

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ "СИСТЕМА
УПРАВЛІННЯ ТОВАРАМИ ДЛЯ УКРАЇНСЬКИХ МАРКЕТПЛЕЙСІВ"
DEVELOPMENT OF THE SOFTWARE APPLICATION "PRODUCT
MANAGEMENT SYSTEM FOR UKRAINIAN MARKETPLACES

Виконав(ла): студент(ка) 4 курсу, групи КНТ-151

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ТРОЦ В. С.

(ПРИЗВИЩЕ та ініціали)

Керівник ГОФМАН Є.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент ЗЕЛЕНЬОВА І.Я.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Факультет КНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (КОД І НАЙМЕНУВАННЯ)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ТРОЦА Віталія Сергійовича
 (ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка програмного застосунку "Система управління товарами для українських маркетплейсів". Development of the Software Application "Product Management System for Ukrainian Marketplaces"

керівник проєкту (роботи) к.т.н., доцент, ГОФМАН Євген Олександрович,
 (науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 28 ” квітня 2025 року № 209

2. Строк подання студентом проєкту (роботи) 02 червня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування системи управління товарами. 3. Розробка програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3 Основна частина	ГОФМАН Є.О., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 14 » квітня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 3
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

(підпис) Віталій ТРОЦ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Євгеній ГОФМАН
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 110 с., 12 табл., 40 рис., 2 дод., 20 джерел.

ПАНЕЛЬ КЕРУВАННЯ, МАРКЕТПЛЕЙС, ІНТЕГРАЦІЯ, API, ROZETKA, PROM.UA, АВТОМАТИЗОВАНА СИСТЕМА, ТОВАР, ЗАМОВЛЕННЯ, ОБЛІК ТОВАРІВ, БАЗА ДАНИХ, PYTHON, FLASK, SQLITE, TELEGRAM.

Об'єкт дослідження – процес автоматизації управління товарами для українських маркетплейсів.

Предмет дослідження – методи та засоби розробки панелі керування товарами для інтеграції з українськими маркетплейсами.

Мета роботи – розробка та програмна реалізація панелі керування товарами, що автоматизує процес менеджменту товарів, замовлень та їх обліку для інтеграції з українськими маркетплейсами.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування Python, JavaScript, фреймворк Flask, SQLite, Visual Studio Code, API маркетплейсів, Telegram-бот, HTML, CSS.

Результати. Проведено аналіз предметної області, спроектовано архітектуру програмного забезпечення, обрано засоби розробки та реалізовано програмний продукт – панель керування товарами з функціями інтеграції, обліку та сповіщень.

Висновки. В результаті виконання роботи розроблено програмний продукт, який дозволяє значно спростити керування товарами та замовленнями, а також обліку продуктів на складі, інтегруючись з відомими українськими маркетплейсами.

Галузь використання – електронна комерція, управління онлайн-магазинами, автоматизація бізнес-процесів.

ABSTRACT

Explanatory note for the qualifying work of the bachelor: 110 pages, 12 tables, 40 figures, 2 appendixes, 20 sources.

CONTROL PANEL, MARKETPLACE, INTEGRATION, API, ROZETKA, PROM.UA, AUTOMATED SYSTEM, PRODUCT, ORDER, PRODUCT ACCOUNTING, DATABASE, PYTHON, FLASK, SQLITE, TELEGRAM.

The object of research is the process of automation of goods management for Ukrainian marketplaces.

Subject of research - methods and tools for development of a product control panel for integration with Ukrainian marketplaces.

The purpose of the study is to develop and implement a product control panel that automates the process of managing goods, orders and their accounting for integration with Ukrainian marketplaces.

Materials, methods and tools: object-oriented programming, Python programming language, JavaScript, Flask framework, SQLite, Visual Studio Code, marketplace APIs, Telegram bot, HTML, CSS.

Results. We analyzed the problem domain, designed the software architecture, selected development tools and implemented a software product - a product control panel with integration, accounting and notification functions.

Conclusions. As a result of the work a software product was developed, which can significantly simplify the management of goods and orders, as well as the accounting of products in the warehouse, integrating with well-known Ukrainian marketplaces.

The field of application is e-commerce, online store management, business process automation.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	9
1.1 Загальні відомості про електронну комерцію та маркетплейси	9
1.2 Огляд функціональних можливостей існуючих систем управління товарами	10
1.3 Опис особливостей українських маркетплейсів	12
1.4 Аналіз проблем і недоліків управління товарами для маркетплейсів ..	16
1.5 Постановка задач дипломної роботи	17
1.6 Висновки до розділу	17
2 Проєктування системи управління товарами.....	19
2.1 Аналіз функціональних та нефункціональних вимог	19
2.2 Проєктування архітектури програмного забезпечення.....	21
2.3 Вибір та обґрунтування засобів реалізації програмного продукту	24
2.4 Висновки до розділу	35
3 Розробка програмного забезпечення.....	37
3.1 Алгоритм роботи системи	37
3.2 Особливості програмної реалізації.....	40
3.3 Тестування програмного забезпечення.....	46
3.4 Висновки до розділу	54
Висновки	56
Перелік джерел посилання	57
Додаток А Тексти програм.....	60
Додаток Б Слайди презентації	103

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CRM	– Customer Relationship Management;
DBMS	– Database Management System;
HTTP	– HyperText Transfer Protocol;
JSON	– JavaScript Object Notation;
MVC	– Model-View-Controller;
REST	– Representational State Transfer;
SDK	– Software Development Kit;
SQL	– Structured Query;
UI	– User Interface;
URL	– Uniform Resource Locator;
XML	– Extensible Markup Language;
БД	– база даних;
ПЗ	– програмне забезпечення;
ПК	– персональний комп'ютер;
СКБД	– систем керування базами даних.

ВСТУП

Останнім часом спостерігається стрімке зростання сфери електронної комерції в Україні, що призводить до значного збільшення кількості суб'єктів господарювання, які реалізують свої товари через різноманітні онлайн-майданчики. Ця динаміка зумовлює зростаючу потребу в ефективних інструментах для централізованого управління асортиментом товарів, обробки замовлень та ведення обліку на цих платформах.

Незважаючи на наявність на ринку рішень для інтеграції з популярними українськими маркетплейсами, аналіз показує, що більшість з них мають певні обмеження функціональності або є складними у використанні для пересічного користувача. Існуючі CRM-системи, хоч і пропонують автоматизацію процесів, можуть бути дорогими для малого та середнього бізнесу або мати недостатньо гнучкі налаштування. Тому ефективне управління товарами, замовленнями та їхньою синхронізацією на кількох платформах залишається складним завданням для продавців.

Враховуючи зазначені недоліки та актуальність ефективного управління продажами в умовах розвитку електронної комерції, тема розробки спеціалізованої панелі керування товарами для інтеграції з провідними українськими маркетплейсами є надзвичайно актуальною. Такий програмний продукт дозволить автоматизувати рутинні операції, підвищити швидкість обробки замовлень та оптимізувати облік товарних залишків.

Таким чином, мета роботи полягає у розробці та програмній реалізації панелі керування товарами, яка забезпечить ефективну інтеграцію з українськими маркетплейсами, автоматизацію процесів менеджменту товарів, замовлень та їх обліку.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні відомості про електронну комерцію та маркетплейси

Електронна комерція (e-commerce) є однією з найбільш динамічних галузей сучасної економіки. Згідно з дослідженнями, обсяг ринку e-commerce в Україні у 2024 році зріс на 25% порівняно з попереднім роком, що зумовлено зростанням доступності інтернету, популяризацією онлайн-платежів і зміною споживацьких звичок [1].

Основними каналами онлайн-продажів є маркетплейси, соціальні мережі та власні вебсайти. Серед маркетплейсів лідирують Rozetka та Prom.ua. Адже наразі кожна з платформ пропонує понад 2 мільйони товарів. Щодня Розетку відвідує в середньому близько 3,5 млн користувачів, а Пром — 2,6 млн відвідувачів [2].

Маркетплейси являють собою платформи, що об'єднують продавців і покупців, надаючи інструменти для управління товарами, обробки замовлень і аналітики продажів. На відміну від власних інтернет-магазинів, маркетплейси забезпечують більшу аудиторію, але вимагають від продавців швидкої адаптації до їхніх технічних стандартів, таких як API для синхронізації даних. Основними категоріями товарів на маркетплейсах є електроніка, одяг, побутова техніка та продукти харчування [2].

Ключовою особливістю маркетплейсів є автоматизація процесів, таких як оновлення залишків, обробка замовлень і формування звітів. Проте ефективність цих процесів залежить від інтеграції з внутрішніми системами продавця, що часто потребує спеціалізованого програмного забезпечення. Аналіз показує, що малі та середні бізнеси стикаються з труднощами через брак доступних і простих у використанні рішень для управління товарами на кількох платформах одночасно [1].

1.2 Огляд функціональних можливостей існуючих систем управління товарами

Сучасні системи управління товарами для маркетплейсів можна умовно поділити на універсальні CRM-системи (KeyCRM, KeepinCRM) та спеціалізовані модулі маркетплейсів. Основні функції таких систем включають:

- управління каталогом товарів. Додавання, редагування, видалення товарів, масове оновлення цін і залишків;
- обробка замовлень. Автоматичний імпорт замовлень, зміна статусів, генерація накладних;
- інтеграція з маркетплейсами. Синхронізація даних через API, підтримка кількох магазинів;
- аналітика продажів. Звіти про продажі, популярність товарів, ефективність каналів;
- сповіщення. Інформування про нові замовлення через email, месенджери чи SMS [3].

Приклад управління замовленнями в KeepinCRM зображено на рис. 1.1 [4].

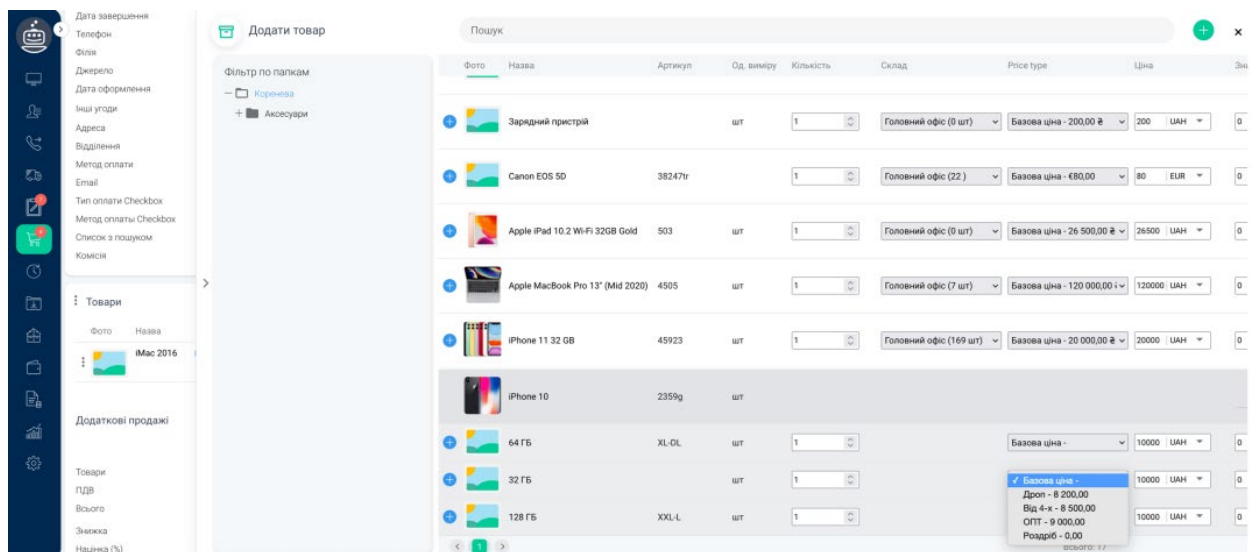


Рисунок 1.1 – Управління замовленнями в KeepinCRM [4]

KeerInCRM орієнтована на простоту використання та інтеграцію з українськими маркетплейсами. Однак її функціонал обмежений у порівнянні з конкурентами, зокрема в управлінні складськими залишками та аналітиці [4].

KeyCRM є однією з популярних систем в Україні, що підтримує інтеграцію з Rozetka, Prom.ua та іншими платформами. Вона пропонує зручний інтерфейс і функції управління складом, але має високу вартість підписки, що робить її менш доступною для малого бізнесу. Приклад інтерфейсу KeyCRM зображено на рис.1.2 [5].

