НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету інформаційних технологій	**Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ	_____ Белла ГОЛУБ
(підпис) (ім'я ПРІЗВИЩЕ)	(підпис) (ім'я ПРІЗВИЩЕ)
"___" _____ 2025 р.	"___" _____ 2025 р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Розробка та дослідження програмного забезпечення для управління товарними запасами з використанням алгоритмів оптимізації

Спеціальність 122 Комп'ютерні науки
(код і найменування)

Освітня програма Інформаційні управляючі системи та технології
(назва)

Орієнтація освітньої програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми
К.т.н., Доцент _____ Белла ГОЛУБ
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Керівник магістерської кваліфікаційної роботи
Д.т.н., професор _____ Наталія ЗАЄЦЬ
(науковий ступінь та вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)

Виконав _____ Олександр ВОРОНА
(підпис) (ім'я ПРІЗВИЩЕ)

КИЇВ – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСТИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук
К.т.н., Доцент _____ **Белла ГОЛУБ**
(науковий ступінь, вчене звання) (підпис) (ім'я ПРІЗВИЩЕ)
"27" Жовтня 2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

ЗДОБУВАЧУ

Вороні Олександру Олександровичу
(прізвище, ім'я, по батькові)

Спеціальність 122 Комп'ютерні науки
(код і найменування)

Освітня програма Інформаційні управляючі системи та технології
(назва)

Орієнтація освітньої програми Освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Тема магістерської кваліфікаційної роботи Розробка та дослідження програмного забезпечення для управління товарними запасами з використанням алгоритмів оптимізації

затверджена наказом від "27" Жовтня 2025р. № 2553 «С»

Термін подання завершеної роботи на кафедру 01 Грудня 2025 року
(рік, місяць, число)

Вихідні дані до магістерської кваліфікаційної роботи

Початкові статистичні дані про рух товарів, попит і залишки, необхідні для аналізу й тестування програмного забезпечення.

Перелік питань, що підлягають дослідженню:

1. Проаналізувати існуючі підходи до управління товарними запасами та алгоритми оптимізації, які застосовуються у цій сфері, і визначити їхні переваги та обмеження.

2. Розробити програмне забезпечення для управління товарними запасами, що реалізує облік, контроль та застосування обраних алгоритмів оптимізації залишків.

3. Провести експериментальні дослідження роботи програмного забезпечення, оцінити ефективність застосованих алгоритмів оптимізації та сформулювати рекомендації щодо їх використання на практиці.

Дата видачі завдання "27" Жовтня 2025 р.

Керівник магістерської кваліфікаційної роботи _____ Наталія ЗАЄЦЬ
(підпис) (ім'я ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ Олександр ВОРОНА
(ім'я ПРІЗВИЩЕ)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

ТЗ – товарний запас

БД – база даних

UML – Unified modelling language

ПО – предметна область

ВСТУП

В сучасних умовах розвитку торгівельних підприємств основною характеристикою є висока конкуренція, динамічний попит та необхідність оперативного прийняття управлінських рішень, тому одним з ключових чинників ефективної діяльності компаній є управління товарними запасами, оскільки від рівня їх оптимізації залежать фінансові результати(виторги та втрати), оборотність ресурсу та ступінь задоволення потреб покупців. Надлишкові запаси призводять до фінансових втрат, в той час як дефіцит товарів несе за собою втрату прибутку та лояльності клієнтів. Тому зараз постає актуальна потреба у розробці програного забезпечення (далі: ПЗ), яке буде прогнозувати та оптимізовувати залишки на основі математичних моделей та алгоритмів.

Об’єктом дослідження є процес управління товарними запасами та торговому підприємстві, то як предметом являються алгоритми оптимізації та програмні засоби для розробки системи прийняття рішень, які призведуть до оптимізації обсягу запасів товарів на складі.

Мета дослідження – це розробка та дослідження програмного забезпечення для управління товарними запасами, зокрема огляд інструментів для створення такого ПЗ та використання і порівняння алгоритмів, математичних моделей для мінімізації надлишкових запасів й підвищення ефективності складської логістики.

Для досягнення мети вирішуються такі завдання:

- Проаналізувати існуючі підходи до управління товарними запасами та алгоритми оптимізації, які застосовуються у цій сфері, і визначити їхні переваги та обмеження.
- Дослідити математичні моделі та алгоритми оптимізації, що є релевантними для дослідження.

- Розробити програмне забезпечення для управління товарними запасами, що реалізує облік, контроль та застосування обраних алгоритмів оптимізації залишків.
- Провести експериментальні дослідження роботи програмного забезпечення, оцінити ефективність застосованих алгоритмів оптимізації та сформулювати рекомендації щодо їх використання на практиці.

У процесі дослідження використовуються такі методи: методи математичного моделювання, методи оптимізації, прогнозування на основі числових рядів, а також методи програної інженерії та проєктування програмних систем.

Наукова новизна роботи полягає в розробці ПЗ, яке поєднуватиме механізми обліку товарних запасів з використанням алгоритмів оптимізації та прогнозування, що дозволить підвищити обгрунтованість управлінських рішень щодо оптимізації залишків та формування замовлень.

Апробація результатів дослідження проходила шляхом тестування ПЗ на реальних наборах даних та аналізу точності сформованих прогнозів, а також оптимальності рівнів запасів, які вийшли в результаті оптимізації. Також було прийнято участь у конференції ITETO 2025, де було представлено результати дослідження та постер.

Магістерська робота включає в себе – сторінок, -- використаних джерел та – додатків. Зокрема робота включає в себе чотири розділи: **Системний аналіз предметної області**, в якому розглядаються процеси та явища, які проходять в межах сфери торгівлі та товарного обліку. **Моделювання системи**, де було змодельовано предметну область, яку розглянуто в даній магістерській роботі. **Розробка системи**: даний розділ включає в себе архітектуру, етапи, методи та інструменти, які було використано для розробки ПЗ. **Результати дослідження** – огляд отриманих результатів дослідження, яке проводилось.

1 СИСТЕМНИЙ АНАЛІЗ

1.1. Аналіз предметної області управління товарними запасами

Управління товарними запасами (далі - ТЗ) є одним з ключових пріоритетів торгівельних підприємств, виробничих компаній та логістичних центрів, які здійснюють обіг матеріальних ресурсів. Оптимальне використання запасів суттєво впливає на фінансові результати підприємств, а раціональне управління ними, зокрема оптимізація обсягів закупівель, своєчасне поповнення та контроль залишків, призводять до стабілізації та підвищення фінансових результатів компанії, за рахунок задоволення потреб покупців та зменшенню відсотку втрат.

Розглянута далі предметна область характеризується складною структурою взаємодії між постачальниками, складами та споживачами. Вплив на процес прийняття рішень має такі фактори, як сезонність попиту (наприклад на цитрусові літом попит менше, чим зимою через сезонність товару), коливання цін, часові лаги постачання, мінливість обсягу продажів та специфіка окремих груп товарів, яка може визначатись низьким терміном придатності, або ж специфічними умовами зберігання. За відсутності ефективної системи управління є два розвитки подій: або запаси починають переходити у надлишок, що призводить до додаткових втрат, або ж навпаки, виникає дефіцит, що відштовхує від підприємства потенційних покупців.

З точки зору бізнесу, проблематика надлишкових запасів полягає в тому, що значна частина оборотного капіталу підприємства знаходяться у товарній формі. Це призводить до того, що збільшується витрати на обслуговування складських площ та підвищує ризик псування товару, в особливості якщо йдеться про продовольчі товари, зокрема «скоропорт» (товари, які мають обмежений строк реалізації, як от овочі, фрукти, молочна продукція, тощо), або зменшення його актуальності.

В свою чергу дефіцит товарів призводить до втрати потенційних, або ж вже наявних постійних покупців, що загрожує зниженню рівня сервісу та несе за собою репутаційні ризики. Наявність таких втрат означає втрату потенційного прибутку, що є дуже критичним у конкурентному середовищі. Саме тому баланс між надлишком та дефіцитом запасів є важливим з точки зору успішності бізнесу.

Раніше управління запасами здійснювалось руками досвідчених менеджерів, або за допомогою простих статистичних правил, таких як EOQ (Economic Order Quantity), метод мінімального та максимального рівнів, тощо. Хоч дані методи і дозволяють класифікувати товарний асортимент та визначити базові параметри для поповнення асортименту, вони не враховують динаміку зміни попиту та тенденцій розвитку ринку.

Сучасні умови вимагають застосування підходів, які базуються на аналізі історичних даних та математичному моделюванні, зокрема застосовуються:

- Моделі прогнозування часових рядів (лінійна регресія, ARIMA, Holt-Winters)
- Оптимізаційні методи (лінійне та нелінійне програмування, евристичні алгоритми)
- Алгоритми машинного навчання (регресійні дерева, нейронні мережі, тощо)

Однак на практиці запровадження таких методів ускладняється тим, що для їх роботи потрібно побудувати відповідну інформаційну інфраструктуру та програмне забезпечення, яке б забезпечувало централізований збір та зберігання інформації, аналіз статистичних закономірностей попиту, побудову прогнозів, визначення оптимальних обсягів закупівель для мінімізації витрат.

Відсутність на українському ринку доступного рішення, яке б поєднувало можливості прогнозування та оптимізації запасів у зручному інтерфейсі, створює проблемну ситуацію та визначає актуальність даного

дослідження**.** Реалізація програмного забезпечення, орієнтованого на малий та середній бізнес, дозволить підвищити ефективність використання товарних запасів, зменшити витрати та забезпечити стабільність логістичних процесів.

1.2. Аналіз існуючих рішень для управління товарними запасами

Управління товарними запасами є предметом активних досліджень у галузях логістики, операційного менеджменту, інформаційних технологій та математичного моделювання. У науковій літературі та практиці представлено різні підходи до вирішення задачі підтримки оптимального рівня запасів, що відрізняються як за складністю застосовуваних алгоритмів, так і за рівнем автоматизації процесів.

1.2.1. Класичні методи управління запасами

До традиційних методів в наш час належать наступні:

- **EOQ (Economic Order Quantity)** — модель економічно обґрунтованого розміру замовлення, що визначає оптимальний обсяг поповнення, виходячи з витрат на замовлення і зберігання. Недоліком моделі є припущення про сталість попиту і незмінність цін.
- **Метод (s, S)** — система поповнення запасу при досягненні мінімального рівня. Дозволяє уникнути дефіциту, але не оптимізує вартість утримання запасів.
- **ABC/XYZ-аналіз** — метод класифікації товарів за цінністю та стабільністю попиту. Дозволяє розподілити запаси за пріоритетом, однак не надає рекомендацій щодо конкретного обсягу замовлень.
- **Метод ковзного середнього і експоненційного згладжування** — використовуються для прогнозування попиту, але не забезпечують адаптації до швидких ринкових змін.

До недоліків цих методів можна віднести їх статистичність, оскільки вони не враховують історичні тренди, сезонність та зовнішні чинники, тому в умовах нестабільного ринку їх ефективність доволі низька.

1.2.2. Сучасні математичні та інтелектуальні методи

В сучасних наукових дослідженнях для прогнозування попиту частіше використовують наступні моделі та методи:

- **моделі часових рядів** (ARIMA, SARIMA, Holt-Winters);
- **регресійні моделі** з урахуванням зовнішніх факторів;
- **методи машинного навчання** (нейронні мережі, градієнтний бустинг, випадкові ліси);

На даний момент саме ці методи показують найвищу точність прогнозування, проте вимагають наявності даних у значному обсязі та відповідної обчислювальної інфраструктури.

1.2.3. Аналіз наявних програмних рішень

Зараз на ринку існує ряд систем для автоматизації товарного обліку та управління запасами. Їх коротка характеристика та порівняння наведено в Табл. 1.1.

Таблиця 1.1.

Порівняння ПЗ для товарного обліку

Програмний продукт	Характеристика	Недоліки
SAP S/4HANA SCM	Комплексне управління логістикою з прогнозуванням попиту	Висока вартість, складність впровадження
Oracle NetSuite ERP	Хмарне управління запасами та закупівлями	Недостатня гнучкість для локальних підприємств
Microsoft Dynamics 365	Інтеграція з CRM та обліковими модулями	Складне налаштування
BAS/1C УТ та Бухгалтерія	Широке використання в Україні, базова аналітика	Відсутність механізмів оптимізації складських залишків
Zoho Inventory / Odoo	Доступні хмарні рішення	Прості моделі прогнозування, обмежена адаптація

Більшість наявних на ринку рішень орієнтовані лише на облік та звітність, тоді як функціонал для прогнозування та оптимізації ТЗ частково реалізовані, або ж не реалізовані взагалі.

1.2.4. Проблема, що вимагає вирішення

В результаті проведеного аналізу було виявлено, що:

- Підприємствам необхідне інструментальне рішення, що дозволить прогнозувати обсяги продажів на основі даних, які збиралися за час роботи підприємства
- Існуючі продукти або занадто складні у впровадженні, або не забезпечують потрібний функціонал для оптимізації ТЗ
- Присутній розров між теоретичними методами, які було описано в наукових роботах та їх практичною реалізацією у ПЗ для малого та середнього бізнесу

1.3. Постановка задачі дослідження

На основі аналізу ПО та огляду існуючих рішень та підходів було встановлено, що управління ТЗ на підприємствах часто здійснюється без завтосування моделей прогнозування та алгоритмів оптимізації, що призводить до виникнення проблем з надлишковими або ж дефіцитними запасами, які негативно впливають на фінансові результати підприємства.

Наявне ПЗ не завжди враховує специфіку діяльності малих та середніх підприємств, де управління запасами має бути адаптивним та економічно обгрунтованим. Здебільшого програмні продукти мають фокус на веденні обліку, тоді як інструментарій для прогнозування та оптимізації запасів або обмежений, або потребує високих витрат на впровадження**.** Водночас наукові дослідження у сфері прогнозування попиту та оптимізації систем запасів демонструють значний потенціал застосування математичних моделей часових рядів, статистичних та інтелектуальних методів для підвищення точності управлінських рішень.

Отже, маємо наступну задачу – інтеграція алгоритмів, які забезпечать можливість:

- Аналізувати дані продажів
- Виявляти закономірності та тенденції в обсягах продажів
- Формувати прогнози попиту на визначений період
- Розраховувати оптимальний рівень ТЗ

Враховуючи зазначене**,** мета даного дослідження полягає у розробці програмного забезпечення для управління товарними запасами із використанням алгоритмів оптимізації та прогнозування попиту, що дозволить мінімізувати надлишкові запаси, та запобігти дефіциту товарів у відповідні моменти.

Для досягнення даної мети необхідно пройти наступні кроки:

- Провести системний аналіз предметної області, визначити фактори, що впливають на формування оптимальних запасів.
- Дослідити існуючі методи прогнозування попиту та алгоритми оптимізації товарних запасів.
- Розробити інформаційну модель і архітектуру програмного забезпечення для управління запасами.
- Реалізувати модулі збору та зберігання даних, модуль прогнозування та модуль оптимізації.
- Інтегрувати методи аналізу та візуалізації результатів у програмний інтерфейс.
- Провести тестування програмного забезпечення на реальних або наближених до реальних даних та оцінити його ефективність

Розв’язання поставлених завдань дозволить створити інструмент підтримки управлінських рішень щодо оптимізації ТЗ на підприємстві. У подальших розділах буде виконано розробку відповідного ПЗ, математичне обгрунтування алгоритмів прогнозування та оптимізації, а також експериментальне оцінювання функціонування системи.

2. МОДЕЛЮВАННЯ СИСТЕМИ

2.1. Функціональне моделювання

Функціональне моделювання є одним із базових підходів до опису інформаційних систем, що дозволяє відобразити логіку роботи процесів, взаємодію між ними, а також потоки даних, які циркулюють у межах системи. Мета цього методу полягає у створенні моделі функціонування системи, яка описує, що саме робить система, які дані вона обробляє *і* які результати формує.

Функціональне моделювання передбачає розбиття складної системи на взаємопов'язані підпроцеси та дозволяє визначити, які саме операції здійснюються на кожному етапі обробки даних. У межах даного дослідження цей підхід використовується для опису бізнес-процесів управління товарними запасами — від формування замовлення до контролю рівня запасів і аналізу ефективності закупівель.

Основна мета моделювання — створення єдиної структурованої картини роботи системи, де кожен елемент (процес, потік, зовнішній об'єкт) має чітко визначене місце і функцію.

Це дозволяє:

- визначити входи (дані, що надходять у систему — наприклад, інформація про продажі або залишки товарів);
- описати обробку (алгоритми, що здійснюють розрахунок оптимального запасу, прогноз попиту, оновлення залишків);
- визначити виходи (результати роботи — звіти, діаграми, рекомендації до замовлень);
- ідентифікувати зовнішніх акторів (користувачі, постачальники, системи обліку, складські модулі).

2.1.1. Опис основних бізнес-процесів управління ТЗ

Процес управління товарними запасами в торгівельних підприємствах включає в себе комплекс взаємопов'язаних операцій, які спрямовані на забезпечення складських залищків на тому рівні, який буде достатнім для

безперебійного задоволення споживчого попиту за умов мінімізації витрат на зберігання та зменшення втрат. Функціонування системи охоплює такі етапи, як закупівля товару в постачальника, облік та зберігання товарних позицій на складі, аналіз продажів та формування прогнозу попиту, що допоможе визначити необхідний обсяг наступного замовлення.

