Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління

Бакалаврська кваліфікаційна робота
на тему:

«РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ СКЛАДУ»

Виконав: студент 4 курсу, групи 2АКІТ-21б
спеціальності 151 – Автоматизація та
комп'ютерно-інтегровані технології

_____ Сергій ІВАНОВ

Керівник: к.т.н., доцент каф. КСУ

_____ Олег КОВАЛЮК

«10» червня 2025 р.

Рецензент: к.т.н., доцент каф. АІІТ

_____ Ярослав КУЛИК

«11» червня 2025 р.

Допущено до захисту
Зав. кафедри КСУ

_____ В'ячеслав КОВТУН
«12» червня 2025 р.

Вінниця ВНТУ – 2025 рік

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління
Рівень вищої освіти І-й (бакалаврський)
Галузь знань – 15 Автоматизація та приладобудування
Спеціальність 151 Автоматизація та комп'ютерно-інтегровані технології
Освітньо-професійна програма Інтелектуальні комп'ютерні системи управління

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ
д.т.н., проф.
В'ячеслав КОВТУН
«24» 05 2025 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Іванову Сергію Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи
Розробка інтелектуальної комп'ютерної системи автоматизації складу
керівник роботи Ковалюк Олег Олександрович, к.т.н., доцент кафедри КСУ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджена наказом ВНТУ № 97 від 20.03.2025 р.
2. Термін подання студентом роботи 14 06 2025 р.
3. Вихідні дані до роботи: Розробка інтелектуальної комп'ютерної системи автоматизації складу; мови графічного інтерфейсу – українська; використання кодування під час передачі даних – так. Джерела розробки: 1) Thompson E. What is Warehouse Automation? Examples, Types, and Benefits. Cyngn | Industrial AMRs to Automate Your Material Handling. 20.06.2024. URL: https://www.cyngn.com/blog/what-is-warehouse-automation-examples-types-and-benefits, 2) Плескач В. Л., Затонацька Т. Г. Інформаційні системи і технології на підприємствах. – Розділ 5: Інтегровані інформаційні системи управління підприємством (ERP-системи). – С. 110–123.
4. Зміст текстової частини: розробка архітектури системи; розробка програмного продукту; аналіз аналогів розробки; тестування програмного забезпечення.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень)
UML-діаграма варіантів використання, Діаграма блок-схема (клієнт), Моделі таблиці товарів, Створення Celery воркерів, Перевизначення методу save, Функція

створення замовлень, Воркер для зміни дат подій та щомісячних перевірок, Таблиці бази даних, Контейнери в Docker-compose, Лист до постачальника, відправлений системою, Тестування переведення дат (до відпрацювання воркеру, після відпрацювання воркеру), Тестування перевірки щомісячних продажів (до відпрацювання воркеру, після відпрацювання воркеру).

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 26 05 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів роботи	Примітка
1	Постановка задач дослідження	26.05.2025 р.	
2	Визначення технічних характеристик системи	31.05.2025 р.	
3	Аналіз і вибір технологій для розробки ПЗ	1.06.2025 р.	
4	Проєктування інтелектуальної комп'ютерної системи автоматизації складу	1.06.2025 р. – 3.06.2025 р.	
5	Розробка програмного забезпечення системи	3.06.2025 р. – 12.06.2025 р.	
6	Оформлення пояснювальної записки, графічного матеріалу і презентації	10.06.2025 р. – 14.06.2025 р.	
7	Графічні матеріали	12.06. 2025 р.	
8	Захист БКР	17.06.2025 р.	

Студент _____ (підпис) Сергій ІВАНОВ

Керівник роботи _____ (підпис) Олег КОВАЛЮК

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 91 сторінки А4, на яких 66 рисунків, 4 додатка, список використаних джерел містить 24 найменування.

Метою роботи є підвищення ефективності складських облікових процесів шляхом аналізу сучасних програмних рішень та створення власної інтелектуальної комп'ютерної системи автоматизації складу.

У теоретичній частині розглянуто особливості предметної області, здійснено огляд наявних програм для автоматизації складського обліку та визначено ключові вимоги до майбутньої системи. Проведено порівняльний аналіз функціональності існуючих рішень, їх переваг, недоліків та відповідності сучасним потребам підприємств. У практичному розділі виконано проєктування, розробку й тестування програмного забезпечення з використанням сучасних технологій, зокрема Python, Django, PostgreSQL, Celery та Docker. Розроблена система реалізує автоматичне формування замовлень, облік залишків товарів, прогнозування попиту та врахування сезонних факторів, що забезпечує зниження рутинного навантаження на персонал.

Ключові слова: склад, автоматизація, прогнозування, поповнення, закупівля, облік товару.

ANNOTATION

The bachelor's qualification work consists of 91 A4 pages, on which there are 66 figures, 4 appendices, the list of used sources contains 24 names.

The purpose of the work is to increase the efficiency of warehouse accounting processes by analyzing modern software solutions and creating our own intelligent computer warehouse automation system.

The theoretical part considers the features of the subject area, reviews existing programs for automating warehouse accounting, and identifies key requirements for the future system. A comparative analysis of the functionality of existing solutions, their advantages, disadvantages, and compliance with modern needs of enterprises is carried out. In the practical section, software design, development, and testing are performed using modern technologies, in particular Python, Django, PostgreSQL, Celery, and Docker. The developed system implements automatic order formation, accounting for product balances, demand forecasting, and consideration of seasonal factors, which reduces the routine workload on personnel.

Keywords: warehouse, automation, forecasting, replenishment, procurement, product accounting.

ЗМІСТ

ВСТУП……4
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ІНТЕЛЕКТУАЛЬНИХ КОМП'ЮТЕРНИХ СИСТЕМ АВТОМАТИЗІЦІЇ СКЛАДУ.....6
1.1 Аналіз предметної області……..……...…...6
1.2 Аналіз існуючих систем автоматизації складу…….……8
1.3 Постановка задачі дослідження……19
1.4 Висновки…….……20
2 ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ АВТОМАТИЗІЦІЇ СКЛАДУ……...21
2.1 Аналіз функцій системи…….…21
2.2 Проєктування архітектури системи……..22
2.3 Проєктування алгоритмів роботи системи……..……23
2.4 Вибір засобів створення програмного забезпечення для розробки інтелектуальної комп'ютерної системи автоматизації складу…….25
2.4.1 Вибір мови програмування…….29
2.4.2 Вибір системи управління базами даних (СУБД)…….…31
2.4.3 Розгортання проєкту……32
2.5 Висновки……...……..33
3 РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ АВТОМАТИЗІЦІЇ СКЛАДУ……..….34
3.1 Написання коду……..34
3.2 Створення докер контейнерів……..…….42
3.3 Тестування програмного продукту……..48
3.4 Висновки…….……...…….....59
ВИСНОВКИ……...…….60
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ……61
ДОДАТКИ
Додаток А (обов'язковий) – Протокол перевірки кваліфікаційної роботи на наявність текстових запозичень…….…64

Додаток Б (обов’язковий) – Технічне завдання……65
Додаток В (довідковий) – Лістинг програми…….……...68
Додаток Г (обов’язковий) – Ілюстративна частина……...……...……..……81

ВСТУП

У другій половині XX століття розпочалася інформаційна та мікропроцесорна революції, які стали основою для активного розвитку засобів автоматизації в усіх сферах людської діяльності. Поява мікропроцесорів, зростання обчислювальної потужності комп'ютерів, розвиток комп'ютерних мереж і програмного забезпечення відкрили нові можливості для створення інтелектуальних систем управління. Це значно вплинуло на виробництво, логістику та управління складськими процесами, особливо у сфері малого та середнього бізнесу.

Автоматизація складу[1] - це впровадження технологій, які дозволяють зменшити участь людини у виконанні рутинних і трудомістких операцій, таких як облік товарів, формування замовлень, контроль залишків, планування постачань тощо. На сьогоднішній день автоматизація складу стає все більш актуальною навіть для компаній із невеликим штатом і обмеженим бюджетом.

Метою дослідження є підвищення ефективності управління товарним обліком шляхом розробки власної системи, що відзначається простотою впровадження та експлуатації, а також не потребує значних фінансових витрат.

Для досягнення поставленої мети необхідно вирішити такі задачі:

– Дослідити особливості предметної області, визначити актуальні потреби підприємств у сфері складського обліку;

– Провести аналіз існуючих програмних рішень для автоматизації складу, оцінити їх переваги та недоліки;

– Спроєктувати архітектуру системи з урахуванням вимог до гнучкості, масштабованості та простоти інтеграції;

– Обрати відповідні технології для реалізації інтелектуальної логіки системи, включаючи засоби обробки даних, автоматичні замовлення і прогнозування потреб у поповненні;

– Реалізувати програмне забезпечення, протестувати його функціональність та перевірити відповідність запланованому функціоналу;

Об'єктом дослідження є процес управління складськими операціями в умовах малого підприємства.

Предметом дослідження є автоматизована система керування складом.

Новизна полягає у створенні бюджетної та простої в реалізації системи автоматизації складських процесів із використанням сучасних програмних рішень.

Практична цінність полягає у впровадженні розробленої системи для оптимізації роботи складу бізнесу без значних фінансових витрат.

Апробація. Представлені в роботі результати апробовані в результаті участі в конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)»[2].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ ІНТЕЛЕКТУАЛЬНИХ КОМП'ЮТЕРНИХ СИСТЕМ АВТОМАТИЗІЦЇ СКЛАДУ

1.1 Аналіз предметної області

Останні роки електронна комерція, зокрема інтернет-магазини, переживає справжній бум - і у світі, і в Україні. Люди дедалі частіше обирають онлайн-шопінг через зручність, швидке оформлення замовлень і великий вибір товарів. У відповідь на ці зміни бізнесу доводиться швидко перебудовувати логістику та роботу складів, щоб ефективно обслуговувати клієнтів, вчасно поповнювати запаси та тримати все під контролем.

Сучасні інтернет-магазини працюють з великим асортиментом товарів, який постійно змінюється - щось залежить від сезону, щось від попиту або акцій. Щоб бізнес працював стабільно, потрібно своєчасно оновлювати запаси: не допускати ні дефіциту, ні надлишків. Перше призводить до втрати прибутку, друге - до "заморожування" грошей у товарі. І якщо раніше можна було вести облік у таблицях, вручну аналізувати продажі та формувати замовлення, то при великому товарообігу такий підхід вже не спрацьовує.

Традиційна система управління залишками вимагає постійного контролю за кожною позицією, аналізу продажів, врахування термінів доставки й навіть майбутніх подій, що впливають на попит: наприклад, акції, сезонні розпродажі або запуск нових товарів. Така робота потребує часу й уваги досвідчених спеціалістів. А коли бізнес росте, виникає потреба або в розширенні команди, або в автоматизації процесів. На щастя, сучасні технології дають змогу значно полегшити цей процес.

Сьогодні можна автоматизувати[1]:

- контроль залишків на складі;
- формування замовлень, коли запаси знижуються до певного рівня;
- прогнозування попиту на основі минулих продажів;
- врахування термінів доставки та поведінки клієнтів;
- створення шаблонних запитів для постачальників;
- збір статистики для подальшого аналізу.

Усе це реалізується за допомогою розумних програмних систем, які можуть працювати у зв'язці з базами даних, CRM, ERP, обліковими системами та системами управління торгівлею. Такі системи можуть бути зовсім простими — працювати за заздалегідь заданими правилами, або ж більш складними й «розумними», що вміють аналізувати дані та прогнозувати попит за допомогою штучного інтелекту чи машинного навчання.[3]

Окремо варто поговорити про автоматизацію самого процесу замовлення. Бо часто це виглядає так: хтось із працівників сідає, відкриває пошту або сайт постачальника, вручну заповнює всі поля — що замовити, скільки, куди доставити. Робота нудна, рутинна і легко помилитися. Сучасна система може взяти це на себе: сформувати типове повідомлення і сама його надіслати — швидко, чітко й без зайвих зусиль.

Для малого й середнього бізнесу це питання особливо важливе. Не кожне підприємство може дозволити собі цілу команду людей або дороге готове програмне рішення. У таких випадках варто подумати про створення власного, простого і зручного інструменту, який буде точно "під розмір" - без зайвого, але з усім необхідним. І головне - такий інструмент можна буде поступово розвивати разом із бізнесом.

Особливо актуально це для малого та середнього бізнесу, де кожна хвилина і кожна гривня на рахунку. Не всі компанії мають змогу тримати великий штат працівників або купувати дорогі "коробкові" рішення. У такій ситуації доцільніше створити власне програмне забезпечення - просте, зрозуміле і налаштоване саме під потреби конкретного бізнесу. У таких випадках доцільно створити власну систему, яка буде адаптована під потреби конкретного бізнесу, з можливістю її подальшого розвитку.

Отже, управління замовленнями на складі охоплює багато задач, які раніше виконувались вручну, але сьогодні можуть бути автоматизовані. Саме тому виникає потреба у створенні інтелектуального рішення, яке візьме на себе основні функції керування запасами та замовленнями, мінімізуючи участь людини. Це й буде темою подальшого дослідження.

1.2 Аналіз існуючих систем автоматизації складу

Автоматизація складських процесів є невід’ємною частиною цифрової трансформації торгових підприємств, зокрема в умовах високої конкуренції та зростання обсягів товарообігу. Сучасні системи дозволяють не лише вести точний облік товарів, а й забезпечують оперативне управління запасами, формування замовлень постачальникам, а також інтеграцію з іншими інформаційними платформами — наприклад, інтернет-магазинами та CRM.

На українському ринку представлено низку рішень, що відрізняються функціональністю, масштабованістю та орієнтацією на різні сегменти бізнесу. До таких систем належать:

- **Торгсофт**[4] — програмне забезпечення, орієнтоване на роздрібну та дрібнооптову торгівлю, яке пропонує широкий спектр інструментів для обліку товару, автоматичного поповнення залишків, ведення клієнтської бази та інтеграції з касовим обладнанням;
- **IT-Enterprise**[5] — потужна ERP-платформа вітчизняного виробництва, що включає модулі для управління складом та логістикою. Ця система дозволяє

оптимізувати обробку товарних потоків, впроваджувати WMS-рішення та адаптувати процеси під потреби великих торговельних мереж;

- **УкрСклад**[6] — це українська програма для автоматизації складського та торговельного обліку, яка підходить для малого та середнього бізнесу. Вона забезпечує ведення обліку товарів, формування та друк документів, управління залишками, а також інтеграцію з фіскальним обладнанням і поштовими службами. Система підтримує багатовалютність, роботу з кількома фірмами, а також має можливості для мережевого використання та синхронізації з інтернет-магазинами.

Першою розглянемо систему **Торгсофт**[4]. За інформацією на офіційному сайті Торгсофт – українська компанія-розробник програмного забезпечення для комплексної автоматизації торгівлі та обліку товару. Заснована у Харкові у 2005 році.

Основні можливості Торгсофт включають:

- облік операцій приходу, продажу та повернення товарів;
- маркування товарів штрих-кодами;
- проведення інвентаризації та переоцінки;
- розподіл товарів між кількома торговельними точками;
- зберігання товарів на складах з фіксацією місць зберігання, розрахунком запасів і контролем сезонної продукції;
- контроль мінімальних і максимальних залишків;
- ведення обліку в кількох валютах, підтримка валютного прайс-листа;
- робота з постачальниками: формування замовлень, імпорт прайс-листів, контроль взаєморозрахунків;
- друк різноманітних торговельних документів: фіскальні/нефіскальні чеки, видаткові накладні, комерційні пропозиції, гарантійні талони, звіти (у т.ч. пРРО, x- та z-звіти), інкасації, чеки повернення, звіти продавця тощо (понад 50 звітів);
- інтеграція з поштовими службами — Новою Поштою та Укрпоштою;

- підтримка функцій прокату та ремонту товарів;
- маркетингові інструменти: CRM, база клієнтів, анкетування, масові розсилки, організація акцій і дисконтних програм;
- фінансовий облік: контроль каси, банківських рахунків, боргів, витрат, основних засобів, аналіз рентабельності бізнесу;
- робота з РРО та пРРО;
- підтримка товарів із ПДВ і підакцизною групою;
- сумісність з обладнанням автоматизації: сканери, принтери етикеток, POS-термінали, ТЗД (термінали збору даних);
- мобільний облік товарів на пристроях Android та iOS;
- управління персоналом: облік зарплат, премій, мотивації, контроль часу та дій працівників;
- облік товарів і замовлень для інтернет-магазинів і маркетплейсів;
- засоби безпеки: обмеження доступу, автоматичне резервне копіювання бази на Google Диск, очищення старих даних.

Торгсофт є одним з найпопулярніших рішень для малого та середнього бізнесу, який прагне автоматизувати складські та торговельні процеси.

Серед їх прикладів впровадження були такі.[7]:

- Автоматизація магазину взуття
- Автоматизація магазину одягу
- Облік у магазині меблів
- Виявлення слабких місць бізнесу за допомогою програми обліку

Це список з останніх подкастів компанії. По цьому списку можна зрозуміти, що сервіси надаються приблизно схожим за схемою бізнесам

Судячи, з відгуків[8], середня оцінка яких становить 8,5/10, послуги, що надає Торгсофт, дійсно якісні. На фоні цього треба дослідити ціни, адже саме за них оцінка друга з кінця по величині (8/10).

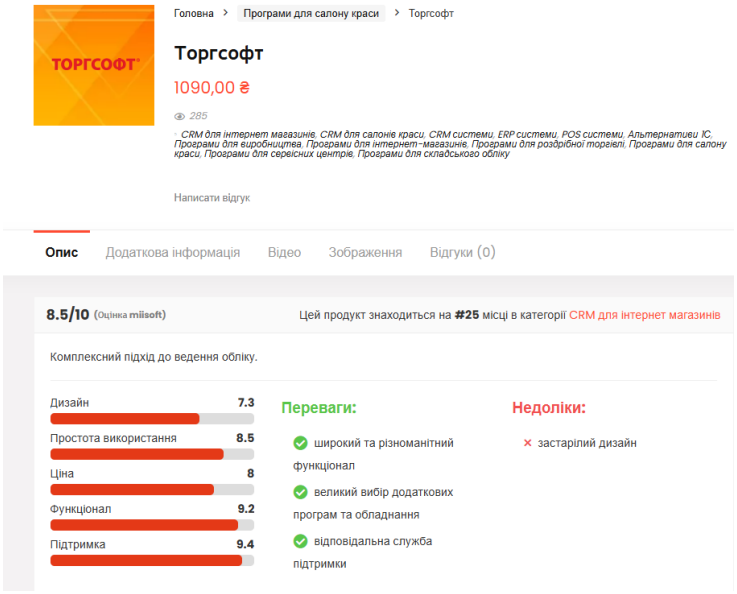


Рисунок 1.1 – Відгуки сервісу Торгсофт.

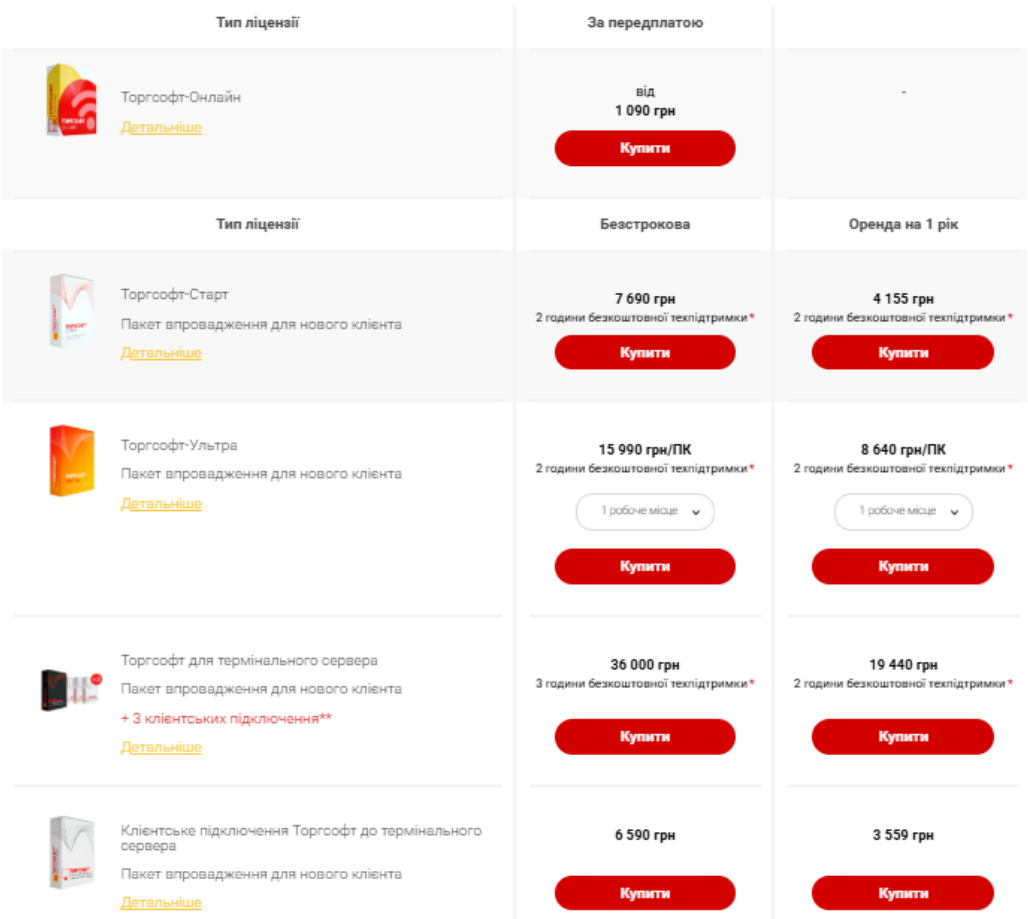


Рисунок 1.2 – Ціни сервісу Торгсофт.

Ціни дійсно виглядять непогано для бізнесів з бюджетом від середнього до високого, малобюджетні компанії, скоріше за все, будуть розраховувати на найдешевший тариф, ціна якого, не враховуючи підключення, становить 7690 грн.

Розглянемо, що постачальник послуг перелічує в функціоналі такого плану.

Рисунок 1.3 – Опис плану “Старт” на сайті Торгсофт.

Як можна побачити, в функції входить облік товарів і фінансів, для більшого провайдер просить більшу ціну.

Тепер розглянемо ERP-систему **IT-Enterprise**[5]. Вона себе позиціонує як єдина українська система, орієнтована на комплексну автоматизацію підприємств або групи підприємств.

Серед плюсів цієї послуги компанія виділяє такі:

- Оптимізація діяльності та бізнес-процесів підприємства
- Підвищення якості та мотивації персоналу завдяки автоматизації
- Надання достовірної інформації для аналізу та прийняття рішень
- Оперативний контроль фінансового стану та собівартості продукції
- Формалізація та контроль усіх бізнес-процесів
- Підвищення пропускної здатності виробництва й рівня обслуговування клієнтів

А перелік рішень і їх прогнозовані ефекти у компанії такі:

- R&D: Скорочення циклу підготовки документації на 40-60%, пришвидшення внесення змін.
- Електронний документообіг і BPM: Узгодження документів у 12 разів швидше, виконання завдань - до 98%.

- Ремонти й обслуговування (EAM): Підвищення ефективності обладнання на 20%, зниження простоїв на 30%.
- Бухгалтерія: Спрощення та пришвидшення звітності.
- Закупівлі та склад: Прискорення закупівель на 20%, зниження запасів на 25%.
- Управління проєктами: Планування та звітність у 2-3 рази швидше, точність прогнозів - в 1,5 раза вища.
- Виробництво: Зростання обсягів до 60%, скорочення термінів випуску на 25%.
- Персонал: Обробка кадрових даних на 30% швидше, зменшення витрат на 10-30%.
- Фінанси: Зменшення оборотних коштів на 10%, автоматизація обліку до 97%.
- Витрати й контролінг: Зменшення відхилень собівартості до 30%.
- Продажі: Зростання продажів на 60%, охоплення аудиторії +400%, кращий контроль етапів.