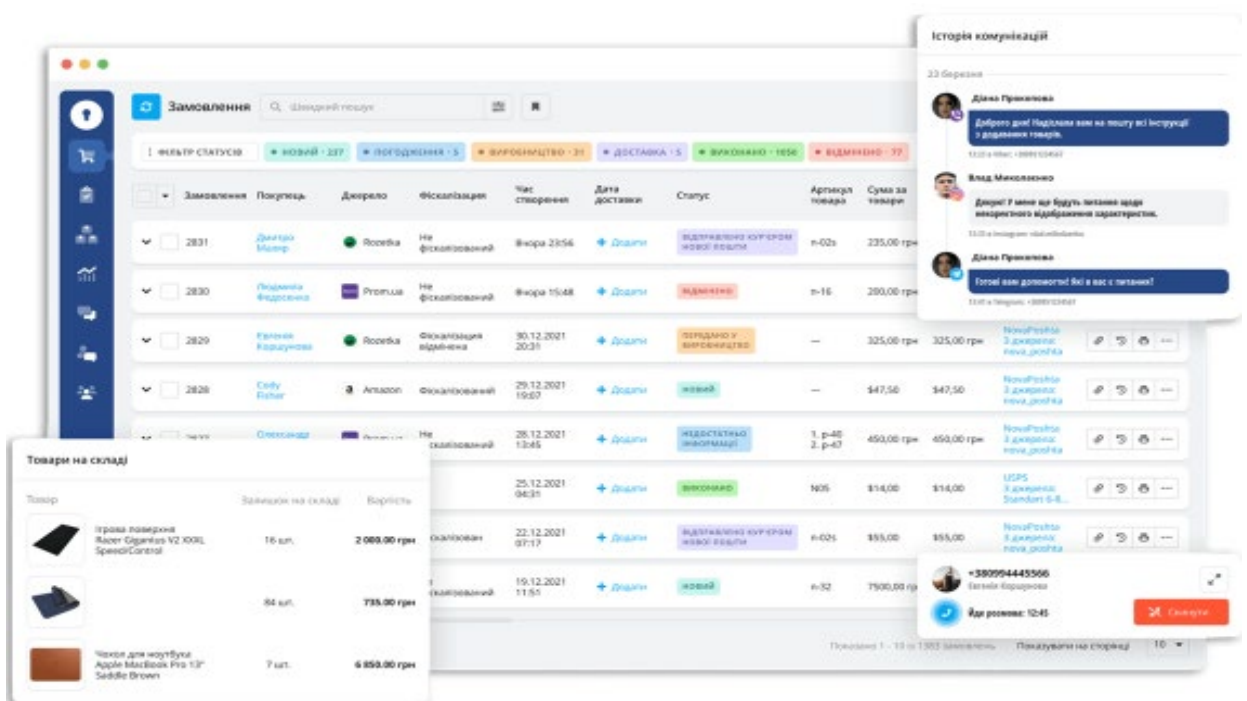


Рисунок 1.2 – Інтерфейс системи KeyCRM [5]

Інші рішення, такі як модулі від самих маркетплейсів (наприклад, кабінет продавця Prom.ua), надають базові інструменти для управління товарами, але не підтримують інтеграцію з іншими платформами чи автоматизацію складних процесів. Крім того, більшість систем не пропонують інтеграцію з месенджерами, такими як Telegram, для оперативного сповіщення про замовлення, що є важливим для швидкої реакції продавців [6].

1.3 Опис особливостей українських маркетплейсів

Українські маркетплейси, такі як Rozetka та Prom.ua, є ключовими платформами для онлайн-торгівлі в Україні, забезпечуючи широкий доступ до клієнтів і розвинену інфраструктуру для продавців. Вони мають унікальні особливості, які впливають на вимоги до систем управління товарами, включаючи специфіку API, функціонал кабінетів продавців і логістичні можливості [7].

Rozetka є найбільшим маркетплейсом України і на 2024 рік займає близько 40% ринку e-commerce. Платформа пропонує широкий асортимент товарів, включаючи електроніку, одяг, товари для дому та продукти харчування. Для продавців Rozetka надає кабінет продавця, який дозволяє керувати каталогом товарів, обробляти замовлення, налаштовувати ціни та відстежувати аналітику продажів [8].

Приклад вигляду маркетплейсу Rozetka наведено на рис. 1.3 [9].



Рисунок 1.3 – Зовнішній вигляд маркетплейсу Rozetka [9]

Кабінет продавця має зручний інтерфейс, але ручне управління товарами стає неефективним для продавців із великим асортиментом (понад 1000 SKU). Наприклад, масове оновлення цін чи залишків потребує завантаження CSV-файлів, що займає час і може призводити до помилок [10].

для автоматизації цих процесів Rozetka пропонує API Marketplace, який підтримує такі операції:

- управління товарами: створення, редагування, видалення карток товарів, оновлення цін і залишків;
- обробка замовлень: імпорт нових замовлень, зміна статусів, відправлення даних про доставку;
- аналітика: отримання звітів про продажі, популярність товарів і поведінку клієнтів [10].

Основні функції кабінету продавця зображено на рис. 1.4 [11].

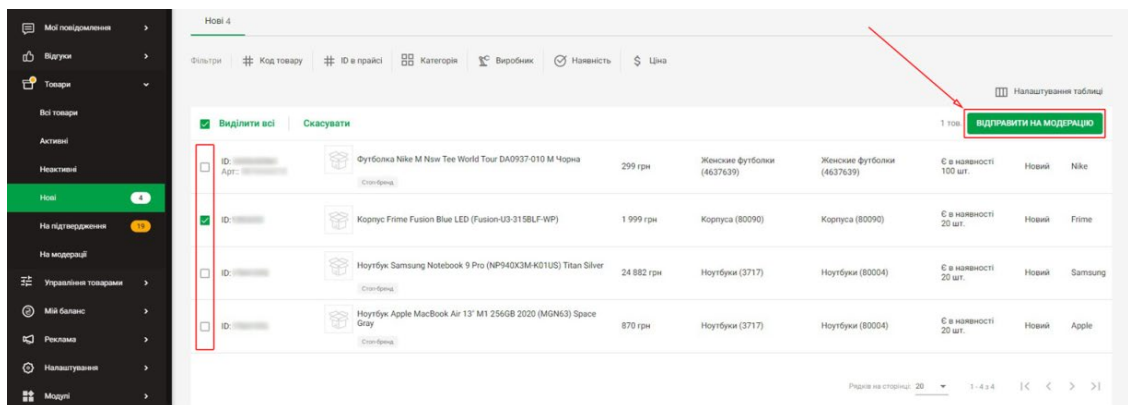


Рисунок 1.4 – Інтерфейс кабінету продавця Rozetka [11]

Prom.ua є другим за популярністю маркетплейсом в Україні, орієнтованим як на B2C, так і на B2B-сегменти. Платформа дозволяє продавцям створювати власні інтернет-магазини в межах маркетплейсу, а також розмішувати товари в загальному каталозі. Prom.ua охоплює близько 25% ринку e-commerce і має ширший асортимент товарів порівняно з Rozetka, включаючи нішеві категорії, такі як ручна робота чи промислове обладнання [7].

Приклад зовнішнього вигляду маркетплейсу Prom.ua зображено на рис. 1.5 [12].



Рисунок 1.5 – Зовнішній вигляд маркетплейсу Prom.ua [12]

Кабінет продавця Prom.ua підтримує управління товарами, замовленнями, відгуками та рекламними кампаніями. Основною перевагою Prom.ua є гнучкість налаштувань, що дозволяє продавцям створювати унікальні сторінки магазинів із власним брендингом. Проте ручне управління товарами, як і в Rozetka, є трудомістким, особливо при роботі з кількома магазинами чи великою кількістю товарів. Для автоматизації Prom.ua пропонує API, який підтримує:

- управління каталогом: додавання, редагування, видалення товарів, масове оновлення характеристик;
- синхронізація замовлень: імпорт і обробка замовлень, зміна статусів, відправлення даних про доставку;

– керування магазинами: підтримка кількох магазинів в одному акаунті, управління їхніми налаштуваннями [<https://public-api.docs.prom.ua>].

Інтерфейс кабінету зображено на рис. 1.6 [7].

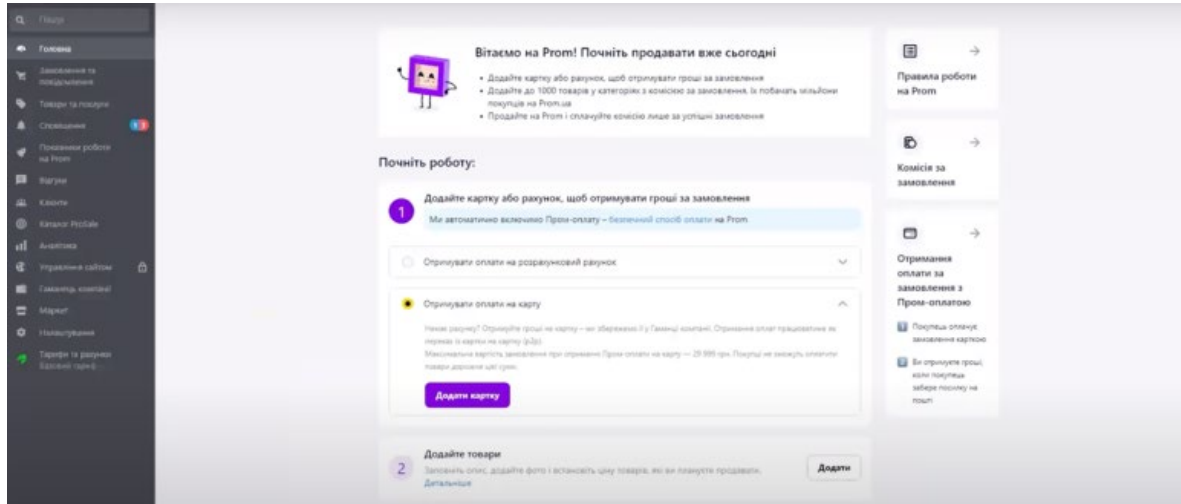


Рисунок 1.6 – Інтерфейс кабінету продавця Prom.ua [7]

Порівняння особливостей маркетплейсів Rozetka та Prom.ua наведене в табл. 1.1 [7].

Таблиця 1.1 – Порівняння характеристик Rozetka та Prom.ua [7]

Характеристика	Rozetka	Prom.ua
Частка ринку	~40%	~25%
Асортимент	Широкий, акцент на електроніку та товари для дому	Широкий, включає нішеві категорії
API	OAuth 2.0, JSON, до 100 запитів/хв	REST, API-токени, до 1000 товарів/запит
Логістика	Власні склади, FBR, Нова Пошта	Нова Пошта, Укрпошта, без власних складів
Управління кількома магазинами	Обмежена підтримка	Повна підтримка, але без синхронізації
Складність інтеграції	Висока (складна документація)	Середня (простіша документація, але збої)

Обидва маркетплейси вимагають регулярного оновлення даних про товари, ціни та залишки, що є критичним для уникнення помилок, таких як

продаж товарів, яких немає в наявності. Однак відмінності в структурі API, частоті оновлення даних і логістичних можливостях ускладнюють створення універсального рішення для роботи з обома платформами одночасно. Наприклад, Rozetka вимагає суворішого дотримання лімітів запитів, тоді як Prom.ua потребує додаткової логіки для управління кількома магазинами [7].

1.4 Аналіз проблем і недоліків управління товарами для маркетплейсів

Аналіз функціонування процесів управління товарами на українських маркетплейсах виявив низку проблем, які впливають на ефективність роботи продавців. Основні проблеми та їхні наслідки наведено в табл. 1.2 [13].

Таблиця 1.2 – Аналіз проблем і їхніх наслідків для управління товарами [13]

Проблема	Наслідки
Відсутність єдиної системи для роботи з кількома маркетплейсами	Подвоєння роботи, помилки в даних, втрата часу
Складність інтеграції з API маркетплейсів	Високі витрати на розробку, обмежена автоматизація
Обмежена підтримка месенджерів для сповіщень	Затримки в обробці замовлень, зниження клієнтської задоволеності
Висока вартість комерційних CRM-систем	Недоступність для малого бізнесу
Недостатня автоматизація складського обліку	Неточності в залишках, скасування замовлень

Ручне введення даних і відсутність синхронізації між маркетплейсами призводять до помилок, таких як продаж товарів, яких немає в наявності, що негативно впливає на репутацію продавця. Крім того, обмеження API (наприклад, ліміт запитів) ускладнюють реальну синхронізацію даних.

Проблема посилюється відсутністю доступних рішень із підтримкою Telegram-ботів для оперативного інформування продавців [13].

1.5 Постановка задач дипломної роботи

На основі проведеного аналізу предметної області сформульовано такі задачі дипломної роботи:

- провести детальний аналіз вимог до системи управління товарами, враховуючи особливості API Rozetka та Prom.ua;
- розробити архітектуру програмного забезпечення, що забезпечує інтеграцію з маркетплейсами, управління базою даних і зручний вебінтерфейс;
- обрати оптимальні технології для реалізації системи, включаючи мову програмування, фреймворк і СУБД;
- реалізувати програмний застосунок із функціоналом управління товарами, обробки замовлень, синхронізації даних і сповіщеннями через Telegram-бот;
- провести тестування системи на відповідність вимогам і забезпечити її стабільну роботу;
- скласти документацію для програмістів і користувачів, що описує функціонал і використання системи.

Метою роботи є створення програмного застосунку, який автоматизує управління товарами для українських маркетплейсів, підвищуючи ефективність роботи продавців і зменшуючи витрати часу на рутинні операції.

1.6 Висновки до розділу

Аналіз предметної області показав, що електронна комерція в Україні активно розвивається, а маркетплейси Rozetka та Prom.ua є ключовими

каналами продажів. Проте відсутність доступних і простих у використанні систем для управління товарами створює проблеми для продавців, такі як помилки в даних, затримки в обробці замовлень і високі витрати на комерційні рішення.

Визначено основні вимоги до розробки програмного застосунку, що включають інтеграцію з API маркетплейсів, зручний вебінтерфейс і підтримку сповіщень через Telegram.

Сформульовані задачі роботи спрямовані на створення ефективного інструменту для автоматизації процесів управління товарами.

2 ПРОЄКТУВАННЯ СИСТЕМИ УПРАВЛІННЯ ТОВАРАМИ

2.1 Аналіз функціональних та нефункціональних вимог

Проектування програмного забезпечення для панелі керування товарами базується на детальному аналізі потреб користувачів та задач, визначених у попередньому розділі роботи. В результаті цього аналізу було сформовано перелік функціональних та нефункціональних вимог, яким має відповідати розроблювана система.

Функціональні вимоги визначають основні функції, які повинна виконувати система для досягнення поставленої мети. До ключових функціональних вимог належать:

- інтеграція з маркетплейсами. Система повинна забезпечувати стабільне та безпечне підключення до API українських маркетплейсів Prom.ua та Rozetka. Передбачається можливість налаштування параметрів підключення, включаючи введення ключів доступу (API токенів або облікових даних);
- підтримка множинності магазинів. Система має підтримувати можливість підключення та одночасної роботи з декількома обліковими записами магазинів на платформі Prom.ua;
- автоматизована обробка замовлень. Реалізація механізму автоматичного відстеження нових замовлень на підключених маркетплейсах та їх автоматичне прийняття;
- синхронізація даних про товари. Забезпечення двосторонньої синхронізації інформації про товари, включаючи залишки на складі, між внутрішньою базою даних системи та підключеними магазинами на маркетплейсах. Будь-які зміни кількості товарів в базі даних системи або внаслідок оформлення замовлення на маркетплейсі мають автоматично відображатися на всіх платформах;

- управління базою даних товарів. Надання користувачеві можливості вести централізовану базу даних усіх товарів. Це включає функції додавання нових товарних позицій, редагування існуючих даних (опис, кількість, посилання на маркетплейсах тощо) та видалення товарів з бази даних;

- ініціалізація бази даних. Можливість початкового завантаження даних про товари з одного з підключених магазинів (наприклад, Prom.ua) до внутрішньої бази даних системи;

- система сповіщень. Реалізація механізму оперативного сповіщення користувача про надходження нових замовлень. Канал сповіщення має бути зручним та легкодоступним (наприклад, інтеграція з Telegram-ботом для надсилання повідомлень, що містять ключові дані про замовлення);

- вебінтерфейс користувача. Розробка інтуїтивно зрозумілого та зручного вебінтерфейсу для взаємодії з системою. Інтерфейс має надавати доступ до функцій управління товарами, перегляду статусу синхронізації та інших ключових параметрів системи з будь-якого пристрою через веб-браузер.

