearchModule,
		ForecastModule,
		LoggerModule.forRoot({
			pinoHttp: {
				level: process.env.NODE_ENV !== 'production' ? 'debug' : 'info',
				transport:
					process.env.NODE_ENV !== 'production'
						? {
								target: 'pino-pretty',
								options: {
									colorize: true,
									singleLine: false,
									translateTime: 'HH:MM:ss'
								}
							}
						: undefined
			}
		})
	],
	controllers: [AppController],
	providers: [AppService]
})
export class AppModule {}
import {
	Controller,
	Post,
```

```
	Body,
	Res,
	Req,
	UnauthorizedException,
	Get,
	UseGuards
} from '@nestjs/common'
import { Response, Request } from 'express'
import { AuthService } from './auth.service'
import { LoginDto } from './dto/login.dto'
import { RegisterDto } from './dto/register.dto'
import { AuthGuard } from '@nestjs/passport'
import { PinoLogger } from 'nestjs-pino'
import { JwtAuthGuard } from 'src/common/guards/jwt-auth.guard'
import { JWTPayload } from 'types/jwt-payload'

@Controller('auth')
export class AuthController {
	constructor(
		private authService: AuthService,
		private readonly logger: PinoLogger
	) {
		this.logger.setContext(AuthController.name)
	}

	@Get('me')
	@UseGuards(JwtAuthGuard)
	async getMe(@Req() req: Request) {
		return this.authService.getMe(req.user as JWTPayload)
	}

	@Post('login')
	async login(@Body() dto: LoginDto, @Res({ passthrough: true }) res: Response) {
		const data = await this.authService.login(dto)

		res.cookie('access_token', data.access_token, {
			httpOnly: true,
			secure: false, // Set to true if using HTTPS
			sameSite: 'strict',
			maxAge: 7 * 24 * 60 * 60 * 1000
		})

		res.cookie('refresh_token', data.refresh_token, {
			httpOnly: true,
			secure: false, // Set to true if using HTTPS
			sameSite: 'strict',
			maxAge: 7 * 24 * 60 * 60 * 1000
		})
		this.logger.info('User logged in successfully', dto.email)
		return { message: 'Login successful' }
	}

	@Post('register')
	async register(@Body() dto: RegisterDto, @Res({ passthrough: true }) res: Response) {
		const data = await this.authService.register(dto)

		res.cookie('access_token', data.access_token, {
			httpOnly: true,
			secure: false, // Set to true if using HTTPS
			sameSite: 'strict',
			maxAge: 7 * 24 * 60 * 60 * 1000
		})
```

```
        res.cookie('refresh_token', data.refresh_token, {
                httpOnly: true,
                secure: false, // Set to true if using HTTPS
                sameSite: 'strict',
                maxAge: 7 * 24 * 60 * 60 * 1000
        })

        this.logger.info('User registered successfully', dto.email)
        return { message: 'Login successful' }
}

@Post('refresh')
async refresh(@Req() req: Request, @Res({ passthrough: true }) res: Response) {
        const refreshToken = req.cookies['refresh_token']
        if (!refreshToken) {
                this.logger.warn('No refresh token provided')
                throw new UnauthorizedException('No refresh token provided')
        }

        const payload = await this.authService.verifyRefreshToken(refreshToken)
        delete payload.iat
        delete payload.exp

        const newAccessToken = this.authService.issueAccessToken(payload)

        res.cookie('access_token', newAccessToken, {
                httpOnly: true,
                secure: false, // Set to true if using HTTPS
                sameSite: 'strict',
                maxAge: 7 * 24 * 60 * 60 * 1000
        })

        // res.cookie('refresh_token', newRefreshToken, {
        //      httpOnly: true,
        //      secure: true,
        //      sameSite: 'strict',
        //      maxAge: 7 * 24 * 60 * 60 * 1000
        // })
        this.logger.info('Token refreshed successfully', payload.email)
        return { message: 'Token refreshed successfully' }
}

@Get('google')
@UseGuards(AuthGuard('google'))
async googleLogin() {}

@Get('google/callback')
@UseGuards(AuthGuard('google'))
async googleAuthRedirect(@Req() req, @Res() res: Response) {
        const tokens = await this.authService.googleLogin(req.user)

        res.cookie('access_token', tokens.access_token, {
                httpOnly: true,
                secure: false,
                sameSite: 'strict',
                maxAge: 7 * 24 * 60 * 60 * 1000
        })

        res.cookie('refresh_token', tokens.refresh_token, {
                httpOnly: true,
                secure: false,
                sameSite: 'strict',
                maxAge: 7 * 24 * 60 * 60 * 1000
```