Уся ця інформація наявна на сайті компанії, що є плюсом, так як вона дає приблизний прогноз щодо ефекту від підключення пакетів послуг. Але є й мінус: на їх веб-сторінках неможливо знайти інформацію про ціни на послуги, замість цього на них наявна велика кнопка “Запит на співпрацю”.

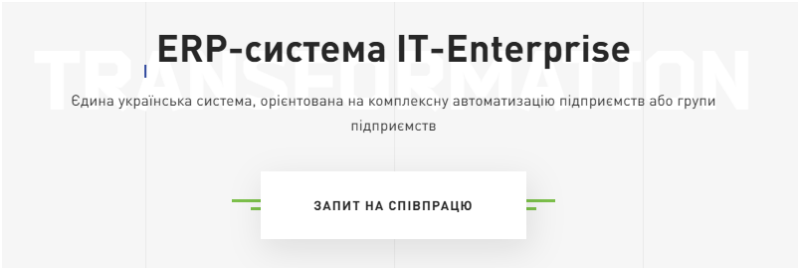


Рисунок 1.4 – Спосіб підключення ERP-системи IT-Enterprise.

На відміну від Торгсофт, IT-Enterprise показує свої приклади впровадження не на подкастах, а на своєму сайті в окремій вкладці у зручному вигляді новин.

Рисунок 1.5 – Останні приклади впровадження на сайті IT-Enterprise[9].

Як можна побачити з цих статей, у компанії були доволі іменити клієнти і співробітники, що говорить про її високий рейтинг довіри, але й також може означати немалі потенціальні витрати на рішення, що вона пропонує.

Компанія має свою сторінку на Фейсбуці, також там можна залишати коментарі та оцінки[10], чим скористались деякі користувачі:

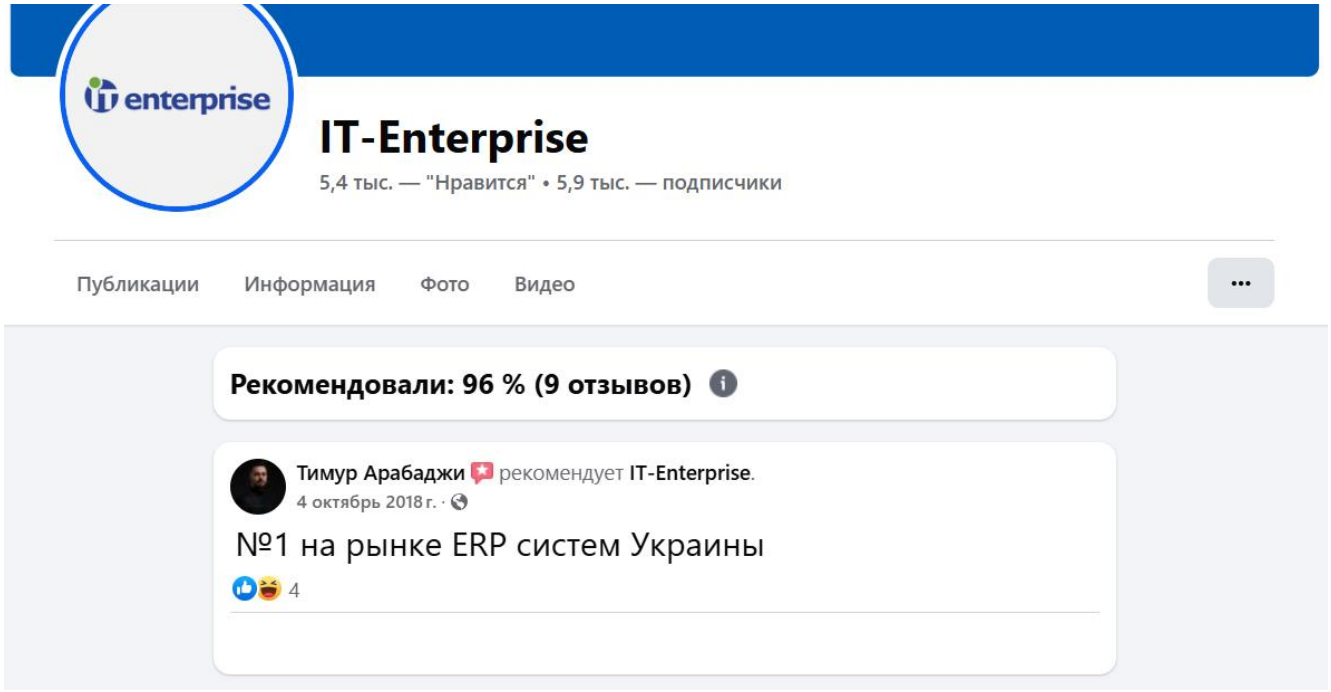


Рисунок 1.6 – Оцінка компанії IT-Enterprise користувачами[10].

Як можна бачити, відгуки про систему дуже позитивні, що означає що в високому бюджеті тяжко буде запропонувати варіант краще.

Якщо пошукати серед новин, можливо знайти деякі ціни на плани в компанії. Оскільки програма, що розробляється, бореться за малий бюджет, має сенс перегляд найдешевшого рішення:

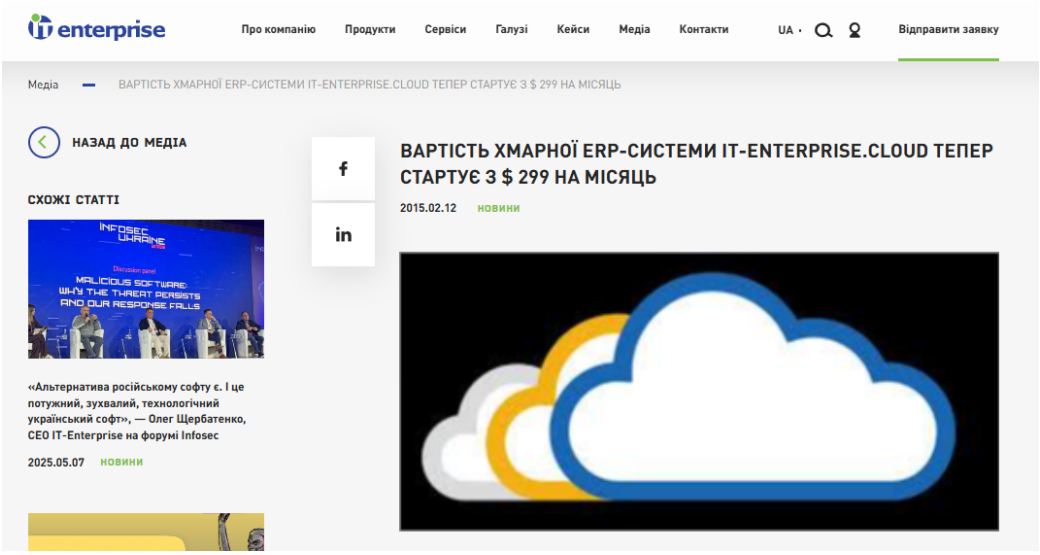


Рисунок 1.7 – Інформація про найдешевший план на сайті IT-Enterprise[11].

З цієї статті[11] можна зробити висновок, що найдешевший план коштує навіть більше ніж у Торгсофт (приблизно на 40%) і пропонує послуги автоматизації логістики, що в цілому робитиме й рішення що розробляється в цій роботі, але за значно менші гроші.

Останнім прикладом буде компанія **УкрСклад**[6]. УкрСклад – програмний продукт, що дозволяє вести складський облік по декількох фірмах і складах. Зокрема, УкрСклад дозволяє:

- Виписувати та друкувати різноманітні документи: накладні, рахунки, акти, касові ордери тощо.
- Вести облік товарів по декількох складах і фірмах.
- Працювати з фіскальними реєстраторами та програмними РРО.
- Інтегруватися з поштовими службами для відправки товарів.
- Підтримувати роботу в мережевому режимі з можливістю віддаленого доступу.
- Синхронізувати дані з інтернет-магазинами.

Для замовлення послуги на сайті доведеться скористатися формою[12], що доволі зручно.

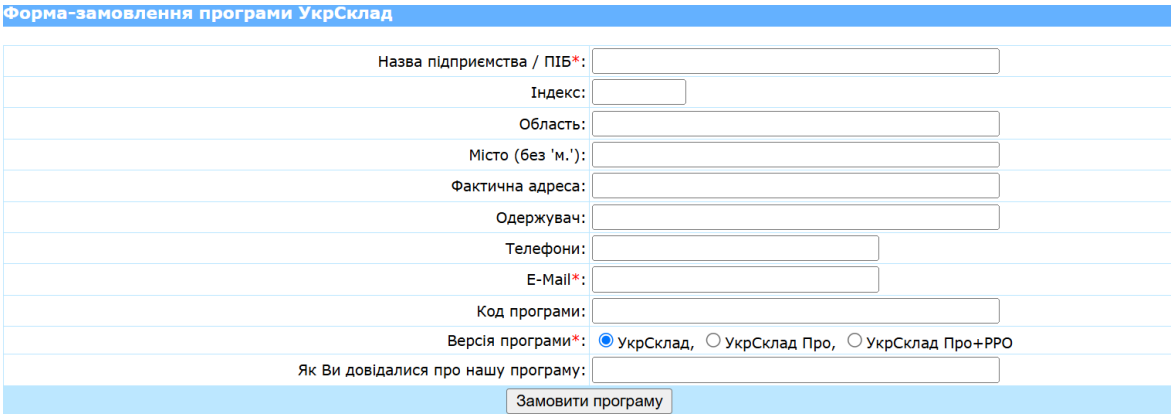


Рисунок 1.8 – Форма для замовлення послуги на сайті УкрСклад.

Переглянемо ціни на послуги:

Програма УкрСклад є платною:

Персональна ліцензія **УкрСклад**
- **995 грн**, ціна першого робочого місця *,
- **500 грн**, ціна кожного додаткового робочого місця *.

Персональна ліцензія **УкрСклад Про**
- **1595 грн**, ціна першого робочого місця *,
- **800 грн**, ціна кожного додаткового робочого місця *.

Рисунок 1.9 – Ціни на послуги УкрСклад[12].

Згідно з інформацією на сайті: *робоче місце - це реєстрація на ОДИН комп'ютер, на нього можна встановити, як мережеву, так і локальну, так і декілька копій програм, все це буде рахуватися як одне робоче місце. Обліковий запис по RDP також є окремим робочим місцем.

За такі ціни компанія пропонує:

УкрСклад Про відрізняється від УкрСклад наявністю додаткових функцій. **В Про-версії доступні:**
- документи **"Вікно касира"** та **"Виробництво"**,
- синхронізація зі службою доставки **"Нова пошта"**,
- оплата через **POS-термінал**,
- **SMS повідомлення клієнтів**,
- синхронізація з програмою **"Мобільний агент"** для Android,
- інтеграція з Віртуальною **АТС Binotel**,
- **функція експорту** податкових накладних, коригування та реєстру ПН **в XML-формат** (для електронної звітності),
- робота з **фіскальними реєстраторами (пРРО/РРО/ФР) (ключ купується окремо)**.

Рисунок 1.10 – Функціонал послуги УкрСклад Про[12].

З якихось причин, на сайті не вказано що робить стандартна програма УкрСклад, що можна віднести до мінусів. Однак, все одно можна побачити, що в описі Про версії все ще нічого не сказано за автоматизацію замовлень. Це той момент, на якому можна зіграти при проєктуванні.

Переглянемо оцінки[13]:

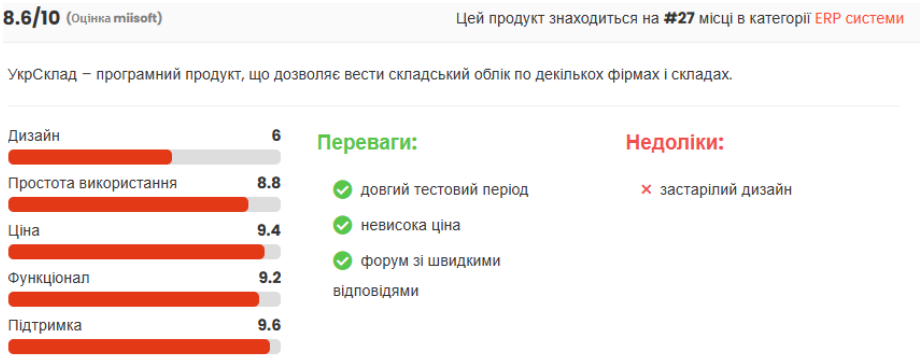
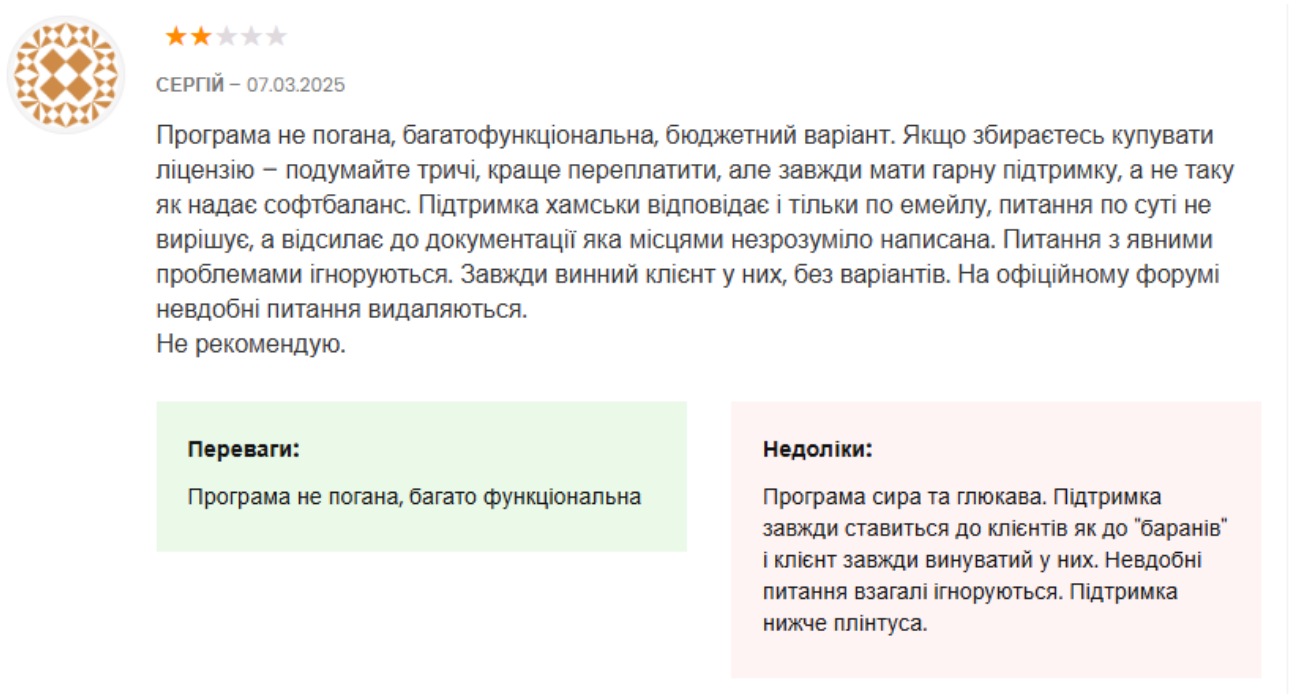


Рисунок 1.11 – Оцінки користувачів УкрСклад.

Як можна бачити, оцінки доволі високі, питання викликає лише застарілий дизайн системи. Судячи з відгуків, система має ряд ще деяких проблем:



★★☆☆☆

СЕРГІЙ – 07.03.2025

Програма не погана, багатофункціональна, бюджетний варіант. Якщо збираєтесь купувати ліцензію – подумайте тричі, краще переплатити, але завжди мати гарну підтримку, а не таку як надає софтбаланс. Підтримка хамськи відповідає і тільки по емейлу, питання по суті не вирішує, а відсилає до документації яка місцями незрозуміло написана. Питання з явними проблемами ігноруються. Завжди винний клієнт у них, без варіантів. На офіційному форумі невдобні питання видаляються.
Не рекомендую.

Переваги:

Програма не погана, багато функціональна

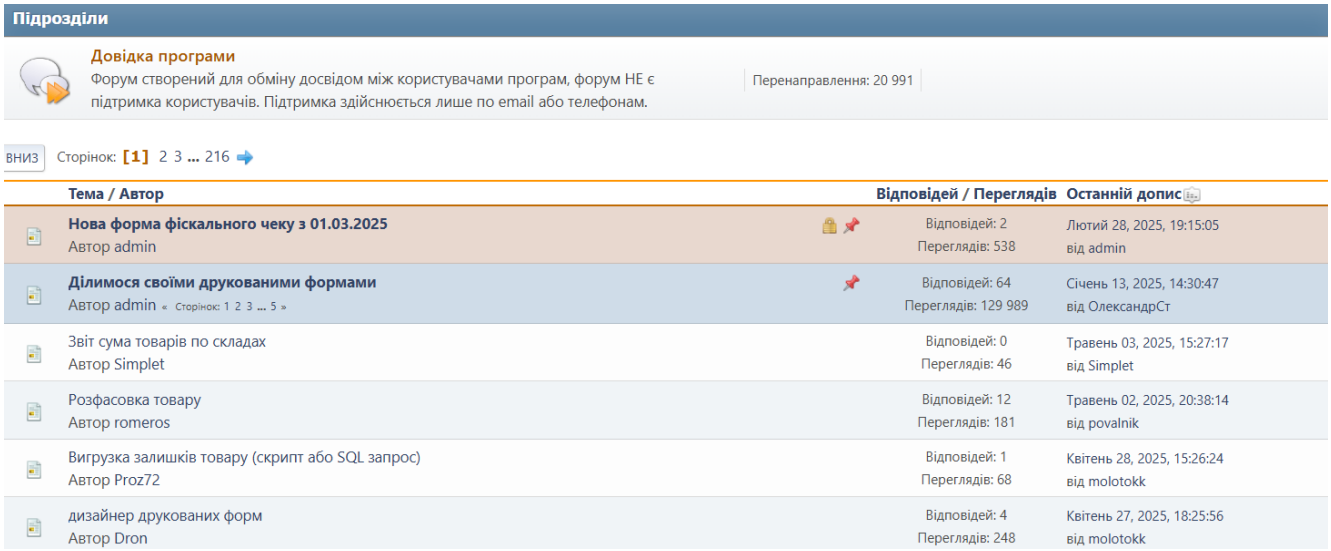
Недоліки:

Програма сира та глюкава. Підтримка завжди ставиться до клієнтів як до "баранів" і клієнт завжди винуватий у них. Невдобні питання взагалі ігноруються. Підтримка нижче плінтуса.

Рисунок 1.12 – Відгук до УкрСклад[13].

Користувач жаліється на повільну роботу програми та погану підтримку сервісу, але все-таки відмічає багатий функціонал та приємну ціну.

Сайт компанії має вкладку з форумом[14], тож користувачі, які з якихось причин могли зіткнутися з поганою роботою підтримки, можуть спробувати вирішити питання завдяки іншим користувачам.



Підрозділи

Довідка програми
Форум створений для обміну досвідом між користувачами програм, форум НЕ є підтримка користувачів. Підтримка здійснюється лише по email або телефонам.
Перенаправлення: 20 991

ВНИЗ Сторінок: [1] 2 3 ... 216

Тема / Автор	Відповідей / Переглядів	Останній допис
Нова форма фіскального чеку з 01.03.2025 Автор admin	Відповідей: 2 Переглядів: 538	Лютий 28, 2025, 19:15:05 від admin
Ділимося своїми друкованими формами Автор admin « Сторінок: 1 2 3 ... 5 »	Відповідей: 64 Переглядів: 129 989	Січень 13, 2025, 14:30:47 від ОлександрСт
Звіт сума товарів по складах Автор Simplet	Відповідей: 0 Переглядів: 46	Травень 03, 2025, 15:27:17 від Simplet
Розфасовка товару Автор romeros	Відповідей: 12 Переглядів: 181	Травень 02, 2025, 20:38:14 від povalnik
Вигрузка залишків товару (скрипт або SQL запрос) Автор Proz72	Відповідей: 1 Переглядів: 68	Квітень 28, 2025, 15:26:24 від molotokk
дизайнер друкованих форм Автор Dron	Відповідей: 4 Переглядів: 248	Квітень 27, 2025, 18:25:56 від molotokk

Рисунок 1.13 – Вкладка “Форум” на сайті УкрСклад.

1.3 Постановка задачі дослідження

Система, що розробляється в цій роботі може взяти найкраще від перелічених раніше варіантів:

- Автоматизацію замовлень
- Облік товарів
- Низькі ціни
- Низьку трудомісткість

На перший час програма буде коштувати дуже символічну суму та буде виступати скоріше в ролі MVP та збирати відгуки про досвід користувачів задля подальшого розвитку.

Отже з процесів, які потрібно реалізувати:

Автоматизовані процеси:

- Перевірка кількості товару на складі - система має регулярно відстежувати залишки продукції та виявляти позиції, що потребують дозамовлення;
- Формування замовлення товару в разі потреби - при досягненні певного порогу кількості, програма автоматично генерує замовлення;
- Контроль параметрів замовлення залежно від чинників - враховується сезонність, попит, статистика продажів;
- Щомісячний облік продажів - збирання та збереження статистичних даних про реалізацію продукції;
- Перенесення інформації про акції (свята) - автоматичне оновлення дат подій на наступний рік.

Неавтоматизовані процеси:

- Початкове налаштування параметрів товару - включає визначення мінімальної кількості для замовлення, назви, категорії та інші характеристики;

- Отримання товару від постачальника - фізичний прийом продукції та її оприбуткування;
- Контроль статусу замовлень - здійснюється менеджером для забезпечення точності постачання та вчасного оновлення інформації.

Таким чином, основна задача полягає в створенні гнучкого та надійного інструменту, який дозволить автоматизувати критично важливі операції підприємства, залишивши за людиною контрольні та адміністративні функції.

1.4 Висновки

У цьому розділі було проведено детальний аналіз предметної області автоматизації складських процесів в умовах сучасної електронної комерції. Встановлено, що зростання обсягів онлайн-продажів зумовлює потребу у впровадженні інтелектуальних систем для автоматизованого управління залишками товарів, формування замовлень та інтеграції з постачальниками.

Розглянуто функціональні можливості та особливості популярних рішень на українському ринку, таких як **Торгсофт**, **IT-Enterprise** та **УкрСклад**. Встановлено, що системи мають високий рівень функціональності, підтримують інтеграцію з інтернет-магазинами та надають інструменти для обліку, аналітики й автоматизації рутинних процесів.

Разом з тим, більшість готових рішень мають певні обмеження щодо гнучкості налаштувань, складності впровадження або високої вартості для малого бізнесу. Це відкриває можливість для розробки власної системи, яка буде орієнтована на потреби невеликої компанії, забезпечуючи автоматизацію ключових етапів замовлення товару при мінімальному залученні людського ресурсу.

Отже, доцільною є розробка власної інтелектуальної комп'ютерної системи автоматизації замовлень, що дозволить зменшити навантаження на персонал і забезпечити ефективне управління складом за обмеженого бюджету.

2 ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ АВТОМАТИЗІЦЇ СКЛАДУ

2.1 Аналіз функцій системи

Для аналізу функцій розроблюваної системи доцільно використати UML-діаграми. UML[15] — це мова моделювання, яка використовується у парадигмі об'єктно-орієнтованого програмування. Вона є невід'ємною частиною уніфікованого процесу розробки програмного забезпечення. UML - мова широкого профілю, яка використовує графічні позначення для створення абстрактної моделі системи, яка називається UML-моделлю. UML був створений для визначення, візуалізації, проєктування й документування в основному програмних систем. UML не є мовою програмування, але в засобах виконання UML-моделей як інтерпретованого коду можлива кодогенерація.

У цьому розділі розглядаються наявні в системі діаграми варіантів використання, розгортання та блок-схеми які детально відображають процес автоматизації поповнення запасів на складі.