Окрім функціональних можливостей, розроблювана система має відповідати певним нефункціональним вимогам, що визначають якісні характеристики її роботи:

- надійність. Система повинна демонструвати високу стабільність роботи, мінімізувати збої та забезпечувати коректну обробку даних навіть в умовах високого навантаження або тимчасових проблем зі зв'язком з маркетплейсами;

- продуктивність. Операції із синхронізації даних та обробки замовлень мають виконуватися швидко, забезпечуючи актуальність інформації на всіх платформах;

- зручність використання. Вебінтерфейс має бути простим та зрозумілим для користувачів без спеціалізованих технічних знань, забезпечуючи легкий доступ до всіх функцій управління;
- безпека. Система повинна забезпечувати захист конфіденційних даних користувачів та інформації про їхні магазини, включаючи безпечне зберігання API ключів та облікових даних;
- масштабованість. Архітектура системи повинна передбачати можливість подальшого розширення функціоналу та інтеграції з іншими торговельними платформами або сервісами.

Виконання цих вимог дозволить створити ефективний та зручний інструмент для автоматизації управління продажами на українських маркетплейсах, що сприятиме оптимізації бізнес-процесів для їх користувачів.

2.2 Проєктування архітектури програмного забезпечення

Для реалізації системи обрано архітектурний шаблон Model-View-Controller (MVC), який забезпечує чіткий розподіл відповідальностей і полегшує масштабування та підтримку. Структура MVC адаптована до потреб системи:

- Model (Модель): відповідає за управління даними, включаючи базу даних для зберігання інформації про товари, замовлення та користувачів, а також логіку синхронізації з API маркетплейсів;
- View (Представлення): реалізує вебінтерфейс для взаємодії користувачів із системою, відображаючи каталоги товарів, замовлення та сповіщення;
- Controller (Контролер): обробляє запити користувачів, координує взаємодію між моделлю та представленням, а також керує викликами до API маркетплейсів і Telegram-бота.

Таким чином, розроблювана система управління товарами для інтеграції з українськими маркетплейсами має модульну архітектуру, що забезпечує гнучкість, масштабованість та полегшує подальшу підтримку та розвиток. Структура системи побудована таким чином, щоб забезпечити ефективну взаємодію між внутрішніми компонентами та зовнішніми сервісами, такими як API маркетплейсів та служба сповіщень.

На високому рівні архітектура системи може бути представлена у вигляді сукупності взаємодіючих модулів:

- модуль взаємодії з маркетплейсами. Цей компонент відповідає за зв'язок із зовнішніми торговельними платформами Prom.ua та Rozetka через їхні програмні інтерфейси API. Його основні функції включають надсилання запитів для отримання інформації про нові замовлення та товари, а також надсилання даних для оновлення інформації про товарні залишки та статуси замовлень на відповідних майданчиках. Модуль інкапсулює логіку роботи з API кожного маркетплейсу, адаптуючи внутрішні дані системи до форматів, що вимагаються зовнішніми сервісами;

- модуль управління базою даних. Цей модуль є центральним сховищем даних системи. Він відповідає за зберігання інформації про товари, їхню кількість, посилання на відповідні позиції на маркетплейсах та іншу необхідну інформацію. Модуль надає інтерфейс для додавання, редагування, видалення та отримання даних про товари. Взаємодія з базою даних відбувається через цей модуль, забезпечуючи цілісність та консистентність збереженої інформації;

- серверна частина. Цей компонент виконує роль координатора в системі. Він ініціалізує та управляє роботою інших модулів, обробляє запити від вебінтерфейсу користувача, запускає процеси автоматичного відстеження замовлень та синхронізації даних. Серверна частина реалізує бізнес-логіку системи та забезпечує взаємодію між модулем маркетплейсів, модулем бази даних та модулем сповіщень;

- модуль сповіщень. Цей компонент відповідає за генерацію та надсилання повідомлень користувачеві про важливі події, зокрема про надходження нових замовлень. Він взаємодіє із зовнішніми службами сповіщень, наприклад Telegram Bot API, для доставки інформації користувачеві в зручному форматі;

- користувацький інтерфейс. Цей компонент представляє собою набір вебсторінок, через які користувач взаємодіє із системою. Він забезпечує візуалізацію даних про товари, надає форми для додавання та редагування інформації, дозволяє ініціювати певні операції. Інтерфейс взаємодіє з основним керуючим модулем (серверною частиною) для отримання та надсилання даних.

Схема архітектури системи наведена на рис. 2.1.

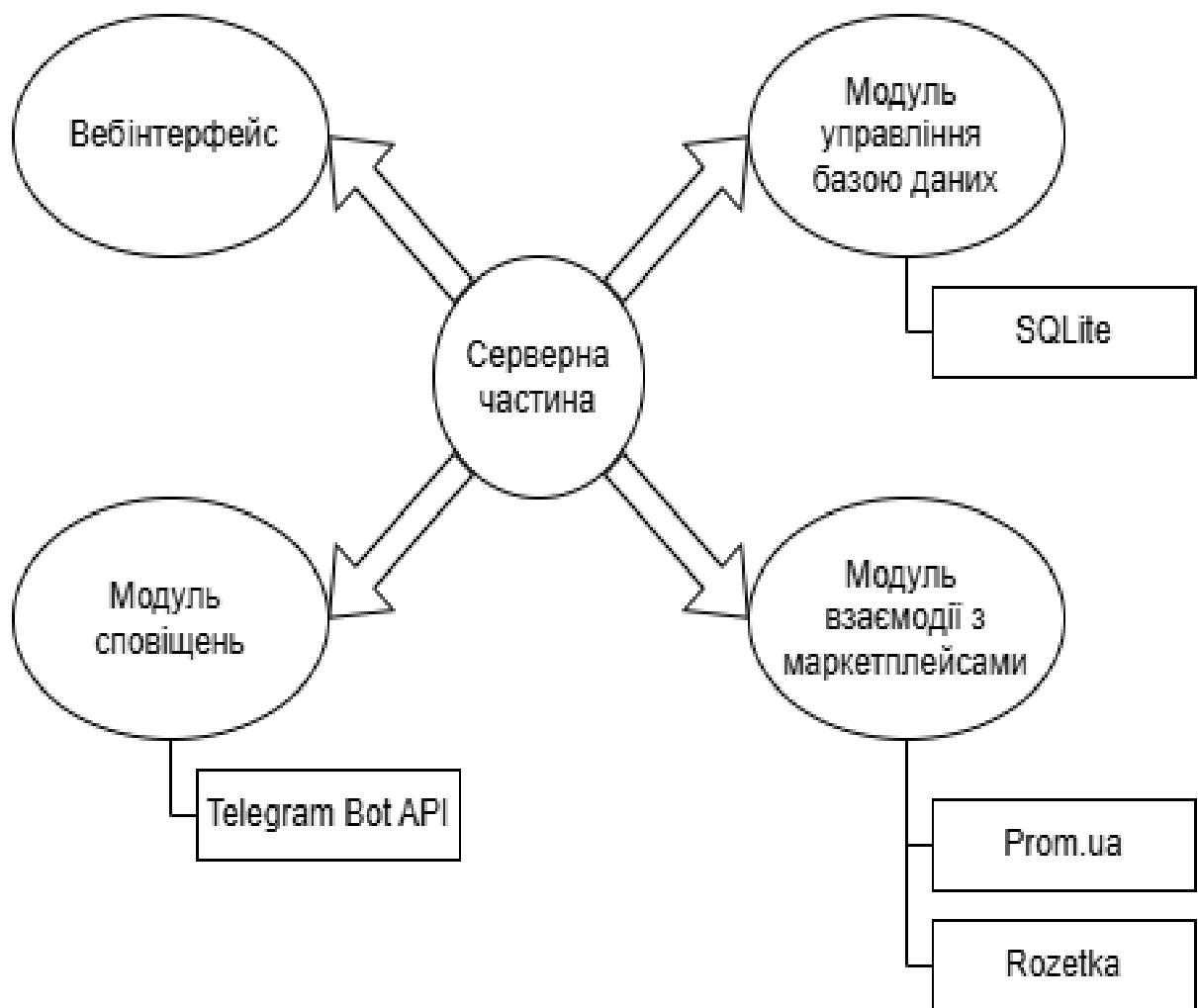


Рисунок 2.1 – Схема архітектури системи

Взаємодія між модулями відбувається наступним чином: основний керуючий модуль отримує дані про нові замовлення через модуль взаємодії з маркетплейсами. Ці дані обробляються, оновлюється інформація в базі даних через модуль управління БД, ініціюється синхронізація даних про залишки через модуль маркетплейсів, і надсилається сповіщення користувачеві через модуль сповіщень. Користувач взаємодіє з системою через вебінтерфейс, який надсилає запити до основного керуючого модуля для виконання операцій з базою даних або отримання актуальної інформації.

2.3 Вибір та обґрунтування засобів реалізації програмного продукту

Ефективна реалізація програмного забезпечення для панелі керування товарами потребує ретельного підходу до вибору відповідного стеку технологій та інструментарію. Цей процес включає ідентифікацію оптимальної мови програмування, вибір зручного середовища розробки, визначення системи керування базами даних, яка відповідає потребам проєкту, вибір відповідного вебфреймворку для побудови серверної частини та інтерфейсу, а також вибір надійного каналу для сповіщень користувачів. Кожен з цих елементів впливає на швидкість розробки, продуктивність системи, її масштабованість та зручність використання

Середовище розробки (IDE) є інтегрованим комплексом інструментів, що значно оптимізує процес написання, налагодження та тестування коду. При виборі IDE для розробки важливі такі аспекти, як підтримка синтаксису мов, наявність функцій автодоповнення, інструментів для налагодження, інтеграція з системами контролю версій та можливість встановлення розширень [14].

Було розглянуто декілька популярних IDE для визначення переваг, недоліків та ключових можливостей. Порівняльний огляд деяких з них наведено у табл. 2.1 [15].

Таблиця 2.1 – Порівняння популярних середовищ розробки [15]

Середовище розробки	Опис та ключові можливості	Переваги	Недоліки
1	2	3	4
Sublime Text	Легкий та швидкий текстовий редактор з можливістю розширення функціоналу через плагіни.	Висока швидкість роботи, мінімалістичний інтерфейс, широка підтримка мов через пакети.	Менш функціональний порівняно з повноцінними IDE, базові можливості налагодження.
Visual Studio Code (VS Code)	Безкоштовний, багатофункціональний редактор коду з потужною системою розширень. Має вбудований термінал, підтримку Git, засоби для налагодження.	Універсальність (підтримка багатьох мов), гнучкість налаштування, велика кількість якісних розширень, активна спільнота.	Може бути дещо повільнішим за Sublime Text на дуже великих проєктах.
Notepad++	Простий редактор коду для платформи Windows. Підтримує синтаксис багатьох мов.	Легкість, швидкість, простота використання.	Обмежена функціональність порівняно з повноцінними IDE, відсутність вбудованих інструментів для налагодження та керування проєктами.
PyCharm	Спеціалізоване інтегроване середовище розробки для Python. Надає можливості аналізу коду, налагодження, тестування, підтримку вебфреймворків Python	Розширені можливості для розробки на Python, потужні інструменти для рефакторингу та аналізу коду.	Переважно орієнтований на Python, може бути менш зручним для роботи з іншими мовами. Платна версія для доступу до повного функціоналу.

Продовження таблиці 2.1

1	2	3	4
Atom	Редактор коду з відкритим вихідним кодом, розроблений GitHub. Гнучкий та налаштовуваний.	Можливість глибокого налаштування, інтеграція з Git, велика кількість пакетів.	Може бути ресурсоємним, швидкість роботи може поступатися Sublime Text або VS Code.

Таким чином, Sublime Text, це легкий і швидкий текстовий редактор коду, який цінується за високу продуктивність та мінімалістичний інтерфейс. Він підтримує багато мов програмування за допомогою плагінів і підходить для розробників, які шукають ефективний інструмент без зайвих функцій. Приклад зовнішнього вигляду інтерфейсу Sublime Text наведено на рис 2.2 [15].

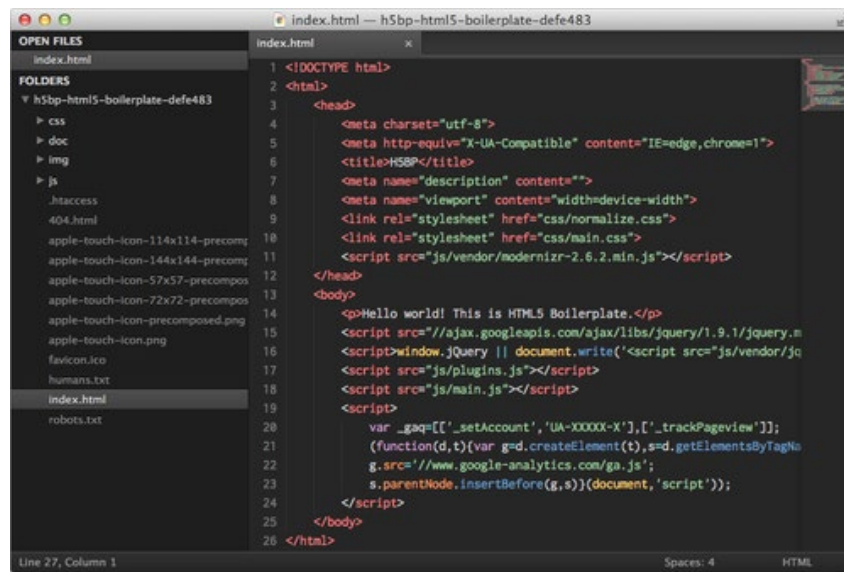


Рисунок 2.2 – Приклад зовнішнього вигляду інтерфейсу Sublime Text [15]

Visual Studio Code (VS Code), це безкоштовний і потужний редактор коду від Microsoft, що став дуже популярним завдяки своїй гнучкості, великій кількості розширень для різних мов та технологій, вбудованому терміналу та підтримці Git. Є універсальним рішенням для розробки на

Python, JavaScript та багатьох інших мовах. Приклад зовнішнього вигляду інтерфейсу VS Code наведено на рис 2.3 [15].