Базовий процес закупівлі починається з формування потреби у поповненні запасів, тому відповідний менеджер аналізує наявні залишки, обсяги продажів за попередні періоди, динаміку попиту та інші дані. У традиційних системах цей процес здійснювався вручну, що часто призводило до прийняття необгрунтованих рішень. У запропонованій системі формування потреби доповнюється розрахунком прогнозу попиту та рекомендаціями для оптимального замовлення товарів.

Процес зберігання та обліку товарів включає в себе реєстрацію поставок, ведення обліку, контроль терміну реалізації та фіксацію переміщень між складами (в деяких торгівельних це можуть бути склади виробничих цехів, склад неліквіду, тощо). Дані про залишки є цлючовими у системі та використовуються для аналітики та прогнозування.

Процес продажів формує так звані «історичні дані» - часові ряди, які є основою для аналітичного модуля прогнозу попиту. Особливу роль в ньому відіграють сезонність, циклічність та загальні тренди в поведінці попиту. Саме від їх коректного врахування буде залежати точність сформованого прогнозу.

Одним з найважливіших процесів у контексті оптимізації є формування замовлення товару. Він передбачає наступні дії:

- Аналіз поточних залишків.
- Визначення очікуваного попиту на період постачання.
- Облік строків і обсягів постачань від постачальників.
- Розрахунок рекомендованого обсягу замовлення.
- Формування накладної на закупівлю.

Для підтримки цього процесу система повинна мати:

- механізм збору та структурування даних про рух товарів;

- модуль прогнозування попиту (на основі історичних продажів);
- модуль оптимізаційного розрахунку обсягу замовлення;
- інструменти візуалізації аналітичної інформації;
- інтерфейс взаємодії користувача з аналітичними рекомендаціями.

Таким чином, основна роль інформаційної системи полягає у зменшенні невизначеності при прийнятті рішень щодо закупівель, що забезпечується шляхом автоматизації аналізу даних та застосування математичних моделей прогнозування і оптимізації. На відміну від статичних підходів, система дозволяє адаптуватися до змін попиту та динаміки ринку.

2.1.2. Діаграма прецедентів

Діаграма прецедентів використовується для опису взаємодії користувачів із системою через набір функціональних можливостей, що надаються ПЗ. Вона дозволяє показати систему візуально, вказуючи зовнішні дії акторів та забезпечує реалізацію визначених сценаріїв використання.

Під час розробки системи було виділено виділено таких акторів:

- Менеджер закупівель — виконує аналіз запасів, формує замовлення, працює з прогнозами та аналітичними звітами.
- Складський оператор — реєструє рух товарів, фіксує поставки, контролює складські залишки.
- Адміністратор системи — організовує довідники номенклатури, постачальників, макрогруп, користувачів, а також здійснює налаштування доступів.

Діаграму системи зображено на Рис. 2.1.

Рис. 2.1. Use-case діаграма системи

Таким чином, система забезпечує три ключові групи можливостей:

- Облік запасів та руху товарів
- Аналітика та прогнозування
- Формування управлінських документів

2.1.3. Діаграма діяльності для процесу формування замовлення

Діаграма діяльності (Activity Diagram) описує послідовність дій, що виконуються під час певного процесу в системі. Вона відображає логіку процесу, починаючи, в моєму випадку, від аналізу залишків та даних продажів і закінчуючи формуванням накладної для постачальника.

Процес формування замовлення складається з таких етапів:

1. Отримання інформації про поточні залишки: система завантажує актуальні дані з бази даних (далі – БД), включаючи поточну кількість товару та динаміку його руху;
2. Аналіз історичних даних: на основі обраного замовлення система отримує дані про обсяг реалізації за певний період(наприклад попередній тиждень/аналогічний період минулого року);
3. Побудова прогнозу попиту: аналітичний модуль виконує прогнозування попиту за допомогою обраної моделі прогнозування(лінійна регресія, ковзне середнє, тощо);
4. Розрахунок оптимального обсягу замовлення: система порівнює прогнозований попит із поточними залишками, враховує час постачання та мінімальні партії замовлення, після чого приступає до формування рекомендації по кожній позиції:
5. Формування накладної: після перегляду рекомендації по позиціях, їх можна підкоригувати вручну, після чого замовлення зберігається та формується накладна на постачальника.
6. Передача накладної постачальнику: після прийому товару, накладна роздруковується з системи, підписується та передається постачальнику. Лише після цього можна провести накладну для оновлення залишків у системі.

На Рис. 2.2. наведено діаграму для процесу формування замовлення та проведення накладної.

Рис. 2.2. Діаграма активності для процесу формування та проведення замовлення

2.2. Об’єктно-орієнтоване моделювання

Об’єктно-орієнтоване моделювання дозволяє подати предметну область системи у вигляді набору сутностей (класів), їхніх властивостей та зв’язків. Даний підхід забезпечує природне відображення реальних бізнес-об’єктів, таких як товар, постачальник, накладна чи складський запис. Використання класів спрощує організацію програмного коду, оскільки кожна сутність має чітку відповідальність та набір операцій, що можуть бути до неї застосовані.

Метою об’єктно-орієнтованого моделювання у даній роботі є формування структури взаємодії основних елементів системи управління товарними запасами, що в подальшому стане основою для програмної реалізації. Для цього була побудована діаграма класів, яка відображає ключові сутності бази даних і зв’язки між ними.

2.2.1. Моделі предметної області та діаграма класів

У межах об’єктно-орієнтованого моделювання системи управління ТЗ було побудовано діаграму класів, що відображає структуру БД для шару Models, відповідно до архітектурного шаблону майбутньої системи. Діаграма класів являє собою модель, яка описує дані, що зберігаються у системі, їх властивості, а також типи зв’язків між об’єктами.

Окрім самих класів в діаграмі даного типу також важливі зв’язки, оскільки вони вказують на рівень залежності класів один від одного. У діаграмі класів системи використано такі типи відношень

- Асоціація (Association) – загальний зв’язок між класами, де об’єкти взаємодіють один з одним.
- Агрегація (Aggregation) – “ціле-частина”, але частина може існувати незалежно від цілого.
- Композиція (Composition) – сильне відношення “ціле-частина”, де частина не існує без цілого.
- Успадкування (Generalization) – використовується тоді, коли один клас наслідує властивості іншого.

- Залежність (Dependency) – один клас використовує методи / об’єкти іншого.

Умовно мою предметну область можна розділити на такі елементи:

- Товари та категорії
- Залишки
- Операції з товарами (закупівлі, продажі)
- Контрагенти (постачальники)
- Документи обліку руху товарів

Далі наведено опис класів діаграми:

1) MacroGroup – визначає категорію товарів. Використовується для групування на рівні асортименту та аналітики продажів.

2) Product (товар) – основна сутність, яка являє собою одиницю асортименту

3) Inventory (Залишки) – вказує, скільки одинись товару знаходиться на складі

4) Supplier (Постачальник) – описує організацію або особу, яка поставляє товар в магазин

5) Purchases (Закупівлі) – являє собою список замовлень на поставки

6) PurchasePosition (Позиції закупівлі) – зберігає в собі окремі товари, які включені в закупівлю

7) Statuses (Статуси) – визначає поточний стан документа в системі, як от «Збережено» для накладних, які в процесі обробки, «Проведено», для оброблених накладних та «Сторновано» - для накладних які було скасовано.

8) Invoices (Накладні) – документ, що фіксує в собі операцію з товаром або замовленням

9) Salse (Продажі) – фіксує факт продажу товару, простіше кажучи чек

10) SalesItems (Позиції продажу) – містить окремі товари, які входять в чек.

Діаграму класів для моєї системи наведено на Рис. 2.3.

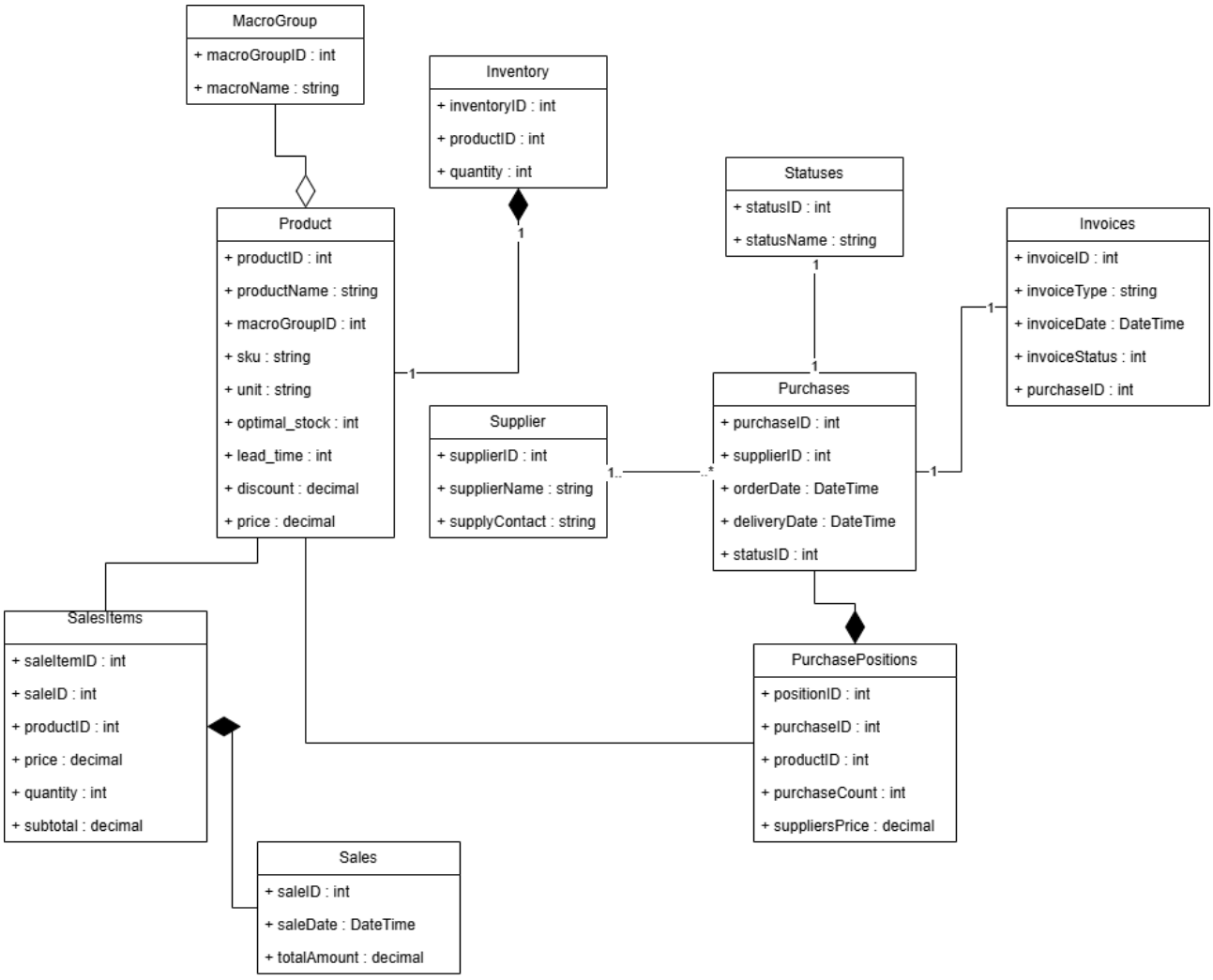


Рис. 2.3. Діаграма класів

2.2.2. Діаграма пакетів

Під час розробки ПЗ важливим завданням є забезпечення логічно впорядкованої структури коду, яка спрощує масштабування, підтримку та розвиток системи. Одним із засобів у об’єктно-орієнтованому моделюванні, що дозволяє зобразити високорівневу структуру системи є діаграма пакетів. Вона демонструє розділ класів та компонентів за функціональними модулями, визначає зв’язок між ними.

У системі для товарного обліку, що розробляється, доцільно виділити наступні пакети:

1) Models – містить сутності предметної області та відображає структуру БД (є необхідним під час використання ORM EntityFrameworkCore)

2) DataAccess – реалізує взаємодію з базою даних. Містить контекст БД (InventoryContext) та конфігурацію сутностей
3) Services – містить бізнес-логіку, зокрема: обробку закупівель, формування звітів, виконання оптимізаційних розрахунків та прогнозування попиту
4) ViewModels – відповідає за логіку представлення даних у намках MVVM. Здійснює зв'язок між інтерфейсом та бізнес-логікою
5) Views – містить файли інтерфейсів користувача, які взаємодіють з ViewModel
6) Reports – в даному пакеті містяться шаблони звітів та класи, що відповідають за їх формування

Для відображення зв'язків на діаграмі даного типу використовується тип відношення залежності, який зображується пунктирною стрілкою. Діаграму пакетів для системи наведено на Рис. 2.4. З переваг використання даної діаграми для аналізу предметної області та моделювання системи можна віднести наступні:

- Така архітектура чітко розділяє відповідальність між рівнями.
- Забезпечує сумісність із шаблоном **MVVM**, що використовується у WPF.
- Полегшує тестування і супровід, оскільки **View** не містить логіки.
- Дозволяє незалежно розвивати бізнес-логіку та інтерфейс.
- Впорядковує роботу з даними через централізований сервісний рівень.

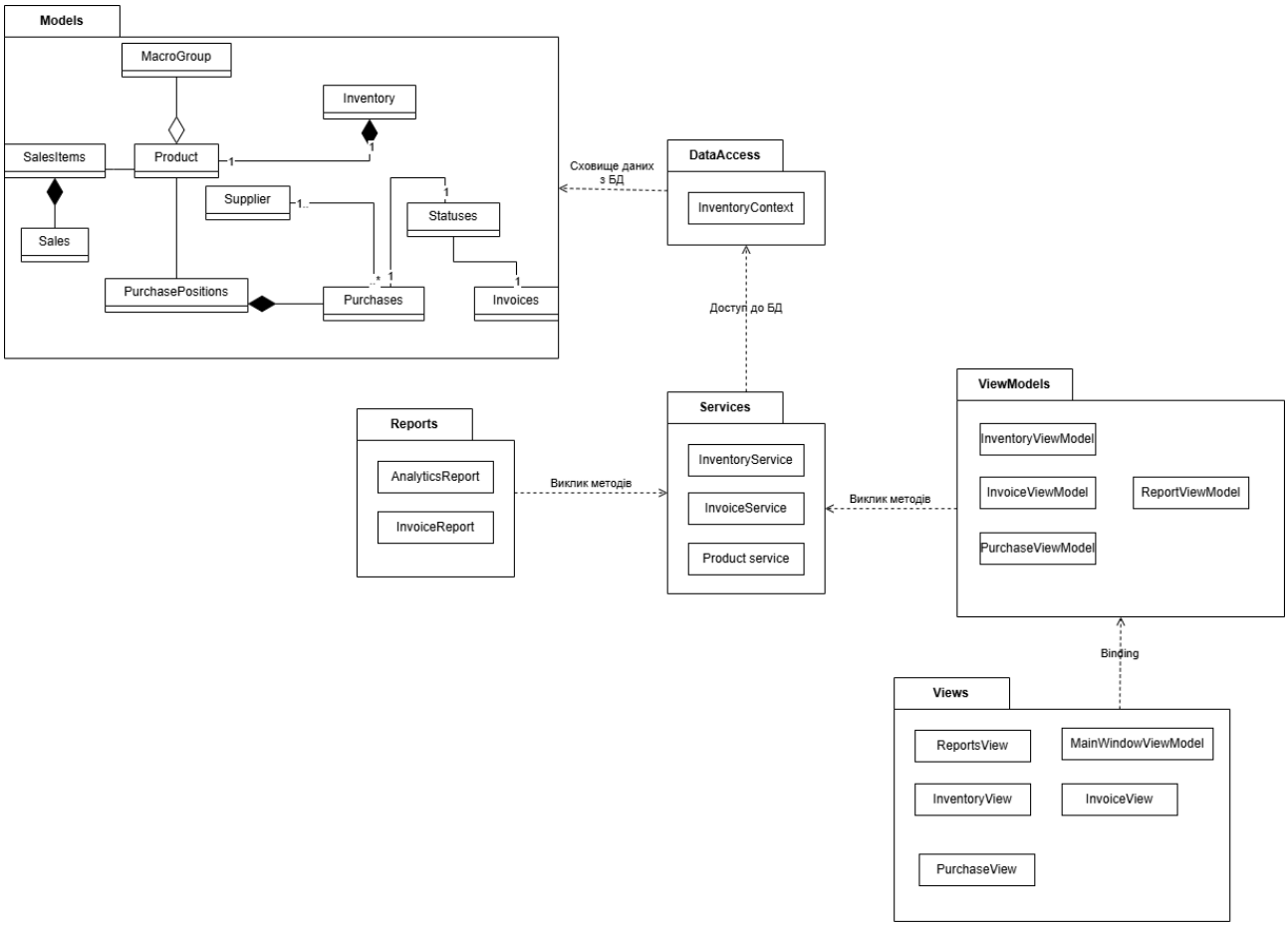


Рис.2.4. Діаграма пакетів

2.2.3. Діаграма об’єктів

Діаграма об’єктів є одним з різновидів статичних діаграм UML, яка використовується для відображення конкретних екземплярів класів у створюваній системі, та зв’язків між ними у визначений момент. На відміну від діаграми класів, що демонструє загальну структуру та відношення між класами, діаграма об’єктів їх деталізує на основі реальних даних, які існують в системи на час виконання функцій/отримання даних з БД. Така діаграма дозволяє зрозуміти, як відбуватиметься взаємодія між сутностями у процесі виконання прикладної задачі.