```
		})

		return res.redirect(process.env.FRONTEND_URL || 'http://localhost:3000')
	}

	@Post('logout')
	logout(@Res({ passthrough: true }) res: Response) {
		res.clearCookie('access_token')
		res.clearCookie('refresh_token')
		return { message: 'Logged out' }
	}
}
import { ConflictException, Injectable, UnauthorizedException } from '@nestjs/common'
import { JwtService } from '@nestjs/jwt'
import { ConfigService } from '@nestjs/config'
import { RegisterDto } from './dto/register.dto'
import { PrismaService } from 'prisma/prisma.service'
import * as bcrypt from 'bcrypt'
import { LoginDto } from './dto/login.dto'
import { JWTPayload } from '../../types/jwt-payload'

@Injectable()
export class AuthService {
	private saltRounds: number
	constructor(
		private jwtService: JwtService,
		private configService: ConfigService,
		private readonly prismaService: PrismaService
	) {
		const saltRoundsStr = this.configService.get<string>('SALT_ROUNDS')
		this.saltRounds = saltRoundsStr ? parseInt(saltRoundsStr, 10) : 12
	}

	async getMe(user: JWTPayload) {
		const _user = await this.prismaService.user.findUnique({
			where: { id: user.id },
			select: { id: true, email: true, name: true, role: true, picture: true }
		})
		if (!_user) throw new UnauthorizedException('Користувача не знайдено')
		return _user
	}

	async googleLogin(user: any) {
		let existingUser = await this.prismaService.user.findUnique({
			where: { email: user.email }
		})

		if (!existingUser) {
			existingUser = await this.prismaService.user.create({
				data: {
					name: user.name,
					email: user.email,
					authMethod: 'GOOGLE',
					picture: user.picture
				}
			})
		}

		const JWTPayload = {
			id: existingUser.id,
			email: existingUser.email,
			name: existingUser.name,
			role: existingUser.role
```

```
		}

		const access_token = this.issueAccessToken(JWTPayload)
		const refresh_token = this.issueRefreshToken(JWTPayload)

		return { access_token, refresh_token }
	}

	async login(user: LoginDto) {
		const existingUser = await this.prismaService.user.findUnique({
			where: { email: user.email }
		})
		if (!existingUser) throw new ConflictException('Користувач з таким email не існує')
		if (!existingUser.password)
			throw new ConflictException(
				'Схоже, що ви зареєстровані через Google. Спробуйте увійти через Google.'
			)
		const isPasswordValid = await bcrypt.compare(user.password, existingUser.password)
		if (isPasswordValid) {
			const JWTPayload = {
				id: existingUser.id.toString(),
				email: existingUser.email,
				name: existingUser.name,
				role: existingUser.role
			}
			const access_token = this.issueAccessToken(JWTPayload)
			const refresh_token = this.issueRefreshToken(JWTPayload)
			return { access_token, refresh_token }
		} else throw new ConflictException('Невірний пароль')
	}

	async register(user: RegisterDto) {
		const oldUser = await this.prismaService.user.findUnique({
			where: { email: user.email }
		})
		if (oldUser) throw new ConflictException('Користувач з таким email вже існує')

		const newUser = await this.prismaService.user.create({
			data: {
				name: user.name,
				email: user.email,
				password: await bcrypt.hash(user.password, this.saltRounds)
			}
		})
		const JWTPayload: JWTPayload = {
			id: newUser.id,
			email: newUser.email,
			name: newUser.name,
			role: newUser.role
		}

		const access_token = this.issueAccessToken(JWTPayload)
		const refresh_token = this.issueRefreshToken(JWTPayload)

		return { access_token, refresh_token }
	}

	issueAccessToken(data: JWTPayload) {
		const access_token = this.jwtService.sign(data, {
			secret: this.configService.get<string>('JWT_SECRET'),
			expiresIn: this.configService.get<string>('JWT_EXPIRATION')
		})
```