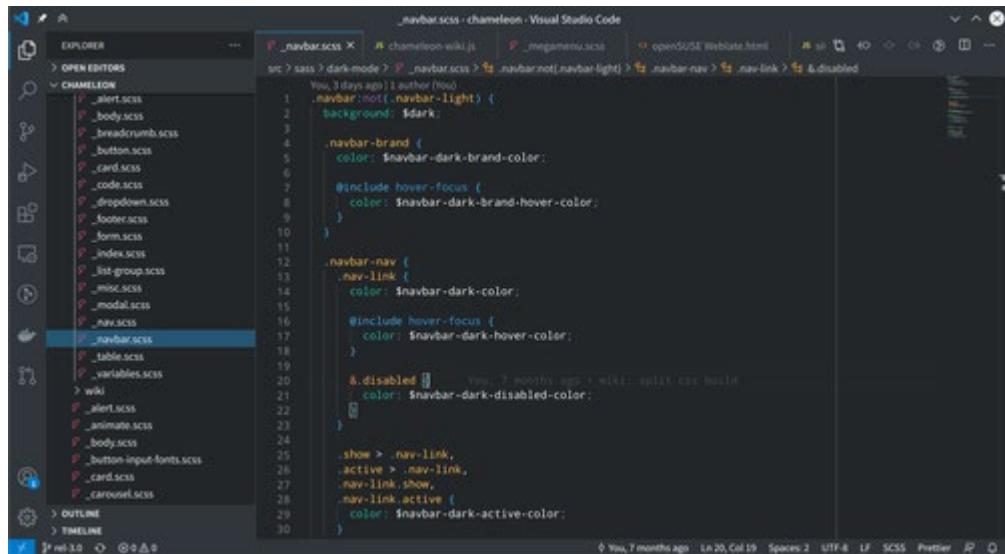


Рисунок 2.3 – Приклад зовнішнього вигляду інтерфейсу VS Code [15]

Notepad++, це простий та ефективний редактор коду, особливо поширений серед користувачів Windows. Він підтримує підсвічування синтаксису для багатьох мов, але має більш базовий функціонал порівняно з повноцінними IDE. Приклад зовнішнього вигляду інтерфейсу Notepad++ наведено на рис 2.4 [15].

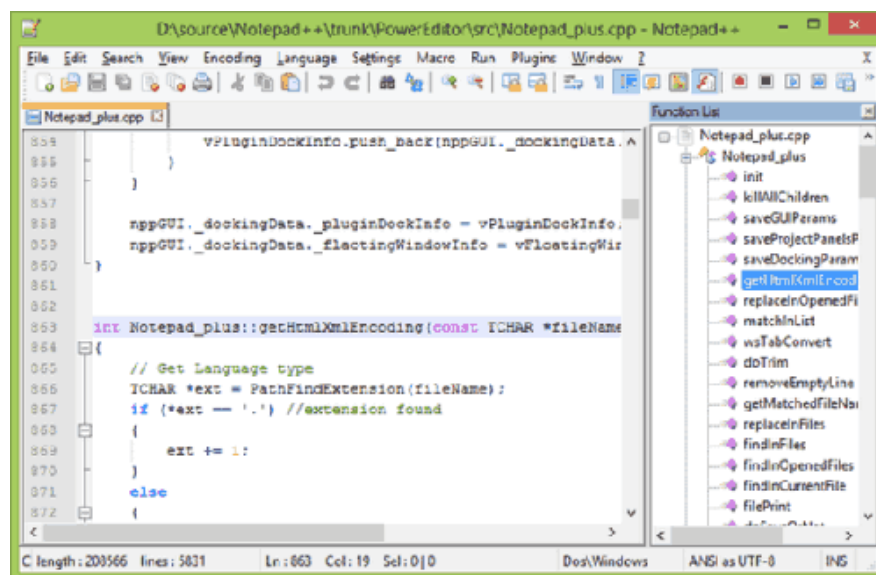


Рисунок 2.4 – Приклад зовнішнього вигляду інтерфейсу Notepad++ [15]

PyCharm, це спеціалізоване та повнофункціональне IDE, розроблене JetBrains спеціально для Python. Надає розширені інструменти для аналізу коду, налагодження, тестування та підтримки фреймворків Python, що робить його ідеальним для професійної розробки на цій мові. Приклад зовнішнього вигляду інтерфейсу PyCharm наведено на рис 2.5 [16].

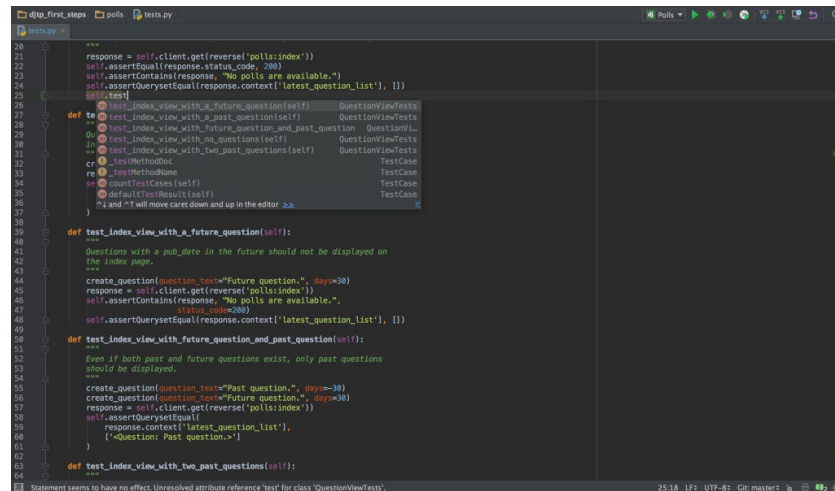


Рисунок 2.5 – Приклад зовнішнього вигляду інтерфейсу PyCharm [16]

Atom, це редактор коду з відкритим вихідним кодом, створений GitHub. Відрізняється високою можливістю налаштування та інтеграцією з Git. Підтримує багато мов програмування і має активну спільноту розробників, яка створює багато пакетів. Приклад зовнішнього вигляду інтерфейсу Atom наведено на рис 2.6 [15].



Рисунок 2.6 – Приклад зовнішнього вигляду інтерфейсу Atom [15]

Враховуючи необхідність роботи як з серверною частиною на Python, так і з клієнтською частиною на JavaScript, а також багатий функціонал, гнучкість та безкоштовність, в якості основного середовища розробки було обрано Visual Studio Code.

Для ефективного зберігання, організації та доступу до інформації про товари, замовлення та інші дані системи необхідна система керування базами даних. Вибір СКБД залежить від типу даних, який необхідно зберігати, вимог до обсягу даних, швидкості доступу та необхідності масштабування. Розглядалися як реляційні, так і NoSQL бази даних. Порівняння деяких з них наведено у табл. 2.2 [17].

Таблиця 2.2 – Порівняння систем керування базами даних [17]

СКБД	Тип	Опис та ключові особливості	Переваги	Недоліки
1	2	3	4	5
MongoDB	NoSQL	Зберігає дані у форматі JSON-подібних документів. Гнучка схема даних, добре підходить для роботи з неструктурованими даними.	Гнучкість схеми, горизонтальна масштабованість, висока продуктивність для певних типів операцій.	Менш підходить для задач, де потрібна суворі цілісність даних та складні зв'язки між сутностями (як у реляційних БД).
PostgreSQL	Об'єктно-реляційна	Потужна та розширювана СКБД з високою надійністю та підтримкою складних типів даних. Відповідає стандартам SQL.	Висока надійність, підтримка ACID-транзакцій, більші можливості для роботи зі складними структурами даних, активна розробка.	Може бути дещо складнішою в налаштуванні та адмініструванні порівняно з MySQL.

Продовження таблиці 2.2

1	2	3	4	5
Redis	NoSQL	Високопродуктивне сховище даних типу "ключ-значення", що працює в оперативній пам'яті. Часто використовується як кеш або брокер повідомлень.	Надзвичайно висока швидкість читання/запису даних.	Переважно використовується як кеш або для специфічних завдань, не є основною СКБД для зберігання великих обсягів постійних даних зі складною структурою.
SQLite	Реляційна	Легка вбудована СКБД, що зберігає всю базу даних в одному файлі на диску. Не вимагає окремого серверного процесу. Ідеально підходить для невеликих додатків та автономних систем.	Простота використання та інтеграції, відсутність необхідності в адмініструванні сервера, портативність.	Обмежена масштабованість та продуктивність при високому навантаженні та великій кількості одночасних запитів. Не підходить для розподілених систем.

Для даного проєкту, враховуючи відносно невеликий обсяг даних та відсутність високих вимог до масштабованості та одночасного доступу багатьох користувачів, а також простоту інтеграції з Python та відсутність потреби в окремому сервері баз даних, оптимальним вибором була визначена вбудована СКБД SQLite.

Було розглянуто декілька популярних мов програмування, які часто використовуються у веброзробці та задачах системної інтеграції. Було виявлено наступні переваги та недоліки:

- C++. Висока продуктивність завдяки прямому доступу до апаратних ресурсів; широке використання в системному програмуванні та високонавантажених системах; велика кількість бібліотек і активна спільнота. Складний синтаксис, що ускладнює розробку для початківців; ручне керування пам'яттю, що підвищує ризик помилок; обмежена підтримка швидкої веброботи;

- Java. Кросплатформність завдяки JVM; об'єктно-орієнтований підхід для структурованого коду; велика екосистема бібліотек і фреймворків. Вища витрата ресурсів порівняно з C++; складніший синтаксис порівняно з Python; менш гнучка для швидкої розробки прототипів;

- Python. Простий і читабельний синтаксис, що прискорює розробку; широка підтримка бібліотек для веброботи, API та баз даних; активна спільнота та легкість інтеграції з Telegram Bot API. Нижча продуктивність порівняно з C++ чи Java; динамічна типізація може спричинити помилки під час виконання; обмежена підтримка низькорівневого програмування;

- JavaScript. Ідеальна для створення інтерактивних веб-інтерфейсів; підтримка серверної розробки через Node.js; велика кількість фреймворків, таких як React та Vue.js. Складність асинхронного програмування для новачків; обмежена ефективність для високонавантажених серверних задач; фрагментація через різноманітність бібліотек [18].

Враховуючи специфіку проєкту, що вимагає ефективної роботи з API маркетплейсів, швидкої розробки серверної логіки та обробки даних, Python був обраний як основна мова для реалізації бекенд-частини системи. Його простий синтаксис та багата екосистема бібліотек роблять його ідеальним інструментом для такого типу задач. Для розробки користувацького інтерфейсу (фронтенду) буде використано JavaScript, що є стандартом для створення інтерактивних вебсторінок.

Вебфреймворк є основою для розробки вебдодатків, надаючи структуру та готові компоненти для обробки HTTP-запитів, маршрутизації,

роботи з шаблонами та інших типових завдань веброзробки. Для мови Python існує декілька популярних вебфреймворків з різною філософією. Порівняння двох найбільш відомих, Django та Flask, наведено у табл. 2.3 [19].

Таблиця 2.3 – Порівняння вебфреймворків Python [19]

Фреймворк	Тип	Опис та ключові особливості	Переваги	Недоліки
Django	Повний функціонал	Надає велику кількість вбудованих компонентів та , такі як ORM, система автентифікації, адмін-панель, система шаблонів тощо.	Висока швидкість розгортання повністю функціональних вебдодатків, наявність готових рішень для типових завдань, сильна спільнота та документація.	Менша гнучкість у виборі окремих компонентів, може здаватися надмірним для невеликих проєктів.
Flask	Простий фреймворк	Надає мінімальний набір базових інструментів для веб-розробки. Дозволяє розробнику самостійно обирати та інтегрувати необхідні бібліотеки та компоненти.	Висока гнучкість, легкість освоєння, ідеально підходить для створення невеликих та середніх додатків або мікросервісів. Дозволяє краще зрозуміти принципи роботи вебдодатків.	Потребує самостійного вибору та інтеграції багатьох компонентів, що може збільшити час розробки для складних проєктів.

Приклад синтаксисів при використанні фреймворків Flask і Django наведено на рис 2.7 [19].