У контексті системи управління ТЗ доцільно розглянути момент, коли формується замовлення до постачальника та створюється накладна на це заповлення. Це один з ключових етапів роботи системи, оскільки він безпосередньо впливає на стан складських залишків та реалізацію механізмів оптимізації. Роль діаграми у проєктування полягає в наступному:

- наочно визначити стан системи в момент операції постачання;
- підтвердити коректність логічних зв’язків та кардинальностей, визначених у діаграмі класів;
- спрогнозувати зміну даних при проведенні накладної (наприклад, оновлення залишків);
- забезпечити коректне формування бізнес-логіки на рівні сервісів.

Діаграму об’єктів наведено на Рис. 2.5.

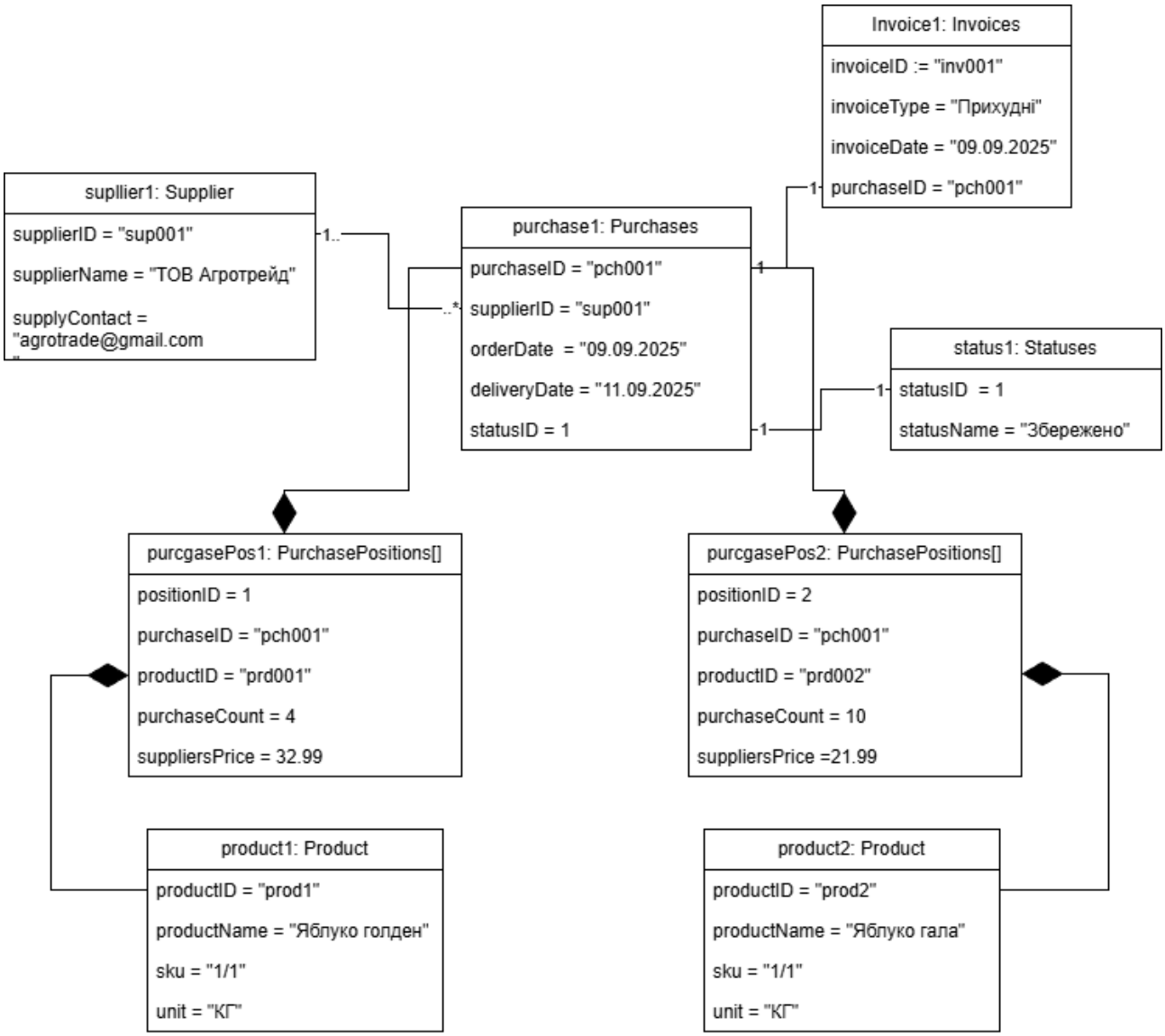


Рис. 2.5. Діаграма об’єктів для процесу оформлення замовлення

3. РОЗРОБКА СИСТЕМИ

3.1. Загальна архітектура системи

В ході розгляду архітектурних шаблонів для системи було виокремлено два, які підходять для таких системи найбільше: MVC та MVVM.

Архітектурна модель MVC (Model–View–Controller) є однією з найпоширеніших парадигм проєктування програмного забезпечення, що забезпечує розділення логіки даних, представлення та управління взаємодією користувача. Вона сформувалась у середині XX століття як універсальний спосіб організації інтерфейсно-орієнтованих систем і надалі стала основою для багатьох сучасних архітектурних підходів.

У класичній структурі MVC:

- Model (Модель) відповідає за доступ до даних, їхню обробку та збереження. Вона містить бізнес-логіку та алгоритми, які визначають поведінку системи. У контексті системи управління товарними запасами модель включає сутності — Product, Inventory, Purchase, Supplier, які відображають таблиці бази даних і реалізують обчислення показників, таких як оптимальний або страховий запас.
- View (Подання) забезпечує візуальне представлення даних користувачеві. У WPF або вебсередовищі це форма, сторінка чи звіт, які відображають інформацію з моделі. Наприклад, діаграми продажів і прогнозів у додатку візуалізують результати аналітичних обчислень із бази даних.
- Controller (Контролер) керує потоком даних між моделлю та поданням, реагує на дії користувача, викликає відповідні методи моделі та оновлює подання. Контролер визначає логіку переходів між сторінками, обробку запитів і загальну поведінку застосунку.

Ключова перевага MVC полягає у розділенні відповідальностей: модифікації інтерфейсу користувача не впливають на бізнес-логіку, а зміни в моделі не порушують структуру подання. Проте зі зростанням складності

інтерфейсів і необхідністю двосторонньої взаємодії (data binding) у десктопних застосунках з’явилась потреба у розширеній архітектурі, яка б спрощувала роботу з поданнями без прямого контролера.

Саме так виникла модель MVVM (Model–View–ViewModel), яка стала логічним розвитком MVC, орієнтованим на застосування у середовищах WPF, UWP та .NET MAUI.
На відміну від MVC, де контролер безпосередньо керує поданням, у MVVM між моделлю та інтерфейсом користувача з’являється проміжна ланка — ViewModel, яка виконує роль абстрактного контролера. Вона реалізує властивості, команди та логіку зв’язування даних, що дозволяє інтерфейсу автоматично реагувати на зміни в моделі.

Розроблене ПЗ для управління товарними запасами з використання оптимізаційних алгоритмів побудоване за багаторівневою архітектурою, яка базується на патерні MVVM (Model-View-ViewModel). Такий підхід забезпечує чітке розділення відповідальностей між рівнями даних, бізнес логіки та інтерфейсу користувача, що суттєво підвищує гнучкість розроблюваної системи, масштабованість та підтримуваність системи.

Система за даним патерном складається з таких рівнів:

1. Рівень даних (Model) - реалізує роботу з базою даних та доступ до сутностей. На цьому рівні визначено класи моделей, а також контекст БД, що використовує EntityFrameworkCore для зв’язку з базою даних Microsoft SQL Server. Такий підхід дозволить працювати з даними на рівні об’єктів без написання SQL записів поміж C# кодом
2. Рівень представлення (View) – реалізований на основі WPF(Windiws Presentation Foundation). Він відповідає за відображення інформації та взаємодію користувача з системою загалом. Головними елементами інтерфейсу в даній системі є наступні: вікно для управління накладними, аналітична панель з діаграмами продажів та прогнозуванням та елементи керування запасами (вікно залишків, вікно замовлень, тощо), перегляду чеків.

3. Рівень бізнес-логіки (ViewModel) – містить основну логіку обробки даних. На даному рівні розташовані сервіси, які виконують бізнес-операції, зокрема управління залишками товарів, формування звітності, проводять аналітику та прогнозування, а також відповідають за обрахунок оптимального замовлення. Даний рівень виступає посередником між View та Model, реалізовуючи реактивну зміну даних через механізми ICommand, INotifyPropertyChanged, тощо.

Взаємодія компонентів реалізована наступним чином:

- View передає команди користувача до ViewModel через механізм Command Binding та відображає дані через Data Binding
- ViewModel викликає методи відповідних Service-класів, які здійснюють операції над даними або аналітичні розрахунки
- Service звертається до Model для виконання CRUD операцій на даними з БД
- Результати обробки повертаються до ViewModel, яка оновлює властивості та відображає їх у View через механізми двостороннього зв’язування даних(раніше згаданий Data Binding).

До переваг обраної архітектури можна віднести те, що вона надає можливості для гнучкого розширення функціоналу, зокрема додавання нових модулів без зміни існуючих, має високу тестованість, так як логіку винесено у ViewModel та сервіси, можливість повторного використання компонентів. Також до плюсів можна віднести інтерактивний та адаптивний інтерфейс, завдяки використанню WPF та Data Binding. На Рис. 3.1. наведено архітектуру у вигляді діаграми.

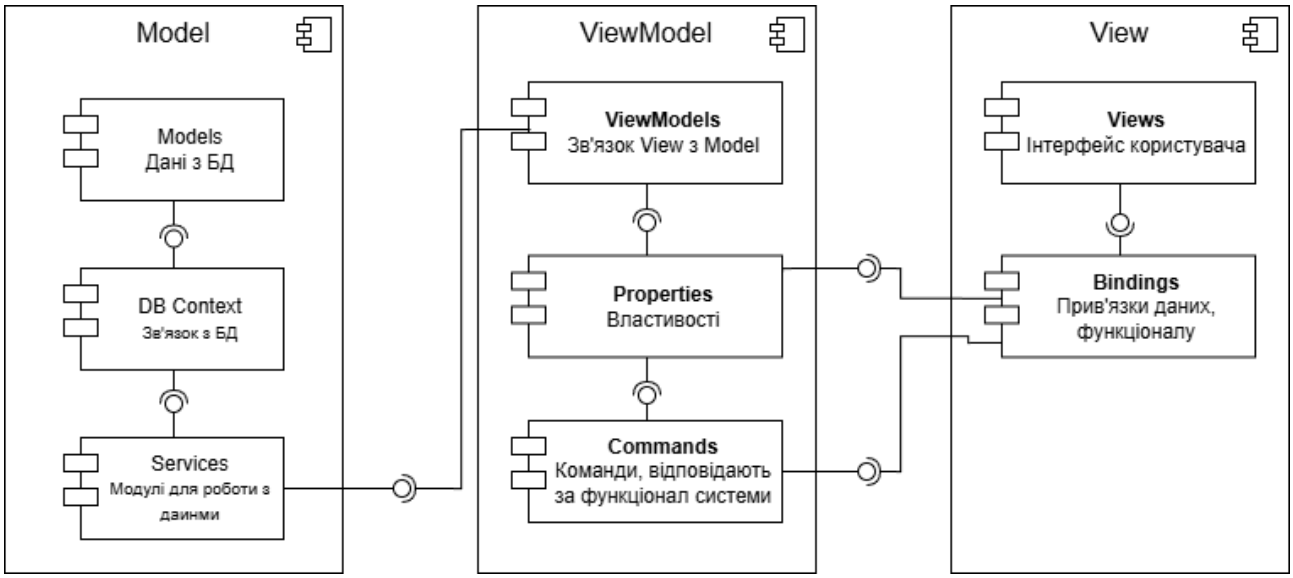


Рис. 3.1. Діаграма архітектурного шаблону MVVM

3.2. Архітектура підсистеми управління ТЗ

Підсистема управління ТЗ є центральним елементом системи, оскільки саме вона забезпечує контроль та оптимізація залишків продукції на складах. Її головна мета – забезпечити баланс між наявними запасами та реальними потребами, мінімізуючи витрати підприємства, що пов’язані з дефіцитом/надлишками.

Загальна структура підсистеми побудована на основі раніше згаданої багаторівневої архітектури з використанням патерну MVVM. Основні складові підсистеми:

1. **Model:**
   - Класи даних: Product, Supplier, Invoice, PurchasePosition, MacroGroup.
   - Клас контексту бази даних: InventoryContext, що забезпечує зв’язок із таблицями БД через Entity Framework Core.
   - Усі сутності описані з використанням атрибутів, що дозволяє автоматично формувати зв’язки один-до-багатьох (наприклад, один постачальник — кілька накладних).
2. **ViewModel/Services:**

- ViewModel: InvoiceViewModel та InventoryViewModel — координують дії користувача, оновлюють відображення даних, обробляють команди (ProcessInvoiceCommand, CancelInvoiceCommand, GenerateInvoiceReportCommand тощо).
- Сервіси: InvoiceService, який відповідає за отримання накладних, зміну їх статусів, тощо. InventoryService – ведення обліку залишків, оновлення кількості товарів після проведення накладної, фіксація змін у запасах. ReportService – сервіс, який відповідає за аналітику та звітність у системі.

3. **View:**
    - Даний рівень підсистеми включає в себе всі вікна та вкладки, які пов’язані з товарами, зокрема InvoicesView, InventoryView, SalesView, тощо.
    - За відображення інформації всередині View відповідають різні елементи з інструментарію WPF, зокрема ListBox, DataGrid, тощо. Також для відображення звітності використано розширення WebView2, який дозволяє відкривати PDF файли всередині себе (дозволяє відкривати, зберігати RDLC звіти).

Загалом підсистема реалізує такі функції:

- Завантаження накладних з бази даних та відображення їх статусів у реальному часі (збережена, проведена, сторнована).
- Розрахунок сумарної вартості кожної накладної (`TotalAmount`), який автоматично оновлюється при зміні позицій.
- Обробка накладних:
    - *Проведення* — додає кількість товарів на склад;
    - *Сторнування* — зменшує залишки;
    - *Повернення в роботу* — відновлює накладну для редагування.
- Генерація звітів про поставки у форматі PDF безпосередньо після вибору накладної.

- Інтеграція з прогнозною підсистемою для аналізу закупівельних тенденцій.
- Перегляд залишків товарів
- Проведення замовлень

Логіка роботи підсистеми реалізована за допомогою асинхронного програмування, що допомагає прискорити роботу ПЗ та проведення обчислень ра рахунок розбиття виконання функцій на потоки. Робота підсистеми на основі функціоналу для роботи з накладними наступна:

- Користувач у вікні вибирає накладну зі списку.
- Властивість SelectedInvoice у InvoiceViewModel оновлюється, що тригерить:
  - завантаження товарів цієї накладної (LoadInvoiceItemsAsync()),
  - оновлення підсумкової суми (TotalAmount),
  - автоматичну генерацію звіту (GenerateInvoiceReportAsync()).
- При натисканні кнопки *«Провести накладну»* викликається команда ProcessInvoiceCommand, яка:
  - змінює статус накладної в БД на *проведено*;
  - додає кількість товарів у таблицю залишків через InventoryService.AddStockAsync();
  - оновлює дані інтерфейсу.
    - Аналогічно, при сторнуванні або поверненні в роботу відбувається зворотна операція.

Для оновлення залишків реалізовано наступний алгоритм: спочатку відбувається отримання списку всіх товарних позицій для обраної накладної. Далі для кожного товару проводиться перевірка наявності у БД за його ProductId. Якщо товар існує, залишок збільшується або зменшується, відповідно до потреби в даний момент. Вже після виконання цих процесів виконується збереження даних в БД за допомогою виклику InventoryContext.SaveChangesAsync(). На Рис. 3.2. наведено приклад реалізації тексту методу для оновлення залишків.

```
2 references
public async Task AddStockAsync(int productId, int delta)
{
    var inventory = await _context.Inventories.FirstOrDefaultAsync(i => i.ProductId == productId);
    if (inventory == null)
    {
        _context.Inventories.Add(new Inventory { ProductId = productId,
            Quantity = Math.Max(0, delta),
            LastUpdate = DateOnly.FromDateTime(DateTime.Now) });
    }
    else
    {
        inventory.Quantity += delta;
        inventory.LastUpdate = DateOnly.FromDateTime(DateTime.Now);
    }
    await _context.SaveChangesAsync();
}
```

Рис. 3.2. Метод для оновлення залишків

3.3. Підсистема звітності

Підсистема звітності в системі управління товарними запасами є невід'ємною частиною аналітичного модуля, яка забезпечує формування, відображення та експорт результатів обчислень у візуальній формі. Основна її мета — **перетворити масиви транзакційних даних** (закупівлі, продажі, залишки) на інформативні аналітичні звіти, що дозволяють користувачеві приймати оптимальні управлінські рішення.

Архітектура звітного модуля побудована за принципом багаторівневого розділення обов'язків, де кожен компонент виконує конкретну функцію:

1. **Рівень даних (Model)**

Цей рівень взаємодіє з базою даних через контекст InventoryContext (Entity Framework Core).