Діаграма варіантів використання[16] являє собою графічну модель, що включає в себе набір акторів, варіанти взаємодії (прецеденти), які обмежені рамками системи, зв'язки між акторами та прецедентами, а також відношення між самими прецедентами й узагальненням акторів. Основна ідея такої діаграми полягає в поданні системи через взаємодію зовнішніх учасників із системою за допомогою певних сценаріїв. Кожен такий сценарій описує набір дій, які виконує система у процесі взаємодії з актором. Щоб точно проаналізувати функції системи та те, за що буде відповідати кожен окремий компонент, створимо UML діаграму варіантів використання.

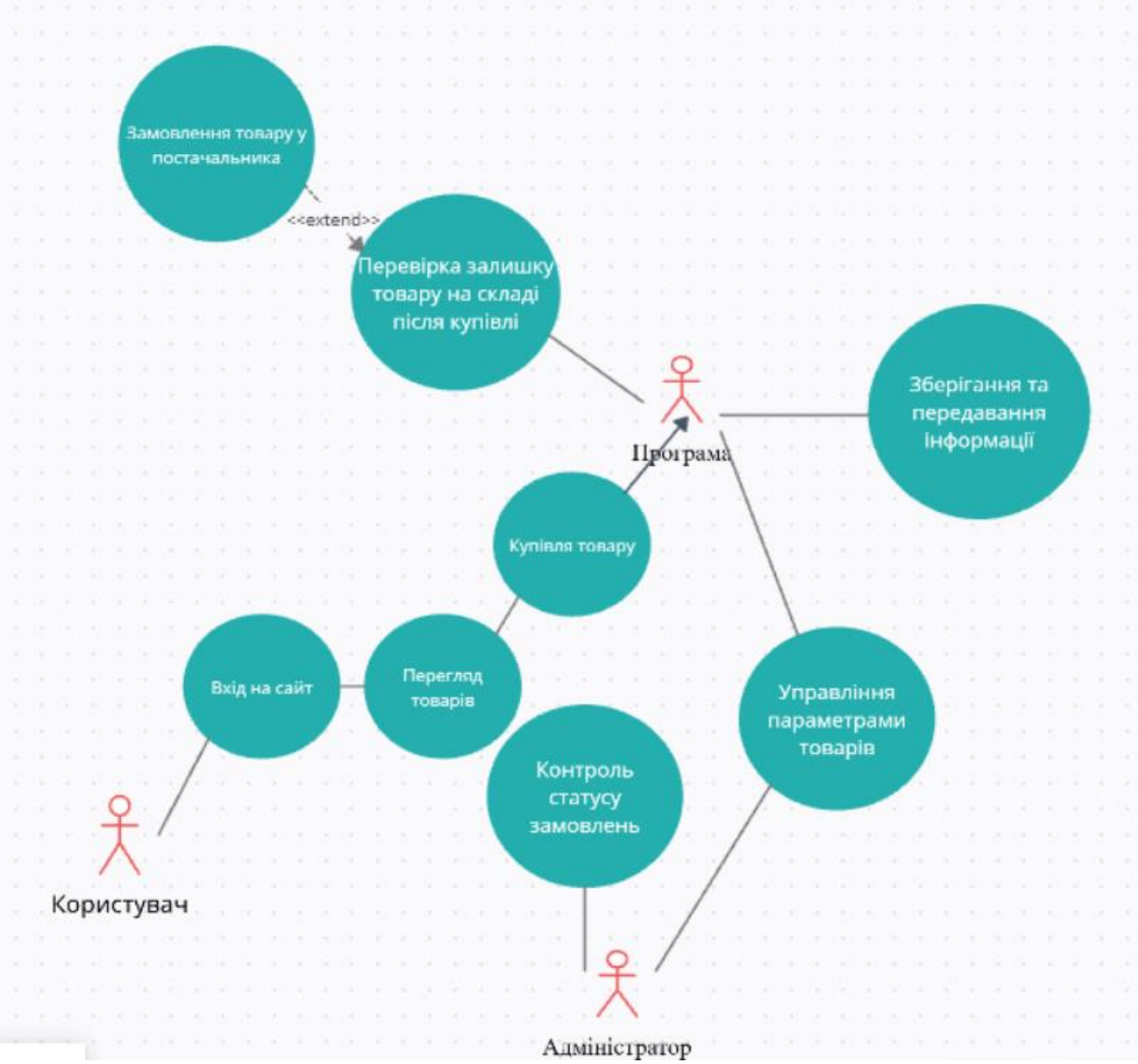


Рисунок 2.1 – UML діаграма варіантів використання.

На рисунку 2.1 зображено діаграму варіантів використання, де учасниками виступають: користувач, сервер, адміністратор, база даних і celery worker-и. Завдяки взаємозв'язкам між цими акторами деталізовано логіку автоматизованого процесу поповнення товару на складі та роль адміністратора в цьому процесі.

2.2 Проєктування архітектури системи

Оскільки система створюється в тому числі й для онлайн-магазинів, усі її процеси можуть відбуватися як на локальному пристрої, так і в мережі Інтернет. Для зберігання й обробки даних, а також виконання основних функцій (згаданих у розділі 1.3), буде використовуватись веб-сервер та база даних.

Деякі задачі (наприклад, перенесення запланованих дат акцій, надсилання листів клієнтам або перевірка календаря на наявність майбутніх подій), будуть виконуватись у воркерах щоб не перевантажувати основну систему. Для цього використовуватимуться спеціальні інструменти, як-от Celery, які допомагають розподіляти навантаження.

Уся система складатиметься з двох основних частин:

- серверної логіки (там відбувається обробка запитів і прийняття рішень),
- та бази даних (де зберігаються всі дані про товари, замовлення тощо).

Частина, яку бачать користувачі (тобто сам сайт магазину), залишиться у повному розпорядженні кожного бізнесу. Це логічно — адже кожен магазин має свій стиль, підхід до клієнтів та власні фішки. А от внутрішня частина — та, яка відповідає за обробку даних — буде єдина для всіх. Вона працює як окрема служба, з якою можна зручно взаємодіяти через API.

Також, будуть надані всі необхідні інтерфейси (ендпоінти), щоб клієнти могли легко підключити систему до свого сайту. Єдине — важливо, щоб структура бази даних відповідала рекомендаціям. Це гарантує, що все працюватиме швидко й без збоїв.

Щоб краще уявити, хто за що відповідає в системі, створено діаграму варіантів використання. А щоб показати, як саме виглядає робота системи в динаміці — діаграму блок-схему. Коли остаточно буде визначено, на яких технологіях усе буде реалізовано, можна буде додати ще й діаграму розгортання.

2.3 Проєктування алгоритмів роботи системи

Тепер потрібно зрозуміти, як покрокова працюватиме система. Для цього на рисунках 2.2 та 2.3 зображено діаграму блок-схему.

Блок-схема[17] - це зручний спосіб зобразити, як працює той чи інший процес або алгоритм. Її суть у тому, щоб передати послідовність дій у вигляді простих фігур, з'єднаних стрілками. Кожна фігура має своє значення: наприклад,

овал означає початок або завершення процесу, прямокутник - якусь дію, а ромб - перевірку умови або вибір напрямку.

Такі схеми часто використовують, коли потрібно розібратися в логіці роботи програми, системи чи навіть бізнес-процесу. Вони допомагають побачити, як усе працює - крок за кроком. Це особливо корисно, коли мова йде про складні речі. В випадку цієї роботи ця діаграма буде кориснішою за діаграму діяльності, через пов'язання потоку роботи клієнта та сервера.

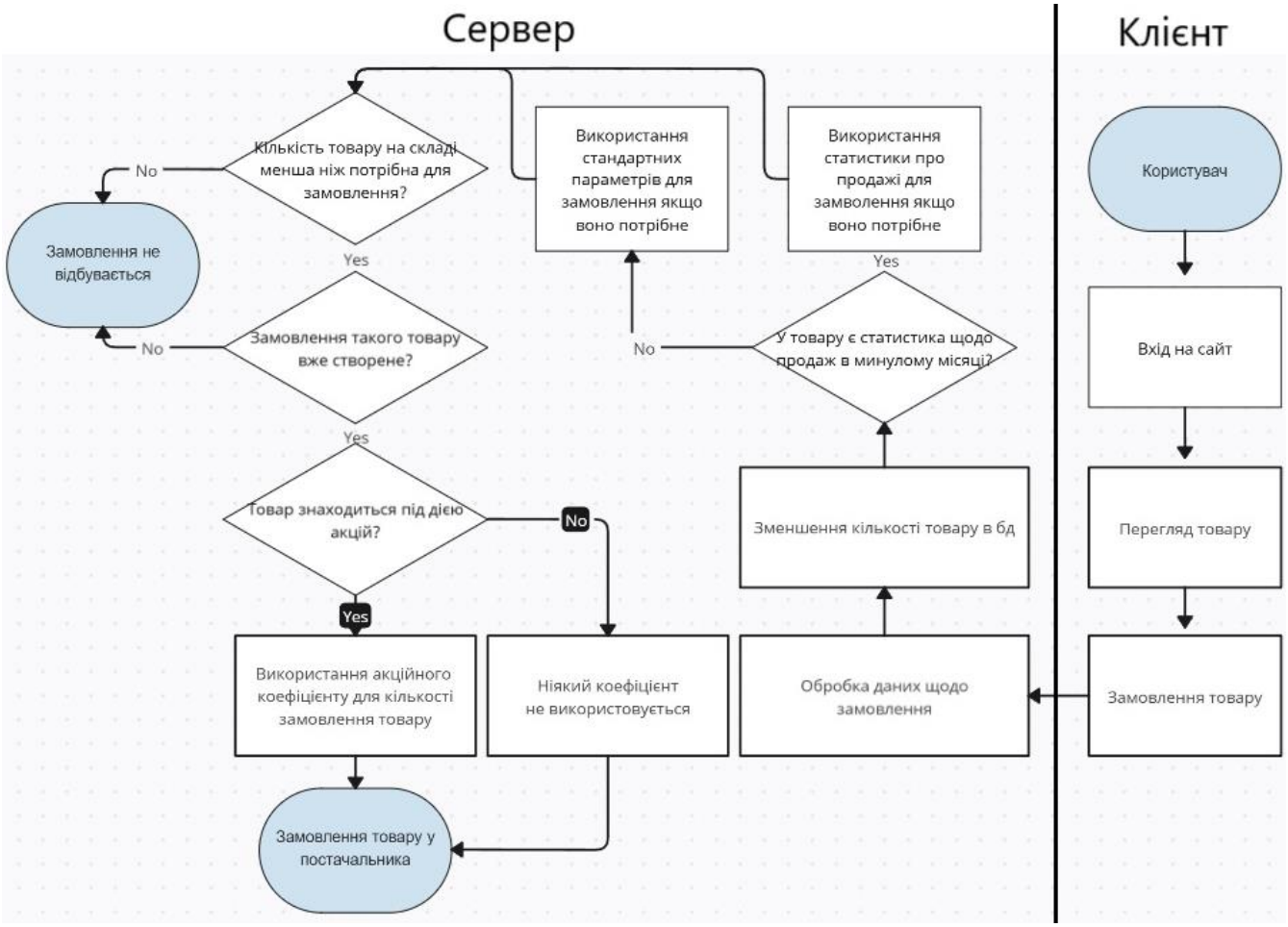


Рисунок 2.2 – Діаграма блок-схема (клієнт).

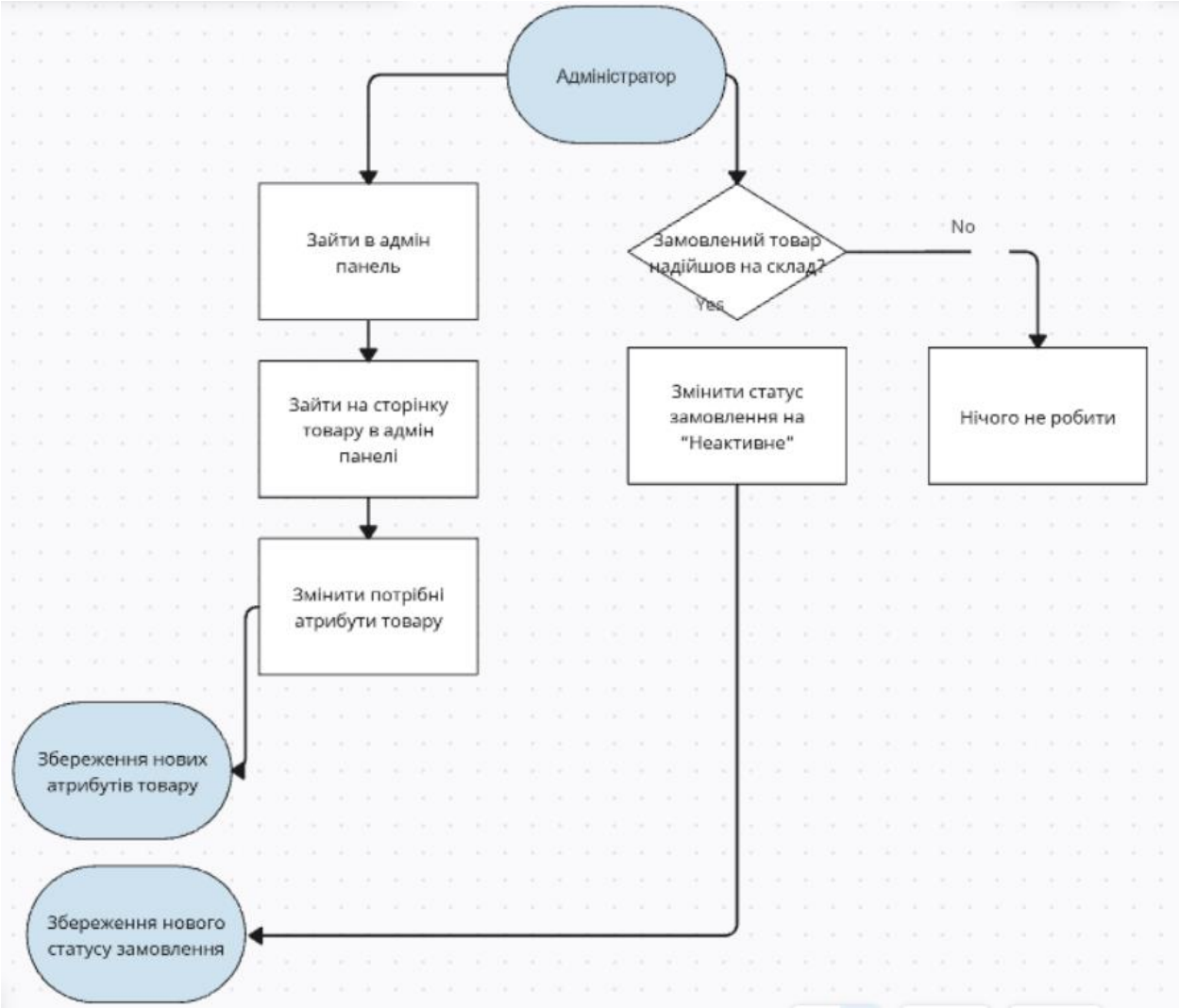


Рисунок 2.3 – Діаграма блок-схема (адміністратор).

Ці діаграми добре описують автоматизований процес від його початку до кінця з урахуванням різних обставин для адміністратора та користувача.

2.4 Вибір засобів створення програмного забезпечення для розробки інтелектуальної комп'ютерної системи автоматизації складу

Зараз автоматизовані системи розвиваються шаленими темпами. Це й не дивно - вони справді полегшують життя не лише для звичайних користувачів, а й для самих розробників. Створювати нове програмне забезпечення стало простіше й зручніше, ніж ще кілька років тому.

Для веб-розробки сьогодні є чимало популярних платформ. Найчастіше згадують Django, ASP. NET і PHP.

Django[18] - це фреймворк, який працює на Python. Його цінують за хорошу безпеку та швидкість. До того ж багато функцій уже є "з коробки", тож не треба кожного разу все писати з нуля. **ASP. NET**[19] - продукт від Microsoft, який часто обирають для створення великих, стабільних систем, особливо в бізнес-середовищі. **PHP**[20] - хоч і доволі стара технологія, але досі дуже популярна. Вона лежить в основі багатьох сайтів і платформ, зокрема WordPress, і дозволяє швидко запускати веб-проєкти. Кожна з цих платформ має свої переваги і застосовується відповідно до потреб конкретного проєкту. **PHP**[20] - це одна з тих мов програмування, які з самого початку створювалися саме для роботи з вебсайтами. Її основна роль - формувати HTML-сторінки на сервері й відправляти вже готовий результат користувачу в браузер. Це зручно, бо все, що "під капотом", залишається прихованим - користувач бачить лише кінцеву версію сторінки, а не сам код. Такий підхід додає безпеки й дозволяє краще контролювати те, що відбувається на сайті.

Хоча PHP з'явилася ще в 90-х, вона й досі тримається в топі мов для веброзробки. І цьому є логічне пояснення: вона проста у вивченні, має зрозумілий синтаксис, працює практично на будь-якому сервері, а головне - підтримується майже всіма хостингами. Ще один важливий плюс - навколо PHP зібралася велика спільнота розробників, які постійно оновлюють документацію, створюють бібліотеки та діляться досвідом. Це дуже допомагає, особливо на початку шляху.

Що ж робить PHP зручною для розробника? Ось кілька ключових моментів:

- Легка робота з базами даних. PHP "дружить" із багатьма СУБД - наприклад, MySQL, PostgreSQL, SQLite, Oracle. Тобто можна підключитися до бази без зайвих складнощів, використовуючи вже готові засоби.
- Зручний синтаксис. Якщо хтось уже мав справу з мовами на кшталт C чи Perl, то з PHP буде легше - багато чого схоже, і це знижує поріг входу.
- Велика швидкість. Попри те, що PHP інтерпретується (а не компілюється), у реальних умовах вона працює досить швидко. Особливо добре себе показує в ситуаціях, коли до сайту одночасно звертається багато користувачів.

- Працює майже скрізь. PHP можна запускати на Windows, Linux, macOS - на будь-якій популярній ОС. Її відкритий код дозволяє змінювати, допрацьовувати та адаптувати під себе.
- Безкоштовна. Ніяких ліцензій чи оплат - усе доступно одразу. Це робить PHP чудовим варіантом і для студентських проєктів, і для стартапів, і для більших систем.

ASP. NET[19] - це сучасна технологія від Microsoft, яка допомагає створювати сайти та онлайн-сервіси. Якщо говорити простіше, вона дозволяє розробникам будувати зручні вебзастосунки - від невеликих сайтів до потужних корпоративних платформ.

Колись була така технологія ASP (Active Server Pages) - вона працювала, але мала багато обмежень. ASP. NET - це її "наступне покоління", тільки значно потужніше, швидше та гнучкіше. Вона працює на платформі .NET, яка дає розробникам більше можливостей і сучасних інструментів.

Основні переваги ASP. NET:

- Швидкість. Коли користувач заходить на сайт, ASP. NET обробляє все на сервері та "видає" вже готовий результат. Це означає менше очікування і більше швидкості.
- Зручність. Microsoft подбала про розробників - у технології вже є безліч готових компонентів. Наприклад, не потрібно вигадувати велосипед, коли хочеш зробити форму входу на сайт чи систему реєстрації.
- Підтримка різних мов. Хтось любить C#, хтось працює з Visual Basic. ASP. NET підтримує різні варіанти, тож розробник зможе писати так, як йому зручно.
- Логіка і зовнішній вигляд - окремо. Це дуже зручно: дизайнери працюють над тим, як виглядає сайт, а програмісти - над тим, як він працює. Ніхто нікому не заважає, і підтримувати код значно легше.

- Гнучке налаштування. Наприклад, якщо потрібно, щоб певні сторінки оброблялись особливим чином - немає проблем. ASP. NET дозволяє це зробити.

Усе це робить ASP. NET чудовим варіантом для тих, хто хоче створити сучасний, швидкий і надійний вебсайт або сервіс. А ще важливо, що ця технологія тісно пов'язана з іншими інструментами Microsoft - тож усе працює разом як одна злагоджена система .

Django[18] - це потужний і зручний фреймворк для створення веб-сайтів на Python. Його назва походить від імені джазового гітариста Джанго Рейнхардта. Це відображає музичні смаки одного з авторів цього проєкту.

Що робить Django особливим? Перш за все, він побудований за принципом розділення коду на окремі, незалежні частини. Це означає, що ваш сайт чи веб-додаток легше підтримувати, розвивати та масштабувати. Такий підхід вирізняє Django серед подібних фреймворків, наприклад, Ruby on Rails.

В основі архітектури Django лежить шаблон проєктування, схожий на MVC (Model-View-Controller), але з невеликими змінами. У Django він називається MTV - Model, Template, View:

- Model описує, як дані зберігаються в базі;
- View відповідає за обробку запитів (тобто, що робити, коли користувач щось просить);
- Template визначає, як виглядатиме сторінка для користувача.

Django спочатку розробляли для створення новинних порталів, тому в ньому багато зручностей для швидкого запуску сайту. Наприклад, вже "з коробки" йде потужна адмін-панель, яку не потрібно писати з нуля. Вона дозволяє керувати контентом, користувачами, правами доступу, а також вести історію змін.

Переваги Django:

- Зручна робота з базами даних (ORM): Ви описуєте структуру даних у вигляді моделей, а Django сам "перекладає" це на мову SQL. Якщо потрібно - можна вручну дописати складні запити.

- Автоматична адмінка: Як тільки ви створюєте модель - Django може згенерувати зручну панель для додавання, редагування чи видалення записів. Дуже економить час.
- Гнучка система маршрутів: Ви самі вирішуєте, як виглядатимуть URL-адреси вашого сайту. Хочете /blog/2025/05/ - без проблем. Все налаштовується у файлі urls.py.
- Шаблони для сторінок: Django має свою мову шаблонів - просту й безпечну. Навіть якщо ви не програміст, ви зможете змінювати зовнішній вигляд сторінок без страху зламати логіку сайту.
- Кешування: Щоб сайт працював швидше, ви можете зберігати частини сторінок у кеші. Django дозволяє легко налаштувати кешування - від окремих фрагментів до цілих сторінок.
- Підтримка багатомовності: Якщо ви плануєте створити сайт для різних мов, Django вас підтримає. Він дозволяє додавати переклади, і навіть якщо для якогось тексту переклад відсутній - відобразиться текст за замовчуванням, без помилок.

У підсумку, Django - це надійний інструмент для розробників, який поєднує в собі зручність, гнучкість і безліч готових рішень. Саме тому він підходить як для невеликих проєктів, так і для великих комерційних веб-додатків. І найголовніше - у Django дуже активна спільнота, яка постійно вдосконалює фреймворк, додає нові можливості та допомагає новачкам. Цей фреймворк в роботі й буде використовуватись.

.

2.4.1 Вибір мови програмування

У процесі розробки інтелектуальної автоматизованої системи управління складом було вирішено використати мову програмування Python. Вона є однією з найпопулярніших мов у світі завдяки своїм простим синтаксису, гнучкості та широких можливостях для розробки сучасного програмного забезпечення.

Python[21] - мова високого рівня, яка підтримує кілька парадигм програмування: об'єктно-орієнтовану, процедурну та функціональну. Завдяки цьому розробники можуть обирати найбільш зручний підхід для вирішення задач. Важливо відмітити, що Python має динамічну типізацію та інтерпретується, а не компілюється, як велика частина інших мов програмування, що значно пришвидшує процес створення і тестування програмного коду, але сповільнює його роботу, що в випадку розроблюваної системи не критично.

Однією з ключових переваг Python є його лаконічний і простий синтаксис, завдяки якому навіть складні алгоритми можна реалізувати зрозуміло та компактно. Це особливо важливо для командної роботи над проєктом і подальшої підтримки коду. Крім того, Python - це кросплатформена мова, тому створені програми можна без змін запускати на різних операційних системах, що робить її універсальним інструментом у розробці.

Python також має велику стандартну бібліотеку, яка включає велику кількість готових модулів для роботи з файлами, мережею, базами даних, графікою, математичними обчисленнями і багато іншого. Це дозволяє не витрачати час на створення власних підпрограм, а використовувати вже перевірені рішення. Додатково Python підтримує інтерактивний режим, що зручно для швидкого тестування окремих фрагментів коду або налагодження.