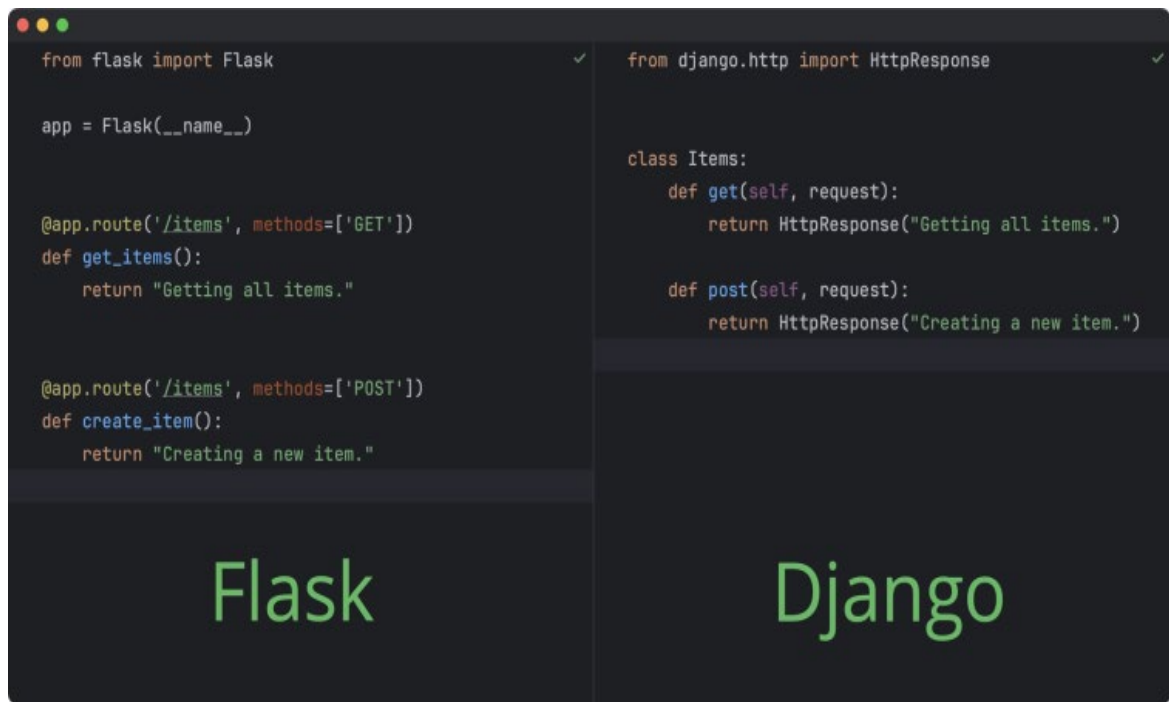


Рисунок 2.7 – Приклад синтаксисів при використанні фреймворків Flask і Django [19]

Для даного проєкту, зважаючи на необхідність у гнучкості при виборі компонентів та відносно невеликий обсяг функціоналу веб-інтерфейсу, був обраний мікрофреймворк Flask. Він надає достатній функціонал для реалізації поставлених задач, дозволяючи при цьому інтегрувати саме ті бібліотеки, які необхідні для специфічних потреб взаємодії з маркетплейсами та базою даних.

Для оперативного інформування користувача про надходження нових замовлень, а також іншої службової інформації та важливі події системи необхідно реалізувати надійний канал сповіщень. Вибір інструменту для надсилання сповіщень визначається вимогами до швидкості доставки повідомлень, надійності, вартості та зручності для кінцевого користувача. В зв'язку з цим було розглянуто декілька основних варіантів засобів надсилання сповіщень, які часто використовуються для автоматичних сповіщень для визначення особливостей, переаг та недоліків. Порівняння характеристика розглянутих варіантів наведено у табл. 2.4 [20].

Таблиця 2.4 – Порівняння інструментів для сповіщень [20]

Інструмент сповіщень	Опис	Переваги	Недоліки
Телеграм-боти	Автоматизовані акаунти в месенджері Telegram, що можуть надсилати повідомлення та взаємодіяти з користувачем за заданим алгоритмом.	Висока швидкість доставки повідомлень, підтримка різних типів контенту (текст, зображення), можливість інтеграції з іншими сервісами, двостороння комунікація, як правило, безкоштовно.	Необхідність наявності встановленого месенджера Telegram та облікового запису у користувача. Ризик блокування бота при порушенні правил платформи.
SMS - повідомлення	Короткі текстові повідомлення, що надсилаються на номер мобільного телефону.	Широке охоплення аудиторії (доступні на будь-якому мобільному телефоні), простота використання.	Можуть бути платними (вартість залежить від оператора та обсягу повідомлень). Обмежений обсяг інформації в одному повідомленні. Ризик потрапляння до спаму.
Електронна пошта	Надсилання повідомлень на адресу електронної пошти користувача.	Можливість надсилання детальної інформації, повідомлення зберігаються в поштової скриньці.	Можливі затримки в доставці, що робить їх менш придатними для термінових сповіщень. Ризик потрапляння до папки "Спам". Не всі користувачі регулярно перевіряють електронну пошту.

Зважаючи на критичну важливість оперативного інформування продавця про нові замовлення, а також зручність, функціональність та економічну вигідність, в якості основного інструменту для сповіщень було

обрано Телеграм-бота. Він дозволяє швидко та надійно доставляти необхідну інформацію користувачеві.

Таким чином, вибір засобів розробки базувався на детальному аналізі вимог проєкту та порівнянні доступних технологій. Обраний стек (Python, JavaScript, VS Code, SQLite, Flask, Telegram Bot API) є оптимальним для ефективної та швидкої реалізації поставлених задач дипломної роботи.

2.4 Висновки до розділу

В ході досліджень було здійснено комплексне проєктування програмного забезпечення панелі керування товарами для інтеграції з українськими маркетплейсами Prom.ua та Rozetka. На основі детального аналізу предметної області та сформульованих функціональних і нефункціональних вимог, було визначено ключові аспекти майбутньої системи.

Проведено обґрунтований вибір засобів розробки програмного забезпечення, включаючи визначення оптимальної мови програмування, середовища розробки, системи керування базами даних, веб-фреймворку та інструментів для реалізації системи сповіщень. При виборі враховувалися вимоги до продуктивності, зручності використання та можливості ефективної взаємодії з зовнішніми сервісами. Таким чином, Python обрано як основну мову програмування завдяки простоті та підтримці бібліотек для роботи з API. Фреймворк Flask забезпечує легкість створення веб-інтерфейсу, а SQLite – гнучкість для тестування та продакшену. Інтеграція з Telegram Bot API дозволяє реалізувати оперативні сповіщення, що відповідає потребам продавців.

Архітектура системи розроблена на основі шаблону MVC, що забезпечує модульність, легкість підтримки та масштабування. Основні компоненти — вебінтерфейс, серверна логіка, база даних, модуль інтеграції з маркетплейсами та Telegram-бот — чітко розмежовані.

Таким чином, даний розділ завершує етап проєктування системи, деталізуючи вимоги та визначаючи технологічні рішення для створення панелі керування товарами.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Алгоритм роботи системи

Функціонування системи поділяється на два основні етапи: ініціалізація під час першого запуску та основний цикл роботи, який забезпечує безперервну взаємодію з маркетплейсами, базою даних і користувачем. Алгоритми розроблено з урахуванням вимог до автоматизації менеджменту товарів, обробки замовлень і синхронізації даних.

На етапі початкового налаштування, тобто етапу ініціалізації, система виконує ряд дій для підготовки до роботи:

- налаштування Telegram-бота: користувачу пропонується ввести токен Telegram-бота та унікальний ідентифікатор чату (Chat ID), куди надсилатимуться сповіщення про нові замовлення, зміни статусів або критичні помилки. Це забезпечує оперативне інформування продавця;
- аутентифікація для Rozetka: користувач вводить облікові дані, такі як логін і пароль, для маркетплейсу Rozetka, що дозволяє системі згенерувати токен доступу до API. Токен зберігається безпечно для подальших запитів;
- підключення магазинів Prom.ua: користувач вказує кількість магазинів на платформі Prom.ua, з якими планує працювати. Для кожного магазину необхідно ввести унікальний API-токен, який забезпечує доступ до даних про товари та замовлення;
- ініціалізація бази даних: система пропонує імпортувати товари з одного з підключених магазинів Prom.ua або Rozetka для початкового наповнення бази даних. Ця функція дозволяє швидко синхронізувати асортимент, включаючи назви товарів, описи, ціни та залишки. Якщо база даних уже існує, система перевіряє її цілісність і оновлює структуру за необхідності;
- перевірка підключення: після введення всіх даних система тестує зв'язок із API маркетплейсів і Telegram-ботом, повідомляючи користувача

про успішне завершення налаштування або можливі помилки, наприклад некоректний токен.

Цей етап забезпечує коректну підготовку системи до роботи, створюючи основу для стабільної інтеграції з зовнішніми сервісами. Після ініціалізації система переходить до основного циклу роботи, який включає фонові процеси та взаємодію з користувачем через вебінтерфейс. Основні компоненти циклу:

а) вебінтерфейс. Користувач отримує доступ до інтуїтивного веб-інтерфейсу, реалізованого через Flask і клієнтські скрипти JavaScript. Інтерфейс дозволяє:

- переглядати список товарів у базі даних із можливістю фільтрації за категоріями, ціною або залишками;
- керувати замовленнями: переглядати деталі, змінювати статуси (наприклад, "нове", "в обробці", "відправлено");
- редагувати товари: додавати нові позиції, змінювати описи, ціни, кількість або видаляти записи;
- аналізувати звіти: переглядати статистику продажів, популярність товарів або помилки синхронізації;

б) фонові обробка замовлень. Система періодично надсилає запити до API Rozetka та Prom.ua для перевірки нових замовлень. У разі виявлення нового замовлення виконуються такі дії:

- 1) замовлення автоматично імпортується до локальної бази даних із збереженням деталей (номер, дата, товари, сума, контактні дані покупця);
- 2) оновлюються залишки відповідних товарів у базі даних, зменшуючи кількість на складі;
- 3) система синхронізує залишки з усіма підключеними магазинами, надсилаючи оновлені дані через API;

4) формується сповіщення про нове замовлення, яке надсилається до Telegram-чату із зазначенням номера замовлення, списку товарів і загальної суми;

в) синхронізація даних. Будь-які зміни в базі даних, ініційовані користувачем через вебінтерфейс (наприклад, редагування ціни товару чи додавання нової позиції), автоматично відображаються на маркетплейсах. Алгоритм синхронізації враховує обмеження API (наприклад, ліміти запитів Rozetka) і оптимізує черговість оновлень, щоб уникнути перевантаження;

г) обробка помилок. Система відстежує можливі збої, такі як втрата зв'язку з API маркетплейсів або некоректні дані. У разі помилки користувач отримує сповіщення через Telegram із описом проблеми та рекомендаціями (наприклад, перевірити токен доступу).

Алгоритми розроблено з урахуванням модульної архітектури системи:

- модульність. Кожен процес (обробка замовлень, синхронізація, сповіщення) реалізовано як окремий компонент, що полегшує підтримку та тестування;

- асинхронність. Для підвищення продуктивності використано асинхронні запити до API маркетплейсів, що дозволяє обробляти кілька магазинів одночасно;

- надійність. Алгоритми включають механізми повторних спроб у разі тимчасових збоїв API та перевірку цілісності даних перед збереженням у базі.

Алгоритм функціонування системи управління товарами для українських маркетплейсів наведено на рис. 3.1.

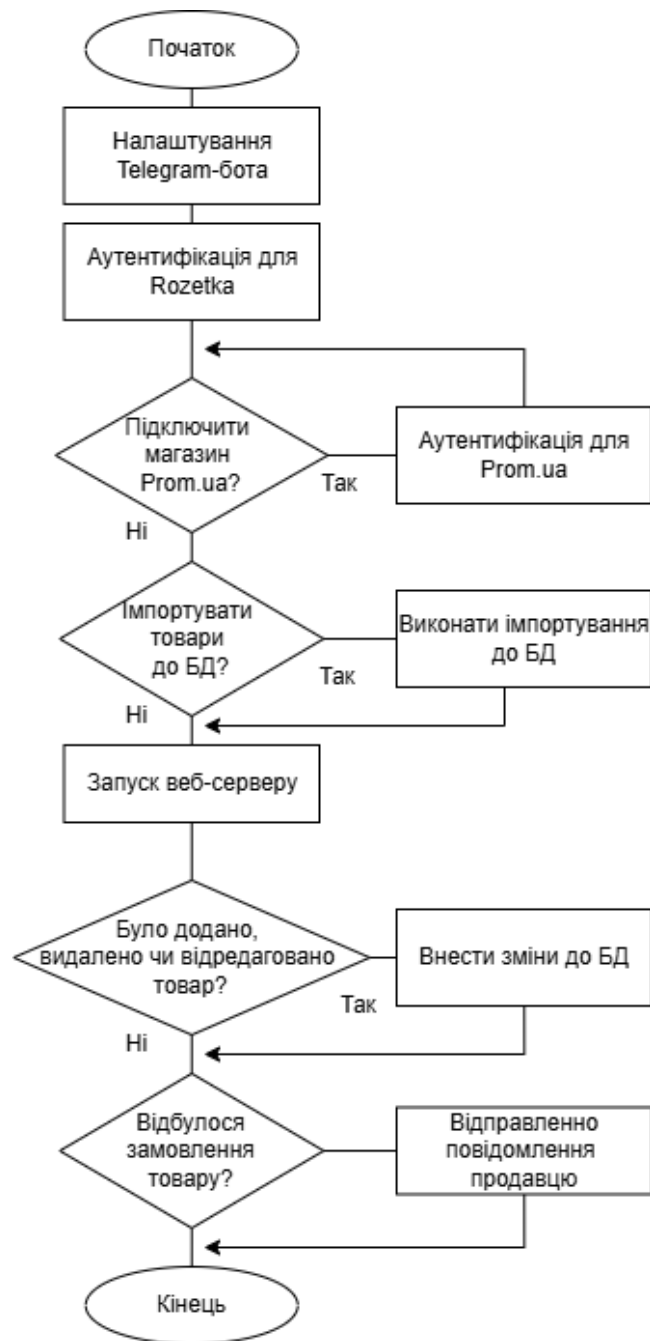


Рисунок 3.1 – Алгоритм роботи системи управління товарами

3.2 Особливості програмної реалізації

Програмне забезпечення системи управління товарами для українських маркетплейсів реалізовано як модульну систему, що базується на архітектурному шаблоні Model-View-Controller (MVC). Основними функціональними блоками системи є:

- серверна частина. Відповідає за обробку запитів, координацію роботи модулів і реалізацію бізнес-логіки;
- управління базою даних. Забезпечує зберігання, оновлення та доступ до інформації про товари, замовлення та налаштування;
- інтеграція з маркетплейсами. Реалізує взаємодію з API Rozetka та Prom.ua для синхронізації даних і обробки замовлень;
- система сповіщень. Надсилає повідомлення про нові замовлення та помилки через Telegram-бот;
- вебінтерфейс. Надає користувачу зручний доступ до функцій системи через браузер.