Основними сутностями є:

- Sale, SalesItem — містять інформацію про реалізацію товарів;
- Purchase, PurchasePosition — фіксують обсяги та постачальників закупівель;
- Product, Inventory — описують наявні товари та їх кількість на складах;
- Status, Invoice, Supplier — допоміжні таблиці для класифікації транзакцій.

Дані агрегуються через LINQ-запити з групуванням і фільтрацією, що дозволяє отримати узагальнені показники для звітів — наприклад, продажі за місяцями, прогноз попиту, ефективність закупівель або динаміку запасів.

2. **Рівень бізнес-логіки (Service)**

Цей рівень реалізований через окремі сервіси, такі як:

- SalesAnalyticsService — розрахунок динаміки продажів і прогнозу на основі ковзного середнього або лінійної регресії;
- ReportService — підготовка джерел даних для RDLC-звітів;
- OptimizationEffectService — аналітика ефективності оптимізації запасів після застосування алгоритмів розрахунку закупівель.

На цьому рівні реалізовано математичні алгоритми обробки даних:

- ковзне середнє (для прогнозування обсягів продажів і замовлень);
- розрахунок відсоткового покращення дефіциту/надлишку;
- регресійні моделі для визначення трендів.

Таким чином, бізнес-рівень підсистеми звітності відповідає за аналітичну підготовку інформації перед її передачею у візуальний рівень.

3. **Рівень подання (View)**

Візуальна частина системи реалізована двома підходами:

- RDLC-звіти — для детальних документів (накладні, рахунки, звіти про продажі). Вони генеруються за допомогою ReportViewer, який взаємодіє з локальними наборами даних.
- OxyPlot-графіки — для інтерактивного аналізу показників.

Наприклад:

- діаграма продажів за місяцями та прогнозом;
- порівняння ефективності алгоритмів (лінійна оптимізація, ковзне середнє);
- візуалізація динаміки залишків і дефіциту товарів.

Кожен графік формується у ViewModel через зв’язування властивостей типу PlotModel, які автоматично оновлюються після виконання асинхронних

методів отримання даних. Вигляд звіту представлено на Рис.3.3., та діаграми Oxyplot На Рис.3.4.

Накладна № 1 Дата створення 22.10.2025 0:00:00

Постачальник ТОВ "FreshME"

Артикул	Назва	Вартість(од)	Кількість	Сума
8/1	Сік апельсиновий 1л	71,25	10	712,50
8/2	Сік яблучний 1л	67,50	11	742,50
8/3	Сік апельсиновий 0,5л	45,00	5	225,00
8/4	Сік яблучний 0,5л	41,25	2	82,50
8/5	Смузі яблучно-моркв'яний 0,25л	31,50	5	157,50
8/6	Смузі зі шпинатом 0,25л	32,25	5	161,25
8/7	Смузі з селерою 0,25л	33,75	5	168,75
8/8	Смузі бананово-полуничний 0,25л	33,00	8	264,00
9/1	Хумус класичний	46,50	8	372,00
9/2	Хумус з чілі	48,75	8	390,00
9/3	Хумус з томатом та базиліком	48,00	8	384,00
9/4	Хумус з прованськими травами	50,25	8	402,00

Сума накладної: 4062,00

Представник постачальника _____ ПІБ Підпис _____

Прийняв поставку _____ ПІБ Підпис _____

Рис. 3.3. Звіт накладної для друку

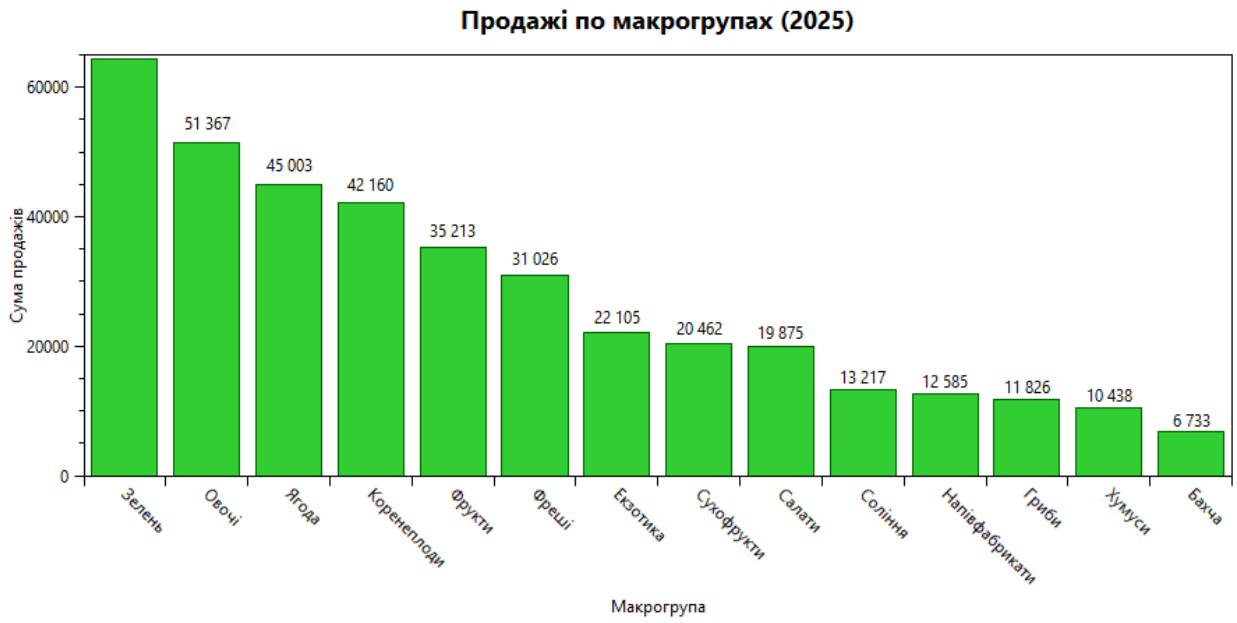


Рис. 3.4. Діаграма продажів по макрогрупах

4. Рівень ViewModel (MVVM зв’язок)

ViewModel виступає посередником між логікою звітів та інтерфейсом користувача.

Основні моделі:

- ReportViewModel — відповідає за формування звітів у ReportViewer;
- OptimizationEffectViewModel — генерує аналітичні діаграми ефективності алгоритмів;
- SalesViewModel — готує дані для графічних звітів продажів.

Завдяки механізму data binding, графіки й таблиці автоматично оновлюються після зміни даних у моделях, без прямого втручання користувача

При формуванні звіту користувач активує команду (для накладної це її відкриття, для діаграм це відкриття вкладки аналітики), яка ініціює виклик сервісу. Той у свою чергу обчислює необхідні показники, створює узагальнений список результатів, який ViewModel перетворює на джерело для PlotModel або RDLC-документа.

Таке багаторівневе розділення забезпечує масштабованість системи — у разі додавання нового типу звіту або алгоритму потрібно лише розширити відповідний сервіс і ViewModel, не змінюючи інші компоненти.

3.4. Архітектура підсистеми аналітики та прогнозування

Підсистема аналітики та прогнозування є інтелектуальним ядром розроблюваного ПЗ. Вона дозволяє аналізувати не лише історичні дані по продажах, а й передбачати майбутні обсяги реалізації продукції, що є основою для прийняття рішень щодо оптимізації запасів. Дана підсистема інтегрована з основним модулем управління запасами та забезпечує аналітичну підтримку менеджера або логіста при плануванні закупівель.

Загальна структура підсистеми:

- **Сервіс аналітики продажів (**SalesAnalyticsService**)** — відповідає за отримання даних із бази, їх обробку, групування за місяцями або макрогрупами та формування прогнозних значень.
- **ViewModel-аналітики (**SalesAnalyticsViewModel**)** — виконує логіку зв'язування даних між сервісом і графічним відображенням, формує колекції для побудови діаграм, обчислює тренди та прогнозні показники.
- **Графічне відображення (**DashboardView.xaml**)** — реалізує інтерактивну діаграму, побудовану на основі бібліотеки **OxyPlot**, яка відображає історичні продажі та прогноз на кілька місяців уперед.

Алгоритм роботи підсистеми наступний:

1. Завантаження даних: дані про продажі обираються з таблиці Sales, де зберігаються дати, суми та ідентифікатори товарів. Запит у SalesAnalyticsService групується за місяцями. Реалізацію методу відбору даних зображено на Рис. 3.5.

```
var rawData = await _context.Sales
    .GroupBy(s => new { s.SaleDate.Year, s.SaleDate.Month })
    .Select(g => new
    {
        Year = g.Key.Year,
        Month = g.Key.Month,
        Total = g.Sum(s => s.TotalAmount)
    })
    .OrderBy(x => x.Year).ThenBy(x => x.Month)
    .ToListAsync();
```

Рис. 3.5. Реалізація відбору «сирих» даних про продажі

2. Формування ряду даних: з отриманих в попередньому пункті даних формується послідовність типу (місяць, сума) для відображення на графіку
3. Обчислення прогнозу: на відміну від ковзного середнього, використовується лінійна регресія, яка дозволяє виявити тенденції росту/спаду продажів. Даний алгоритм було реалізовано за допомогою MathNet.Numerics. Для стабілізації прогнозу система також перевіряє, чи різниця між прогнозом і середнім значенням останніх трьох місяців не перевищувала 50%, що дозволяє уникнути різких стрибків або помилкових значень при сезонних коливаннях. В ДОДАТОК А наведено повну реалізацію алгоритму з коментарями.
4. Додавання прогнозних точок: до масиву даних для діаграми додається 3 прогнозні значення для наступних місяців, які позначаються на діаграмі іншим кольором та пунктирною лінією. Для цього в структуру даних для діаграми додано булеве поле IsForecast, де true-прогнозоване значення, а false – значення яке вже існує в БД.
5. Відображення результатів на діаграмі: за допомогою методів з бібліотеки OxyPlot будується комбінована діаграма(Рис. 3.6.), де: зелена лінія – фактичні значення, пунктирна червона – прогноз.

Рис. 3.6. Прогноз продажів до кінця року

Для візуалізації даних система будує на сторінці OxyPlot-графік типу LineSeris, на якому відображено: фактичні продажі, прогноз та координатні осі, зокрема підписи місяців і легенду. Текст програми для побудови діаграми наведено на Рис.3.7.

```
private async Task LoadChartAsync(){
    var data = await _analyticsService.GetSalesWithForecastAsync();
    var model = new PlotModel { Title = "Продажі за місяцями + прогноз" };
    model.Axes.Add(new DateTimeAxis
    {
        Position = AxisPosition.Bottom,
        StringFormat = "MM.yyyy",
        Title = "Місяць",
        IntervalType = DateTimeIntervalType.Months,
        MinorIntervalType = DateTimeIntervalType.Days
    });
    model.Axes.Add(new LinearAxis
    {
        Position = AxisPosition.Left,
        Title = "Сума продажів"
    });
    var actualLine = new LineSeries
    {
        Title = "Фактичні продажі",
        MarkerType = MarkerType.Circle,
        Color = OxyColors.Green
    };

    var forecastLine = new LineSeries
    {
        Title = "Прогноз",
        MarkerType = MarkerType.Diamond,
        Color = OxyColors.Red,
        LineStyle = LineStyle.Dash
    };

    foreach (var d in data)
    {
        var date = DateTime.Parse($"01.{d.Month}");
        actualLine.Points.Add(new DataPoint(DateTimeAxis.ToDouble(date), (double)d.Total));
        if (d.IsForecast)
            forecastLine.Points.Add(new DataPoint(DateTimeAxis.ToDouble(date), (double)d.Total));
    }
    model.Series.Add(actualLine);
    model.Series.Add(forecastLine);
    SalesChart = model;
    SalesChart = model;
    OnPropertyChanged(nameof(SalesChart));
```

Рис. 3.7. Текст методу для побудови графіка.

Перевагами архітектури підсистеми є те, що вона забезпечує динамічну побудову прогнозів на основі історичних даних, використаннч асинхронної обробки даних для забезпечення високої продуктивності, а сам алгоритм прогнозування піддається легкому розширенню, або ж взагалі заміні

(наприклад на ARIMA чи експоненційне згладжування) Також до переваг можна віднести використання Oxyplot, який є одним з найпростіших у експлуатації.

3.5. Алгоритм оптимізації запасів

Підсистема оптимізації товарних запасів є ключовою для розроблюваної системи, оскільки саме вона забезпечує виконання мети розробки системи – мінімізацію надлишкових запасів та підтримки оптимального рівня товарів на складі. Цей модуль тісно інтегрований з підсистемами аналітики та управління запасами ,що дозволяє йому приймати рішення на основі прогнозів продажів, історії постачань та обсягів продажів.

Загальна ідея оптимізації наступна: знайти баланс між витратами на зберігання надлишкових товарів і ризиком дефіциту продукції. Метою є визначення такого рівня запасів, який:

- Забезпечує безперебійний процес продажу
- Мінімізує сумарні витрати на втрати по товару
- Враховує прогнозований попит у майбутніх періодах

Загалом в системі реалізовано підхід, який включає:

- Статистичний аналіз попиту на основі історичних даних
- Розрахунок оптимального залишку
- Використання прогнозних значень для адаптації під сезонні коливання

Одним із ключових компонентів підсистеми оптимізації є метод UpdateOptimalStocksAsync**()**, який реалізує автоматичне оновлення оптимальних рівнів запасів у базі даних на основі статистичних параметрів надійності постачань і коливань попиту.
Цей метод забезпечує постійну актуальність значень поля OptimalStock у таблиці Products, що використовується при формуванні звітів, прогнозів і плануванні нових закупівель (Рис 3.8). Метою даного алгоритму є динамічне коригування оптимального рівня запасів з урахуванням факторів ризику,

зокрема варіативність попиту (стандартне відхилення) та часу постачання. Завдяки цьому система тримає баланс між надлишковими запасами та дефіцитом продукції, забезпечуючи стабільний товарний потік.

```
1 reference
public async Task UpdateOptimalStocksAsync()
{
    var products = await _context.Products.Include(p => p.Inventories).ToListAsync();

    const decimal Z = 1.65m;   // коефіцієнт надійності
    const decimal Sigma = 0.2m; // стандартне відхилення (20%)
    const decimal LeadTime = 3m; // середній час постачання (днів)

    foreach (var p in products)
    {
        var safety = p.SafetyStock;
        if (safety <= 0) continue;

        var delta = Z * Sigma * (decimal)Math.Sqrt((double)LeadTime);
        var optimal = safety + delta * safety;

        p.OptimalStock = (int?)Math.Round((decimal)optimal, 2);
    }

    await _context.SaveChangesAsync();
}
```

Рис. 3.8. Текс методу для обрахунку оптимального залишку

Принцип роботи алгоритму наступний:

1. Отримання даних – система отримує дані з БД
2. Ініціалізація параметрів розрахунку, де Z = 1,65 (коефіцієнт надійності, що відповідає 95% рівню сервісу) , Sigma = 0.2 (стандартне відхилення, що характеризує середній рівень варіації попиту (20%)) та LeadTime = 3 – середній час постачання продукції
3. Обхід усіх товарів та перевірка даних – для кожного товару перевіряється поточний оптимальний залишок, який за потреби змінюється
4. Розрахунок корекційного коефіцієнта – для кожного товару обчислюється поправка, що враховує коливання попиту та час постачання. Обрахунок відбувається за наступною формулою:

$$\Delta = Z \times \sigma \times \sqrt{LT}$$

Де: Z – коефіцієнт надійності, σ – стандартне відхилення, а LT – середній час постачання.

5. Обчислення оптимального рівня запасів відбувається за наступною формулою:

$$Q^* = S + \Delta \times S$$

Де S – поточний оптимальний залишок.

6. Оновлення БД: результат округлюється і записується у відповідне поле OptimalStock

Особливостями реалізації даного алгоритму є асинхронність(метод виконується асинхронно), інкапсуляція логіки (усі обчислення реалізовані в межах одного методу, що спрощує повторне використання) та гнучкість параметрів (коефіцієнти можна адаптувати для різних категорій продукції).

Алгоритм оптимізації кількості замовлення реалізовано в методі RecalculateRecomendedQuantities() (Рис. 3.9.). Він виконує динамічний розрахунок рекомендованих обсягів закупівель для кожного товару замовлення, враховуючи його поточний залишок, оптимальний залишок та обраний метод оптимізації. Цей алгоритм є важливою складовою модуля планування закупівель і забезпечує автоматичне оновлення поля PurchaseCount у списку позицій замовлення.

```
199     private void RecalculateRecommendedQuantities()
200     {
201         if (PurchasePositions == null || PurchasePositions.Count == 0)
202             return;
203
204         foreach (var pos in PurchasePositions)
205         {
206
207             // Поточний залишок
208             var currentStock = pos.Product?.Inventories?.Sum(i => i.Quantity) ?? 0;
209             var safetyStock = pos.Product?.SafetyStock ?? 0;
210             var optimalStock = pos.Product?.OptimalStock ?? 0;
211
212             decimal recommended = 0;
213
214             if (IsLinearOptimization)
215             {
216                 // Лінійна оптимізація: орієнтація на дефіцит запасу
217                 recommended = Math.Max(0, optimalStock - currentStock);
218             }
219             else if (IsMovingAverage)
220             {
221                 // Прогноз за ковзним середнім (умовно)
222                 var randomFluctuation = new Random().NextDouble() * 0.1; // невелике відхилення
223                 recommended = Math.Max(0, (optimalStock - currentStock) * (decimal)(1 + randomFluctuation));
224             }
225
226             pos.PurchaseCount = (int)Math.Round(recommended, 0);
227         }
```

Рис. 3.9. Метод обрахунку оптимального замовлення

Метою даного методу є розрахунок кількості одиниць товару, яку необхідно дозамовити, щоб забезпечити оптимальний рівень запасів відповідно до поточного стану складу. Алгоритм ураховує два різні підходи до оптимізації:

- Лінійну оптимізацію — орієнтовану на компенсацію дефіциту запасу.
- Метод ковзного середнього — що включає елемент випадкового відхилення для моделювання сезонних або непередбачуваних змін попиту.