Також перевагою мови є її відкритість. Це проєкт із відкритим вихідним кодом, тому кожен охочий може долучитися до його розвитку, вивчати внутрішню логіку роботи або створювати власні розширення. До того ж, мову легко інтегрувати з іншими програмами - зокрема, Python часто використовується як скриптова мова в більш складних системах, або як зв'язуюча ланка між модулями, написаними іншими мовами (наприклад, C чи C++).

Загалом, Python добре підходить як для створення невеликих додатків, так і для проєктування повноцінних програмних систем. У контексті автоматизації складських процесів він надає всі необхідні інструменти - від обробки даних до побудови складної логіки взаємодії між елементами системи. Усе необхідне для

початку роботи з Python - інтерпретатор, документація, інструменти та бібліотеки - можна знайти на офіційному сайті www.python.org.

2.4.2 Вибір системи управління базами даних (СУБД)

Для зберігання даних в роботі було обрано **PostgreSQL**[22]. Це потужна система керування базами даних, яку часто обирають як альтернативу популярним платним продуктам на кшталт Oracle чи Microsoft SQL Server. Але на відміну від них, PostgreSQL - повністю безкоштовна, з відкритим вихідним кодом. Її розвиває спільнота ентузіастів та компаній з усього світу, а не одна централізована компанія. Завдяки цьому PostgreSQL швидко розвивається й отримує нові можливості.

Ця БД також підтримує об'єктно-орієнтовані фішки, що дає більше гнучкості в проєктах. Її ядро написане на мові C, а саму систему зазвичай встановлюють із вихідного коду - хоча зараз є й зручні готові інсталятори для різних операційних систем.

Основні переваги PostgreSQL:

- Клієнт-серверна архітектура. PostgreSQL працює за знайомим принципом: є сервер, який обслуговує запити клієнтів. Це дозволяє будувати масштабовані рішення - від невеликих проєктів до систем, які обробляють великі обсяги даних у режимі 24/7.
- Стабільна робота з багатьма користувачами. Завдяки механізму багатоверсійності (MVCC), PostgreSQL може одночасно виконувати багато транзакцій без "гальм" і конфліктів. Це важливо для стабільної роботи в багатокористувацькому середовищі. До того ж, вона дотримується ACID-принципів - тобто забезпечує надійність і цілісність даних.
- Широкі можливості. PostgreSQL підтримує не лише стандартний SQL, а й багато додаткових функцій: роботу з JSON, тригери, процедури, представлення, повнотекстовий пошук тощо. Це робить її дуже гнучкою та зручною для розробників.

- Гнучке встановлення. Хоча PostgreSQL працює як окремий сервер, її досить просто встановити на будь-яку сучасну ОС - Windows, Linux або macOS. А головне - після встановлення її можна точно налаштувати під свої потреби: виділення пам'яті, обробка запитів, кешування тощо.
- Висока продуктивність. PostgreSQL добре справляється з великими масивами даних і складними запитами. Вона має продуману систему індексації та оптимізації, що дозволяє досягти високої швидкості навіть за великого навантаження .

2.4.3 Розгортання проєкту

Тепер, коли ми вже визначились зі стеком технологій, можемо перейти до побудови діаграми розгортання[23].

Ця діаграма (deployment diagram) в UML використовується для того, щоб показати, як система поводиться саме під час виконання. Вона дає змогу наочно уявити, які обчислювальні вузли (сервери, пристрої тощо) беруть участь у роботі застосунку, які програмні компоненти на них запускаються, а також які між ними відбуваються взаємодії.

У випадку розроблюваного застосунку це дозволяє відобразити, як клієнтська частина взаємодіє з Docker-хостом, на якому розгорнуто серверну логіку. Це виявилось особливо зручно, бо Docker дозволяє не заморочуватись з ручним налаштуванням середовища. Все запускається всередині контейнера, і при цьому неважливо, яка саме операційна система - все працює так само.

Детальний опис переваг Docker та особливості його налаштування будуть розглянуті у наступному розділі

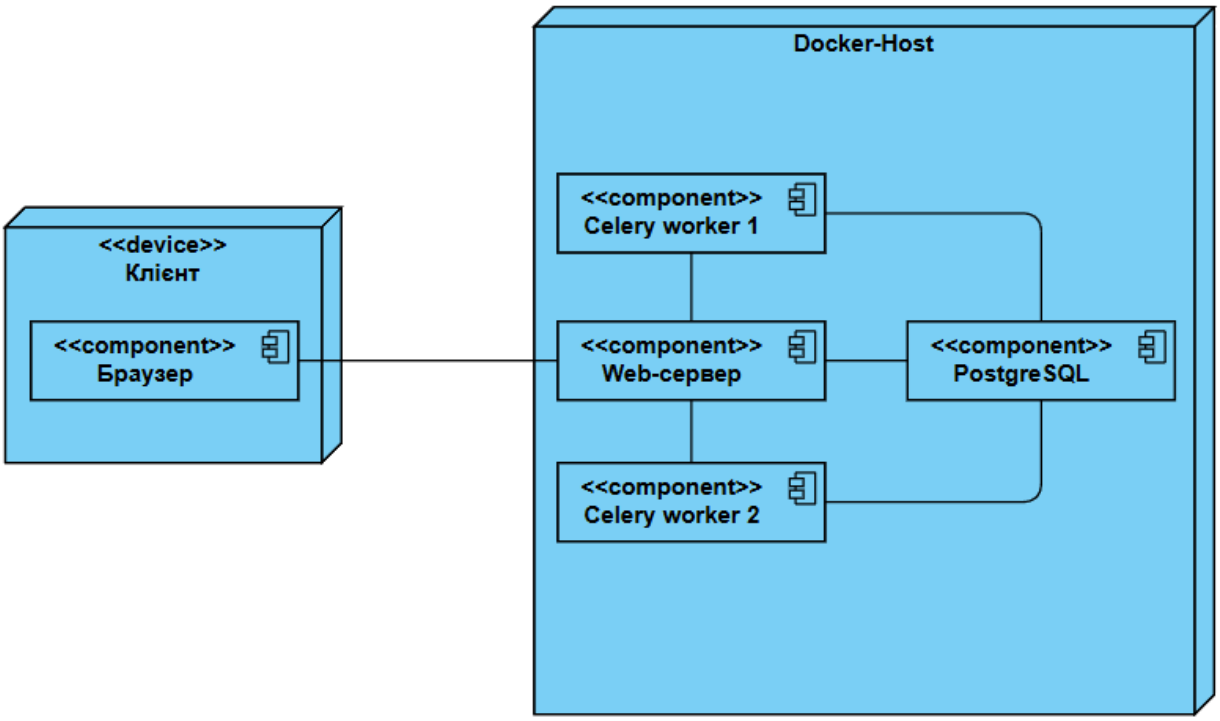


Рисунок 2.4 – UML діаграма розгортання.

Діаграма розгортання відображає обчислювальні вузли, задіяні під час виконання програми, а також компоненти й об'єкти, які функціонують на цих вузлах.

2.5 Висновки

У цьому розділі було створено UML-діаграми варіантів використання, розгортання та діаграму блок-схему які детально відображають процес пошуку транспортних квитків. Також здійснено вибір програмного забезпечення, мови програмування, системи управління базами даних та фреймворків, необхідних для розробки автоматизованої системи пошуку квитків.

3 РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ АВТОМАТИЗІЦІЇ СКЛАДУ

3.1 Написання коду

Процес розробки в налаштованому Django проєкті починається з написання моделей. Моделі розроблюваного застосунку "shop" виглядають так:

```python
from django.core.exceptions import ValidationError
from django.db import models
from django.utils import timezone
from replenishment.tasks import stock_task

class Category(models.Model):
    name = models.CharField(verbose_name='Назва кареторії', unique=True, blank=False, null=False, max_length=50)
    slug = models.SlugField(default='', null=False, db_index=True)

    def __str__(self):
        return f'Категорія: {self.name}'


class ProductType(models.Model):
    name = models.CharField(verbose_name='Назва типу продукту', unique=True, blank=False, null=False, max_length=50)
    category = models.ForeignKey(Category, on_delete=models.CASCADE, blank=False, null=False,
    verbose_name='Категорія', related_name='producttype')
    slug = models.SlugField(default='', null=False, db_index=True)

```

Рисунок 3.1 – Моделі Category та ProductType.

Через те, що архітектура магазину, яка використовується в розробці, використовує парадигму EAV (Entity-Attribute-Value), база даних також містить моделі Property та PropertyName:

```python
class PropertyName(models.Model):
    name = models.CharField(verbose_name='Назва характеристики', unique=True, blank=False, null=False, max_length=200)
    producttype = models.ManyToManyField(ProductType, related_name='propertyname', verbose_name='Тип товару')
    type_field = models.CharField(max_length=1, choices=typeslist, verbose_name='Тип характеристики')

    def __str__(self):
        return f'Характеристика: {self.name}'


class Property(models.Model):


    propertyname = models.ForeignKey(PropertyName, on_delete=models.CASCADE, verbose_name='Назва характеристики',
    related_name='property')
    product = models.ForeignKey(Product, on_delete=models.CASCADE, verbose_name='Назва товару',
                                related_name='property')
    dynamic_type_value = models.CharField('Значення',max_length=100, null=True, blank=True)

    class Meta:
        unique_together = (('propertyname', 'product'),)


    def clean(self):
        if self.propertyname not in self.product.producttype.propertyname.all():
            raise ValidationError("Значення поля `propertyname` не належить до списку `propertyname` для типу продукту")

        return super().clean()
```

Рисунок 3.2 – Моделі Property, PropertyName.

Але, безперечно, найважливішою моделлю є Product, поля якої містять потрібну для контролю кількості товару на складі інформацію.

```python
class Product(models.Model):

    QUANTCATS = [
        (0, 5),
        (1, 10),
        (2, 100),
        (3, 1000)
    ]

    def content_file_name(instance, filename):
        return '/'.join([f'images/products/{instance.producttype.name}', instance.name[0], filename])

    name = models.CharField(verbose_name='Назва товару', unique=True, blank=False, null=False, max_length=200)
    price = models.DecimalField(verbose_name='Ціна товару',max_digits=8, decimal_places=2, blank=False, null=False)
    slug = models.SlugField(default='', null=False, db_index=True)
    description = models.TextField('Опис', blank=True)
    quantity = models.IntegerField('Кількість')
    created = models.DateTimeField('Дата і час створення товару', auto_now_add=True)
    updated = models.DateTimeField('Дата і час останньої зміни товару', auto_now_add=True)
    image = models.ImageField('Зображення', upload_to=content_file_name, default='defaults/pc.png', blank=True)
    category = models.ForeignKey(Category, on_delete=models.SET_NULL, null=True, verbose_name='Категорія', related_name='product')
    producttype = models.ForeignKey(ProductType, on_delete=models.SET_NULL, null=True, verbose_name='Тип продукту', related_name='product')
    min_quantity = models.IntegerField('Мінімальна кількість для замовлення', choices=QUANTCATS)
    supplier_email = models.EmailField('Емейл постачальника', blank=True)
    supplier_product_name = models.CharField('Назва продукту у постачальника', blank=True)
    last_monthly_sales_check = models.DateField('Дата і час останньої перевірки кількості продаж', default=timezone.now)
    current_monthly_sales = models.IntegerField('Нинішня кількість продаж за місяць', default=0)
    last_monthly_sales = models.IntegerField('Кількість продаж за минулий місяць', default=0)
```

Рисунок 3.3 – Поля моделі Product.

План проєкту передбачає замовлення у різних постачальників, настільки, що обмежуватись одним сайтом з можливістю оптової закупівлі не потрібно. В цьому випадку гарним варіантом є відправка email дистриб’ютору з вказанням потрібного товару та його кількості.

Поля, які грають роль в поповненні складу:

- Через те, що єдиним полем, що не описане через дескриптори є QUANTCATS, воно одне являється статичним, і представляє собою список, значення якого використовуються в атрибуті min_quantity, який і містить ті значення, що будуть використовуватись при замовленні товарів до того моменту, коли збереться перша місячна статистка.
- supplier_email - поле для збереження електронної адреси постачальника. Значення вводиться вручну оператором під час додавання нового товару до бази.
- supplier_product_name - використовується для фіксації назви товару згідно з інформацією постачальника, у разі відмінності від внутрішньої назви. Також заповнюється вручну під час створення товарної позиції.

- last_monthly_sales_check - зберігає дату останньої перевірки щомісячних продажів. Використовується як контрольна точка для запуску наступної перевірки через 30 днів. Значення встановлюється автоматично під час створення запису.
- current_monthly_sales - містить кількість продажів за поточний місяць. При створенні нового запису встановлюється значення за замовчуванням - 0. Після кожної перевірки обнуляється, а попереднє значення передається до відповідного поля архіву.
- last_monthly_sales - зберігає дані про кількість продажів за попередній місяць. Використовується при формуванні замовлення нової партії товару та при визначенні потреби поповнення.

Також на запит на придбання впливають моделі:

- Holiday - модель, що зберігає інформацію про святкові дні, включаючи їх назви та відповідні дати проведення.
- HolidayDiscount - модель, яка містить дані про розмір знижок, що діють у святкові дні, а також перелік категорій товарів, на які поширюються ці знижки.

```
126 class Holiday(models.Model):
127     name = models.CharField('Назва свята', blank=False, default='Unnamed')
128     date_start = models.DateField('Дата початку свята', blank=True)
129     date_end = models.DateField('Дата кінця свята', blank=True)
130 
131     def __str__(self):
132         return self.name
133 
134 
135 class HolidayDiscount(models.Model):
136     holiday = models.ForeignKey(Holiday, verbose_name='Свято', related_name='discounts', on_delete=models.CASCADE)
137     category = models.ForeignKey(Category, verbose_name='Категорія товарів для знижок', related_name='holiday_discounts',
138                                  on_delete=models.CASCADE)
139     discount_value = models.DecimalField('Знижка', max_digits=2, decimal_places=0, default=0)
140 
141     class Meta:
142         unique_together = ['holiday', 'category']
143 
144     def __str__(self):
145         return f'Свято: {self.holiday.name}, Категорія: {self.category.name}, Знижка: {self.discount_value}%'
```

Рисунок 3.4 – Моделі Holiday та HolidayDiscount.

Хоч ці таблиці і містять інформацію про знижки, але кількість товару при замовленні збільшується пропорційно до цієї знижки, бо приблизно на стільки ж прогнозується збільшення попиту відповідно.

Текст замовлення виглядатиме так:

“Шановна команда {Ім’я постачальика},

Мене звати Іванов Сергій, я представляю компанію VNTUTM. Ми зацікавлені у закупівлі наступного товару:

{Назва товару у постачальника} у кількості {(**last_monthly_sales** , якщо є, інакше - **min_quantity***2)*коефіцієнт від знижки} шт.

Термін доставки: 5 робочих днів.

Будь ласка, підтвердьте можливість виконання цього замовлення,

а також надішліть рахунок для оплати. Якщо потрібні додаткові уточнення,

зв’яжіться зі мною за контактами нижче.

Контактні дані:

1) Email: {Емейл замвоника}

2) Номер телефону: {Номер замовника}

Дякую за співпрацю! Очікую на вашу відповідь.”

Згідно з діаграмою блок-схемою клієнта (Рисунок 2.2), замовлення у постачальника буде створюватись після придбання клієнтом товару на сайті та перевірці, чи залишилось товару на складі менше ніж значення поля min_quantity, або last_monthly_sales (у разі наявності значення у нього). Досягається це переписуванням методу save в моделі Product, а точніше його доповненням:

```
56      def save(self, *args, **kwargs):
57          if self.id and self.quantity<=(self.last_monthly_sales if self.last_monthly_sales else
58          self.get_min_quantity_display()) and \
59          not OrderFoReplenishment.objects.filter(product__id=self.id, active=True).exists():
60              stock_task.delay(self.id)
61          return super().save(*args, **kwargs)
```

Рисунок 3.5 – Оновлений метод save моделі Product.

stock_task – це Celery таска, яка прописана в файлі tasks.py. Але щоб її запустити, треба ініціалізувати воркер, що робиться в файлі celery_app.py. В цьому файлі ініціалізується одразу 2 воркери:

```
12  def workerconfig(name: str, num: int):
13      app = Celery(name)
14      app.config_from_object('django.conf:settings')
15      app.conf.broker_url = settings.CELERY_BROKER_URL[num]
16      app.autodiscover_tasks()
17      return app
18
19  app = workerconfig('stocks', 0)
20  app2 = workerconfig('sales_checking', 1)
```

Рисунок 3.6 - Функція workerconfig для ініціалізації Celery воркеру, створення 2 воркерів за допомогою її виклику.

Функція workerconfig має параметри name та num, котрі використовує при створенні та налаштуванні екземпляру класу Celery. name це просто ім’я для воркеру, а num ключ для вибору бази даних Redis, яка буде керувати чергами тасок.

```
132  CELERY_BROKER_URL = ['redis://redis:6379/0', 'redis://redis:6379/1']
```

Рисунок 3.7 – Список посилань на бази даних Redis в файлі settings.py.

Це зроблено для того, щоб таски першого виду та другого не заважали один одному, так як обидва завдання одночасно можуть викликатись багато разів. stock_task використовує 0 БД. Для відправки Email задіяний SMTP-сервер компанії Google.

```
134  EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
135  EMAIL_HOST = 'smtp.gmail.com'
136  EMAIL_PORT = 587
137  EMAIL_USE_TLS = True
138  EMAIL_HOST_USER = os.getenv('EMAIL_USER')
139  EMAIL_HOST_PASSWORD = os.getenv('EMAIL_PASS')
140  DATA_UPLOAD_MAX_MEMORY_SIZE = 10485760
```

Рисунок 3.8 – Налаштування під’єднання до SMTP-серверу Google в файлі settings.py.

```python
@app.task
def stock_task(id):
    from shop.models import Product, OrderFoReplenishment
    product = Product.objects.get(id=id)
    coefficient = get_coefficient(product=product)
    subject = f'Замовленян товару VNTUTM'
    email_user = os.getenv("EMAIL_USER")
    supplier_name = product.supplier_email.split('@')[0]
    quantity = math.floor((product.last_monthly_sales if product.last_monthly_sales else
    product.get_min_quantity_display()) * 2 * coefficient)
    message = f"""
    Шановна команда {supplier_name},

    Мене звати Іванов Сергій, я представляю компанію VNTUTM. Ми зацікавлені у закупівлі наступного товару:
    {product.supplier_product_name} у кількості {quantity} шт.

    **Термін доставки:** 5 робочих днів.
    Будь ласка, підтвердьте можливість виконання цього замовлення,
    а також надішліть рахунок для оплати. Якщо потрібні додаткові уточнення,
    зв'яжіться зі мною за контактами нижче.

    **Контактні дані:**
    1) Email: {email_user}
    2) Номер телефону: 3592953267235

    Дякую за співпрацю! Очікую на вашу відповідь.
    """
    send_mail(subject=subject, message=message, from_email=email_user,recipient_list=[product.supplier_email], fail_silently=False)
    OrderFoReplenishment.objects.create(product=product, quantity=quantity)
```

Рисунок 3.9 – Функція stock_task.

Для обчислення змінної coefficient було написано та використано функцію get_coefficient:

```python
def get_coefficient(product):
    from shop.models import HolidayDiscount

    today = timezone.now().date()

    active_holidays = HolidayDiscount.objects.filter(
        Q(holiday__date_start__lte=today) & Q(holiday__date_end__gte=today),
        category=product.category
    ).order_by('-discount_value')

    if active_holidays.exists():
        discount = active_holidays.first().discount_value
        return 1 + discount/100
    return 1
```

Рисунок 3.10 – Функція get_coefficient.

Безпосередньо відправка Email-у відбувалася завдяки використанню вбудованої в Django функції send_mail, а після цього в базі даних створився запис про замовлення до таблиці order_for_replenishment, модель якої виглядає так:

```python
class OrderForReplenishment(models.Model):

    product = models.ForeignKey(Product, on_delete=models.DO_NOTHING, verbose_name='Продукт', related_name='order')
    active = models.BooleanField('Активний', default=True)
    quantity = models.IntegerField('Кількість', null=False)


    def __str__(self):
        return f'Товар: {self.product.name}, кількість: {self.quantity}'
```

Рисунок 3.11 – Модель OrderForReplenishment.

Для роботи другого воркера потрібно запустити Celery beat, який вже розписано в docker-compose. Старт сервісу відбувається за допомогою конфігурації beat_schedule, в який передається шлях до виконуваної фукнції та проміжок часу між виконаннями.

```python
app2.conf.beat_schedule = {
    "test": {
        "task": 'replenishment.tasks.check_and_run_function',
        "schedule": timedelta(days=1),
    },
}
```

Рисунок 3.12 – Конфігурація beat_schedule в celery_app.py.

Функціонал другого воркера полягає у переведенні всіх свят на рік вперед якщо дата 1 січня та перевірці всіх товарів з метою виявлення тих, для яких остання перевірка здійснювалася понад 30 днів тому. Для таких записів виконується перенесення значення продажів за поточний місяць у поле минуломісячних продажів, обнулення поля поточних продажів, а також оновлення дати останньої перевірки на поточну, що є важливою функцією для прогнозування.

```python
@shared_task
def check_and_run_function():
    from shop.models import Product, Holiday
    time_now = timezone.now().date()
    if (time_now.month, time_now.day) == (1, 1):
        holidays = Holiday.objects.all()
        for holiday in holidays:
            holiday.date_start = holiday.date_start + relativedelta(years=1)
            holiday.date_end = holiday.date_end + relativedelta(years=1)
            holiday.save()

    month_before = time_now - relativedelta(days=30)
    products_to_check = Product.objects.filter(last_monthly_sales_check__lte=month_before)
    for product in products_to_check:
        product.last_monthly_sales_check = time_now
        product.last_monthly_sales = product.current_monthly_sales
        product.current_monthly_sales = 0
        product.save()
```

Рисунок 3.13 – Функція оновлення статистики продажів та переведення свят.

Тепер, все що залишається, це провести міграції за допомогою команд docker-compose exec web_server python manage.py makemigrations та docker-compose exec web_server python manage.py migrate та отримати базу даних з таблицями:

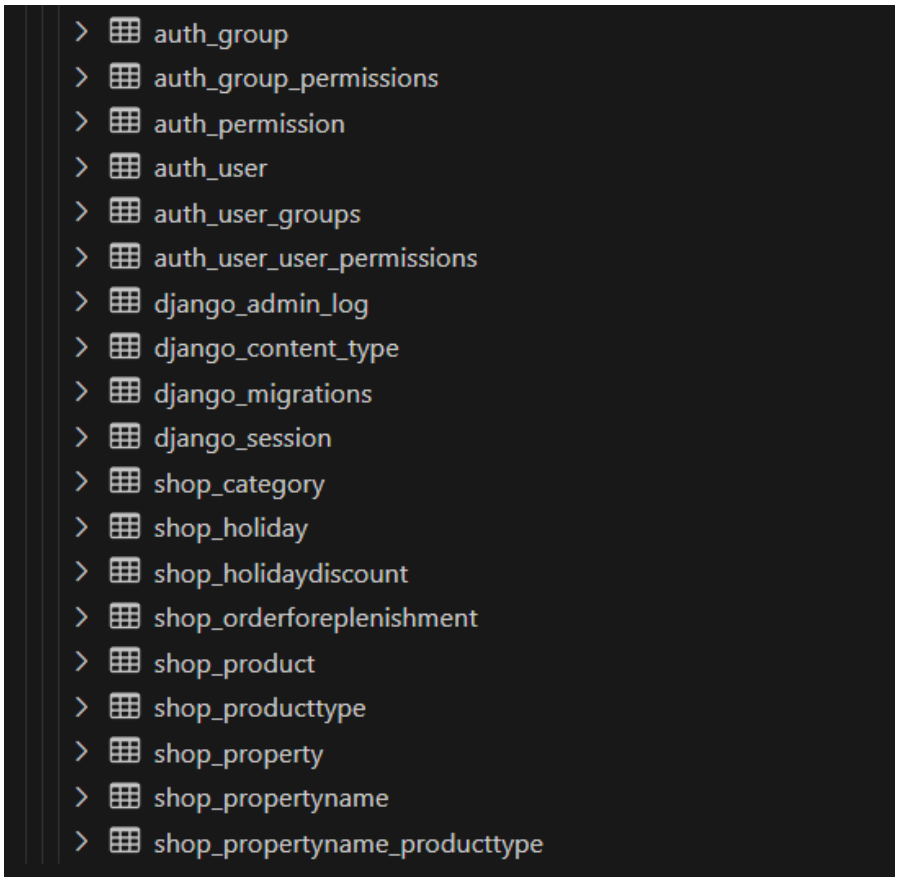


Рисунок 3.14 – таблиці бази даних, сформовані з Django моделей.