Реалізація системи виконана з використанням технологій Python (Flask), SQLite, JavaScript, HTML і CSS. Основні компоненти програмного забезпечення розподілені між серверними та клієнтськими файлами, які взаємодіють для забезпечення повного циклу роботи системи. Нижче наведено опис файлів, що становлять кодову базу системи.

Опис елементів файлу index.html наведено в табл. 3.1.

Таблиця 3.1 – Опис елементів файлу index.html

Змінна	Призначення
1	2
inventory_loader	Елемент для відображення лоадера під час завантаження даних.
showSidebar	Кнопка для показу бічної панелі.
app_container	Головний контейнер для бічної панелі та сітки записів.
addForm	Форма додавання з полями для назви, кількості, зображення, ID Prom і Rozetka.
toastNotification	Елемент для відображення тост-повідомлень про дії користувача.
sidebar	Бічна панель із елементами керування (пошук, сортування, додавання).
hideSidebar	Кнопка для приховування бічної панелі.

Продовження таблиці 3.1

1	2
sidebar_header	Заголовок бічної панелі з назвою системи та кнопкою закриття.
sidebar_controls	Контейнер для елементів керування (пошук, сортування, кнопка додавання).
searchInput	Поле введення для пошуку товарів за назвою.
sortSelect	Випадаючий список для вибору критерію сортування (назва, кількість).
sort_options	Радіокнопки для вибору напрямку сортування (зростання, спадання).
addItemBtn	Кнопка для відкриття модального вікна додавання товару.
itemCountLabel	Елемент для відображення загальної кількості записів.
recordsGrid	Сітка для відображення карток товарів.
editModal	Модальне вікно для редагування товару.
editForm	Форма редагування з полями для назви, кількості, зображення, ID Prom і Rozetka.
deleteModal	Модальне вікно для підтвердження видалення товару.
addModal	Модальне вікно для додавання нового товару.

Опис змінних та методів файлу app.py наведено в табл. 3.2.

Таблиця 3.2 – Опис змінних та методів файлу app.py

Змінна	Призначення
1	2
app	Об'єкт класу Flask для створення та управління веб-додатком.
storage_name	Змінна, що зберігає назву файлу бази даних SQLite ("inventory.db").
stock_db	Об'єкт класу StockDatabase для взаємодії з базою даних.
updated_products	Список для відстеження продуктів із зміненням залишком для подальшої синхронізації.

Продовження таблиці 3.2

1	2
create_product	Додає новий продукт до бази даних через POST-запит.
messenger_client	Об'єкт класу MessengerClient для надсилання сповіщень через Telegram.
ecom_apis	Список об'єктів класу EcommerceAPI для взаємодії з магазинами Prom.ua.
marketplace_api	Об'єкт класу MarketplaceAPI для взаємодії з маркетплейсом Rozetka.
thread1	Об'єкт потоку для запуску асинхронної задачі manage_inventory.
thread2	Об'єкт потоку для запуску асинхронної задачі track_orders.
check_inventory_change	Перевіряє, чи змінився залишок товару, та додає продукт до списку оновлень.
get_user_input	Отримує введення від користувача з валідацією та обмеженням спроб.
initialize_credentials	Ініціалізує credentials для Telegram, Prom.ua та Rozetka, повертаючи об'єкти API.
show_dashboard	Рендерить головну сторінку вебінтерфейсу (index.html).
get_products	Повертає список усіх продуктів із бази даних.
update_product	Оновлює дані продукту в базі даних через POST-запит.
delete_product	Видаляє продукт із бази даних за ідентифікатором через POST-запит.
notify_marketplace_order	Формує та надсилає сповіщення про нові замовлення з Rozetka у Telegram.
track_orders	Асинхронно відстежує нові замовлення з Prom.ua та Rozetka, оновлює статуси.
run_async_task	Запускає асинхронну задачу в окремому циклі подій.
manage_inventory	Асинхронно оновлює залишки товарів на маркетплейсах із списку updated_products.
sync_product_data	Синхронізує оновлення залишків продуктів у базі даних для Prom.ua та Rozetka.
notify_order_alert	Формує та надсилає сповіщення замовлень у Telegram.

Продовження таблиці 3.2

1	2
upload_media	Завантажує медіафайл на сервер Telegra.ph і повертає пряме посилання.

Опис змінних та методів файлу messenger.py наведено в табл. 3.3.

Таблиця 3.3 – Опис змінних та методів файлу messenger.py

Змінна	Призначення
MessengerClient	Клас для взаємодії з Telegram Bot API.
_key	Змінна, що зберігає токен Telegram-бота.
_action	Змінна, що зберігає метод API для надсилання повідомлень ("dispatchText").
_group	Змінна, що зберігає ідентифікатор чату для сповіщень.
init	Конструктор класу, ініціалізує клієнта та налаштовує логування.
_init_logger	Налаштовує логування операцій клієнта у файл.
_send_alert	Надсилає повідомлення до Telegram-чату через API.
verify_connection	Перевіряє підключення до Telegram API, повертаючи статус.

Опис змінних та методів файлу marketplace.py наведено в табл. 3.4.

Таблиця 3.4 – Опис змінних та методів файлу marketplace.py

Змінна	Призначення
1	2
MarketplaceAPI	Клас для взаємодії з API Rozetka.
_update_order_state	Оновлює статус замовлення через API Rozetka.
_update_products	Оновлює дані кількох продуктів через API Rozetka.
auth_token	Змінна, що зберігає токен авторизації для API Rozetka.
get_pending_orders	Отримує список нових замовлень із Rozetka із деталями.
_retrieve_statuses	Отримує список статусів замовлень із API Rozetka.

Продовження таблиці 3.4

1	2
init	Конструктор класу, виконує авторизацію та отримує токен доступу.
request_timeout	Змінна, що визначає таймаут для HTTP-запитів (15 секунд).
_execute_request	Виконує HTTP-запит до API Rozetka із обробкою помилок.
adjust_product_inventory	Оновлює залишки товару за ідентифікатором продукту.
_list_all_products	Отримує список усіх продуктів із Rozetka (до 10000 елементів).

Опис змінних та методів `storage_manager.py` наведено в табл. 3.5.

Таблиця 3.5 – Опис змінних та методів файлу `storage_manager.py`

Змінна	Призначення
StockDatabase	Клас для взаємодії з базою даних SQLite.
storage_file	Змінна, що зберігає шлях до файлу бази даних SQLite.
init	Конструктор класу, ініціалізує підключення до бази даних і створює таблиці.
_create_storage	Створює таблицю products у базі даних, якщо вона ще не існує.
add_product	Додає новий продукт до таблиці products із валідацією залишків.
list_all_products	Повертає список усіх продуктів із бази даних у форматі словників.
get_product_by_id	Отримує дані продукту за ідентифікатором.
update_product	Оновлює дані продукту в базі даних із валідацією залишків.
delete_product	Видаляє продукт із бази даних за ідентифікатором.

Опис змінних та методів файлу `script.js` наведено в табл. 3.6.

Таблиця 3.6 – Опис змінних та методів файлу script.js

Змінна	Призначення
items_list	Список усіх товарів, отриманих із сервера.
items_pagination	Список товарів для відображення (з урахуванням пагінації).
filtered_items	Список відфільтрованих і відсортованих товарів.
records_grid	Елемент DOM для сітки карток товарів.
sort_select	Елемент DOM для вибору критерію сортування.
search_input	Елемент DOM для поля пошуку.
edit_modal	Елемент DOM для модального вікна редагування.
delete_modal	Елемент DOM для модального вікна видалення.
confirm_delete	Елемент DOM для кнопки підтвердження видалення.
add_item_btn	Елемент DOM для кнопки додавання товару.
add_modal	Елемент DOM для модального вікна додавання.
sort_direction_radios	Елементи DOM для радіокнопок напрямку сортування.
sidebar	Елемент DOM для бічної панелі.
show_sidebar_btn	Елемент DOM для кнопки показу бічної панелі.
hide_sidebar_btn	Елемент DOM для кнопки приховування бічної панелі.
delete_item_id	Змінна для зберігання ID товару, що видаляється.
show_toast	Відображає тост-повідомлення з текстом на 3 секунди.
render_items	Рендерить картки товарів у сітці на основі items_pagination.
draw_grid	Оновлює сітку товарів, копіюючи відфільтровані дані.
sort_and_filter_items	Виконує сортування та фільтрацію товарів, отримуючи дані з сервера.
deep_copy	Виконує глибоке копіювання об'єкта для уникнення мутацій.

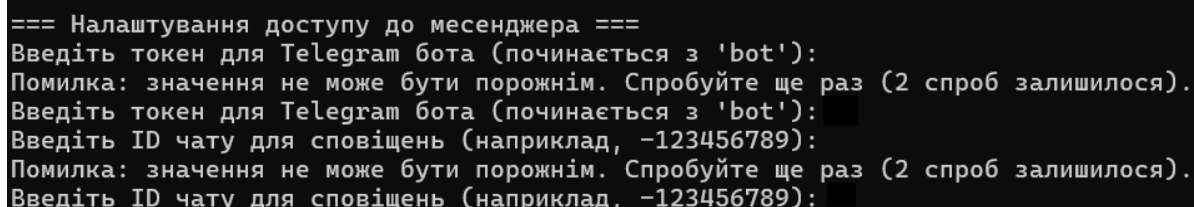
3.3 Тестування програмного забезпечення

Тестування розробленого програмного забезпечення проведено для підтвердження його відповідності функціональним і нефункціональним

вимогам, визначеним вище, включаючи інтеграцію з API маркетплейсів Rozetka та Prom.ua, автоматизовану обробку замовлень, синхронізацію даних, надійність сповіщень через Telegram-бот і зручність вебінтерфейсу. Тестування охоплювало модульне, інтеграційне та користувацьке тестування, що дозволило перевірити коректність роботи всіх компонентів системи, виявити недоліки та оптимізувати продуктивність. Процес тестування поділено на чотири основні етапи: ініціалізація системи, тестування вебінтерфейсу, тестування інтеграції та синхронізації, а також перевірка надійності й продуктивності.

На першому етапі роботи, система пропонує виконати налаштування доступу до месенджера. Для цього необхідно ввести дані токена для Telegram бота, а також ID чату для сповіщень. Валідація системи перевіряє рядок на порожність. Для цього надається 3 спроби. Також в дужках надається підказка.

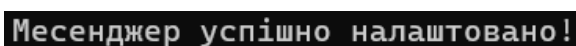
Спроба підключення до системи Telegram бота для отримання сповіщень наведено на рис. 3.2.



```
=== Налаштування доступу до месенджера ===  
Введіть токен для Telegram бота (починається з 'bot'):  
Помилка: значення не може бути порожнім. Спробуйте ще раз (2 спроб залишилося).  
Введіть токен для Telegram бота (починається з 'bot'):  
Введіть ID чату для сповіщень (наприклад, -123456789):  
Помилка: значення не може бути порожнім. Спробуйте ще раз (2 спроб залишилося).  
Введіть ID чату для сповіщень (наприклад, -123456789):
```

Рисунок 3.2 – Спроба підключення до системи Telegram бота для отримання сповіщень

В випадку успішного підключення в консолі користувача буде відображено відповідне повідомлення, як на рис. 3.3. Також Telegram бот сповістить про це в чаті, як на рис. 3.4.



```
Месенджер успішно налаштовано!
```

Рисунок 3.3 – Приклад успішного підключення Telegram бота

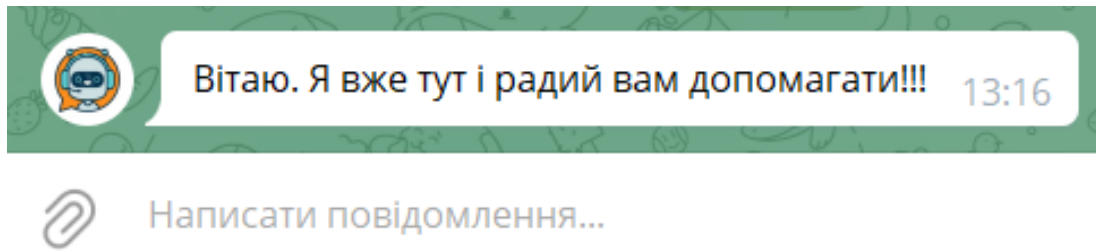


Рисунок 3.4 – Сповідження Telegram бота про початок роботи

На наступному етапі система запропонує обрати кількість магазинів Prom.ua, яку користувач бажає підключити для синхронізації. Процес валідації виключає можливість введення пустого рядку, або підозріло короткого. Приклад наведено на рис. 3.5.

```
=== Налаштування доступу до маркетплейсів ===
Скільки магазинів Prom.ua ви хочете підключити? (введіть число, 0 для пропуску): 1

Налаштування магазину #1
Введіть API-ключ для Prom.ua магазину #1:
Помилка: значення не може бути порожнім. Спробуйте ще раз (2 спроб залишилося).
Введіть API-ключ для Prom.ua магазину #1:
Помилка: API-ключ виглядає невалідним (занадто короткий). Спробуйте ще раз.
Введіть API-ключ для Prom.ua магазину #1:
```

Рисунок 3.5 – Спроба підключити магазин Prom.ua

Як видно з рис. 3.5, є можливість взагалі пропустити цей етап, якщо користувач не бажає підключати магазин Prom.ua. Окрім цього видно, що в дужках надано підказки, а також повідомлення про кількість спроб, що залишилася.