Алгоритм працює за наступною послідовністю:

1. Перевірка наявності даних – алгоритм починає виконання лише в тому випадку, якщо PurchasePositions має в собі дані. Якщо колекція порожня, метод припиняє роботу
2. Обхід усіх позицій закупівлі – відбувається обхід кожної позиції замовлення. Для кожного елемента визначається поточний залишок, та оптимальний залишок.
3. Розрахунок рекомендованої кількості замовлення – в залежності від того, який метод обрахунку обере користувач, запускається один з алгоритмів: перший це лінійна оптимізація: якщо поточний запас менший за оптимальний, система визначає різницю та пропонує дозамовити товар до оптимального значення. Другий алгоритм – це метод ковзного середнього: додається невелике випадкове відхилення (до ±10%), що моделює вплив непередбачуваних факторів: коливань попиту, змін сезонності або поведінки клієнтів.
4. Наступний крок – округлення результатів за запис їх у відповідні поля.

Приклади роботи алгоритмів наведено на Рис. 3.10-3.11.

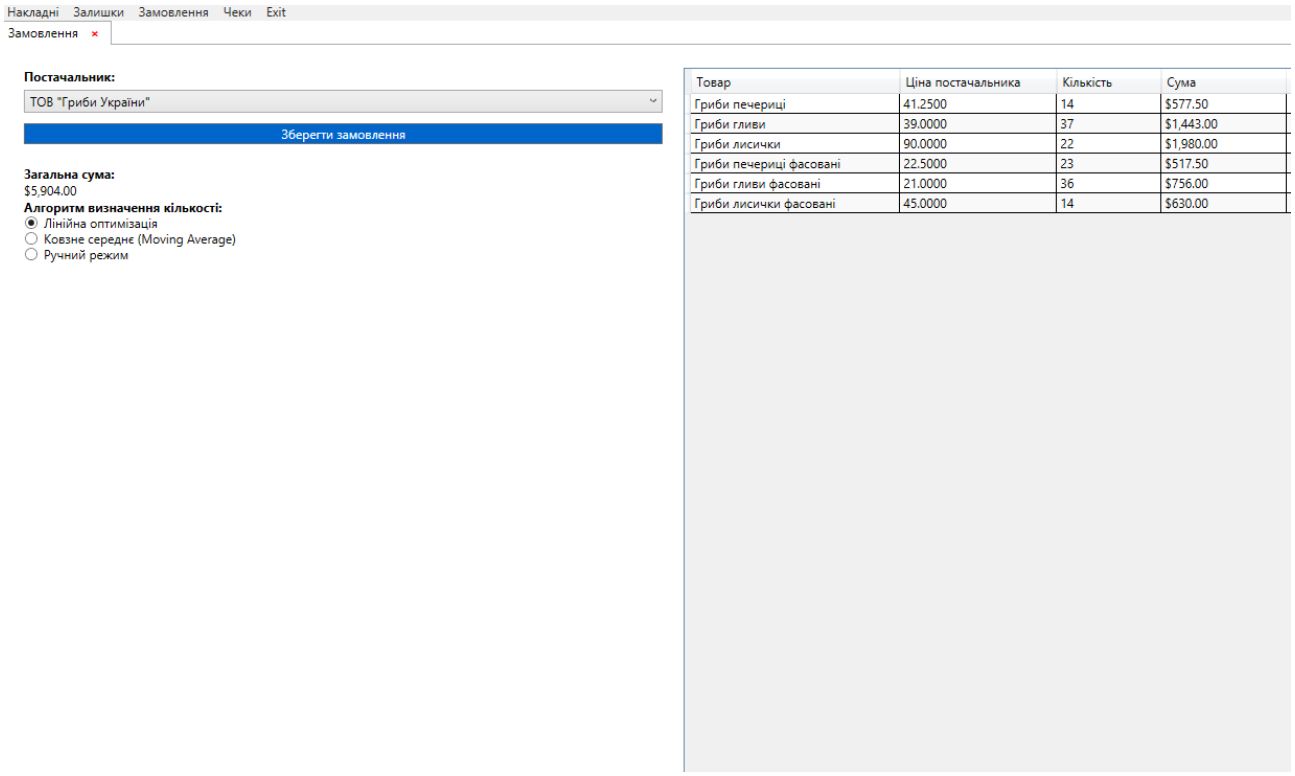


Рис. 3.10. Приклад роботи алгоритму лінійної оптимізації

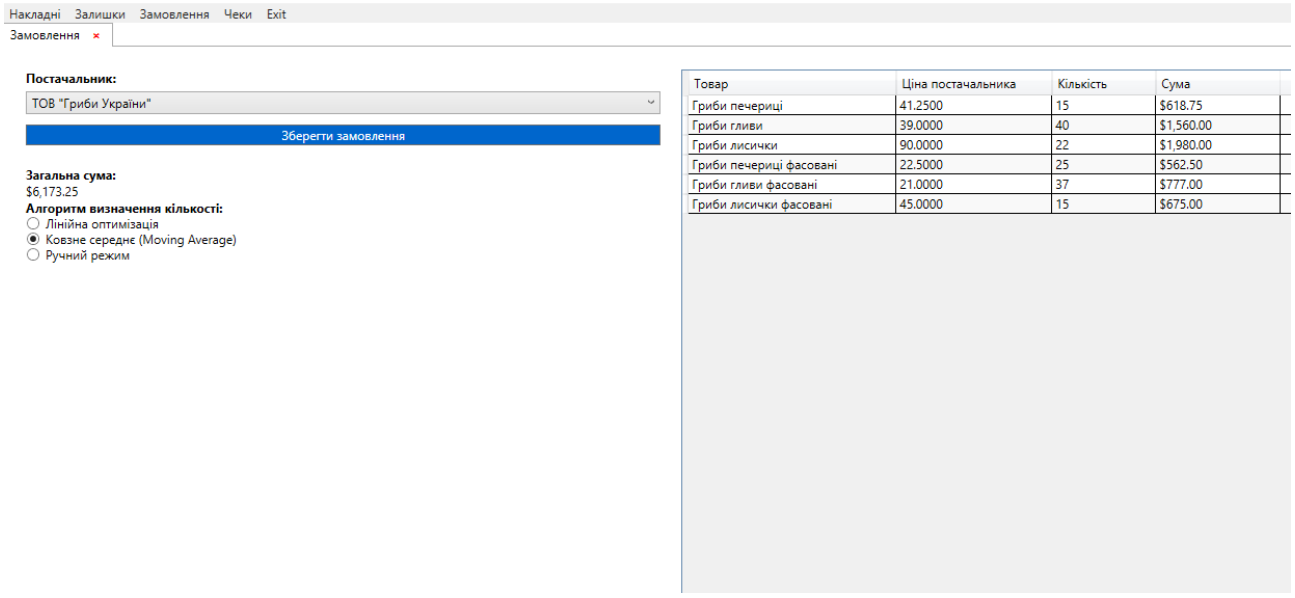


Рис. 3.11. Приклад роботи алгоритму ковзного середнього

Завдяки цьому методу користувач може в реальному часі отримати оновлені рекомендації для замовлень у відповідності до фактичного стану складу. У комплексі з методами розрахунку оптимальних запасів (див. рисунок 3.6), цей підхід утворює замкнутий цикл управління запасами, що охоплює: аналіз → прогноз → розрахунок → рекомендацію → звіт.

3.6. Структура бази даних

База даних системи управління запасами була побудована за принципами реляційної моделі та реалізована в середовищі Microsoft SQL Server із використанням ORM-технології Entity Framework Core. Її структура орієнтована на забезпечення цілісності даних та створена за правилами 3 Нормальної форми.

Структура бази даних наступна: вона містить 9 таблиць, зокрема: Product (містить «картки» товарів – загальні відомості про товар), Supplier (дані про постачальників, зокрема назва, контакти, тощо), MacroGroup (дані про макрогрупи товарів), Inventories (в даній таблиці знаходяться відомості про залишки кожного товару та дату останнього надходження), за аналогічним до таблиць Product та Inventories побудовано таблиці PurchaseInvoice та PurchasePosition / Sale та SalesPositions, одна таблиця – загальні відомості, інша – перелік позицій які входять в чек/закупівлю. Під час проєктування було побудовано ER-діаграму для БД, яку зображено на Рис. 3.12.

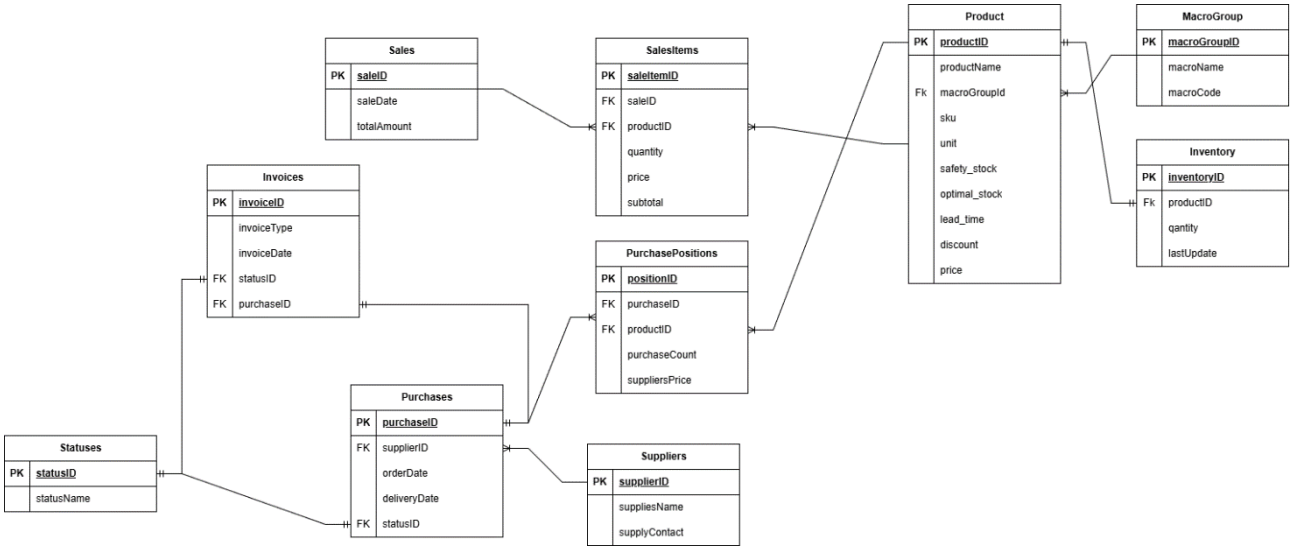


Рис. 3.12. ER-діаграма БД

Реалізація через EntityFrameworkCore забезпечує спрощення взаємодії ПЗ та БД та дає доступ до використання LinQ, функціоналу для запитів, який вбудовано в C#. Використання даного ORM також забезпечує автоматичне

створення сутностей через міграції, двостороннє відображення сутностей у вигляді об’єктів C#, гнучке керування запитами без прямого SQL-коду та інтеграцію з асинхронними методами. На Рис. 3.13. зображено мігровану з БД сутність у вигляді C# класу.

```
using System;
using System.Collections.Generic;

namespace InventoryOptimization.Models;

11 references
public partial class Supplier
{
    5 references
    public int SupplierId { get; set; }

    4 references
    public string SuppliesName { get; set; } = null!;

    1 reference
    public string? SupplyContact { get; set; }

    1 reference
    public virtual ICollection<Purchase> Purchases { get; set; } = new List<Purchase>();
    1 reference
    public ICollection<Product> Products { get; set; } = new List<Product>();
}
```

Рис. 3.13. Структура класу сутності БД

Також створена БД містить статичні дані, які вносяться відразу після її створення та створення таблиць. Саме в моєму варіанті, статичними даними є дані про макрогрупи та статуси накладних. Приклад внесення статичних даних зображено на Рис. 3.14.

```
delete from MacroGroup
INSERT INTO MacroGroup (macroName, macroCode)
VALUES
    (N'Коренеплоди', N'1'),
    (N'Овочі', N'2'),
    (N'Фрукти', N'3'),
    (N'Екзотика', N'4'),
    (N'Гриби', N'5'),
    (N'Зелень', N'6'),
    (N'Салати', N'7'),
    (N'Фреші', N'9'),
    (N'Хумуси', N'09'),
    (N'Ягода', N'10'),
    (N'Бахча', N'11'),
    (N'Соління', N'12'),
    (N'Напівфабрикати', N'13'),
    (N'Сухофрукти', N'14');
```

Рис. 3.14. Внесення статичних даних в БД

Також для взаємодії з БД через EntityFrameworkCore потрібен клас контексту, який показано на Рис. 3.15.

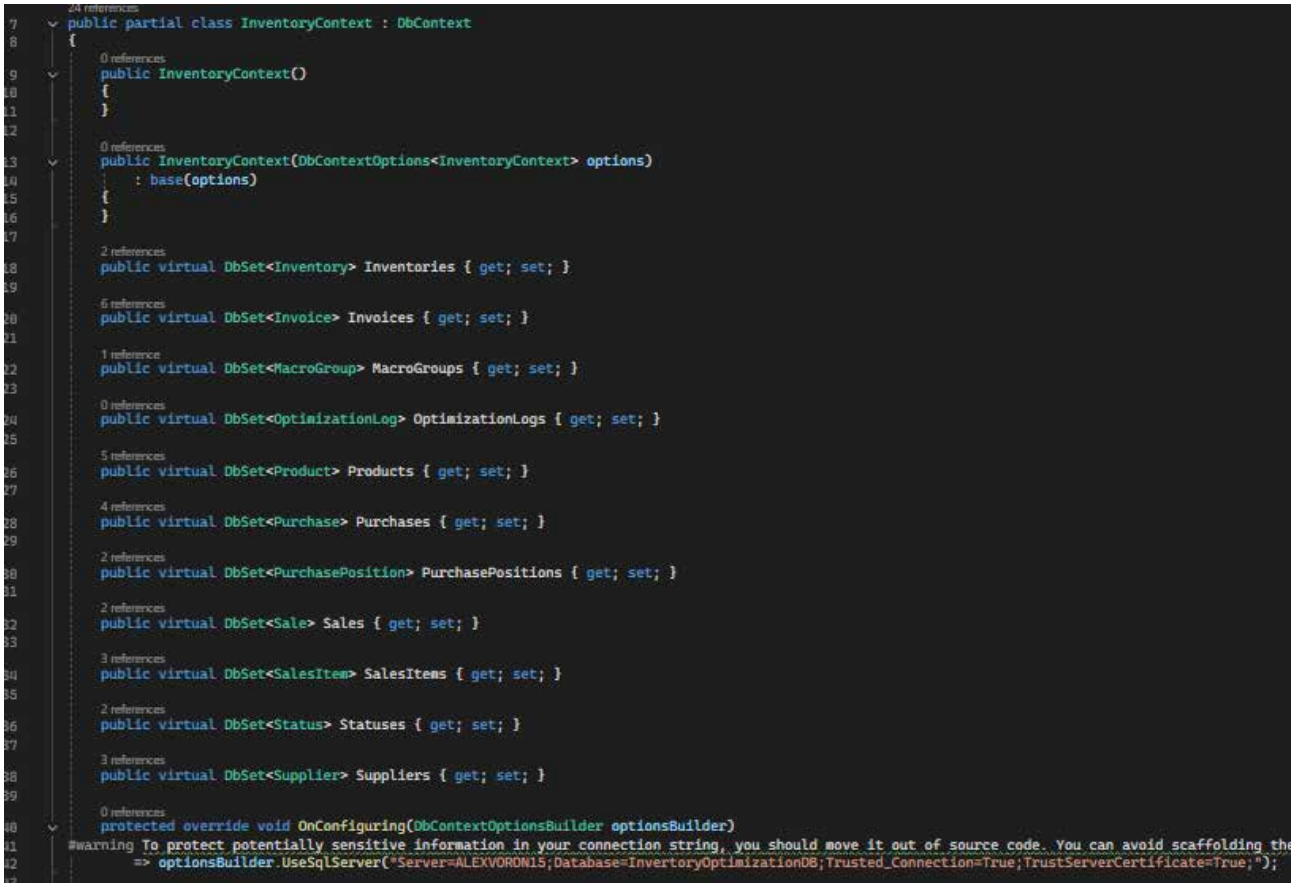


Рис. 3.15. Клас контексту

Даний клас відповідає за розмітку структури БД всередині середовища C#, також він містить строку підключення до БД та методи для взаємодії з нею.

3.7. Технологічне забезпечення

Технологічне забезпечення розробленої системи управління ТЗ визначає набір інструментів, технологій та платформ, за допомогою яких було реалізовано всі програмні, аналітичні та візуалізаційні компоненти проєкту. Вибір технологічного стеку ґрунтується на вимогах до надійності, продуктивності, гнучкості та масштабованості системи, а також на зручності інтеграції з сучасними засобами аналітики та звітності.