3.2 Створення докер контейнерів

У цьому проєкті активно використовувався **Docker**[24] для контейнеризації сервісів, що значно спростило процес розгортання, масштабування та підтримки системи. Docker дозволив ізолювати середовище кожного сервісу, що усунуло проблеми з конфліктами залежностей і забезпечило стабільність роботи незалежно від оточення.

Основні переваги Docker у проєкті:

- Ізоляція середовищ: завдяки Docker можна створювати однакові середовища для розробки, тестування та продакшн-оточення. Це дозволило уникнути ситуацій, коли на одному пристрої програма працює, а на іншому – ні.
- Простота розгортання: контейнери можна запускати за лічені секунди, що значно пришвидшило CI/CD процеси. Кожен сервіс мав свій Dockerfile або вже створений образ, а вся система конфігурувалась через docker-compose.
- Масштабованість: Docker дав змогу легко масштабувати окремі сервіси. Наприклад, за потреби ми могли підняти кілька інстансів одного з мікросервісів без складної настройки.
- Портативність: контейнери запускались однаково на будь-якій машині з Docker, що спростило колаборацію в команді. Нові учасники могли швидко стартувати, просто запустивши docker-compose up.
- Безпека та контроль: кожен контейнер працював у власному ізольованому середовищі, що підвищувало загальну безпеку системи. Крім того, ми використовували офіційні образи та регулярно оновлювали їх для уникнення вразливостей.

Docker став незамінним інструментом у проєкті. Він забезпечив гнучкість у розробці, надійність у продакшн-середовищі та зменшив витрати часу на конфігурацію інфраструктури.

Одним з основних контейнерів, які використовувались у docker-compose файлі є “web_server”, він був створений на основі власноруч прописаного Dockerfile:

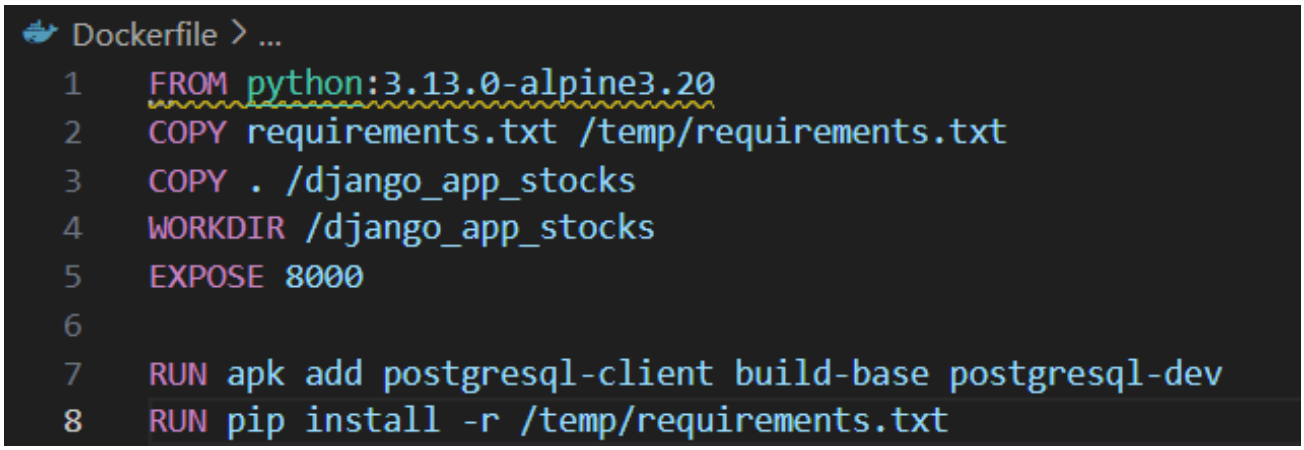


Рисунок 3.15 - Dockerfile до контейнеру web_server.

Цей Dockerfile містить в собі полегшений alpine-образ python а також інструкції для встанвлення потрібних бібліотек при збірці.

Щодо самого контейнеру в компоуз-файлі, так виглядає його код:

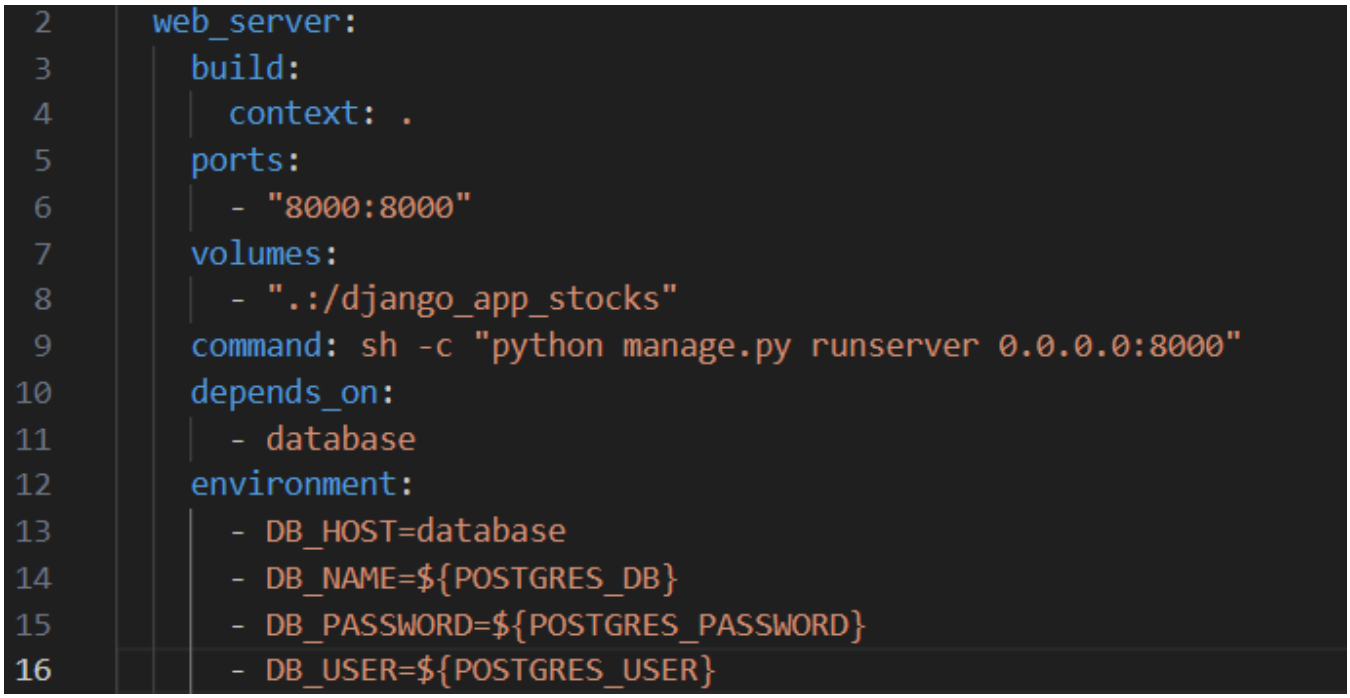


Рисунок 3.16 – Контейнер web_server в docker-compose файлі.

З примітного тут можна виділити прокидання 8000 порту (порт який за стандартом використовується в Django та використовується і в цьому проєкті, так як на даному порті ніяких конфліктів немає), директива command, в яку записано

команду для запуску Django додатку, директиву depends_on, яка каже, що контейнер залежний від контейнеру з базою даних та директива environment, яка записала в контейнер змінні середовища для підключення до POSTGRESQL.

Далі в compose-файлі створюється контейнер database, тематика якого зрозуміла з самої назви. В ньому так само як і в модулі web_server використано полегшену версію образу, але на цей раз postgres.

```
18    database:
19      image: postgres:14.6-alpine
20      environment:
21        - POSTGRES_DB=${POSTGRES_DB}
22        - POSTGRES_USER=${POSTGRES_USER}
23        - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
```

Рисунок 3.17 – Контейнер database в docker-compose файлі.

В нього також було записано environment, але цього разу для налаштування бази даних.

Наступним є контейнер для Redis з однойменною назвою, який потрібен для роботи Celery задля контролю черги завдань. Він також використовує полегшений образ.

```
25    redis:
26      image: redis:7.0.5-alpine
27      hostname: redis
```

Рисунок 3.18 - Контейнер redis в docker-compose файлі.

Параметр hostname використано для чіткого візуального відображення залежностей між контейнерами через depends_on, щоб спростити розуміння їх взаємодії.

Наступними йдуть контейнери celery_worker та celery_worker 2, які повністю повторюють структуру web_server, так як вони являють собою окремо запущені екземпляри веб-серверу, кожен з яких має свою окрему задачу.

```
29    celery_worker:
30      build:
31        context: .
32      hostname: worker
33      entrypoint: celery
34      command: -A replenishment.celery_app.app worker --loglevel=info
35      volumes:
36        - ".:/django_app_stocks"
37      links:
38        - redis
39      depends_on:
40        - redis
41      environment:
42        - DB_HOST=database
43        - DB_NAME=${POSTGRES_DB}
44        - DB_PASSWORD=${POSTGRES_PASSWORD}
45        - DB_USER=${POSTGRES_USER}
```

Рисунок 3.19 - Контейнер celery_worker в docker-compose файлі.

```
47    celery_worker2:
48      build:
49        context: .
50      hostname: worker2
51      entrypoint: celery
52      command: -A replenishment.celery_app.app2 worker --loglevel=info
53      volumes:
54        - ".:/django_app_stocks"
55      links:
56        - redis
57      depends_on:
58        - redis
59      environment:
60        - DB_HOST=database
61        - DB_NAME=${POSTGRES_DB}
62        - DB_PASSWORD=${POSTGRES_PASSWORD}
63        - DB_USER=${POSTGRES_USER}
```

Рисунок 3.20 - Контейнер celery_worker2 в docker-compose файлі.

З відмінностей від оригіналу: директива, що вказує на залежність від redis та параметр links, який не є обов'язковим, але так само як hostname є вказівником явного зв'язку.

Відокремлені ці сервіси тому що вони виконують різні ролі: перший – для розсилання email, другий – для перевірки щомісячних замовлень.

celery_beat – контейнер, який також повторює конфігурацію web_server, але його завданням є відслідковування стану товарів, а точніше їх дат перевірок, кожного дня.

```
celery_beat:
  build:
    context: .
  hostname: beat
  entrypoint: celery
  command: -A replenishment.celery_app.app2 beat --loglevel=info
  volumes:
    - ".:/django_app_stocks"
  links:
    - redis
  depends_on:
    - redis
  environment:
    - DB_HOST=database
    - DB_NAME=${POSTGRES_DB}
    - DB_PASSWORD=${POSTGRES_PASSWORD}
    - DB_USER=${POSTGRES_USER}
```

Рисунок 3.21 - Контейнер celery_beat в docker-compose файлі.

Останніми модулями є 2 flower-и, завдання кожного з яких це спостерігання за celery-тасками (їх кількість, наявність запущених та помилок).

```
flower:
  build:
    context: .
  hostname: flower
  entrypoint: celery
  command: -A replenishment.celery_app.app flower
  volumes:
    - ".:/django_app_stocks"
  links:
    - redis
  depends_on:
    - redis
  ports:
    - "5555:5555"
```

Рисунок 3.22 - Контейнер flower в docker-compose файлі.

```
flower2:
  build:
    context: .
  hostname: flower
  entrypoint: celery
  command: -A replenishment.celery_app.app2 flower
  volumes:
    - ".:/django_app_stocks"
  links:
    - redis
  depends_on:
    - redis
  ports:
    - "5556:5555"
```

Рисунок 3.23 - Контейнер flower2 в docker-compose файлі.

3.3 Тестування програмного продукту

Для тестування програмного забезпечення потрібно запустити Docker-контейнери. Щоб це зробити, треба в командний рядок написати docker-compose up –build. Це команда, що замінює дві: docker-compose build та docker-compose up.

```
[+] Building 26.3s (23/50)
 => => transferring context: 1.25MB
 => CACHED [flower 2/6] COPY requirements.txt /temp/requirements.txt
 => [web_server 3/6] COPY . /django_app_stocks
 => [celery_beat 4/6] WORKDIR /django_app_stocks
 => [celery_worker2 5/6] RUN apk add postgresql-client build-base postgresql-dev
```

Рисунок 3.24 – Процес побудови та запуску контейнерів.

Для забезпечення ефективної роботи оператора в системі необхідна адміністративна панель, яка дозволяє здійснювати управління даними, контролювати користувачів, редагувати записи та виконувати інші адміністративні функції. У фреймворку Django така панель вже реалізована "з коробки" у вигляді вбудованого адміністративного інтерфейсу. Django Admin дозволяє швидко організувати доступ до бази даних через зручний веб-інтерфейс, не потребуючи розробки окремого інструменту для адміністрування. Це значно спрощує процес розробки, оскільки забезпечує базову функціональність для роботи оператора без додаткових витрат часу та ресурсів. Щоб створені таблиці з’явились в цій панелі, необхідно їх зареєструвати та налаштувати в файлі admin.py.

```python
from django.contrib import admin
from .models import *

@admin.register(ProductType)
class ProductTypeAdmin(admin.ModelAdmin):
    list_display = ['__str__']
    search_fields = ['name']


@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ['__str__']
    search_fields = ['name']
```

Рисунок 3.25 – Приклад налаштування відображення таблиць в admin.py.

Робота звичайного менеджера полягає у зміні статусу замовлень та перевірці узгодженості даних у таблицях. Щоб створити менеджера, треба створити звичайного користувача, а потім присвоїти йому групу “managers” в адмінпанелі, або скориставшись відповідним API-ендпоінтом та надати йому staff статус.

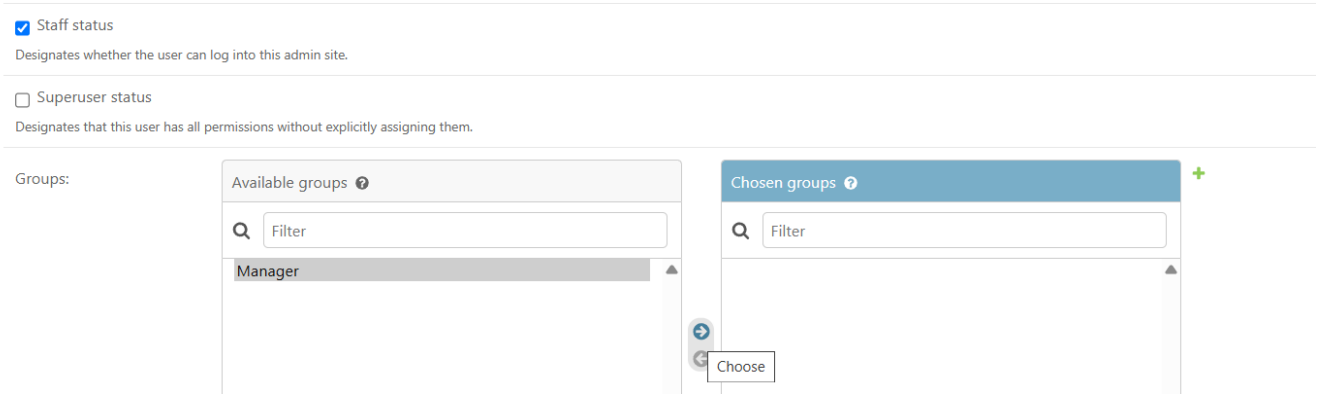


Рисунок 3.26 – Створення менеджера.

Для більших можливостей роботи з базою даних, таких як створення, зміна та видалення будь-якого запису, потрібно мати статус суперюзера. Він надається одною галочкою “superuser status”, або використанням команди magage.py createsuperuser.

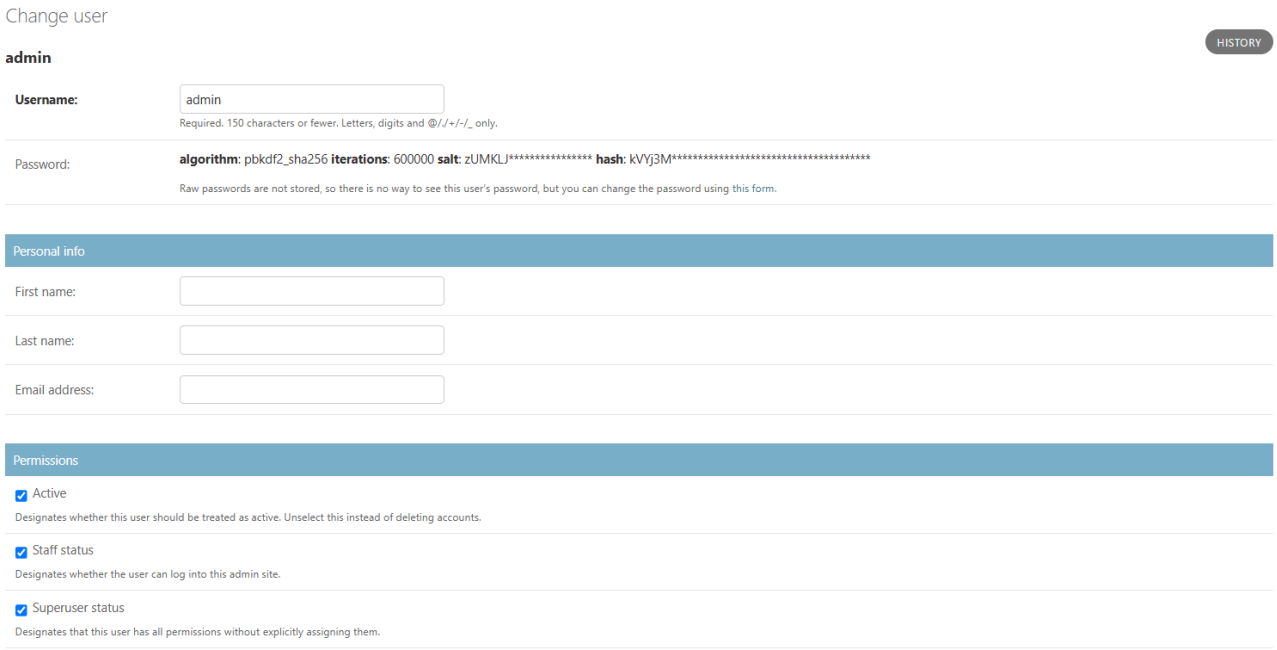


Рисунок 3.27 – Профіль суперюзера в адмін панелі.

Перевіримо інтерфейс для створення або зміни записів у адмін панелі:

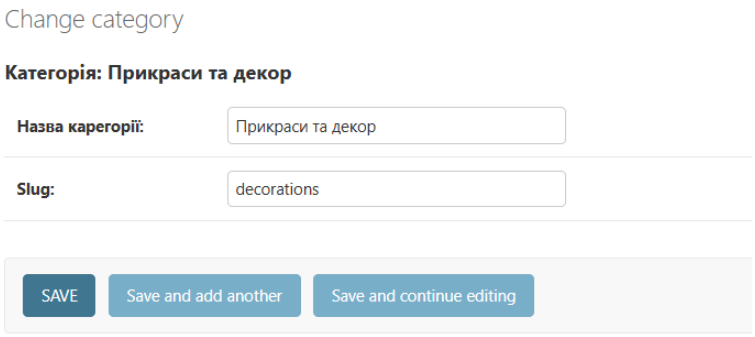


Рисунок 3.28 - Інтерфейс для створення або зміни записів у адмін панелі.

Завдяки налашутванням в файлі admin.py, можна зробити зручний вигляд сторінки з замовленнями для того щоб швидко перемикати Bool поле запису, що в цьому випадку від оператора і потрібно.

Рисунок 3.29 – Налаштований інтерфейс сторінки з замовленнями.

Оскільки воркер №2 має періодичну роботу, можна виставити менший проміжок задля тестування, поставимо значення в 20 секунд.

```
23  app2.conf.beat_schedule = {
24      "test": {
25          "task": 'replenishment.tasks.check_and_run_function',
26          "schedule": timedelta(seconds=20),
27      },
28  }
```

Рисунок 3.30 – Змінена конфігурація Celery beat для тестування.

Також, треба змінити дату переведення днів акцій (1 січня стандартно, 6 травня на момент тестування).

```python
@shared_task
def check_and_run_function():
    from shop.models import Product, Holiday
    time_now = timezone.now().date()
    if (time_now.month, time_now.day) == (5, 6):
        holidays = Holiday.objects.all()
        for holiday in holidays:
            holiday.date_start = holiday.date_start + relativedelta(years=1)
            holiday.date_end = holiday.date_end + relativedelta(years=1)
            holiday.save()

    month_before = time_now - relativedelta(days=30)
    products_to_check = Product.objects.filter(last_monthly_sales_check__lte=month_before)
    for product in products_to_check:
        product.last_monthly_sales_check = time_now
        product.last_monthly_sales = product.current_monthly_sales
        product.current_monthly_sales = 0
        product.save()
```

Рисунок 3.31 – Змінена конфігурація 2 celery task для тестування.

Перевіримо, чи зміниться дата проведення свята після роботи воркера:

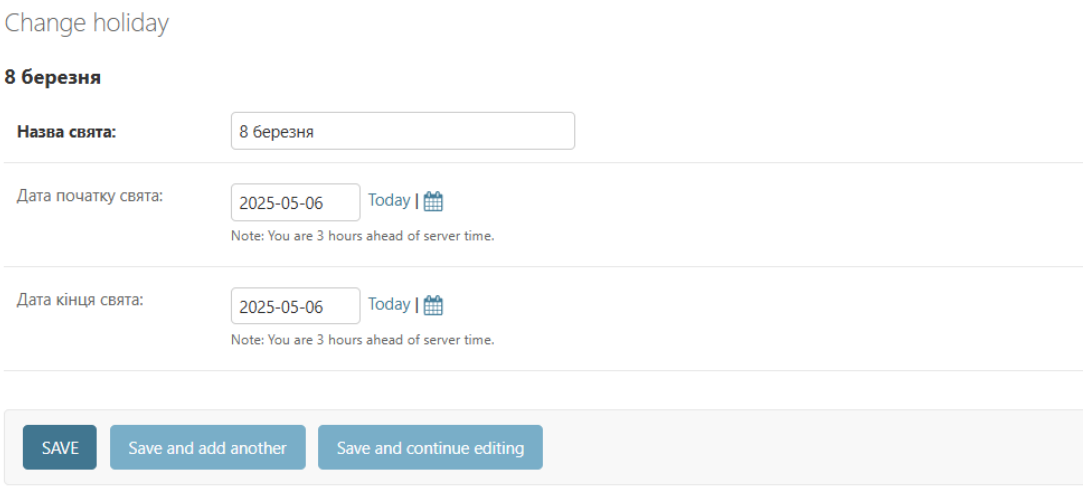


Рисунок 3.32 – Дати проведення свята до роботи воркера.