```
		return access_token
	}

	issueRefreshToken(data: JWTPayload) {
		const refreshToken = this.jwtService.sign(data, {
			secret: this.configService.get<string>('REFRESH_JWT_SECRET'),
			expiresIn: this.configService.get<string>('REFRESH_JWT_EXPIRATION')
		})

		return refreshToken
	}

	async verifyRefreshToken(token: string) {
		try {
			return await this.jwtService.verifyAsync(token, {
				secret: this.configService.get<string>('REFRESH_JWT_SECRET')
			})
		} catch (error) {
			throw new UnauthorizedException('Недійсний токен оновлення.')
		}
	}
}
import { Controller, Get, Param, HttpException, HttpStatus, ParseIntPipe, BadRequestException,
InternalServerErrorException } from '@nestjs/common'
import { ForecastService } from './forecast.service'

@Controller('forecast')
export class ForecastController {
	constructor(private readonly forecastService: ForecastService) {}

	@Get(':productId')
	async createForecast(@Param('productId', ParseIntPipe) productId: number) {
		try {
			const result = await this.forecastService.runForecast(productId)

			return {
				success: true,
				data: {
					forecast: result.forecast,
					algorithm: result.algorithm,
					chartData: result.chartData,
					historicalData: result.historicalData
				},
				message: 'Прогноз успішно створено'
			}
		} catch (error) {
			// Спробуємо розпарсити детальну помилку з сервісу
			let errorDetails
			try {
				errorDetails = JSON.parse(error.message)
			} catch {
				// Якщо не вдалося розпарсити, використовуємо базову помилку
				errorDetails = {
					message: 'Помилка при створенні прогнозу',
					details: error.message
				}
			}

			// Визначаємо тип помилки та відповідний HTTP статус
			if (error.name === 'NotFoundException') {
				throw new HttpException(
					{
						success: false,
```

```
                                        message: errorDetails.message,
                                        details: errorDetails.details,
                                        productId
                                },
                                HttpStatus.NOT_FOUND
                        )
                }

                // Для помилок Python скрипту або алгоритмів
                if (errorDetails.algorithm) {
                        throw new HttpException(
                                {
                                        success: false,
                                        message: errorDetails.message,
                                        details: errorDetails.details,
                                        algorithm: errorDetails.algorithm,
                                        dataLength: errorDetails.dataLength,
                                        productId
                                },
                                HttpStatus.UNPROCESSABLE_ENTITY
                        )
                }

                // Загальна помилка сервера
                throw new HttpException(
                        {
                                success: false,
                                message: errorDetails.message,
                                details: errorDetails.details,
                                productId
                        },
                        HttpStatus.INTERNAL_SERVER_ERROR
                )
        }
    }
}
import { Injectable, Logger, NotFoundException } from '@nestjs/common'
import { spawn } from 'child_process'
import { PrismaService } from 'prisma/prisma.service'

@Injectable()
export class ForecastService {
    private readonly logger = new Logger(ForecastService.name)

    constructor(private readonly prismaService: PrismaService) {}

    async runForecast(productId: number): Promise<any> {
        // Витягуємо дані з БД
        const productHistories = await this.prismaService.productHistory.findMany({
            where: {
                productId: productId
            },
            orderBy: {
                date: 'asc'
            }
        })

        if (productHistories.length === 0) {
            throw new NotFoundException(`Продукт з ID ${productId} не знайдено або немає історії`)
        }

        // Перетворюємо дані в формат, який очікує Python скрипт
        const data = productHistories.map(history => ({
```

```
                        quantity: history.quantity,
                        max_rate: history.max_rate,
                        cost_usd: history.cost_usd,
                        cost_price_usd: history.cost_price_usd || 0,
                        profit_usd: history.profit_usd,
                        cost_in_uah: history.cost_in_uah || 0,
                        cost_price_uah: history.cost_price_uah || 0,
                        profit_uah: history.profit_uah || 0,
                        date: history.date.toISOString()
                }))