1. Середовище розробки

Основне середовище розробки — Microsoft Visual Studio 2022 Community Edition**,** яке забезпечує повноцінну підтримку мов C# і XAML, інтеграцію з системою контролю версій Git, а також вбудовані інструменти

налагодження, профілювання і тестування. Для розробки графічного інтерфейсу застосовано WPF (Windows Presentation Foundation) — сучасну платформу від Microsoft для створення настільних застосунків із підтримкою шаблонів, прив’язки даних і гнучкої кастомізації інтерфейсу.

2. Технологічна структура ПЗ

Система побудована на базі архітектурного патерну MVVM (Model–View–ViewModel), який забезпечує чітке розділення відповідальностей:

- **Model** — містить сутності доменної області, що відображають таблиці бази даних (наприклад, Product, Supplier, Sale, Inventory);
- **ViewModel** — реалізує бізнес-логіку, асинхронну взаємодію з сервісами та оновлення даних інтерфейсу користувача;
- **View** — реалізована у XAML із використанням механізму DataBinding для зв’язку з ViewModel без безпосереднього кодування в Code-Behind.

Такий підхід забезпечує високу тестованість, масштабованість та зручність підтримки програмного коду.

3. Система керування даними

Для збереження даних використано Microsoft SQL Server 2022 Express, який забезпечує:

- підтримку реляційної моделі з транзакційною цілісністю;
- використання запитів SQL та T-SQL процедур;
- інтеграцію з ORM-фреймворком Entity Framework Core 8.0.

EF Core дозволяє працювати з даними через об’єктну модель без написання SQL-запитів, забезпечує механізм міграцій (автоматичного оновлення схеми бази) та асинхронні запити для оптимальної продуктивності при великих обсягах інформації.

4. Засоби формування звітів

Для створення звітів і друкованих документів застосовано Microsoft RDLC (Report Definition Language Client-Side) у поєднанні з бібліотекою BoldReports**,** що підтримує локальне рендерення звітів у форматах PDF, Excel та HTML. Звіти формуються автоматично на основі DataSet, що передається з ViewModel у сервіс ReportService, де генерується PDF-документ із накладними, продажами або аналітичними даними.
В інтерфейсі користувача результати відображаються через компонент WebView2**,** що дозволяє вбудовано переглядати сформований звіт без зовнішніх засобів.

5. Інструменти візуалізації аналітики

Для побудови інтерактивних діаграм використано бібліотеку OxyPlot, яка інтегрується з WPF і підтримує широкий спектр візуальних елементів (лінійні, стовпчикові, кругові діаграми, тренди, гістограми тощо).

У межах системи реалізовано:

- графіки продажів за місяцями із трендом прогнозу;
- аналітику продажів по макрогрупах;
- динаміку змін запасів у реальному часі.

OxyPlot обрано завдяки простоті використання, відкритому вихідному коду та можливості налаштування стилів у межах MVVM-підходу.

6. Додаткові бібліотеки

У процесі розробки використано низку допоміжних бібліотек:

- MathNet.Numerics — для виконання статистичних розрахунків і побудови трендових моделей (лінійна регресія, ковзне середнє);
- System.Text.Json — для серіалізації даних;
- Microsoft.Web.WebView2.Wpf — для відображення звітів у PDF/HTML форматах;
- EntityFrameWork Core – ORM фреймворк для взаємодії з БД

7. Технологічна сумісність і розширюваність

Застосунок розроблено з урахуванням можливості масштабованого розгортання у корпоративному середовищі Windows, що забезпечує стабільну роботу без необхідності встановлення додаткових зовнішніх бібліотек чи спеціалізованого серверного програмного забезпечення. Такий підхід дає змогу легко інтегрувати систему в існуючу IT-інфраструктуру підприємства, мінімізуючи витрати на обслуговування та підтримку.

Система побудована на базі технологій .NET (WPF, Entity Framework Core, LINQ), що забезпечує кросплатформенну сумісність у межах екосистеми Microsoft і дозволяє використовувати її як у вигляді локального десктопного додатка, так і як частину корпоративної інтегрованої платформи.

Передбачено можливість міграції системи на хмарні середовища, такі як Microsoft Azure або Amazon Web Services (AWS), із використанням контейнеризації (Docker) або публікації у вигляді вебсервісів. Це відкриває перспективу реалізації концепції Software as a Service (SaaS), коли доступ до функціоналу системи можна отримувати через вебінтерфейс або корпоративну мережу.

Окремо реалізовано модульну структуру для інтеграції з аналітичними системами бізнес-інтелекту (BI), зокрема Power BI, QlikView та Tableau, через REST API або механізм Data Export. Експорт даних можливий у форматах JSON, CSV та XML, що дозволяє будувати гнучкі звіти, інтерактивні панелі моніторингу та розширювати можливості системи без змін у її основному коді.

Завдяки такій архітектурі застосунок може використовуватись як самостійна інформаційна система для управління запасами, так і як аналітичний модуль у складі корпоративної ERP-системи, забезпечуючи єдине інформаційне середовище для прийняття управлінських рішень.

4 РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

У даному розділі наведено результати практичної реалізації та дослідження розробленої системи управління товарними запасами з використанням алгоритмів оптимізації, зокрема описано етапи експериментального дослідження, методику перевірки ефективності алгоритмів і аналіз отриманих результатів.

Дослідження мало на меті підтвердити, що застосування запропонованих методів — лінійної оптимізації та ковзного середнього — дозволяє досягнути підвищення ефективності управління запасами, зменшення надлишкових залишків і покращення точності прогнозування закупівель.

Під час експериментів перевірялася коректність функціонування алгоритмів підсистеми оптимізації, що відповідає за розрахунок оптимальних рівнів запасів і рекомендованих обсягів замовлення. Особливу увагу приділено достовірності прогнозів, які формуються на основі історичних даних про продажі, та відповідності фактичних результатів теоретично очікуваним значенням.

4.1. Хід проведення дослідження

Дослідження проводилось у кілька етапів, відповідно до логічної послідовності розробки самої системи:

1. Підготовчий етап – на даному етапі проводився збір вимог до системи, аналіз предметної області та розгляд вже існуючих рішень на ринку
2. Етап моделювання – в ході даного етапу було проведено моделювання предметної області з використанням UML-діаграм класів, об’єктів та пакетів з об’єктно-орієнтованого моделювання, та діаграм Use-case, діяльності по методології функціонального моделювання.

3. Розробка програмної частини – під час даного етапу було розроблено програмне рішення для проведення дослідження за архітектурою MVVM, як результат було отримано структуру проєкту, зображену на Рис 4.1-4.3.Також в ДОДАТОК Б показано вигляд самої системи.

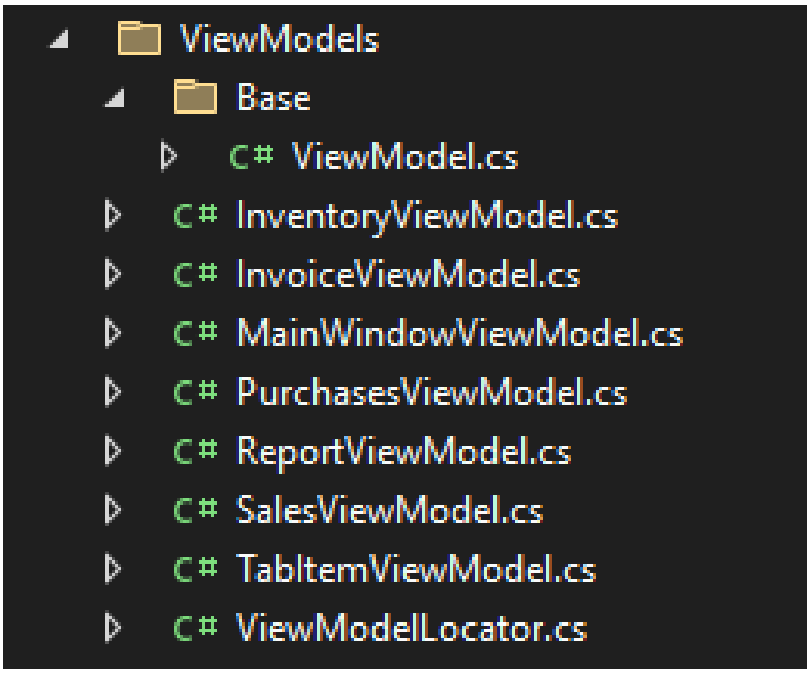


Рис. 4.1. Елементи шару ViewModel

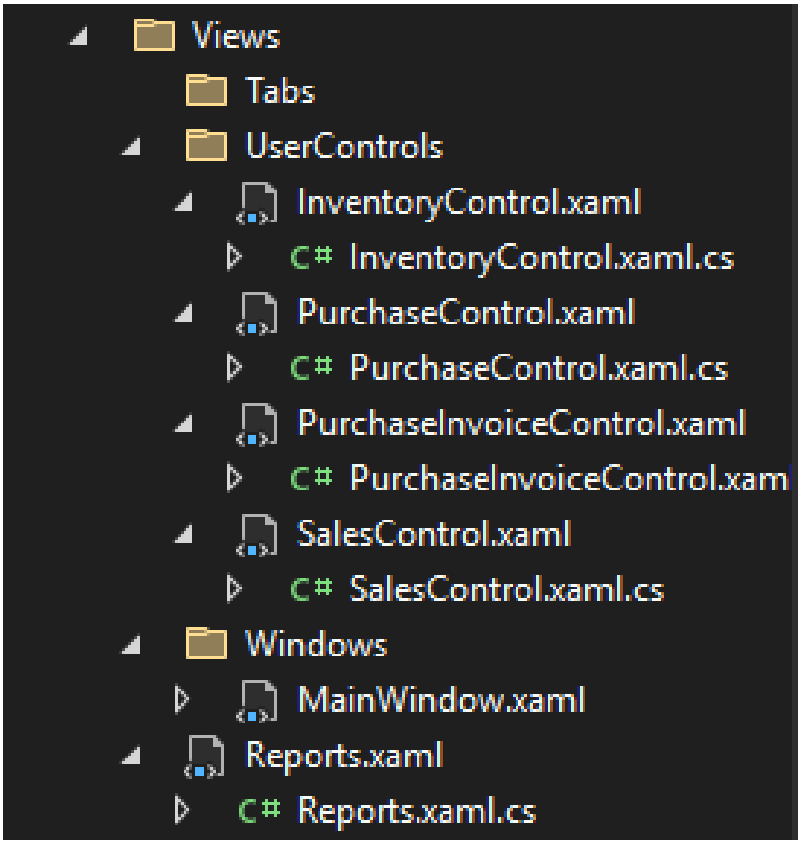


Рис. 4.2. Елементи шару View

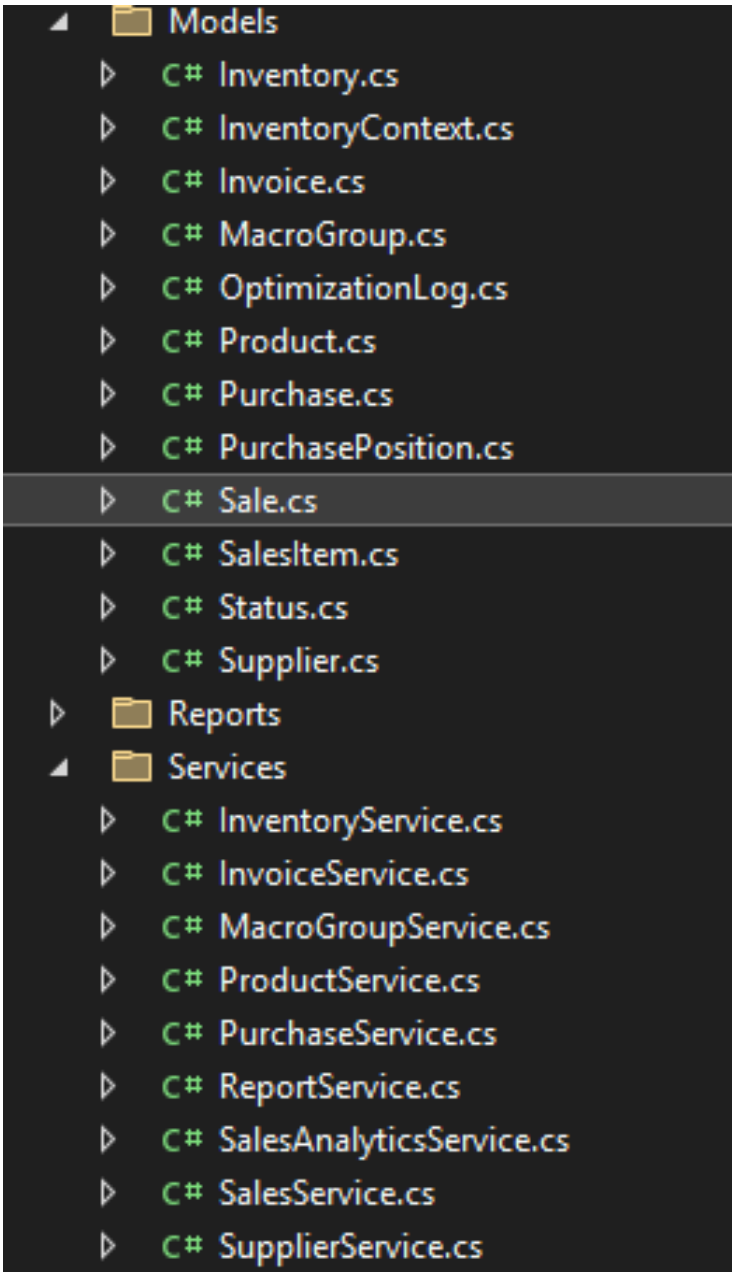


Рис. 4.3. Елементи шару Model

4. Етап експериментального дослідження – на даному етапі було проведено працездатність алгоритмів оптимізації, проведено моделювання різних сценаріїв закупівлі та їх порівняння з точки зору ефективності, побудовано аналітичні діаграми прогнозів продажів на 3 місяці.

4.2. Аналіз отриманих результатів

Результати дослідження показали, що розроблена система дозволяє доволі ефективно керувати запасами й зменшувати обсяг надлишкової продукції, зокрема в ході експерименту було додатково розроблено

алгоритм, який буде обраховувати якість покращень з точки зору оптимізації закупівель, який базується на порівнянні стану таблиць до закупівлі та після. Для розрахунку було використано два значення: before (середній обсяг показника до оптимізації) та after (середній обсяг показника після оптимізації).

Основним критерієм ефективності виступає відсоткова зміна між початковим та оптимальним значеннями, тому формула для обрахунку має наступний вигляд:

$$Improvement = \frac{After - Before}{Before} \times 100$$

Де:

- Improvement – відсоток покращення
- Before – початкове значення показника
- After – Значення показника після застосування алгоритму.

Таким чином, якщо after < before, то результат буде позитивним і це означатиме покращення в результаті оптимізації, відповідно якщо after > before, то результат вийде від'ємним, що означатиме погіршення. Текст методу наведено на Рис 4.4. Результати було візуалізовано за допомогою OxyPlot, як і інші візуалізації в системі. Вісь Y містить категорії ("Надлишкові запаси", "Дефіцит товарів"),а вісь X — відсоткове значення покращення.

Під час побудови графіка додатково застосовується модифікація від'ємних значень: значення з негативним покращенням відображаються іншим кольором або із міткою "↓", що дозволяє візуально оцінити, у яких категоріях спостерігалось зниження ефективності.

```
4 references
public class OptimizationResult
{
    2 references
    public string Metric { get; set; } = string.Empty;
    4 references
    public double Before { get; set; }
    3 references
    public double After { get; set; }
    1 reference
    public double Improvement => Math.Round((After - Before) / Before * 100, 2);
}

1 reference
public async Task<List<OptimizationResult>> EvaluateOptimizationAsync()
{
    // 1️⃣ До оптимізації (історичні дані)
    var beforeAvgStock = await _context.Products
        .AverageAsync(p => p.SafetyStock);

    // 2️⃣ Після оптимізації
    var afterAvgStock = await _context.Products
        .AverageAsync(p => p.OptimalStock);

    // 3️⃣ Дефіцит (товари нижче SafetyStock)
    var total = await _context.Products.CountAsync();
    var deficitBefore = await _context.Inventories
        .Where(i => i.Quantity < i.Product.SafetyStock)
        .CountAsync();
    var deficitAfter = await _context.Inventories
        .Where(i => i.Quantity < i.Product.OptimalStock)
        .CountAsync();
```

Рис. 4.4. Метод для перевірки результатів оптимізації

Тепер при виконання бачимо результат оптимізації(Рис. 4.5.), де покращення надлишкових запасів склало ~54%, а покращення в області дефіциту - ~63%. Таким чином є сенс стверджувати, що алгоритм оптимізації принесе користь підприємству, хоч це і більше теоретичні дані.