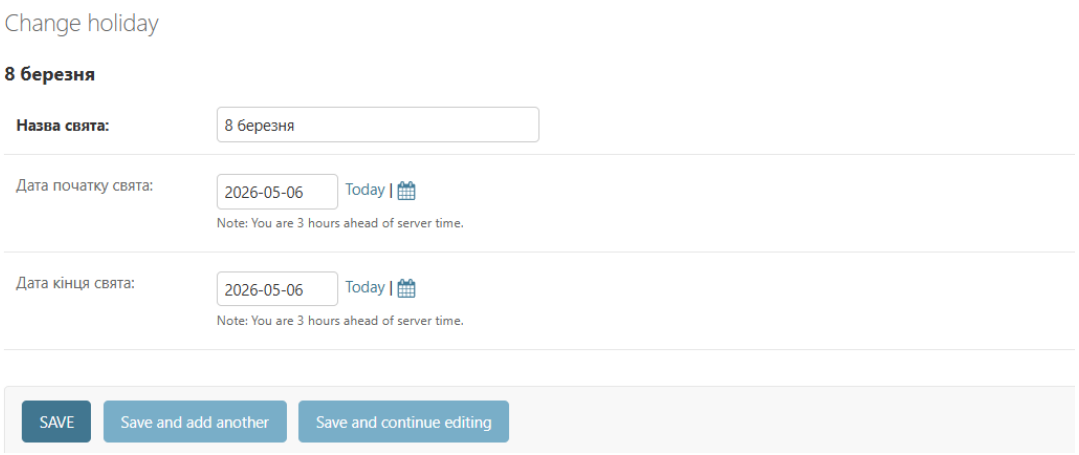


Рисунок 3.33 - Дати проведення свята після роботи воркера.

Тепер перевіримо чи працює перевірка та зміна щомісячної статистики продаж. Для цього штучно зменшимо дату останньої перевірки на 30 днів.

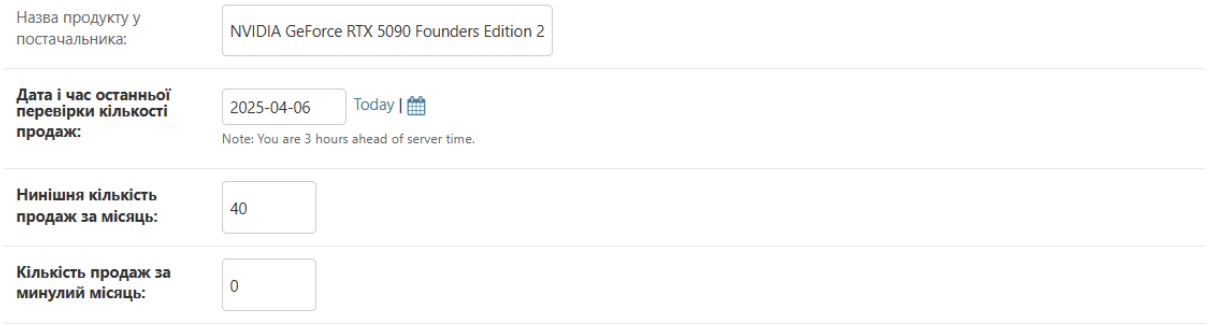


Рисунок 3.34 – Сторінка товару до проведення щомісячної перевірки.

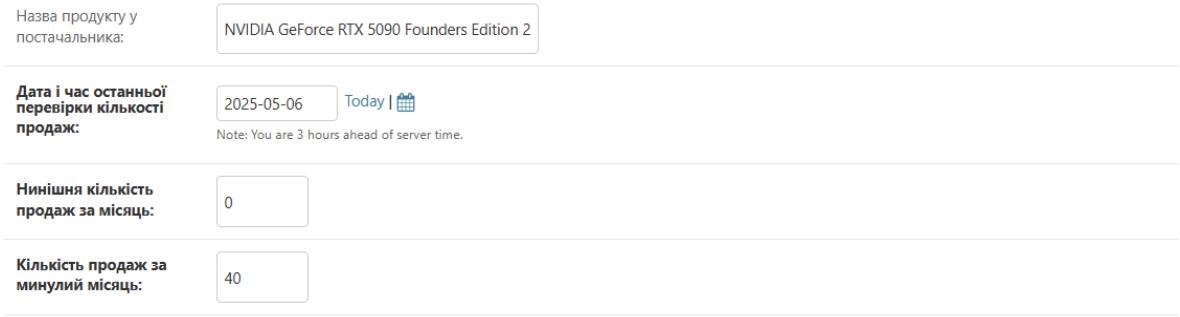


Рисунок 3.35 – Сторінка товару після проведення щомісячної перевірки.

Щоб подивитись на успішність всіх запущених тасок, використаємо flower та подивимось як завершились початі завдання:

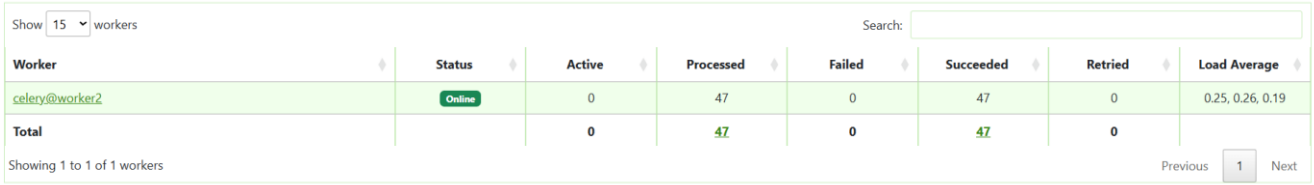


Рисунок 3.36 – Статистика роботи воркера: 47 успішних завдань з 47 запущених.

Виходячи з цієї інформації, можна зробити висновок, що воркер №2 працює правильно.

Тепер, перевіримо як працює воркер №1. Робота цього воркера полягає в створенні замовлень та надсиланні листа. В разі необхідності замовлення є 2 сценарії:

1. Товар новий та не має статистики продажів, в такому випадку використовується стандартне значення
2. Товар вже має якусь статистику попиту та програма може від неї відштовхуватись.

Спочатку розглянемо сценарій №1:

Маємо такий товар: NVIDIA GeForce RTX 5090 Founders Edition. Поки у неї немає продажів, кількість подібних на складі – 30, а потрібна мінімальна кількість для замовлення - 10.

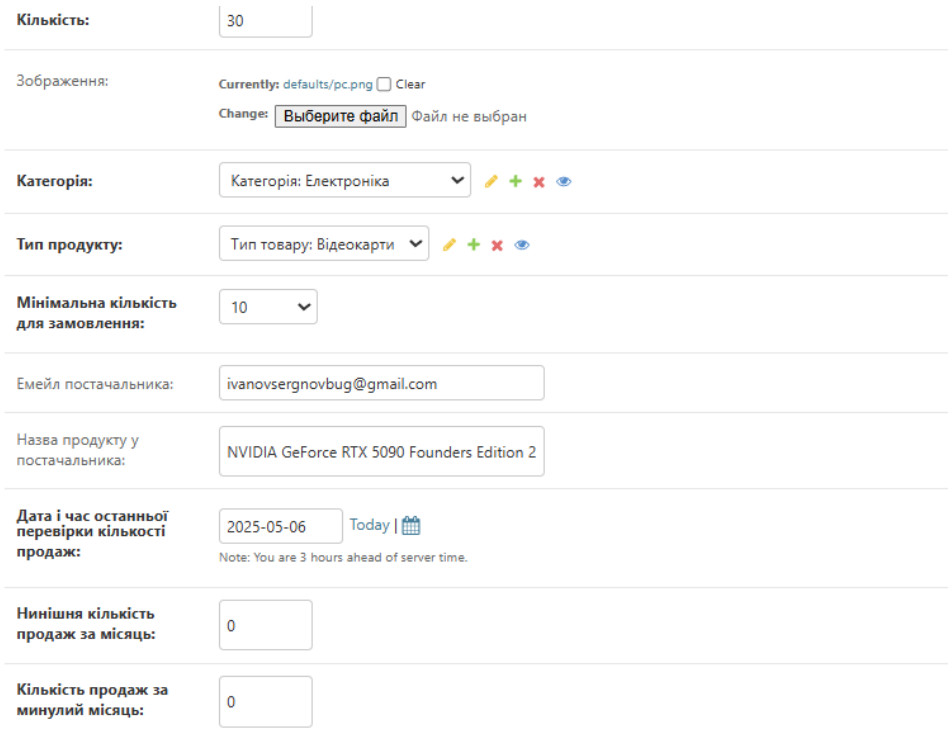


Рисунок 3.37 – Сторінка товару NVIDIA GeForce RTX 5090 Founders Edition в адмін панелі – статистики продажів немає, кількість на слкаді перевищує 10.

Зімітуємо створення одного замовлення на позицію, збільшивши нинішні її продажі на 1 і зменшивши її кількість до 29, що все ще перевищує мінімальний поріг у 10 одиниць.

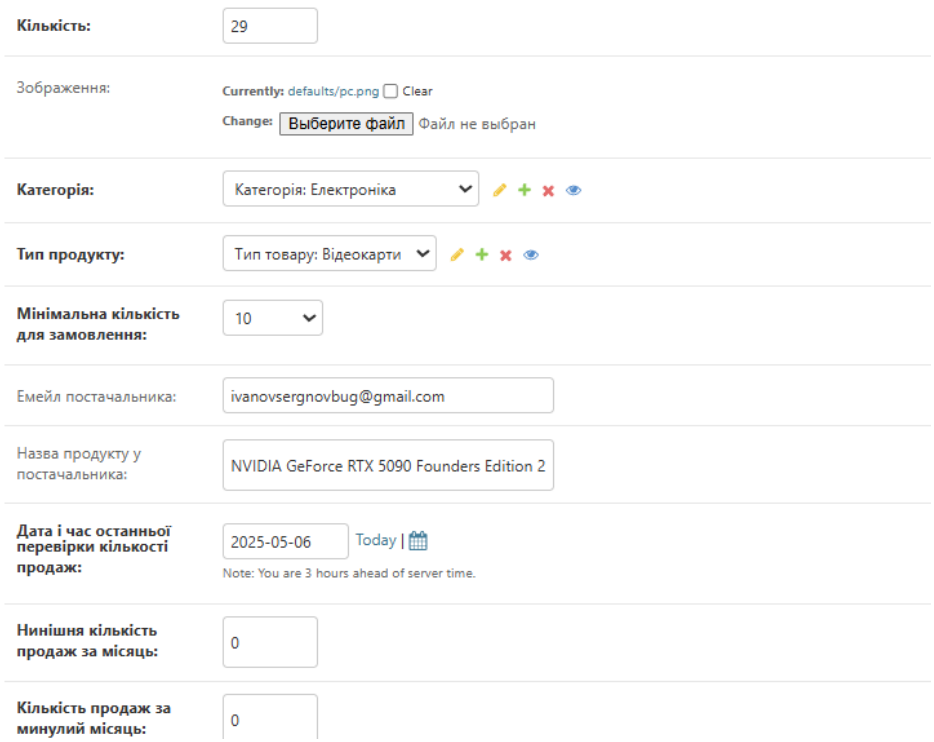


Рисунок 3.38 – Сторінка товару NVIDIA GeForce RTX 5090 Founders Edition в адмін панелі – статистики продажів немає, кількість все ще перевищує 10.

Якщо кількість товару була б меншою за вказану як потрібну для замовлення, в таблиці Order for replenishments з’явився би відповідний запис, але в цьому випадку його нема:

Рисунок 3.39 – Всі рядки таблиці Order for replenishments, потрібного товару немає.

Тепер припустимо, що в магазині купили 21 подібний товар, а це означає, що на складі його менше 10.

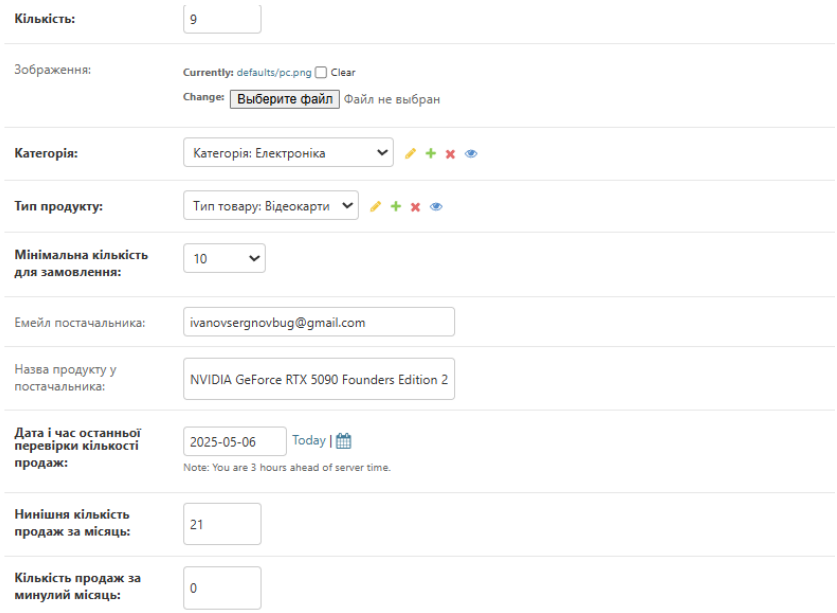


Рисунок 3.40 - Сторінка товару NVIDIA GeForce RTX 5090 Founders Edition в адмін панелі – статистики продажів немає, кількість менша 10.

Така ситуація спровокує відпрацьовування воркеру, який створить відповідне замовлення в БД та відправить лист постачальнику.

Рисунок 3.41 - Всі рядки таблиці Order for replenishments, є активне замовлення на потрібний товар

Перевіримо, чи надійшов лист указаному постачальнику:

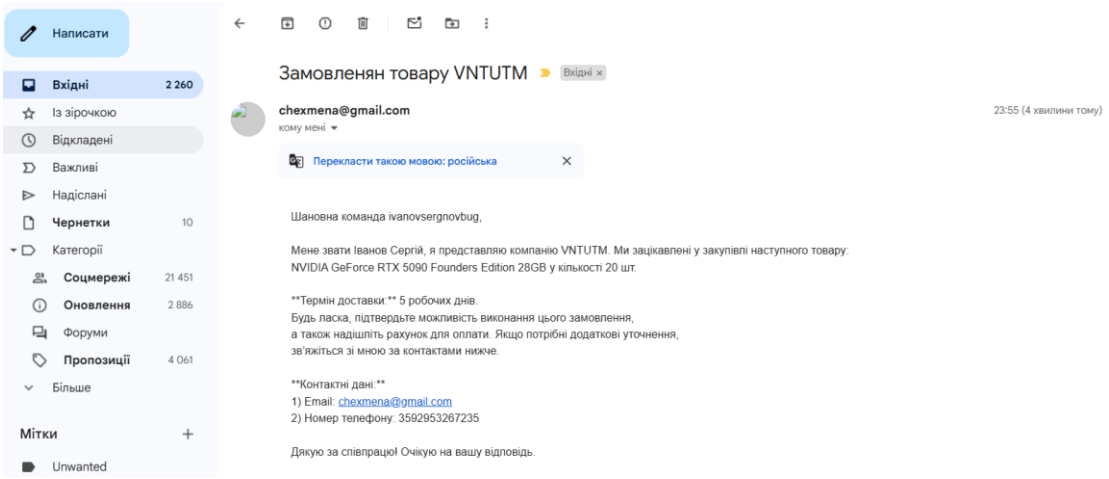


Рисунок 3.42 – Лист із замовленням у дистриб’ютора.

Як можна побачити, лист прийшов та весь текст та значення коректні.

Тепер спробуємо зменшити кількість товару ще на 1, щоб перевірити чи створиться нове замовлення при наявному активному. Для цього проводимо таку ж процедуру, як для попередніх тестів.

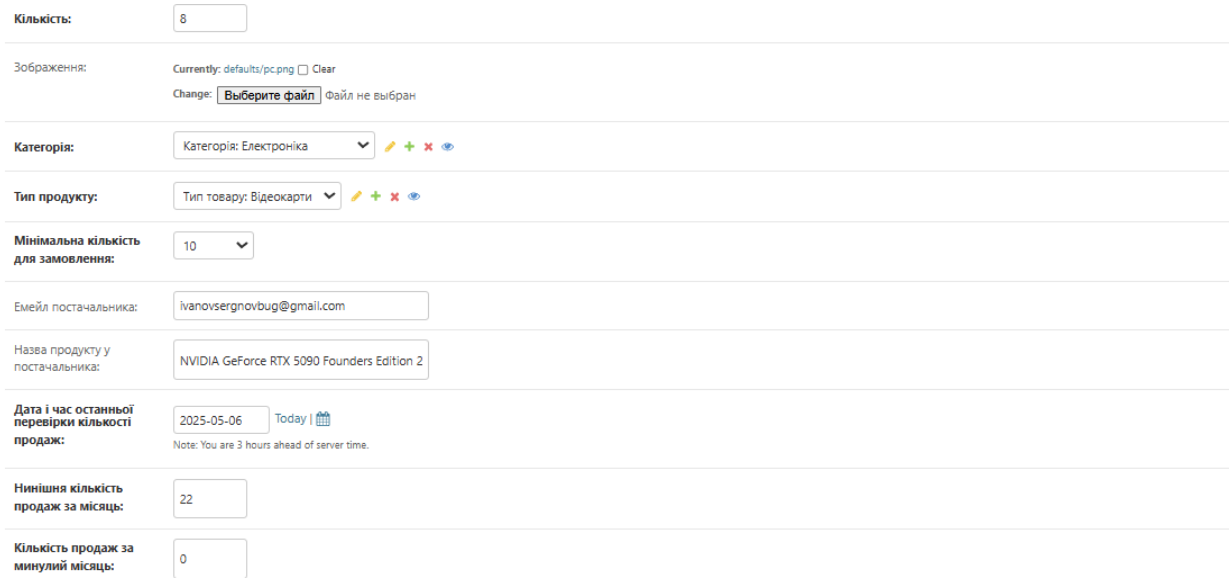


Рисунок 3.43 - Сторінка товару NVIDIA GeForce RTX 5090 Founders Edition в адмін панелі, залишок товару став ще менше.

Перевіримо, чи змінилось щось в Order for replenishments:

Рисунок 3.44 - Всі рядки таблиці Order for replenishments, нових замовлень немає.

Тепер перейдемо до сценарію №2, в якому у нас є статистика за попередній місяць. Також зімітуємо успішне завершення замовлення тестового товару.

Рисунок 3.45 - Всі записи таблиці Order for replenishments, запит на постачання NVIDIA GeForce RTX 5090 Founders Edition став неактивним.

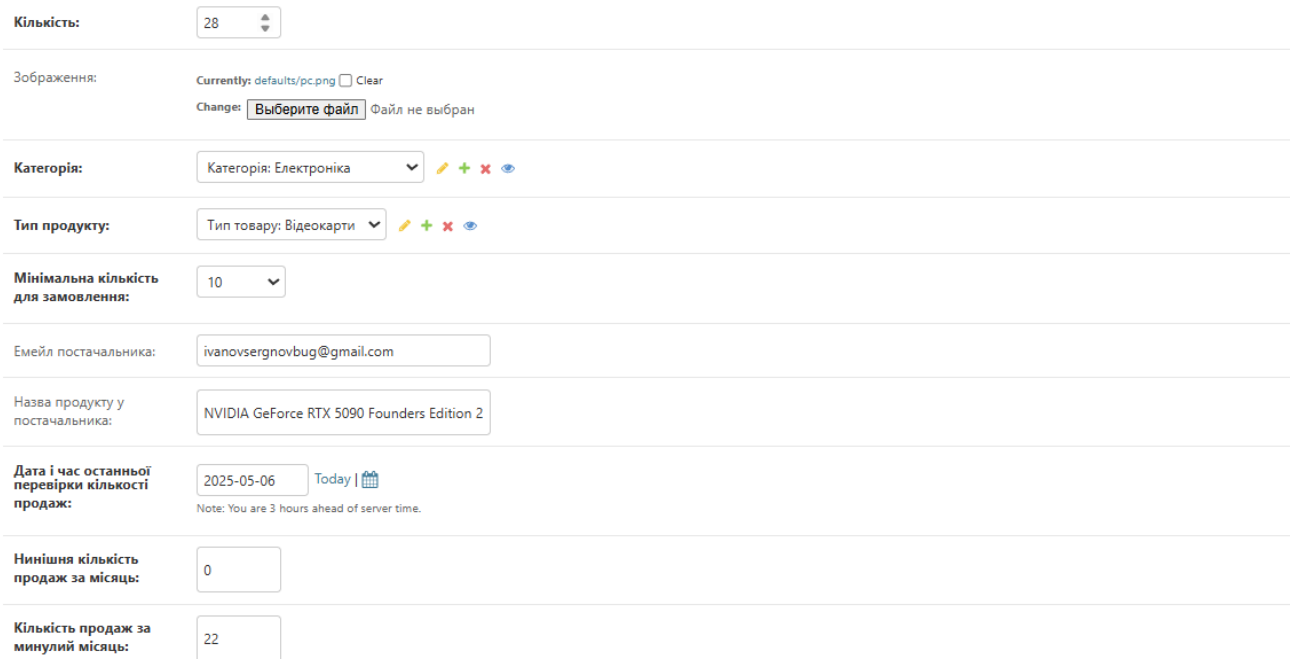


Рисунок 3.46 - Сторінка товару NVIDIA GeForce RTX 5090 Founders Edition в адмін панелі, відбулось поповнення, з’явилась статистка за минулий місяць.

Тепер знову зімітуємо декілька покупок позиції клієнтом, щоб кількість стала меншою, але зараз для поповнення потрібно, щоб залишок товару на складі був меншим за минуломісячні продажі.

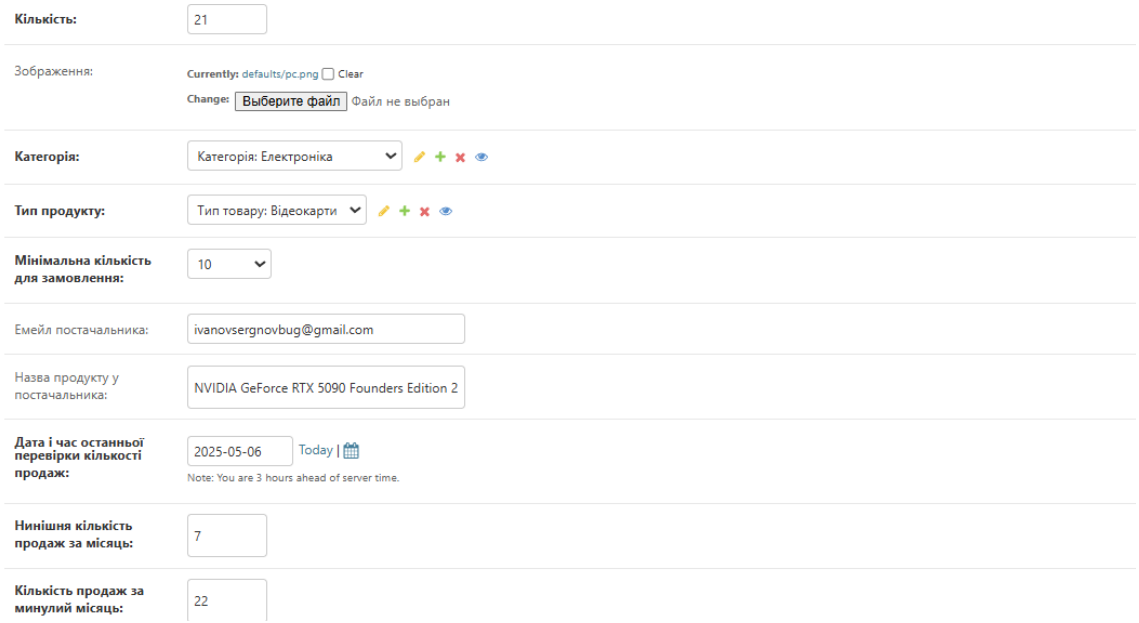


Рисунок 3.47 - Сторінка товару NVIDIA GeForce RTX 5090 Founders Edition в адмін панелі, статистика продажів є і залишок на складі мешний, ніж ніж кількість проданого товару.

Знову переглянемо таблицю Order for replenishments, повинне з’явитись нове замовлення, так як спрацювала відповідна умова та йому не заважає інше активне замовлення на товар. Також, цей запит на постачання тепер опирається на інформацію про збут за минулий місяць.

Рисунок 3.48 - Всі записи таблиці Order for replenishments, з’явився новий активний запит на постачання NVIDIA GeForce RTX 5090 Founders Edition зі значенням ґрунтованим на статистикі продажів.