                // Перевіряємо, чи є ненульові значення quantity
                const nonZeroQuantities = data.filter(item => item.quantity > 0)
                if (nonZeroQuantities.length === 0) {
                        throw new NotFoundException(`Всі значення quantity дорівнюють 0. Неможливо створити прогноз`)
                }
                const dataLength = data.length
                let algorithm: string
                // Визначаємо алгоритм залежно від довжини масиву
                if (dataLength < 2) {
                        throw new NotFoundException(`Недостатньо даних для прогнозу. Потрібно мінімум 2 записи, отримано ${dataLength}`)
                } else if (dataLength < 10) {
                        algorithm = 'forecast_prophet'
                } else if (dataLength < 12) {
                        algorithm = 'forecast_prophet' // SSARIMA потребує мінімум 12 записів
                } else if (dataLength >= 12 && dataLength <= 36) {
                        algorithm = 'forecast_sSARIMA'
                } else {
                        algorithm = 'forecast_lstm'
                }

                this.logger.log(`Використовуємо алгоритм: ${algorithm} для ${dataLength} записів`)

                return new Promise((resolve, reject) => {
                        // Запускаємо Python скрипт
                        const pythonProcess = spawn('python3', [
                                '/Users/vladyslav/Documents/MasterDiploma/Algorithms/Forecast/forecast.py',
                                algorithm,
                                JSON.stringify(data)
                        ])

                        let result = ''
                        let error = ''

                        pythonProcess.stdout.on('data', data => {
                                result += data.toString()
                        })

                        pythonProcess.stderr.on('data', data => {
                                const errorData = data.toString()
                                // Ігноруємо попередження TensorFlow та Prophet, але зберігаємо справжні помилки
                                if (!errorData.includes('Epoch') &&
                                        !errorData.includes('WARNING') &&
                                        !errorData.includes('INFO') &&
                                        !errorData.includes('Initializing') &&
                                        !errorData.includes('Disabling') &&
                                        !errorData.includes('Optimization terminated') &&
                                        !errorData.includes('INFO:prophet') &&
                                        !errorData.includes('INFO:cmdstanpy')) {
                                        error += errorData
                                }
```

```
})

pythonProcess.on('close', code => {
	this.logger.log(`Python скрипт завершився з кодом ${code}`)
	this.logger.log(`Stderr: ${error}`)
	this.logger.log(`Stdout: ${result}`)
	this.logger.log(`Алгоритм: ${algorithm}`)
	this.logger.log(`Кількість записів: ${dataLength}`)

	if (code !== 0) {
		// Створюємо детальну помилку
		const errorMessage = error || 'Невідома помилка Python скрипту'
		const errorDetails = {
			message: `Помилка виконання алгоритму ${algorithm}`,
			details: errorMessage,
			algorithm,
			dataLength,
			exitCode: code,
			stdout: result
		}
		reject(new Error(JSON.stringify(errorDetails)))
		return
	}

	try {
		// Парсимо результат JSON
		const parsedResult = JSON.parse(result)

		// Перевіряємо, чи є помилка в результаті
		if (parsedResult.error) {
			const errorDetails = {
				message: `Помилка в алгоритмі ${algorithm}`,
				details: parsedResult.error,
				algorithm,
				dataLength
			}
			reject(new Error(JSON.stringify(errorDetails)))
			return
		}

		this.logger.log(`Прогноз успішно виконано за допомогою ${algorithm}`)

		// Додаємо історичні дані для побудови графіка
		const chartData = data.map(item => ({
			date: item.date,
			quantity: item.quantity
		}))

		// Формуємо повну відповідь з прогнозом та даними для графіка
		const response = {
			...parsedResult,
			chartData: chartData,
			historicalData: {
				dates: chartData.map(item => item.date),
				quantities: chartData.map(item => item.quantity)
			}
		}

		resolve(response)
	} catch (parseError) {
		this.logger.error(`Помилка парсингу результату: ${parseError}`)
		this.logger.error(`Raw result: ${result}`)
```

```
                                        const errorDetails = {
                                                message: 'Помилка обробки результату прогнозу',
                                                details: `Не вдалося розпарсити результат: ${parseError.message}`,
                                                algorithm,
                                                dataLength,
                                                rawResult: result
                                        }
                                        reject(new Error(JSON.stringify(errorDetails)))
                                }
                        })

                        pythonProcess.on('error', err => {
                                this.logger.error(`Помилка запуску Python процесу: ${err.message}`)
                                reject(new Error(`Помилка запуску Python процесу: ${err.message}`))
                        })
                })
        }
}
```

ДОДАТОК В - Фрагменти програмного коду UI