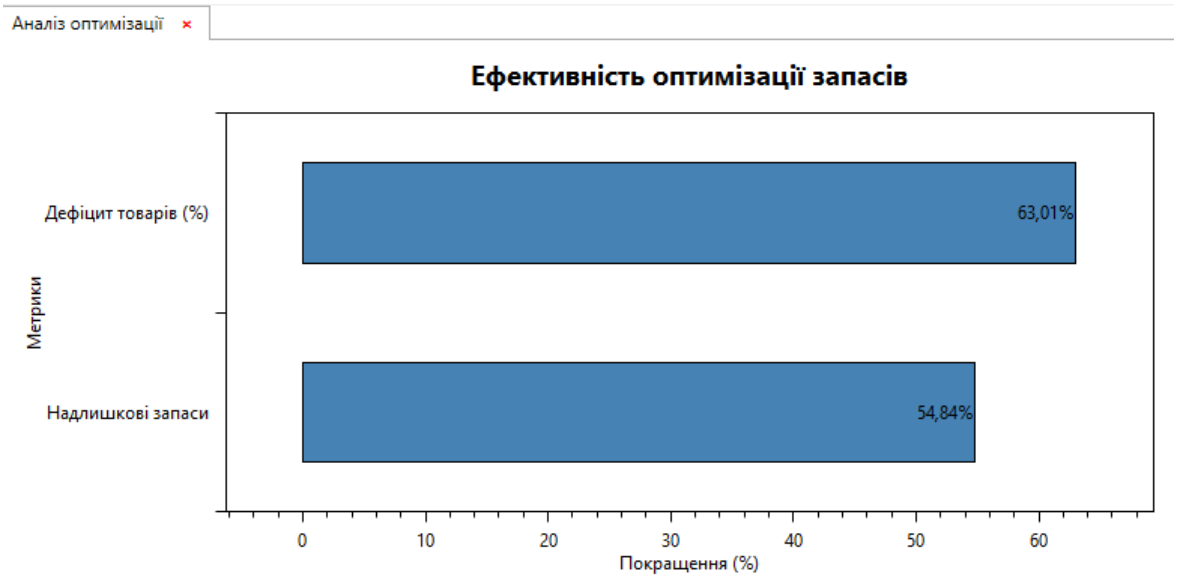


Рис. 4.5. Ефективність оптимізації закупівель (Лінійна оптимізація)

ВИСНОВОК

У ході виконання магістерської кваліфікаційної роботи було проведено комплексне дослідження процесів управління товарними запасами, що дало змогу визначити основні проблеми у сфері оптимізації складських залишків та обґрунтувати необхідність використання сучасних інформаційних технологій для їх вирішення. На основі системного аналізу встановлено, що традиційні методи управління запасами (EOQ, ABC/XYZ-аналіз, ковзне середнє тощо) мають суттєві обмеження, оскільки не враховують динаміку попиту, сезонність і зовнішні фактори, що знижує точність управлінських рішень.

Було розроблено концепцію та архітектуру програмного забезпечення для управління товарними запасами, яке поєднує функції обліку, прогнозування та оптимізації. Система побудована за архітектурним шаблоном MVVM, що забезпечує логічне розділення рівнів даних, бізнес-логіки та інтерфейсу користувача, спрощуючи супровід і масштабування. У процесі моделювання розроблено діаграми прецедентів, діаграми діяльності, діаграму класів і діаграму пакетів, які відображають структуру та взаємодію основних компонентів системи.

Практичний результат роботи полягає у створенні прототипу програмного забезпечення, яке дозволяє аналізувати продажі, прогнозувати попит і формувати оптимальні замовлення постачальникам. Реалізовані модулі збору, обробки та візуалізації даних забезпечують автоматизацію прийняття рішень щодо поповнення товарних запасів. Проведене експериментальне тестування підтвердило ефективність використаних алгоритмів оптимізації — отримані результати показали зниження надлишкових запасів і стабілізацію рівня дефіциту товарів.

Таким чином, розроблена система може бути використана торговельними підприємствами малого та середнього бізнесу для підвищення ефективності логістичних процесів, зменшення витрат і підвищення обґрунтованості управлінських рішень, зокрема дослідження показало, що покращення

надлишкових запасів склало ~54%, а покращення в області дефіциту - ~63%. У подальшому розвиток проєкту може бути спрямований на інтеграцію з BI-системами, хмарними платформами та впровадження більш складних методів машинного навчання для підвищення точності прогнозів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dependency Injection in .NET | Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection.
2. Diagram Types | draw.io Documentation. [Електронний ресурс] – Режим доступу: https://www.drawio.com/doc/getting-started-diagram-types.
3. Documents in WPF | Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/advanced/documents-in-wpf.
4. Model-View-ViewModel (MVVM) Pattern | .NET MAUI Architecture | Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/uk-ua/dotnet/architecture/maui/mvvm.
5. MVC Design Pattern | GeeksforGeeks. [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/system-design/mvc-design-pattern/.
6. MVVM Framework | WPF Controls | DevExpress Documentation. [Електронний ресурс] – Режим доступу: https://docs.devexpress.com/WPF/15112/mvvm-framework.
7. OxyPlot Documentation. [Електронний ресурс] – Режим доступу: https://oxyplot.readthedocs.io/en/latest/.
8. RDLC Report Layouts | Business Central | Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/uk-ua/microsoft-edge/webview2/?form=MA13LH.
9. SQL Server Management Studio (SSMS) | Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/ssms/.
10. T-SQL Language Reference | Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/sql/t-sql/language-reference?view=sql-server-ver17.
11. WebView2 | Microsoft Edge Developer. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/uk-ua/microsoft-edge/webview2/?form=MA13LH.

12. Windows Presentation Foundation documentation | Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/desktop/wpf/.
13. UML-діаграми | Evergreen. [Електронний ресурс] – Режим доступу: https://evergreens.com.ua/ua/articles/uml-diagrams.html.
14. Огляд Entity Framework Core | Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/ru-ru/ef/core/.
15. Оптимізація управління запасами: стратегії, методи, переваги | ABM Cloud. [Електронний ресурс] – Режим доступу: https://abmcloud.com/uk/optimizacziya-upravlinnya-zapasami/.
16. Розроблення програмного забезпечення | BukLib.net. [Електронний ресурс] – Режим доступу: https://buklib.net/books/27325/.

ДОДАТОК А

Текст методу прогнозування продажів

Сторінок 3

```
#region Sales
 public class MonthlySalesDto
 {
    public string Month { get; set; }
    public decimal Total { get; set; }
    public bool IsForecast { get; set; } = false;
 }

 /// Отримуємо продажі по місяцях + прогноз наступного
 public async Task<List<MonthlySalesDto>> GetSalesWithForecastAsync()
 {
    var rawData = await _context.Sales
       .GroupBy(s => new { s.SaleDate.Year, s.SaleDate.Month })
       .Select(g => new
       {
          Year = g.Key.Year,
          Month = g.Key.Month,
          Total = g.Sum(s => s.TotalAmount)
       })
       .OrderBy(x => x.Year).ThenBy(x => x.Month)
       .ToListAsync();

    // Перетворення → DTO
    var data = rawData
       .Select(x => new MonthlySalesDto
       {
          Month = $"{x.Month}.{x.Year}",
          Total = (decimal)x.Total
       })
       .ToList();

    // Масиви для регресії
    double[] x = data.Select(d =>
       DateTime.Parse($"01.{d.Month}").ToOADate()
    ).ToArray();

    double[] y = data.Select(d => (double)d.Total).ToArray();

    // Якщо даних мало → ковзне середнє
    bool useAverage = data.Count < 4;

    double last3avg = y.TakeLast(Math.Min(3, y.Length)).Average();

    // Лінійна регресія, якщо даних достатньо
    double a = 0, b = 0;
```

```
if (!useAverage)
    (a, b) = Fit.Line(x, y);

// --- Прогнозуємо наступні 3 місяці ---
DateTime startDate = DateTime.Parse($"01.{data.Last().Month}");

// Використовуємо список для x і y, щоб динамічно розширювати
var dynX = x.ToList();
var dynY = y.ToList();

for (int i = 1; i <= 3; i++)
{
    DateTime forecastDate = startDate.AddMonths(i);
    double forecastX = forecastDate.ToOADate();

    double predicted;

    if (dynY.Count < 4)
    {
        predicted = dynY.Average();
    }
    else
    {
        var (a2, b2) = Fit.Line(dynX.ToArray(), dynY.ToArray());
        predicted = a2 * forecastX + b2;

        double last3avg2 = dynY.TakeLast(Math.Min(3, dynY.Count)).Average();

        if (predicted < last3avg2 * 0.5)
            predicted = last3avg2;
    }

    if (predicted < 0) predicted = 0;
    predicted = Math.Round(predicted, 2);

    // Додаємо в результат
    data.Add(new MonthlySalesDto
    {
        Month = $"{forecastDate.Month}.{forecastDate.Year}",
        Total = (decimal)predicted,
        IsForecast = true
    });

    // ОНОВЛЮЄМО МАСИВИ!
    dynX.Add(forecastX);
```

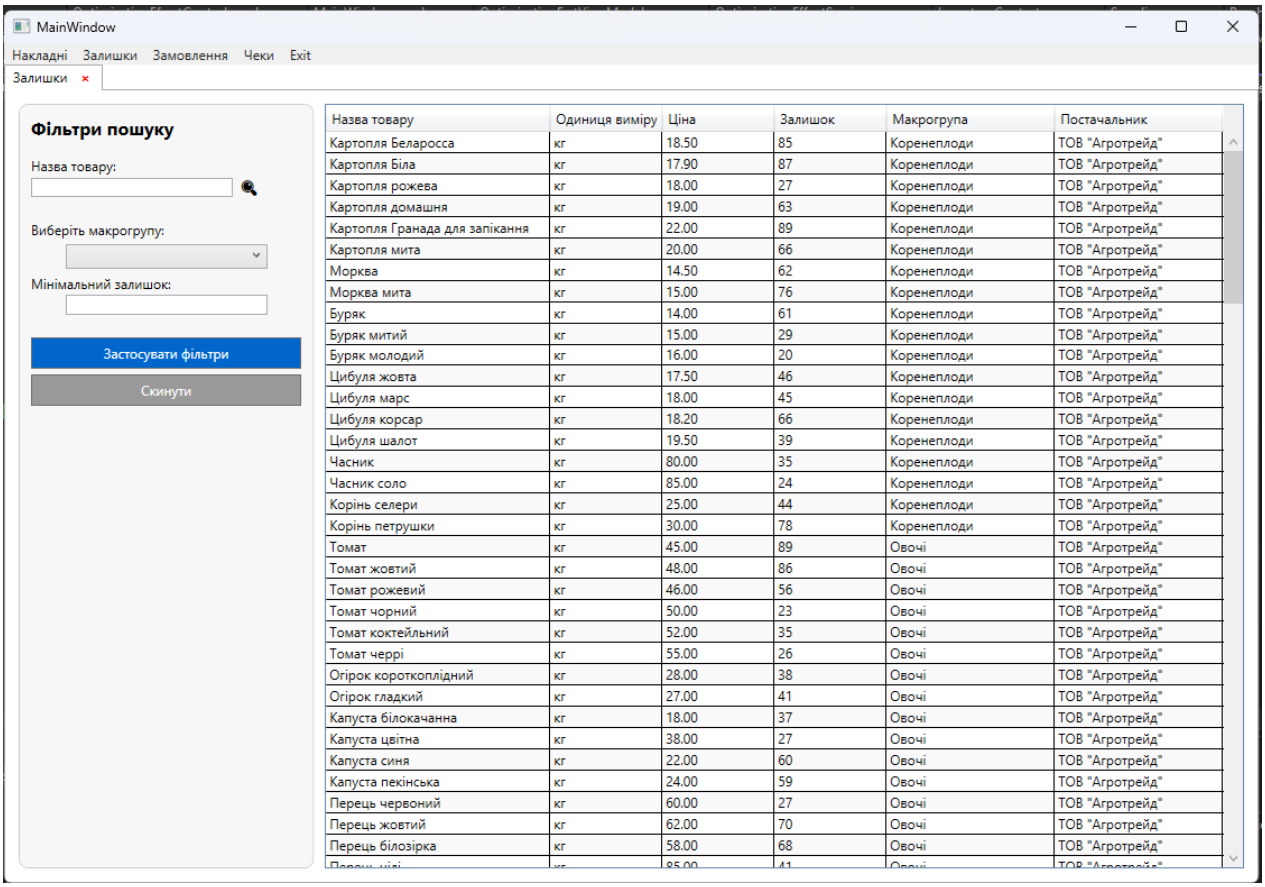
```
            dynY.Add(predicted);
        }

        return data;
    }
    #endregion
```

ДОДАТОК Б

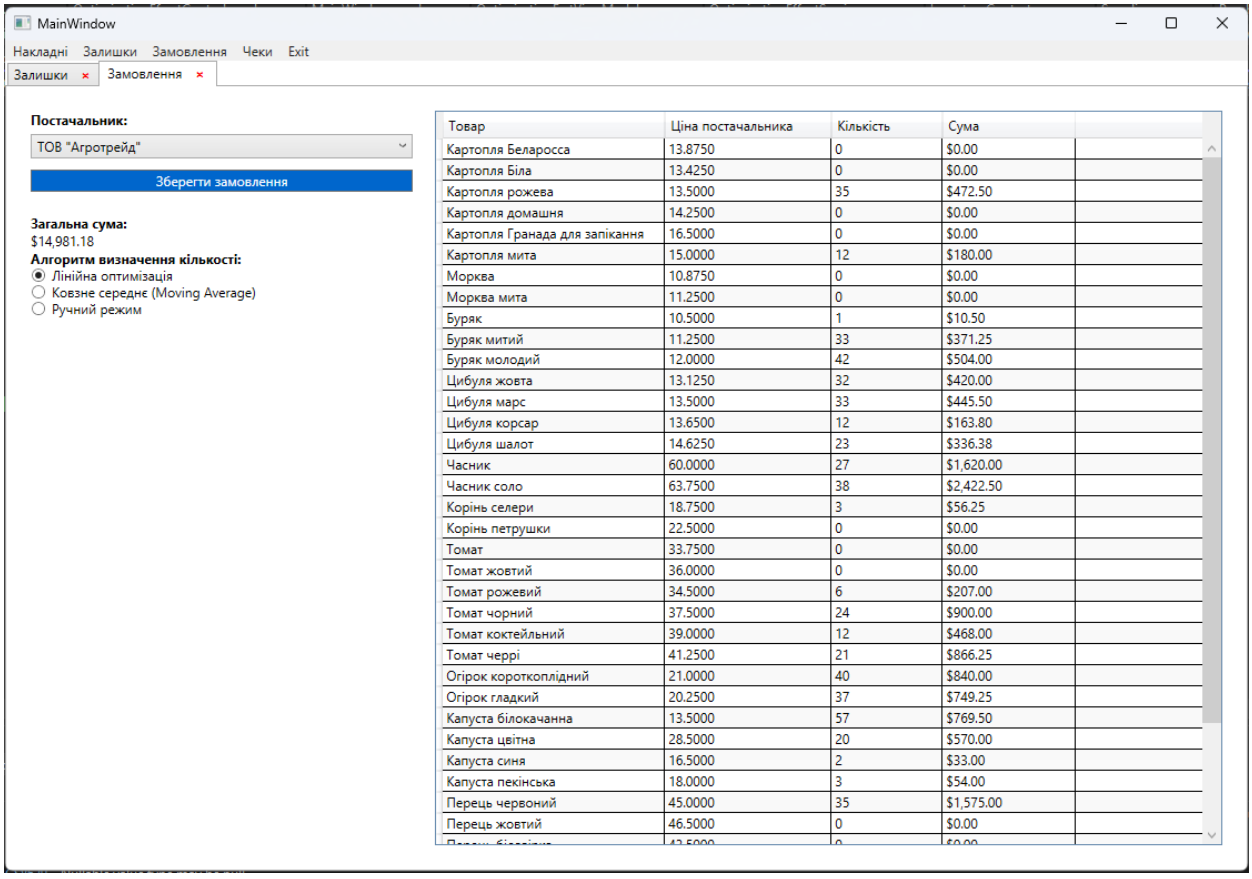
Загальний вигляд інтерфейсу

Сторінок 3



Назва товару	Одиниця виміру	Ціна	Залишок	Макрогрупа	Постачальник
Картопля Белароссa	кг	18.50	85	Коренеплоди	ТОВ "Агротрейд"
Картопля Біла	кг	17.90	87	Коренеплоди	ТОВ "Агротрейд"
Картопля рожева	кг	18.00	27	Коренеплоди	ТОВ "Агротрейд"
Картопля домашня	кг	19.00	63	Коренеплоди	ТОВ "Агротрейд"
Картопля Гранада для запікання	кг	22.00	89	Коренеплоди	ТОВ "Агротрейд"
Картопля мита	кг	20.00	66	Коренеплоди	ТОВ "Агротрейд"
Морква	кг	14.50	62	Коренеплоди	ТОВ "Агротрейд"
Морква мита	кг	15.00	76	Коренеплоди	ТОВ "Агротрейд"
Буряк	кг	14.00	61	Коренеплоди	ТОВ "Агротрейд"
Буряк митий	кг	15.00	29	Коренеплоди	ТОВ "Агротрейд"
Буряк молодий	кг	16.00	20	Коренеплоди	ТОВ "Агротрейд"
Цибуля жовта	кг	17.50	46	Коренеплоди	ТОВ "Агротрейд"
Цибуля марс	кг	18.00	45	Коренеплоди	ТОВ "Агротрейд"
Цибуля корсар	кг	18.20	66	Коренеплоди	ТОВ "Агротрейд"
Цибуля шалот	кг	19.50	39	Коренеплоди	ТОВ "Агротрейд"
Часник	кг	80.00	35	Коренеплоди	ТОВ "Агротрейд"
Часник соло	кг	85.00	24	Коренеплоди	ТОВ "Агротрейд"
Корінь селери	кг	25.00	44	Коренеплоди	ТОВ "Агротрейд"
Корінь петрушки	кг	30.00	78	Коренеплоди	ТОВ "Агротрейд"
Томат	кг	45.00	89	Овочі	ТОВ "Агротрейд"
Томат жовтий	кг	48.00	86	Овочі	ТОВ "Агротрейд"
Томат рожевий	кг	46.00	56	Овочі	ТОВ "Агротрейд"
Томат чорний	кг	50.00	23	Овочі	ТОВ "Агротрейд"
Томат коктейльний	кг	52.00	35	Овочі	ТОВ "Агротрейд"
Томат черрі	кг	55.00	26	Овочі	ТОВ "Агротрейд"
Огірок короткоплідний	кг	28.00	38	Овочі	ТОВ "Агротрейд"
Огірок гладкий	кг	27.00	41	Овочі	ТОВ "Агротрейд"
Капуста білокачанна	кг	18.00	37	Овочі	ТОВ "Агротрейд"
Капуста цвітна	кг	38.00	27	Овочі	ТОВ "Агротрейд"
Капуста синя	кг	22.00	60	Овочі	ТОВ "Агротрейд"
Капуста пекінська	кг	24.00	59	Овочі	ТОВ "Агротрейд"
Перець червоний	кг	60.00	27	Овочі	ТОВ "Агротрейд"
Перець жовтий	кг	62.00	70	Овочі	ТОВ "Агротрейд"
Перець білозірка	кг	58.00	68	Овочі	ТОВ "Агротрейд"