В випадку будь-яких помилок, пов'язаних з підключенням магазину Prom.ua, в консолі буде виведено відповідне повідомлення. Приклад такого повідомлення наведено на рис. 3.6.

```
Помилка: API повернув невалідний JSON для https://my.prom.ua/api/v1/payment_options/list. Отримано: <html>
<head>
  <title>401 Not Authenticated</title>
</head>
<body>
  <h1>401 Not Authenticated</...
Помилка: не вдалося отримати методи оплати від Prom.ua. Використовуються порожні значення.
```

Рисунок 3.6 – Спроба невдалого підключення магазину Prom.ua

Аналогічним чином виконується й підключення до магазину Rozetka. Приклад наведено на рис. 3.7.

```

=== Налаштування доступу до Rozetka ===
Введіть ID користувача для Rozetka:
Помилка: значення не може бути порожнім. Спробуйте ще раз (2 спроб залишилося).
Введіть ID користувача для Rozetka:
Введіть пароль для Rozetka:
Помилка: значення не може бути порожнім. Спробуйте ще раз (2 спроб залишилося).
Введіть пароль для Rozetka:
Rozetka успішно налаштовано!

```

Рисунок 3.7 – Спроба підключити магазин Rozetka

На даному етапі система завершує початкові налаштування. Останнім етапом ініціалізації є пропозиція імпортувати товари з одного з підключених магазинів до бази даних, що актуально для першого запуску. При цьому перевіряється наявність файлу бази даних SQLite. Якщо файл відсутній, створюється нова база даних із відповідною таблицею.

В результаті сервер Flask буде запущено за локальною адресою і портом, що вказується в коді. При переході за посиланням буде відображено панель продавця з усіма доданими картками товарів. Приклад наведено на рис. 3.8.

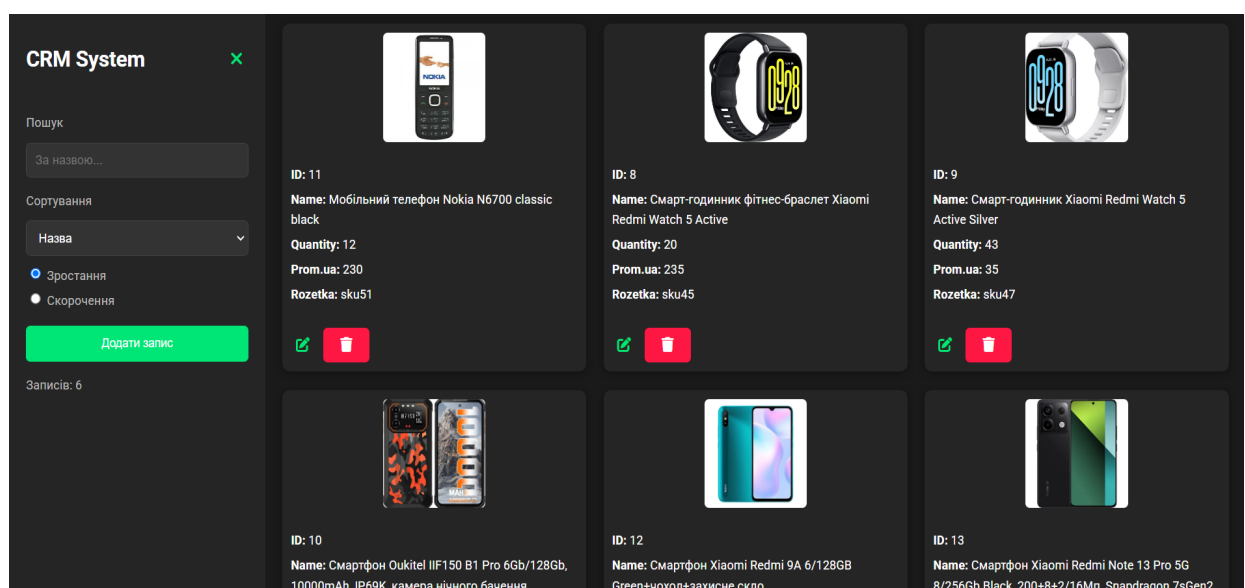


Рисунок 3.8 – Зовнішній вигляд панелі продавця

В системі надається можливість виконувати сортування, як це наведено на рис. 3.9 та 3.10.

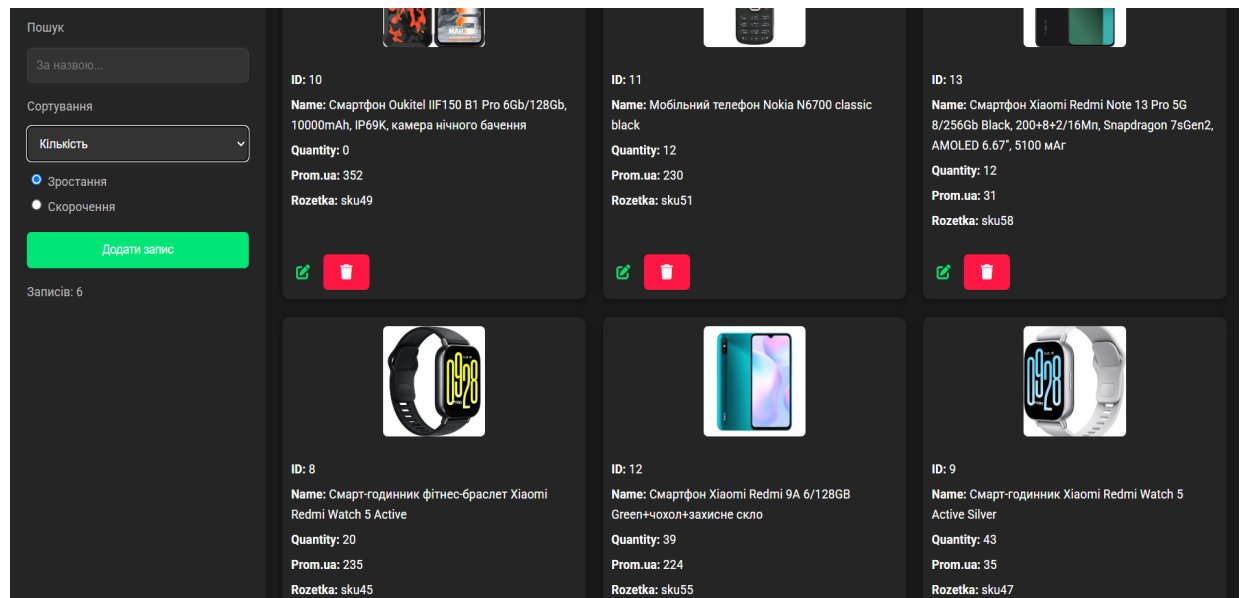


Рисунок 3.9 – Приклад сортування товарів

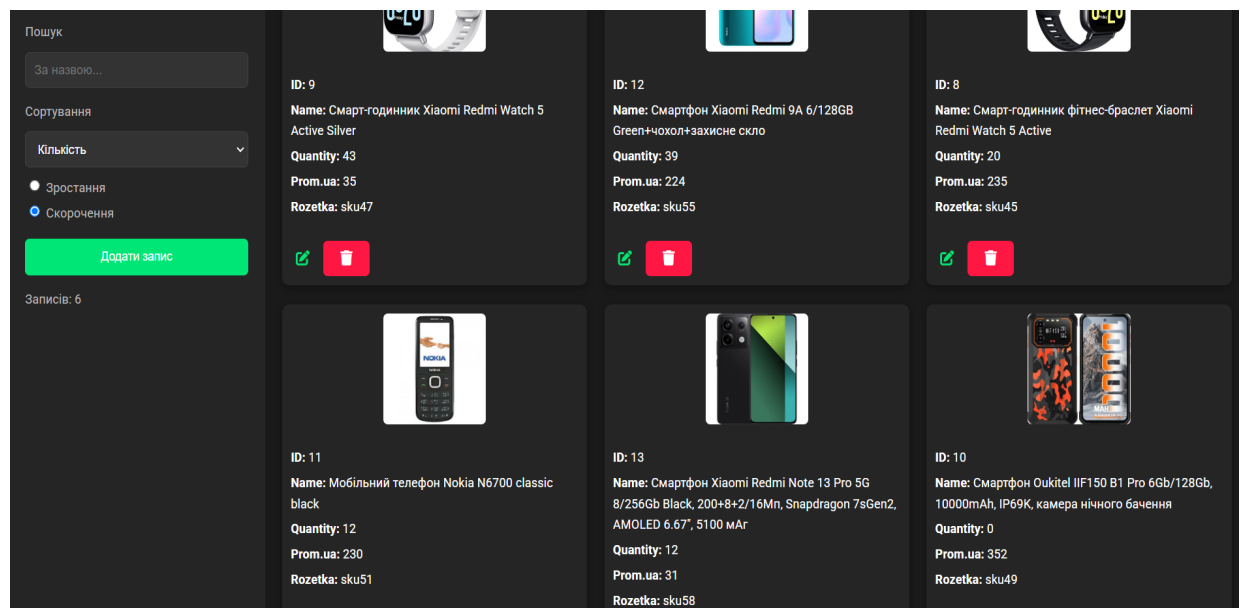


Рисунок 3.10 – Приклад сортування товарів

Також надається можливість фільтрації товарів за частинами слів в назві. Для цього використовується відповідне поле пошуку. Після кожного редагування назви зміни відбуваються динамічно. Приклад такої фільтрації товарів наведено на рис. 3.11.

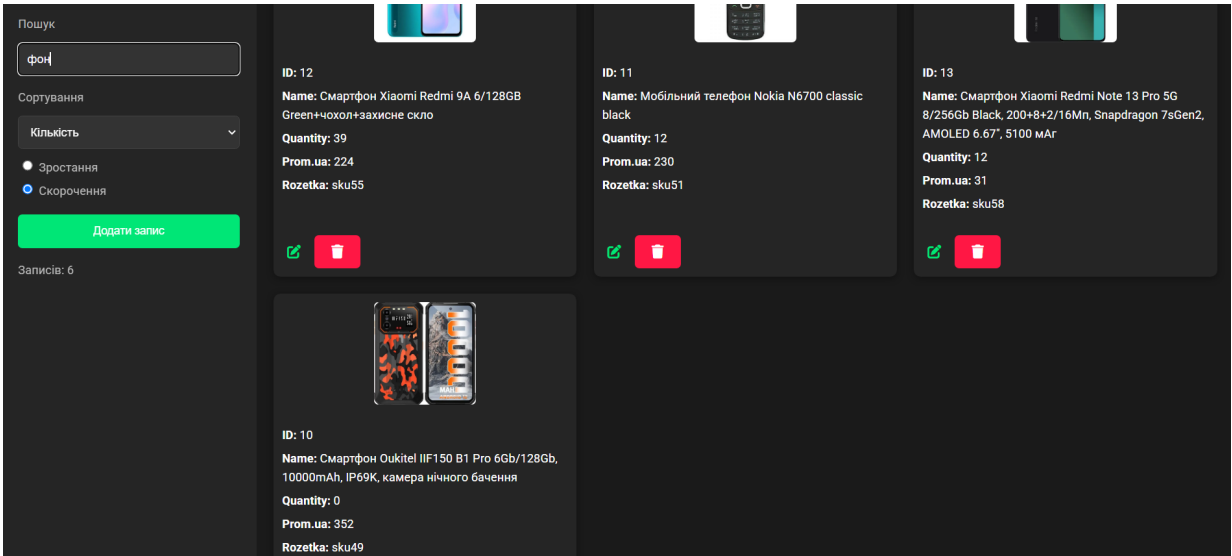


Рисунок 3.11 – Приклад пошуку по назві

Як видно з рис. 3.9-3.11, фільтрація і сортування виконуються одночасно.

Успішне додавання, редагування чи видалення картки товару буде відображено спливаючим повідомленням в правому нижньому куті, як на рис. 3.12.