```
'use client'

import { Header } from '@/components/core/header/Header'
import { SearchResults } from '@/components/core/search/SearchResults'
import { ForecastDisplay } from '@/components/core/forecast/ForecastDisplay'
import { useSearch } from '@/hooks/useSearch'
import { useForecast } from '@/hooks/useForecast'
import { useState } from 'react'

export const Home = () => {
        const { query, setQuery, results, isLoading, error } = useSearch()
        const { forecast, isLoading: isForecastLoading, error: forecastError, getForecast } = useForecast()
        const [selectedProductId, setSelectedProductId] = useState<number | null>(null)

        const handleForecast = (productId: number) => {
                setSelectedProductId(productId)
                getForecast(productId)
        }

        return (
                <div className='flex flex-col items-center '>
                        <div className='w-full '>
                                <Header onSearch={setQuery} />
                                <div className='flex flex-row h-[calc(100vh-4rem)]'>
                                        <div className='bg-gray-50 flex flex-col min-w-80 max-w-80 overflow-y-auto'>
                                                <SearchResults
                                                        results={results}
                                                        isLoading={isLoading}
                                                        error={error}
                                                        query={query}
                                                        onForecast={handleForecast}
                                                        selectedProductId={selectedProductId}
                                                />
                                        </div>
                                        <div className='flex flex-col flex-1 bg-white'>
                                                <ForecastDisplay
                                                        forecast={forecast}
                                                        isLoading={isForecastLoading}
                                                        error={forecastError}
                                                        productId={selectedProductId}
```

```
                                        />
                                    </div>
                                </div>
                            </div>
                        </div>
                )
        }

'use client'

import { Product } from '@/types/product'
import { Card } from '@/components/ui/card'
import { Button } from '@/components/ui/button'
import { Loader2, TrendingUp } from 'lucide-react'

interface SearchResultsProps {
        results: Product[]
        isLoading: boolean
        error: string | null
        query: string
        onForecast: (productId: number) => void
        selectedProductId: number | null
}

export const SearchResults = ({ results, isLoading, error, query, onForecast, selectedProductId }: SearchResultsProps) => {
        if (!query.trim()) {
                return (
                        <div className='p-3 text-center text-muted-foreground text-sm'>
                                Введіть текст для пошуку товарів
                        </div>
                )
        }

        if (isLoading) {
                return (
                        <div className='flex items-center justify-center p-6'>
                                <Loader2 className='h-4 w-4 animate-spin' />
                                <span className='ml-2 text-sm'>Пошук...</span>
                        </div>
                )
        }

        if (error) {
                return (
                        <div className='p-3 text-center text-red-500 text-sm'>
                                {error}
                        </div>
                )
        }

        if (results.length === 0) {
                return (
                        <div className='p-3 text-center text-muted-foreground text-sm'>
                                Товари не знайдені
                        </div>
                )
        }

        return (
                <div className='p-3'>
                        <div className='mb-3'>
                                <h3 className='text-sm font-semibold text-gray-700'>
                                        Результати пошуку ({results.length})
```

```
                                        </h3>
                                </div>
                                <div className='space-y-2'>
                                        {results.map((product) => (
                                                <Card
                                                        key={product.id}
                                                        className={`p-3 hover:shadow-md transition-shadow border ${
                                                                selectedProductId === product.id
                                                                        ? 'border-blue-500 bg-blue-50'
                                                                        : 'border-gray-200'
                                                        }`}
                                                >
                                                        <div className='space-y-2'>
                                                                <div className='text-xs text-muted-foreground font-mono'>
                                                                        ID: {product.id}
                                                                </div>
                                                                <div       className='font-medium       text-xs       leading-tight
text-gray-800 line-clamp-3'>
                                                                        {product.nomenclature}
                                                                </div>
                                                                <Button
                                                                        size='sm'
                                                                        variant={selectedProductId === product.id ? 'default' :
'outline'}
                                                                        onClick={() => onForecast(product.id)}
                                                                        className='w-full h-7 text-xs'
                                                                >
                                                                        <TrendingUp className='h-3 w-3 mr-1' />
                                                                        Прогноз
                                                                </Button>
                                                        </div>
                                                </Card>
                                        ))}
                                </div>
                        </div>
                )
}
'use client'

import { ForecastData } from '@/types/forecast'
import { Card } from '@/components/ui/card'
import { Loader2, TrendingUp, Brain, BarChart3 } from 'lucide-react'
import { LineChart, Line, XAxis, YAxis, CartesianGrid, Tooltip, ResponsiveContainer, BarChart, Bar } from 'recharts'

interface ForecastDisplayProps {
        forecast: ForecastData | null
        isLoading: boolean
        error: string | null
        productId: number | null
}

export const ForecastDisplay = ({ forecast, isLoading, error, productId }: ForecastDisplayProps) => {
        if (!productId) {
                return (
                        <div className='p-3 text-center text-muted-foreground text-sm'>
                                Оберіть товар для прогнозування
                        </div>
                )
        }

        if (isLoading) {
                return (
                        <div className='flex items-center justify-center p-6'>
```