Рис. 1. Вкладка «Залишки»



Товар	Ціна постачальника	Кількість	Сума
Картопля Белароссa	13.8750	0	$0.00
Картопля Біла	13.4250	0	$0.00
Картопля рожева	13.5000	35	$472.50
Картопля домашня	14.2500	0	$0.00
Картопля Гранада для запікання	16.5000	0	$0.00
Картопля мита	15.0000	12	$180.00
Морква	10.8750	0	$0.00
Морква мита	11.2500	0	$0.00
Буряк	10.5000	1	$10.50
Буряк митий	11.2500	33	$371.25
Буряк молодий	12.0000	42	$504.00
Цибуля жовта	13.1250	32	$420.00
Цибуля марс	13.5000	33	$445.50
Цибуля корсар	13.6500	12	$163.80
Цибуля шалот	14.6250	23	$336.38
Часник	60.0000	27	$1,620.00
Часник соло	63.7500	38	$2,422.50
Корінь селери	18.7500	3	$56.25
Корінь петрушки	22.5000	0	$0.00
Томат	33.7500	0	$0.00
Томат жовтий	36.0000	0	$0.00
Томат рожевий	34.5000	6	$207.00
Томат чорний	37.5000	24	$900.00
Томат коктейльний	39.0000	12	$468.00
Томат черрі	41.2500	21	$866.25
Огірок короткоплідний	21.0000	40	$840.00
Огірок гладкий	20.2500	37	$749.25
Капуста білокачанна	13.5000	57	$769.50
Капуста цвітна	28.5000	20	$570.00
Капуста синя	16.5000	2	$33.00
Капуста пекінська	18.0000	3	$54.00
Перець червоний	45.0000	35	$1,575.00
Перець жовтий	46.5000	0	$0.00

Рис. 2. Вкладка «Замовлення»

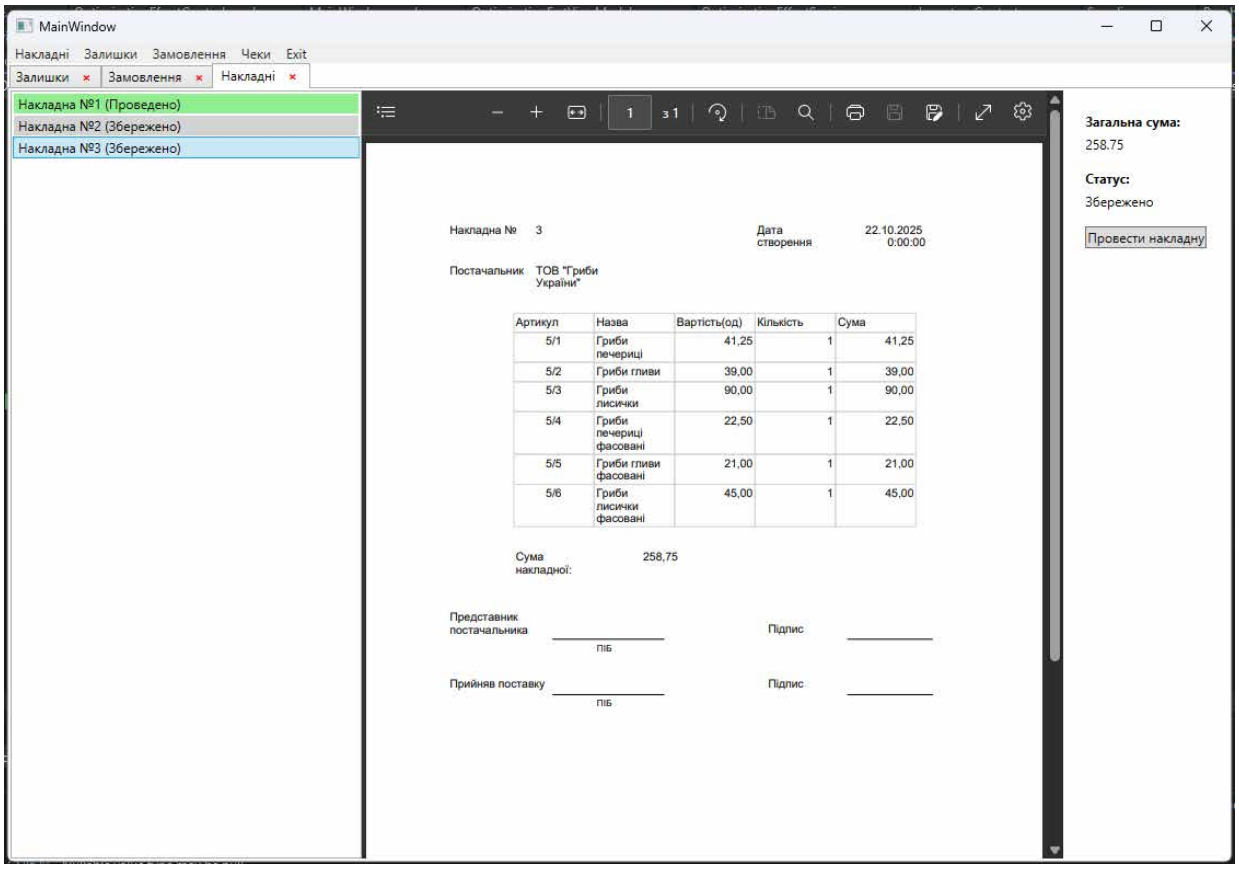


Рис. 3. Вкладка «Накладні»

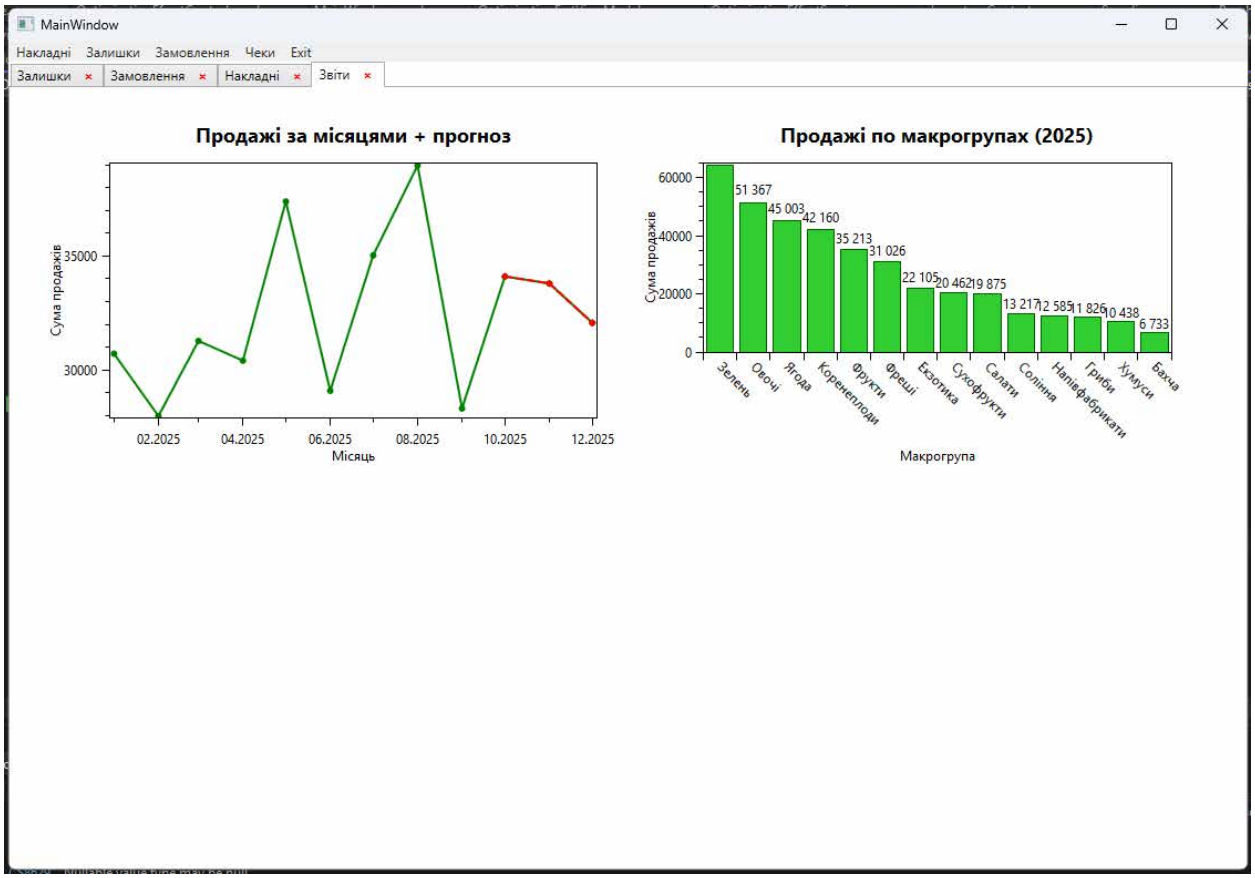


Рис. 4. Вкладка «Аналітика»

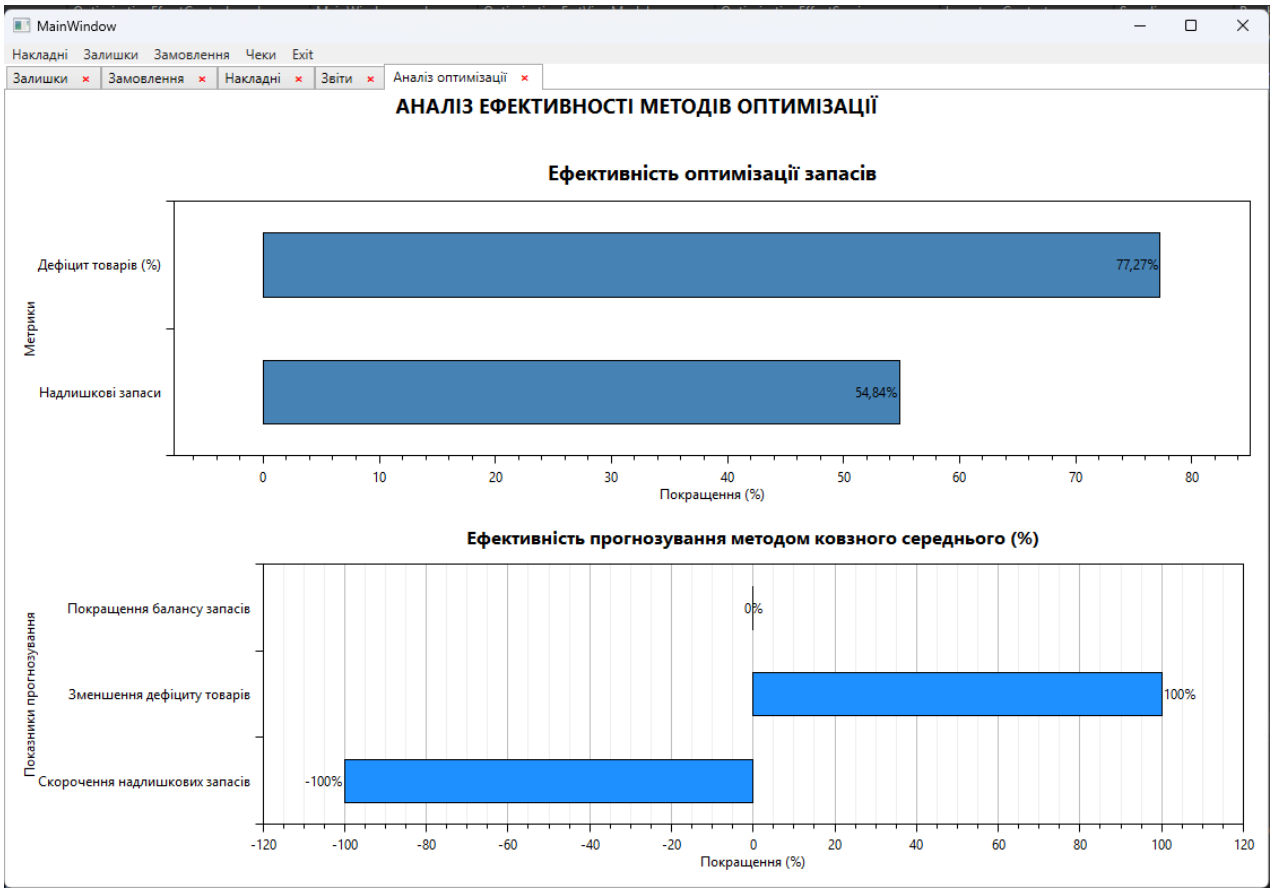


Рис. 5. Вкладка «Аналіз оптимізації»

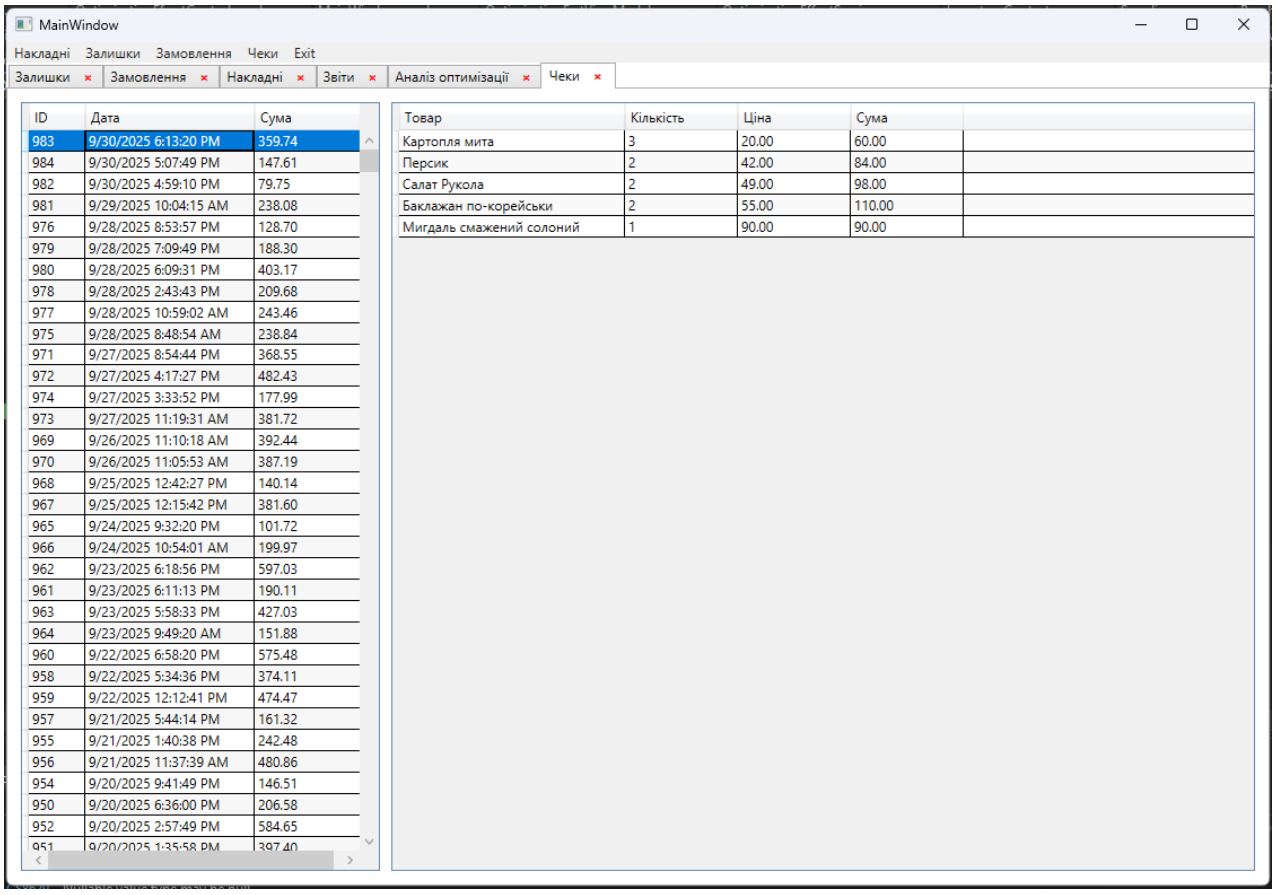


Рис. 6. Вкладка «Чеки»