Переглянемо пошту, щоб остаточно переконатися в успішності замовлення:

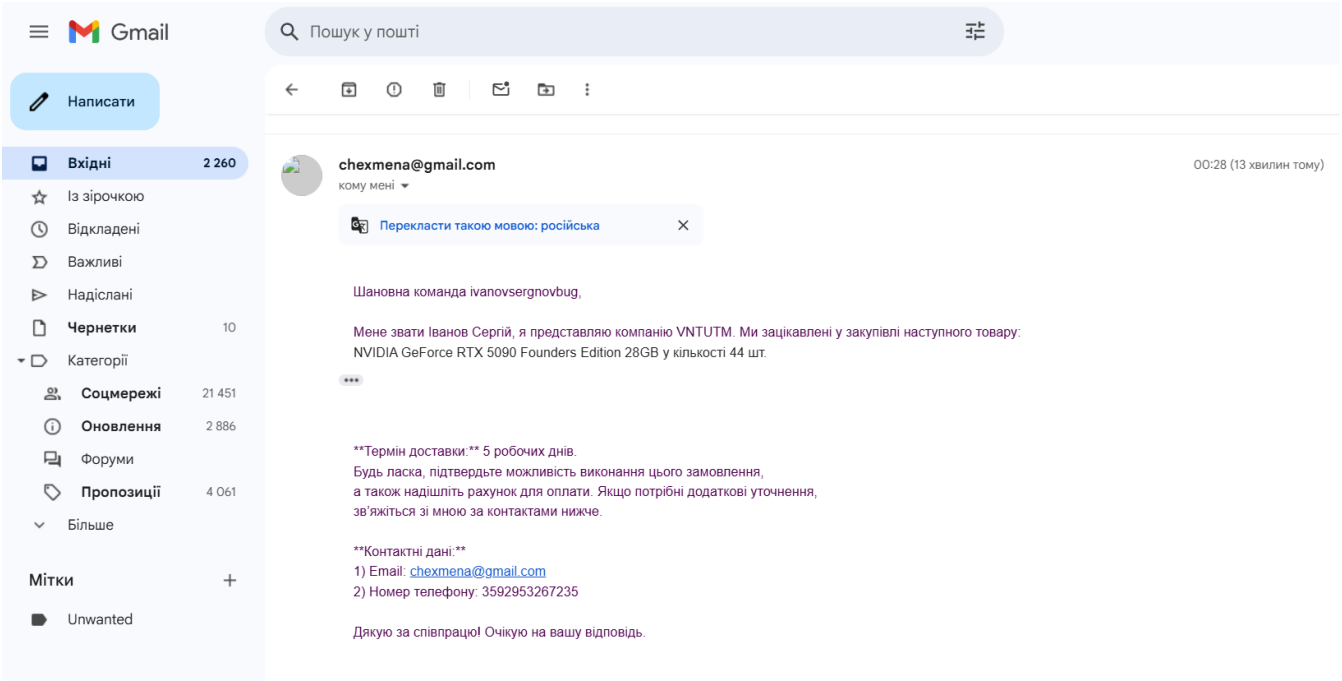


Рисунок 3.49 – Другий лист із замовленням у дистриб’ютора.

Також, якщо проводити тести в св’яткові дні, які можна налаштувати в БД, то кількість товару для замволення зросте пропорційно до найбільших знижок в цей момент.

Тепер можна бути упевненими, що всі методи та воркери працюють коректно, забезпечуючи стабільну і ефективну роботу системи. Всі функції виконуються без помилок, а обробка завдань відбувається згідно з встановленими вимогами. Система демонструє високий рівень надійності та продуктивності, що підтверджується відсутністю збоїв у процесі виконання задач.

3.4 Висновки

В даному розділі було розроблено програмну частину системи для управління складом та замовленнями. Були створені моделі для товарів, категорій та замовлень, а також розроблено функцію автоматичного генерування замовлень для поповнення складу на основі порогових значень. Також було налаштовано систему асинхронної обробки завдань з використанням Celery для забезпечення ефективної роботи фонових процесів, таких як перевірка складу і відправка повідомлень. В ході роботи було налаштовано адміністративну панель для зручного управління даними через Django. Для забезпечення безперебійної роботи програми було використано Docker для контейнеризації всіх компонентів системи. Було проведено тестування для перевірки коректності роботи функціональності, включаючи автоматичне поповнення складу та надсилання повідомлень.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було проаналізовано сучасні рішення для автоматизації складських процесів та виявлено їхні основні переваги й недоліки. На основі проведеного аналізу сформовано вимоги до власної інтелектуальної комп’ютерної системи автоматизації складу, що має забезпечити підвищення ефективності управління товарними запасами при мінімальних фінансових витратах.

Було спроєктовано архітектуру системи, визначено функціональні модулі та створено програмне забезпечення, яке реалізує автоматичне формування замовлень, облік залишків, прогнозування попиту та врахування сезонних чинників. У межах розробки здійснено вибір відповідного інструментарію та технологій, таких як Python, Django, PostgreSQL, Docker і Celery, що забезпечило надійну, гнучку та масштабовану реалізацію програмного продукту.

Проведене тестування підтвердило коректність роботи системи відповідно до поставлених задач. Розроблене рішення демонструє хороші показники продуктивності та зручності використання, що робить його придатним для впровадження на підприємствах малого та середнього бізнесу.

Таким чином, запропонована інтелектуальна система автоматизації складу дозволяє значно скоротити участь людини у рутинних процесах, оптимізувати управління залишками товарів і забезпечити більш оперативне реагування на зміни в попиті.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Thompson E. What is Warehouse Automation? Examples, Types, and Benefits. Cyngn | Industrial AMRs to Automate Your Material Handling. 20.06.2024. [Електронний ресурс] URL: https://www.cyngn.com/blog/what-is-warehouse-automation-examples-types-and-benefits (дата звернення: 26.05.2025).
2. Іванов С.С., Ковалюк О.О. «Розробка інтелектуальної комп'ютерної системи автоматизації складу», конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)». [Електронний ресурс] URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/25421
3. В.Л. Плескач, Т.Г. Затонацька Інформаційні системи і технології на підприємствах. – Книга. – Розділ 5: Інтегровані інформаційні системи управління підприємством (ERP-системи). – С. 110–123. (дата звернення: 27.05.2025).
4. TorgSoft. Про компанію. [Електронний ресурс] URL: https://torgsoft.ua/about/company/ (дата звернення: 28.05.2025).
5. ERP-система IT-Enterprise. IT-Enterprise – твоя універсальна платформа для цифрової трансформації | www.it.ua. [Електронний ресурс] URL: https://www.it.ua/erp-system-it-enterprise (дата звернення: 28.05.2025).
6. Програма Склад Україна - універсальний складський облік | Про програму. Програма Склад Україна - універсальний складський облік | Про програму. [Електронний ресурс] URL: http://www.ukrsklad.com (дата звернення: 28.05.2025).
7. TorgSoft. Приклади впровадження програмного забезпечення. Apple Podcasts. [Електронний ресурс] URL: https://podcasts.apple.com/gb/podcast/торгсофт/id1512571703 (дата звернення: 28.05.2025).
8. Торгсофт - Огляд, Відгуки, Ціна та Аналоги - MIISOFT. [Електронний ресурс] URL: https://miisoft.com.ua/product/torgsoft/ (дата звернення: 28.05.2025).

9. Кейси та кращі практики. IT-Enterprise – твоя універсальна платформа для цифрової трансформації | www.it.ua. [Електронний ресурс] URL: https://www.it.ua/cases (дата звернення: 28.05.2025).
10. Відгуки про ITEnterprise на Facebook. [Електронний ресурс] URL: https://www.facebook.com/ITEnterprise/reviews (дата звернення: 28.05.2025).
11. ВАРТІСТЬ ХМАРНОЇ ERP-СИСТЕМИ IT-ENTERPRISE.CLOUD ТЕПЕР СТАРТУЄ З $ 299 НА МІСЯЦЬ - новина з розвитку бізнесу та інформаційних технологій на підприємстві | IT-Enterprise. IT-Enterprise – твоя універсальна платформа для цифрової трансформації | www.it.ua. [Електронний ресурс] URL: https://www.it.ua/news/stoimost-oblachnoj-erp-sistemy-it-enterprisecloud-teper-startuet-s-299-v-mesjac (дата звернення: 28.05.2025).
12. Програма Склад Україна - універсальний складський облік | Як придбати. Програма Склад Україна - універсальний складський облік | Про програму. [Електронний ресурс] URL: https://www.ukrsklad.com/ua/register.html (дата звернення: 28.05.2025).
13. Укрсклад - Огляд, Відгуки, Ціна та Аналоги – MIISOFT. [Електронний ресурс] URL: https://miisoft.com.ua/product/ukrsklad/ (дата звернення: 28.05.2025).
14. УкрСклад. Спільнота для обміну досвідом між користувачами програм УкрБланк, УкрСклад, УкрЗарплата. [Електронний ресурс] URL: https://www.softbalance.com.ua/forum/index.php?board=4.0 (дата звернення: 29.05.2025).
15. Учасники проектів Вікімедіа. Unified Modeling Language – Вікіпедія. Вікіпедія. 14.11.2024. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Unified_Modeling_Language (дата звернення: 1.06.2025).
16. Моралес Дж. In-depth Knowledge of UML Use Case Diagram: with Tutorial. MindOnMap | Free Mind Mapping Tool to Draw Ideas Easily Online. 10.03.2023. [Електронний ресурс]

URL: https://www.mindonmap.com/uk/blog/what-is-a-uml-use-case-diagram (дата звернення: 01.06.2025).

17. Maxym Z. Блок-схема (Flowchart). Maxym Zosym. 12.06.2023. [Електронний ресурс] URL: https://www.maxzosim.com/blok-skhema/ (дата звернення: 01.06.2025).
18. Учасники проектів Вікімедіа. Django – Вікіпедія. Вікіпедія. 25.11.2024. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Django (дата звернення: 02.06.2025).
19. Учасники проектів Вікімедіа. ASP.NET – Вікіпедія. Вікіпедія. 18.04.2022. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/ASP.NET (дата звернення: 02.06.2025).
20. Учасники проектів Вікімедіа. PHP – Вікіпедія. Вікіпедія. 06.02.2025 [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/PHP (дата звернення: 02.06.2025).
21. Учасники проектів Вікімедіа. Python – Вікіпедія. Вікіпедія. 21.12.2024 [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Python (дата звернення: 02.06.2025).
22. Учасники проектів Вікімедіа. PostgreSQL – Вікіпедія. Вікіпедія. 07.04.2025. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/PostgreSQL (дата звернення: 02.06.2025).
23. Учасники проектів Вікімедіа. Діаграма розгортання – Вікіпедія. Вікіпедія. 01.12.2020. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Діаграма_розгортання (дата звернення: 03.06.2025).
24. Учасники проектів Вікімедіа. Docker – Вікіпедія. Вікіпедія. 16.04.2025. [Електронний ресурс] URL: https://uk.wikipedia.org/wiki/Docker (дата звернення: 11.06.2025).

Додаток А

Протокол перевірки на плагіат

(обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: _ Розробка інтелектуальної комп'ютерної системи автоматизації складу _

Тип роботи: _____бакалаврська дипломна робота_____
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ _____кафедра КСУ_____
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism __2,7_ %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

☒ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

_Завідувач кафедри КСУ_____ _____ В'ячеслав КОВТУН
(прізвище, ініціали, посада) (підпис)

_Гарант ОПП_____ _____ Олег КОВАЛЮК
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Володимир ДУБОВОЙ_
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ _____ Олег КОВАЛЮК _____
(підпис) (прізвище, ініціали, посада)

Здобувач _____ _____ Сергій ІВАНОВ _____
(підпис) (прізвище, ініціали)

Додаток Б

(обов'язковий)

ВНТУ

ЗАТВЕРДЖЕНО

Зав. кафедри КСУ ВНТУ,

д.т.н., проф.

В'ячеслав КОВТУН

" 22 " травня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи

«РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ СКЛАДУ»

Студент групи

2АКІТ-21б Сергій ІВАНОВ

Керівник

Доц. каф. КСУ, к.т.н.

Олег КОВАЛЮК

Вінниця 2025

1. Назва та галузь застосування

1.1. Назва – Інтелектуальна комп'ютерна система автоматизації складу. Частина 1. Програмна частина.

1.2. Галузь застосування – Роздрібна торгівля, логістика.

2. Підстава для проведення розробки.

Тема бакалаврської кваліфікаційної роботи затверджена наказом по ВНТУ №_97_ від _20 березня_.2025 р.

3. Мета та призначення розробки.

Метою бакалаврської кваліфікаційної роботи є підвищення ефективності управління товарним обліком шляхом розробки власної системи, що відзначається простотою впровадження та експлуатації, а також не потребує значних фінансових витрат.

4. Джерела розробки.

Бакалаврська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Thompson E. What is Warehouse Automation? Examples, Types, and Benefits. Cyngn | Industrial AMRs to Automate Your Material Handling. 20.06.2024. URL: https://www.cyngn.com/blog/what-is-warehouse-automation-examples-types-and-benefits
2. В.Л. Плескач, Т.Г. Затонацька Інформаційні системи і технології на підприємствах. – Книга. – Розділ 5: Інтегровані інформаційні системи управління підприємством (ERP-системи). – С. 110–123.
3. Django documentation. URL: https://docs.djangoproject.com/en

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- Перевірка кількості товару на складі
- Формування замовлення товару в разі потреби
- Щомісячний облік продажів
- Контроль параметрів замовлення залежно від чинників

- Перенесення інформації про акції (свята)

5.2. Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:

- Будь-яка ОС на комп’ютері

5.2.2. Умови експлуатації системи:

- робота одного оператора для підтвердження замовлень що завершились
- можливість цілодобового функціонування системи;

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

1. Аналіз методів, принципів, підходів і засобів реалізації задачі автоматизації процесами в об’єкті управління відповідно до теми кваліфікаційної роботи. Постановка задач дослідження «10» травня 2025 р.
2. Визначення технічних характеристик системи «12» травня 2025 р.
3. Розробка програмного забезпечення системи «30» травня 2025 р.

6.2 Графічні матеріали:

1. Розробка моделі системи «24» травня 2025 р.
2. Тестування програмного забезпечення «5» червня 2025 р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «4» червня 2025 р.

7.2. Атестація проєкту здійснюється на попередньому захисті. Попередній захист бакалаврської кваліфікаційної роботи провести до «10» червня 2025 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист бакалаврської кваліфікаційної роботи провести до «20» червня 2025 р.

Додаток В

Лістинг програми

shop/models.py:

```
from django.core.exceptions import ValidationError
from django.db import models
from django.utils import timezone
from django.contrib.auth.models import User

from replenishment.tasks import stock_task

class Category(models.Model):
    name = models.CharField(verbose_name='Назва карегорії', unique=True,
blank=False, null=False, max_length=50)
    slug = models.SlugField(default='', null=False, db_index=True)

    def __str__(self):
        return f'Категорія: {self.name}'

class ProductType(models.Model):
    name = models.CharField(verbose_name='Назва типу продукту', unique=True,
blank=False, null=False, max_length=50)
    category = models.ForeignKey(Category, on_delete=models.CASCADE,
blank=False, null=False,
    verbose_name='Категорія', related_name='producttype')
    slug = models.SlugField(default='', null=False, db_index=True)

    def __str__(self):
        return f'Тип товару: {self.name}'
```

```
class Product(models.Model):

    QUANTCATS = [
        (0, 5),
        (1, 10),
        (2, 100),
        (3, 1000)
    ]

    def content_file_name(instance, filename):
        return '/'.join([f'images/products/{instance.producttype.name}', instance.name[0], filename])

    name = models.CharField(verbose_name='Назва товару', unique=True, blank=False, null=False, max_length=200)
    price = models.DecimalField(verbose_name='Ціна товару',max_digits=8, decimal_places=2, blank=False, null=False)
    slug = models.SlugField(default='', null=False, db_index=True)
    description = models.TextField('Опис', blank=True)
    quantity = models.IntegerField('Кількість')
    created = models.DateTimeField('Дата і час створення товару', auto_now_add=True)
    updated = models.DateTimeField('Дата і час останньої зміни товару', auto_now_add=True)
    image = models.ImageField('Зображення', upload_to=content_file_name, default='defaults/pc.png', blank=True)
    category = models.ForeignKey(Category, on_delete=models.SET_NULL, null=True, verbose_name='Категорія', related_name='product')
```

```
    producttype = models.ForeignKey(ProductType, on_delete=models.SET_NULL,
null=True, verbose_name='Тип продукту', related_name='product')
    min_quantity = models.IntegerField('Мінімальна кількість для замовлення',
choices=QUANTCATS)
    supplier_email = models.EmailField('Емейл постачальника', blank=True)
    supplier_product_name = models.CharField('Назва продукту у постачальника',
blank=True, max_length=200)
    last_monthly_sales_check = models.DateField('Дата і час останньої перевірки
кількості продаж', default=timezone.now)
    current_monthly_sales = models.IntegerField('Нинішня кількість продаж за
місяць', default=0)
    last_monthly_sales = models.IntegerField('Кількість продаж за минулий місяць',
default=0)

    def __str__(self):
        return f'{self.name}, id:{self.id}'

    def save(self, *args, **kwargs):
        if self.id and self.quantity<=(self.last_monthly_sales if self.last_monthly_sales else
        self.get_min_quantity_display()) and \
        not OrderFoReplenishment.objects.filter(product__id=self.id,
active=True).exists():
            stock_task.delay(self.id)
        return super().save(*args, **kwargs)

typeslist = [
    ('1', 'String'),
    ('2', 'Float'),
    ('3', 'Bool'),
```

```
    ('4', 'Int')
]

class PropertyName(models.Model):
    name = models.CharField(verbose_name='Назва характеристики', unique=True,
blank=False, null=False, max_length=200)
    producttype = models.ManyToManyField(ProductType,
related_name='propertyname', verbose_name='Тип товару')
    type_field = models.CharField(max_length=1, choices=typeslist,
verbose_name='Тип характеристики')

    def __str__(self):
        return f'Характеристика: {self.name}'

class Property(models.Model):

    propertyname = models.ForeignKey(PropertyName, on_delete=models.CASCADE,
verbose_name='Назва характеристики',
    related_name='property')
    product = models.ForeignKey(Product, on_delete=models.CASCADE,
verbose_name='Назва товару',
                                related_name='property')
    dynamic_type_value = models.CharField('Значення',max_length=100, null=True,
blank=True)

    class Meta:
        unique_together = (('propertyname', 'product'),)

    def clean(self):
```

```
        if self.propertyname not in self.product.producttype.propertyname.all():
            raise ValidationError("Значення поля `propertyname` не належить до списку `propertyname` для типу продукту")

        return super().clean()

    def save(self, *args, **kwargs):
        if self.propertyname.type_field == '1':
            self.dynamic_type_value = str(self.dynamic_type_value)
        elif self.propertyname.type_field == '2':
            self.dynamic_type_value = float(self.dynamic_type_value)
        elif self.propertyname.type_field == '3':
            self.dynamic_type_value = bool(self.dynamic_type_value)
        elif self.propertyname.type_field == '4':
            self.dynamic_type_value = int(self.dynamic_type_value)

        super().save(*args, **kwargs)

    def __str__(self):
        return f'{self.product} - {self.propertyname}: {self.dynamic_type_value}'


class OrderFoReplenishment(models.Model):

    product = models.ForeignKey(Product, on_delete=models.DO_NOTHING, verbose_name='Продукт', related_name='order')
    active = models.BooleanField('Активний', default=True)
    quantity = models.IntegerField('Кількість', null=False)
```

```
    def save(self, *args, **kwargs):
        if self.active == False:
            self.product.quantity += self.quantity
            self.product.save()
        return super().save(*args, **kwargs)



    def __str__(self):
        return f'Товар: {self.product.name}, кількість: {self.quantity}'



class Holiday(models.Model):
    name = models.CharField('Назва свята', blank=False, default='Unnamed',
max_length=50)
    date_start = models.DateField('Дата початку свята', blank=True)
    date_end = models.DateField('Дата кінця свята', blank=True)

    def __str__(self):
        return self.name

class HolidayDiscount(models.Model):
    holiday = models.ForeignKey(Holiday, verbose_name='Свято',
related_name='discounts', on_delete=models.CASCADE)
    category = models.ForeignKey(Category, verbose_name='Категорія товарів для
знижок', related_name='holiday_discounts', on_delete=models.CASCADE)
    discount_value = models.DecimalField('Знижка', max_digits=2, decimal_places=0,
default=0)

    class Meta:
```

```
        unique_together = ['holiday', 'category']

    def __str__(self):
        return f'Свято: {self.holiday.name}, Категорія: {self.category.name}, Знижка:
{self.discount_value}%'


class Cart(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name='cart')
    product = models.ForeignKey(Product, on_delete=models.CASCADE,
related_name='products')
    quantity = models.SmallIntegerField()

    def __str__(self):
        return 'User: ' + self.user.username + ', cart obect: ' + self.product.name + f'
quantity, pcs: ' + str(self.quantity)

    class Meta:
        unique_together = ['user', 'product']


class Order(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    status = models.BooleanField(db_index=True, default=False)
    total = models.DecimalField(max_digits=11, decimal_places=2, db_index=True)
    date = models.DateField(auto_now_add=True, db_index=True)

    def __str__(self) -> str:
        return f'ORDER: USER: {self.user.username}, id: {self.id}'
```

```
class OrderItem(models.Model):
    order = models.ForeignKey(Order, on_delete=models.CASCADE,
related_name='order_item')
    product = models.ForeignKey(Product, on_delete=models.CASCADE,
related_name='product')
    quantity = models.SmallIntegerField()
    unit_price = models.DecimalField(max_digits=8, decimal_places=2, db_index=True)
    price = models.DecimalField(max_digits=9, decimal_places=2, db_index=True)

    def __str__(self):
        return self.product.name + ' in the order of ' + self.order.user.username + f' user,
quantity: {self.quantity} pcs'

    class Meta:
        unique_together = ['order', 'product']
```

shop/admin.py:

```
from django.contrib import admin
from .models import *
# Register your models here.

@admin.register(ProductType)
class ProductTypeAdmin(admin.ModelAdmin):
    list_display = ['__str__']
    search_fields = ['name']
```

```
@admin.register(Category)
class CategoryAdmin(admin.ModelAdmin):
    list_display = ['__str__']
    search_fields = ['name']

@admin.register(Property)
class PropertyAdmin(admin.ModelAdmin):
    list_display = ['__str__']
    search_fields = ['propertyname__name', 'product__name']

@admin.register(Product)
class ProductAdmin(admin.ModelAdmin):
    list_display = ['__str__']
    search_fields = ['id', 'name']

@admin.register(PropertyName)
class PropertyNameAdmin(admin.ModelAdmin):
    list_display = ['__str__']
    search_fields = ['name']


@admin.register(OrderFoReplenishment)
class OrderForRepelishmentAdmin(admin.ModelAdmin):
    list_display = ['__str__', 'active']
    list_editable = ['active']
    search_fields = ['id']

@admin.register(Holiday)
class HolidayAdmin(admin.ModelAdmin):
```

```
    pass

@admin.register(HolidayDiscount)
class HolidayDiscountsAdmin(admin.ModelAdmin):
    pass

admin.site.register(Order)
admin.site.register(OrderItem)
admin.site.register(Cart)
```

replenishment/tasks.py:

```
from django.core.mail import send_mail
from .celery_app import app
from celery import shared_task
from django.utils import timezone
import math
from dateutil.relativedelta import relativedelta
from django.db.models import Q
from datetime import timedelta

import os

def get_coefficient(product):
    from shop.models import HolidayDiscount

    today = timezone.now().date()

    active_holidays = HolidayDiscount.objects.filter(
```

```
        Q(holiday__date_start__lte=today) & Q(holiday__date_end__gte=today),
        category=product.category
    ).order_by('-discount_value')

    if active_holidays.exists():
        discount = active_holidays.first().discount_value
        return 1 + discount/100
    return 1

@app.task
def stock_task(id):
    from shop.models import Product, OrderFoReplenishment
    product = Product.objects.get(id=id)
    coefficient = get_coefficient(product=product)
    subject = f'Замовленян товару VNTUTM'
    email_user = os.getenv("EMAIL_USER")
    supplier_name = product.supplier_email.split('@')[0]
    quantity = math.floor((product.last_monthly_sales if product.last_monthly_sales else
    product.get_min_quantity_display()) * 2 * coefficient)
    message = f"""
    Шановна команда {supplier_name},

    Мене звати Іванов Сергій, я представляю компанію VNTUTM. Ми зацікавлені у
закупівлі наступного товару:
    {product.supplier_product_name} у кількості {quantity} шт.

    **Термін доставки:** 5 робочих днів.
    Будь ласка, підтвердьте можливість виконання цього замовлення,
    а також надішліть рахунок для оплати. Якщо потрібні додаткові уточнення,
```

```
    зв’яжіться зі мною за контактами нижче.

    **Контактні дані:**
    1) Email: {email_user}
    2) Номер телефону: 3592953267235

    Дякую за співпрацю! Очікую на вашу відповідь.
    """
    send_mail(subject=subject, message=message,
from_email=email_user,recipient_list=[product.supplier_email], fail_silently=False)
    OrderFoReplenishment.objects.create(product=product, quantity=quantity)

@shared_task
def check_and_run_function():
    from shop.models import Product, Holiday
    time_now = timezone.now().date()
    if (time_now.month, time_now.day) == (1, 1):
        holidays = Holiday.objects.all()
        for holiday in holidays:
            holiday.date_start = holiday.date_start + relativedelta(years=1)
            holiday.date_end = holiday.date_end + relativedelta(years=1)
            holiday.save()

    month_before = time_now - relativedelta(days=30)
    products_to_check =
Product.objects.filter(last_monthly_sales_check__lte=month_before)
    for product in products_to_check:
        product.last_monthly_sales_check = time_now
        product.last_monthly_sales = product.current_monthly_sales
```

```
        product.current_monthly_sales = 0
        product.save()
```

replenishment/celery_app.py:

```
from django.conf import settings

import os
from datetime import timedelta

from celery import Celery



os.environ.setdefault("DJANGO_SETTINGS_MODULE", 'stocks.settings')

def workerconfig(name: str, num: int):
    app = Celery(name)
    app.config_from_object('django.conf:settings')
    app.conf.broker_url = settings.CELERY_BROKER_URL[num]
    app.autodiscover_tasks()
    return app

app = workerconfig('stocks', 0)
app2 = workerconfig('sales_checking', 1)

app2.conf.beat_schedule = {
    "test": {
        "task": 'replenishment.tasks.check_and_run_function',
        "schedule": timedelta(days=1),}, }
```