```
                        <Loader2 className='h-4 w-4 animate-spin' />
                        <span className='ml-2 text-sm'>Створення прогнозу...</span>
                    </div>
            )
}

if (error) {
            return (
                        <div className='p-3 text-center text-red-500 text-sm'>
                                    {error}
                        </div>
            )
}

if (!forecast) {
            return (
                        <div className='p-3 text-center text-muted-foreground text-sm'>
                                    Прогноз не знайдено
                        </div>
            )
}

// Підготовка даних для графіка
const chartData = forecast.chartData.map(item => ({
            ...item,
            date: new Date(item.date).toLocaleDateString('uk-UA', {
                        month: 'short',
                        day: 'numeric'
            })
}))

// Додаємо прогнозовані значення до графіка
const forecastData = forecast.forecast.map((value, index) => ({
            date: `Рекомендація ${index + 1}`,
            quantity: value,
            isForecast: true
}))

const combinedChartData = [...chartData, ...forecastData]

return (
            <div className='p-3 space-y-4'>
                        {/* Заголовок */}
                        <div className='mb-3'>
                                    <h3 className='text-sm font-semibold text-gray-700 flex items-center gap-2'>
                                                <TrendingUp className='h-4 w-4' />
                                                Рекомендації, щодо закупівлі
                                    </h3>
                        </div>

                        {/* Графік */}
                        <Card className='p-4 border border-gray-200'>
                                    <div className='mb-3'>
                                                <h4 className='text-sm font-medium text-gray-700 flex items-center gap-2'>
                                                            <BarChart3 className='h-4 w-4' />
                                                            Графік попиту
                                                </h4>
                                    </div>

                                    <div className='h-64 w-full'>
                                                <ResponsiveContainer width="100%" height="100%">
                                                            <LineChart data={combinedChartData}>
                                                                        <CartesianGrid strokeDasharray="3 3" />
```

```
                                        <XAxis
                                                dataKey="date"
                                                tick={{ fontSize: 12 }}
                                                angle={-45}
                                                textAnchor="end"
                                                height={60}
                                        />
                                        <YAxis tick={{ fontSize: 12 }} />
                                        <Tooltip
                                                formatter={(value, name) => [value, name === 'quantity' ? 'Кількість' : 'Рекомендація']}
                                                labelFormatter={(label) => `Дата: ${label}`}
                                        />
                                        <Line
                                                type="monotone"
                                                dataKey="quantity"
                                                stroke="#3b82f6"
                                                strokeWidth={2}
                                                dot={{ fill: '#3b82f6', strokeWidth: 2, r: 4 }}
                                                activeDot={{ r: 6 }}
                                        />
                                </LineChart>
                        </ResponsiveContainer>
                </div>
        </Card>

        {/* Деталі прогнозу */}
        <Card className='p-3 border border-blue-200 bg-blue-50'>
                <div className='space-y-2'>
                        <div className='flex items-center gap-2 text-xs text-blue-700'>
                                <Brain className='h-3 w-3' />
                                <span>Алгоритм: {forecast.algorithm}</span>
                        </div>

                        <div className='space-y-1'>
                                <div className='text-xs text-gray-600'>Реколмендована кількість:</div>
                                <div className='flex flex-wrap gap-1'>
                                        {forecast.forecast.map((value, index) => (
                                                <span
                                                        key={index}
                                                        className='inline-flex items-center px-2 py-1 rounded-md bg-blue-100 text-blue-800 text-xs font-medium'
                                                >
                                                        {value}
                                                </span>
                                        ))}
                                </div>
                        </div>

                        {forecast.forecast.length > 0 && (
                                <div className='text-xs text-gray-600'>
                                        Середнє значення: <span className='font-medium'>{forecast.forecast.reduce((a, b) => a + b, 0) / forecast.forecast.length}</span>
                                </div>
                        )}
                </div>
        </Card>
</div>
```
)
}