Додаток Г
(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ІНТЕЛЕКТУАЛЬНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ АВТОМАТИЗАЦІЇ СКЛАДУ

Студент групи 2АКІТ-21б

_____ **Сергій ІВАНОВ**

Керівник: к.т.н., доцент каф. КСУ

_____ **Олег КОВАЛЮК**

Вінниця 2025

UML- діаграма варіантів використання

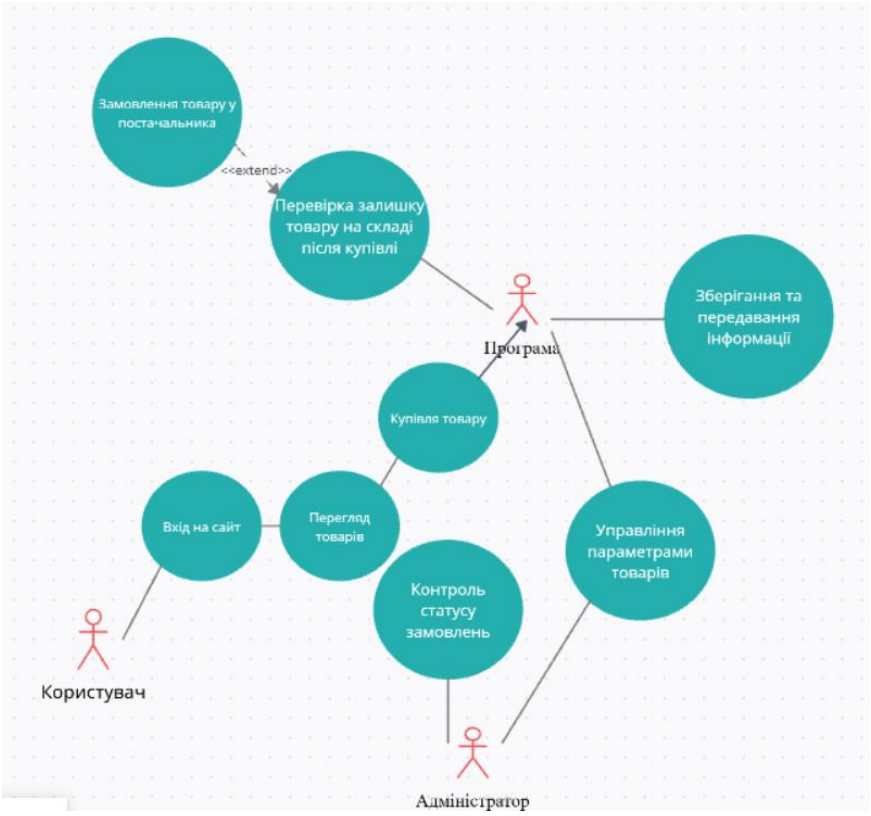


Рисунок Г.1 – UML- діаграма варіантів використання.

Діаграма блок-схема (клієнт)

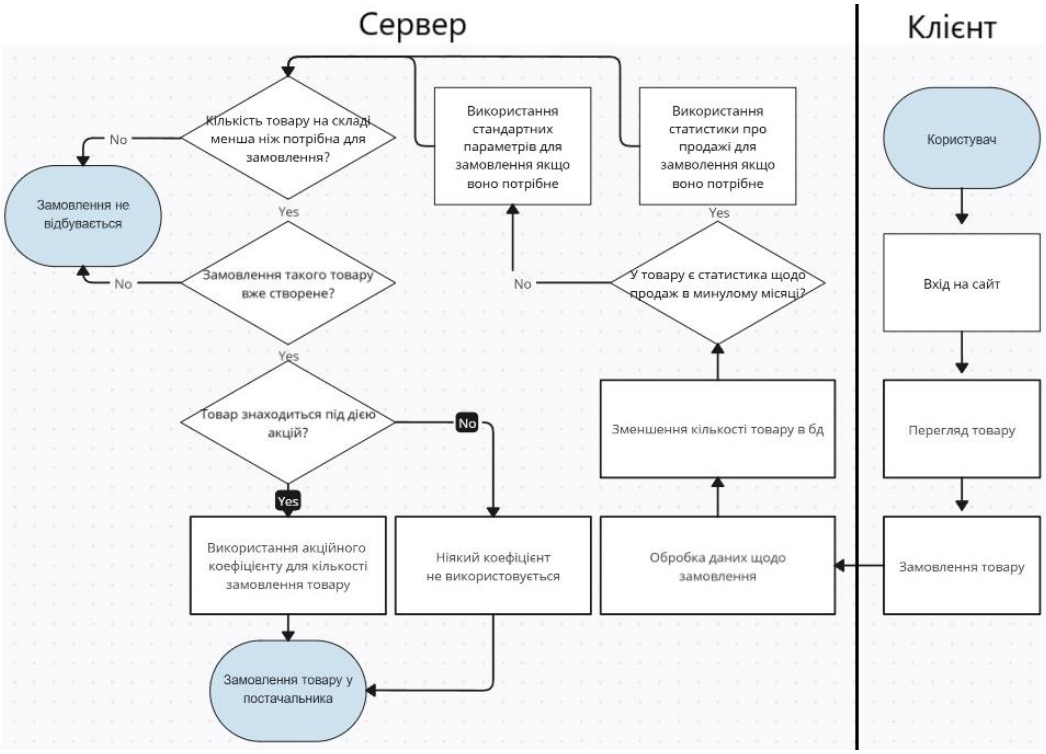


Рисунок Г.2 – Діаграма блок-схема 1 (клієнт).

Моделі таблиці товарів

```python
class Product(models.Model):

    QUANTCATS = [
        (0, 5),
        (1, 10),
        (2, 100),
        (3, 1000)
    ]

    def content_file_name(instance, filename):
        return '/'.join([f'images/products/{instance.producttype.name}', instance.name[0], filename])

    name = models.CharField(verbose_name='Назва товару', unique=True, blank=False, null=False, max_length=200)
    price = models.DecimalField(verbose_name='Ціна товару',max_digits=8, decimal_places=2, blank=False, null=False)
    slug = models.SlugField(default='', null=False, db_index=True)
    description = models.TextField('Опис', blank=True)
    quantity = models.IntegerField('Кількість')
    created = models.DateTimeField('Дата і час створення товару', auto_now_add=True)
    updated = models.DateTimeField('Дата і час останньої зміни товару', auto_now_add=True)
    image = models.ImageField('Зображення', upload_to=content_file_name, default='defaults/pc.png', blank=True)
    category = models.ForeignKey(Category, on_delete=models.SET_NULL, null=True, verbose_name='Категорія', related_name='product')
    producttype = models.ForeignKey(ProductType, on_delete=models.SET_NULL, null=True, verbose_name='Тип продукту', related_name='product')
    min_quantity = models.IntegerField('Мінімальна кількість для замовлення', choices=QUANTCATS)
    supplier_email = models.EmailField('Емейл постачальника', blank=True)
    supplier_product_name = models.CharField('Назва продукту у постачальника', blank=True)
    last_monthly_sales_check = models.DateField('Дата і час останньої перевірки кількості продаж', default=timezone.now)
    current_monthly_sales = models.IntegerField('Нинішня кількість продаж за місяць', default=0)
    last_monthly_sales = models.IntegerField('Кількість продаж за минулий місяць', default=0)
```

Рисунок Г.3 – Моделі таблиці товарів.

Створення Celery воркерів

```python
def workerconfig(name: str, num: int):
    app = Celery(name)
    app.config_from_object('django.conf:settings')
    app.conf.broker_url = settings.CELERY_BROKER_URL[num]
    app.autodiscover_tasks()
    return app

app = workerconfig('stocks', 0)
app2 = workerconfig('sales_checking', 1)
```

Ртсунок Г.4 – Створення Celery воркерів.

Перевизначення методу save

```python
    def save(self, *args, **kwargs):
        if self.id and self.quantity<=(self.last_monthly_sales if self.last_monthly_sales else
        self.get_min_quantity_display()) and \
        not OrderFoReplenishment.objects.filter(product__id=self.id, active=True).exists():
            stock_task.delay(self.id)
        return super().save(*args, **kwargs)
```

Рисунок Г.5 – Перевизначення методу save.

Функція створення замовлень

```python
@app.task
def stock_task(id):
    from shop.models import Product, OrderFoReplenishment
    product = Product.objects.get(id=id)
    coefficient = get_coefficient(product=product)
    subject = f'Замовленян товару VNTUTM'
    email_user = os.getenv("EMAIL_USER")
    supplier_name = product.supplier_email.split('@')[0]
    quantity = math.floor((product.last_monthly_sales if product.last_monthly_sales else
    product.get_min_quantity_display()) * 2 * coefficient)
    message = f"""
    Шановна команда {supplier_name},

    Мене звати Іванов Сергій, я представляю компанію VNTUTM. Ми зацікавлені у закупівлі наступного товару:
    {product.supplier_product_name} у кількості {quantity} шт.

    **Термін доставки:** 5 робочих днів.
    Будь ласка, підтвердьте можливість виконання цього замовлення,
    а також надішліть рахунок для оплати. Якщо потрібні додаткові уточнення,
    зв'яжіться зі мною за контактами нижче.

    **Контактні дані:**
    1) Email: {email_user}
    2) Номер телефону: 3592953267235

    Дякую за співпрацю! Очікую на вашу відповідь.
    """
    send_mail(subject=subject, message=message, from_email=email_user,recipient_list=[product.supplier_email], fail_silently=False)
    OrderFoReplenishment.objects.create(product=product, quantity=quantity)
```

Рисунок Г.6 – Функція створення замовлень.

Воркер для зміни дат подій та щомісячних перевірок

```python
@shared_task
def check_and_run_function():
    from shop.models import Product, Holiday
    time_now = timezone.now().date()
    if (time_now.month, time_now.day) == (1, 1):
        holidays = Holiday.objects.all()
        for holiday in holidays:
            holiday.date_start = holiday.date_start + relativedelta(years=1)
            holiday.date_end = holiday.date_end + relativedelta(years=1)
            holiday.save()

    month_before = time_now - relativedelta(days=30)
    products_to_check = Product.objects.filter(last_monthly_sales_check__lte=month_before)
    for product in products_to_check:
        product.last_monthly_sales_check = time_now
        product.last_monthly_sales = product.current_monthly_sales
        product.current_monthly_sales = 0
        product.save()
```

Рисунок Г.7 – Воркер для зміни дат подій та щомісячних перевірок.

Таблиці бази даних

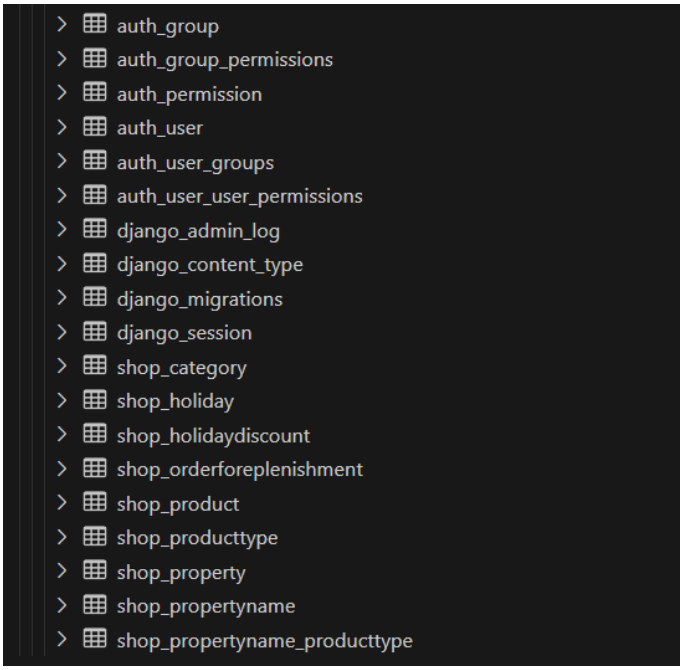


Рисунок Г.8 – Таблиці бази даних.

Контейнери в Docker-compose

```
services:
  Run Service
  web_server:
    build:
      context: .
    ports:
      - "8000:8000"
    volumes:
      - ".:/django_app_stocks"
    command: sh -c "python manage.py runserver 0.0.0.0:8000"
    depends_on:
      - database
    environment:
      - DB_HOST=database
      - DB_NAME=${POSTGRES_DB}
      - DB_PASSWORD=${POSTGRES_PASSWORD}
      - DB_USER=${POSTGRES_USER}

  Run Service
  database:
    image: postgres:14.6-alpine
    environment:
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
    ports:
      - "5432:5432"

  Run Service
  redis:
    image: redis:7.0.5-alpine
    hostname: redis
```

Рисунок Г.9 – Контейнери в docker-compose.

Лист до постачальника, відправлений системою

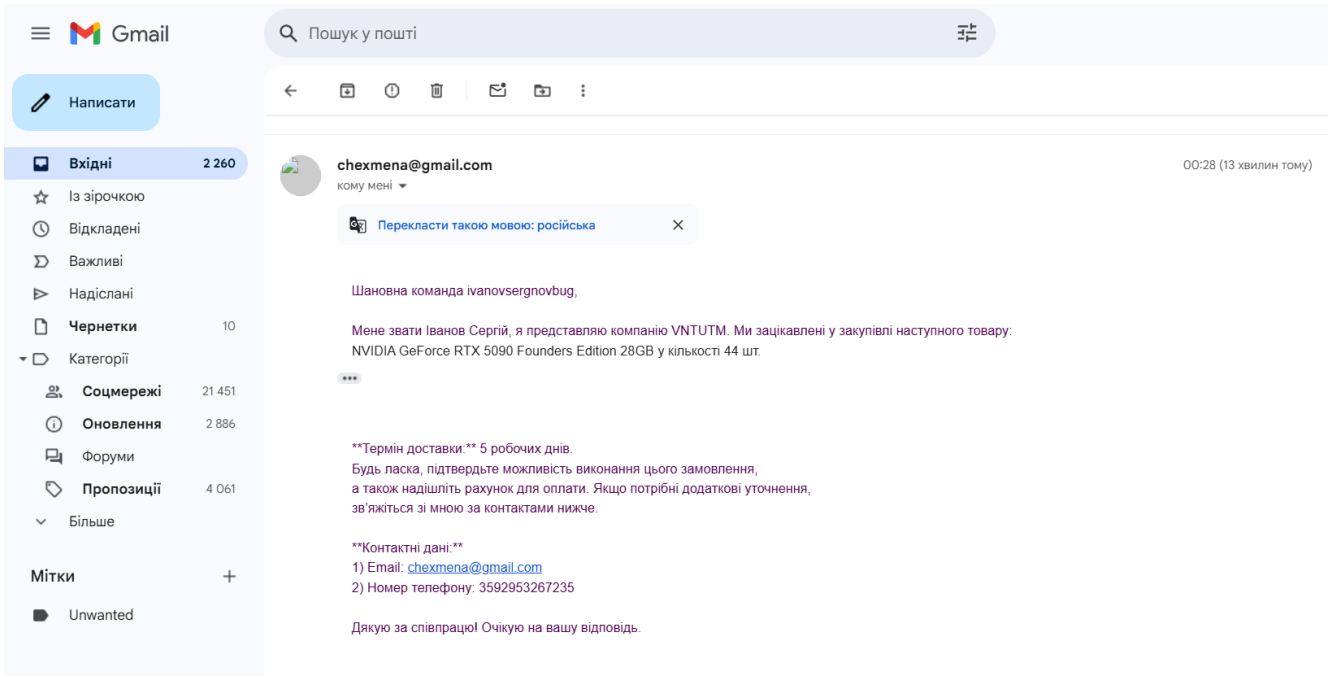


Рисунок Г.10 – Лист до постачальника відправлений системою.

Тестування переведення дат до відпрацювання воркеру

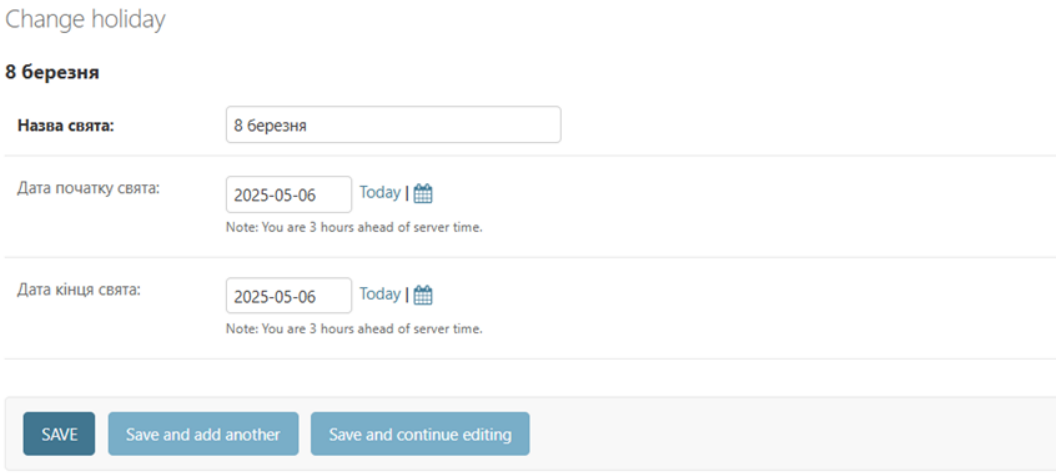


Рисунок Г.11.1 – Тестування переведення дат до відпрацювання воркеру.

Тестування переведення дат після відпрацювання воркеру

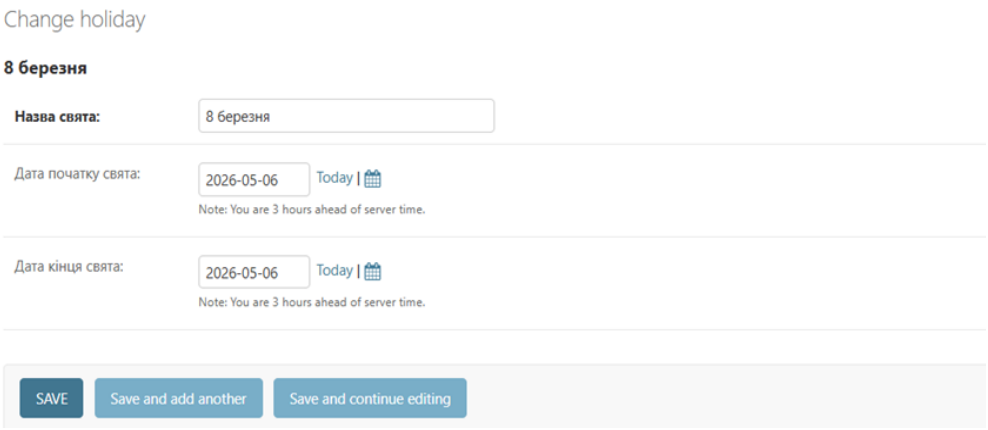


Рисунок Г.11.2 – Тестування переведення дат після відпрацювання воркеру.

Тестування перевірки щомісячних продажів до відпрацювання воркеру

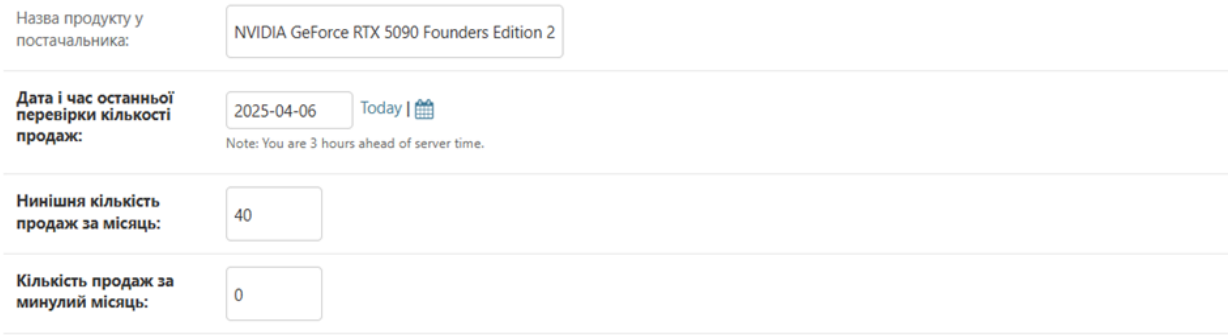


Рисунок Г.12.1 – Тестування перевірки щомісячних продажів до відпрацювання воркеру.

Тестування перевірки щомісячних продажів після відпрацювання воркеру

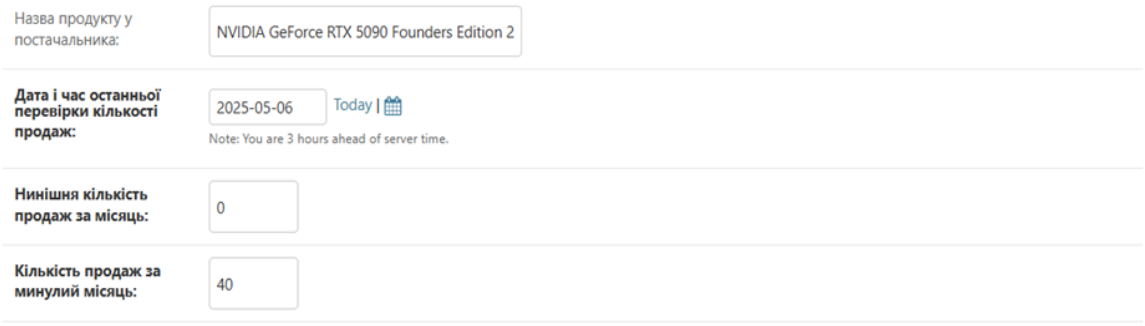


Рисунок Г.12.2 – Тестування перевірки щомісячних продажів після відпрацювання воркеру.