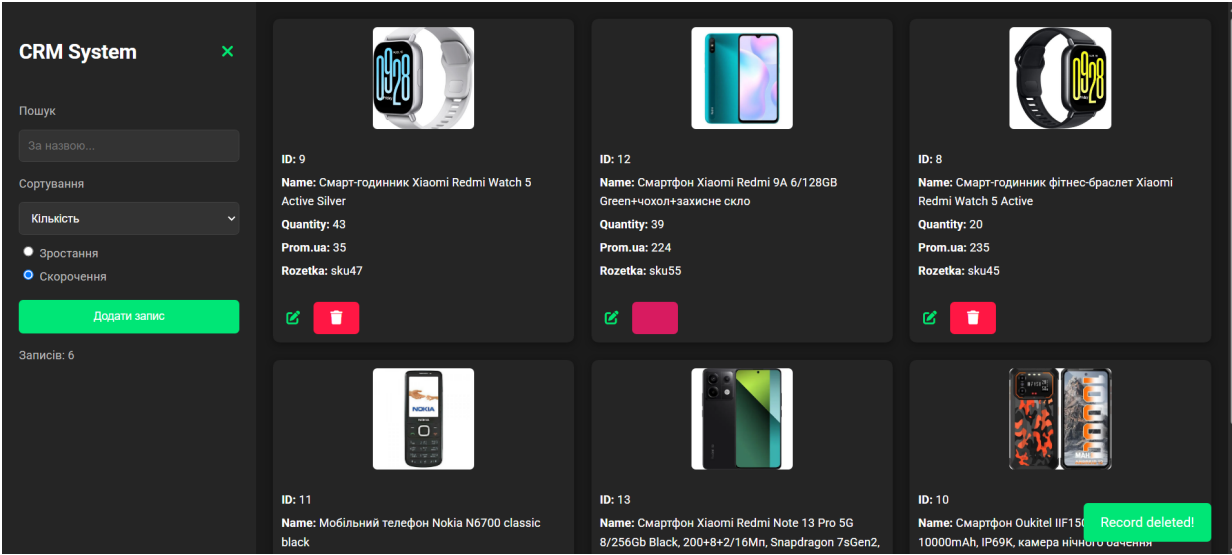
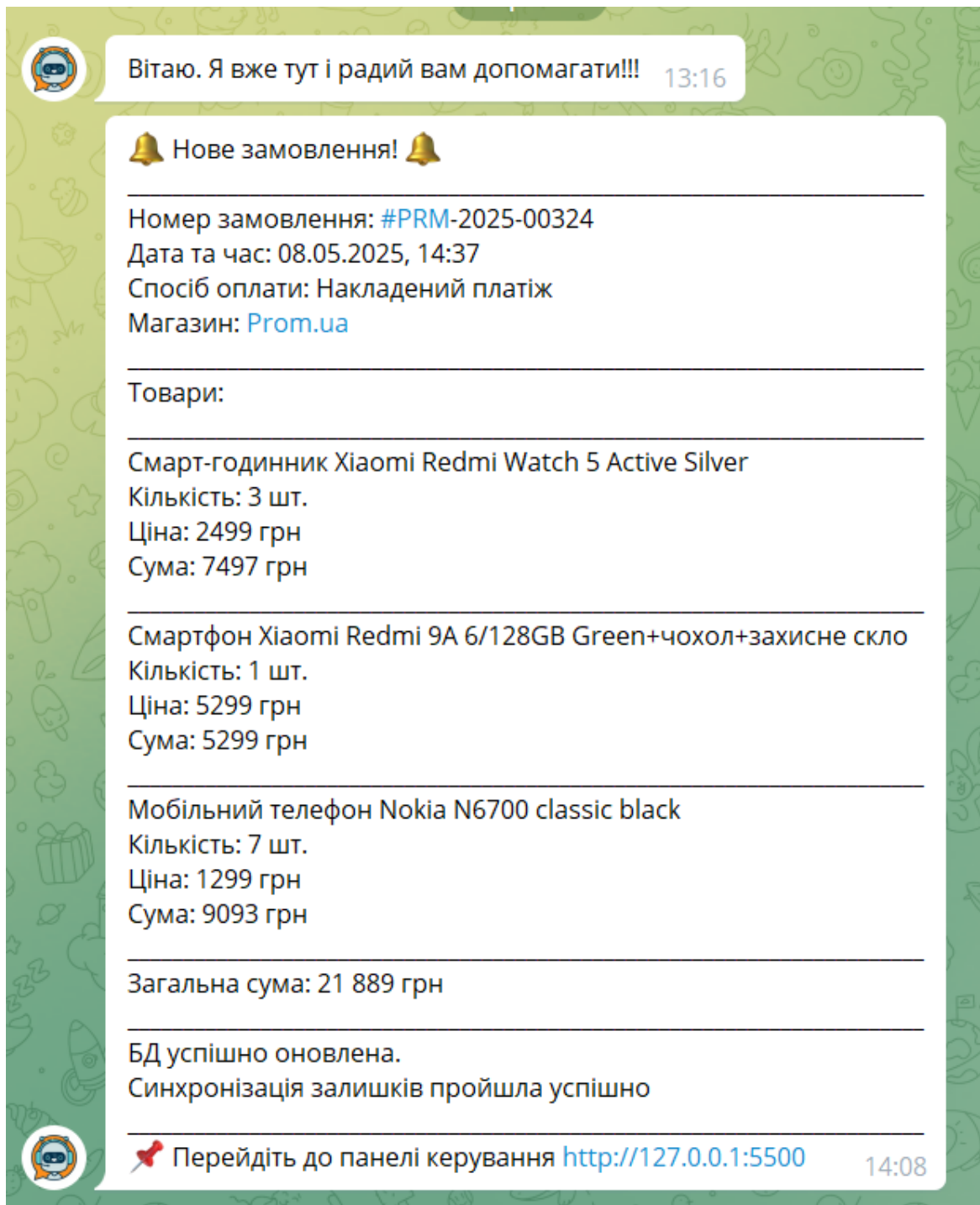


Рисунок 3.12 – Приклад спливаючого повідомлення про успішне додавання, редагування чи видалення картки товару

Приклад успішного замовлення наведено на рис. 3.13.



Написати повідомлення...

Рисунок 3.13 – Приклад сповіщення про нове замовлення

Як видно з рис. 3.13, система автоматично обробляє замовлення з маркетплейсів, надсилаючи детальні сповіщення через Telegram-бот із інформацією про товари, спосіб оплати та магазин. Вона забезпечує

синхронізацію залишків і оновлення бази даних, дозволяючи користувачу керувати замовленнями через вебінтерфейс.

Також додавання, видалення чи редагування виконується в відповідному модальному вікні. Приклад такого модального вікна наведено на рис. 3.14.

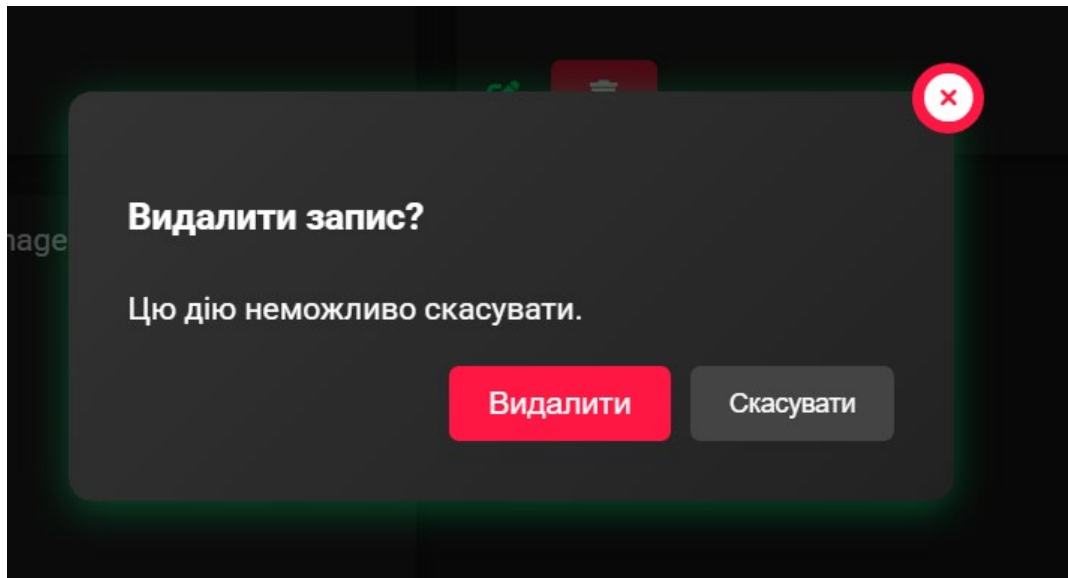


Рисунок 3.14 – Приклад модального вікна про видалення товару

Для перевірки надійності системи проведено тестування на стійкість до помилок:

- некоректні токени API. Система видала сповіщення про помилку підключення та запропонувала повторити введення даних;
- відсутність зв'язку з API. Система виконала повторні спроби підключення та надіслала сповіщення в Telegram про тимчасову проблему;
- великий обсяг даних. Імпорт 1000 товарів із Prom.ua виконано за 45 секунд, що відповідає вимогам продуктивності.

Користувацьке тестування показало, що вебінтерфейс є інтуїтивно зрозумілим, а час реакції на такі дії, як додавання, редагування та фільтрація становить менше 1 секунди.

Тестування підтвердило, що система коректно виконує всі заявлені функції:

- ініціалізація та підключення до API маркетплейсів і Telegram-бота;
- управління товарами через вебінтерфейс (додавання, редагування, видалення, сортування, фільтрація);
- автоматична обробка замовлень із синхронізацією залишків;
- надійна доставка сповіщень через Telegram;
- стійкість до помилок і швидка обробка даних.

3.4 Висновки до розділу

У даному розділі дипломної роботи виконано практичну реалізацію програмного застосунку "Система управління товарами для українських маркетплейсів", що забезпечує автоматизацію процесів менеджменту товарів, обробки замовлень та їх обліку. Розроблене програмне забезпечення включає модулі для інтеграції з API маркетплейсів Rozetka та Prom.ua, управління локальною базою даних, сповіщення через Telegram-бот та зручний вебінтерфейс для користувачів.

Реалізовані алгоритми синхронізації даних, обробки замовлень та ініціалізації бази даних забезпечують стабільну взаємодію системи з маркетплейсами, дозволяючи автоматично оновлювати ціни, залишки товарів та статуси замовлень. Модульна архітектура, побудована на основі шаблону MVC, сприяє легкості підтримки та масштабування системи. Використання технологій Python, Flask, SQLite, JavaScript та Telegram Bot API дозволило створити ефективне рішення, яке відповідає сформульованим функціональним і нефункціональним вимогам, зокрема щодо надійності, продуктивності, зручності використання та безпеки.

Тестування програмного забезпечення, яке включало модульне, інтеграційне та користувацьке тестування, підтвердило коректність роботи всіх компонентів системи. Виявлені під час тестування недоліки, такі як затримки при обробці великих обсягів даних, були усунені шляхом

оптимізації запитів до API та бази даних. Розроблений вебінтерфейс виявився інтуїтивно зрозумілим, що спрощує роботу користувачів без спеціальних технічних знань.

ВИСНОВКИ

В результаті виконання роботи було створено програму для управління товарами на маркетплейсах Rozetka та Prom.ua. Вона спрощує обробку замовлень, оновлення даних і облік товарів, роблячи онлайн-торгівлю зручнішою.

Було проведено досліді, як працюють маркетплейси та які проблеми є у продавців. Багато програм виявилися складними чи дорогими, тому створили просте рішення. Програма автоматично з'єднується з Rozetka та Prom.ua, щоб оновлювати товари й замовлення. У ній є вебінтерфейс, де легко додавати чи змінювати товари. Telegram-бот сповіщає про нові замовлення, економлячи час продавців.

Програму розроблено за допомогою Python і Flask, щоб вона була швидкою. Для даних вибрали SQLite, бо вона проста й підходить для невеликих бізнесів. Вебінтерфейс на JavaScript, HTML і CSS зрозумілий усім. Тестування показало, що програма працює без збоїв, швидко обробляє замовлення й оновлює дані. Проблеми з повільною роботою виправляли для зручності.

Програма економить час і зменшує помилки від ручного введення даних. Telegram-бот допомагає швидко реагувати на замовлення. Вона підходить для невеликих магазинів і дешевша за схожі програми, як KeyCRM.

У майбутньому можна додати підтримку маркетплейсів OLX чи Shafa, функцію звітів про продажі та додаток для телефону. Це зробить програму ще кориснішою.

Усі завдання роботи виконано. Програма готова до використання в онлайн-торгівлі й допомагає продавцям працювати ефективніше.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Promodo | Дослідження українського ринку eCommerce. Друге півріччя 2024 [Електрон. ресурс]. – Режим доступу: <https://www.promodo.ua/ukrayinskiy-ecommerce-2024-2>.
2. RemOnline | Як працюють Розетка та Пром маркетплейс [Електрон. ресурс]. – Режим доступу: <https://remonline.ua/blog/prom-marketplace>.
3. KeyCRM | KeyCRM vs KeepinCRM: вибрати українську сrm [Електрон. ресурс]. – Режим доступу: <https://blog.keycrm.app/uk/keycrm-vs-keepincrm-vibrati-ukrainsku-srm/>.
4. KeepinCRM | Зрозуміла CRM для керування компанії [Електрон. ресурс]. – Режим доступу: <https://keepincrm.com/>.
5. Phonet | Що вміє українська KeyCRM? [Електрон. ресурс]. – Режим доступу: <https://blog.phonet.ua/what-can-keycrm/>.
6. prom.продавцям | Що таке «Сторінка продавця»? [Електрон. ресурс]. – Режим доступу: <https://support.prom.ua/hc/uk/articles/360013465937-Сторінка-продавця>.
7. KeyCRM | Як правильно оформити інтернет-магазин на українських маркетплейсах Rozetka та Prom [Електрон. ресурс]. – Режим доступу: <https://blog.keycrm.app/uk/yak-pravilno-oformiti-internet-magazin-na-ukrainskih-marketplejsah-rozetka-ta-prom/>.
8. Асоціація ритейлерів України RAU.UA | Як стати лідером на ринку e-commerce. Історія компанії Rozetka [Електрон. ресурс]. – Режим доступу: <https://rau.ua/novyni/e-commerce-istoriya-rozetka/>.
9. Rozetka | Головна сторінка [Електрон. ресурс]. – Режим доступу: <https://rozetka.com.ua/>.
10. Rozetka | API ROZETKA Маркетплейс [Електрон. ресурс]. – Режим доступу: <https://api-seller.rozetka.com.ua/apidoc/>.

11. Довідковий центр продавця Rozetka | Додати новий товар [Електрон. ресурс]. – Режим доступу: <https://sellerhelp.rozetka.com.ua/p203-add-new-items.html>.
12. Prom.ua | Головна сторінка [Електрон. ресурс]. – Режим доступу: <https://prom.ua/>.
13. NetHunt CRM | Що таке CRM: повний гід по вибору CRM для початківців [Електрон. ресурс]. – Режим доступу: <https://nethunt.ua/blog/shcho-takie-crm-sistiema-povnii-ghid-po-viboru-crm-dlia-pochatktivtsiv/>.
14. Medium | Best Text Editor? Atom vs Sublime vs Visual Studio Code vs Vim [Electronic resource]. – Access mode: <https://codementorio.medium.com/best-text-editor-atom-vs-sublime-vs-visual-studio-code-vs-vim-85b651d36665>.
15. Major-Innovations | Exploring Text Editors: Why Developers Prefer Visual Studio Code [Electronic resource]. – Access mode: <https://major-innovations.com/tutorial/5>.
16. JetBrains | PyCharm. The only Python IDE you need [Electronic resource]. – Access mode: <https://www.jetbrains.com/pycharm>.
17. Dev.to | Database Comparison - SQL vs. NoSQL (MySQL vs PostgreSQL vs Redis vs MongoDB) [Electronic resource]. – Access mode: <https://dev.to/profilsoftware/database-comparison-sql-vs-nosql-mysql-vs-postgresql-vs-redis-vs-mongodb-2e01>.
18. Medium | Decoding the Secrets of Top Programming Languages: JavaScript, Python, Java, C#, and C++ [Electronic resource]. – Access mode: <https://medium.com/@keyboardwordsmith/decoding-the-secrets-of-top-programming-languages-javascript-python-java-c-and-c-8b184f966bb7>.
19. JetBrains | Django vs. Flask: Which Is the Best Python Web Framework? [Electronic resource]. – Access mode: <https://blog.jetbrains.com/pycharm/2023/11/django-vs-flask-which-is-the-best-python-web-framework/>.

20. StartADAM | Telegram vs SMS: Which Messaging Service Suits Your Needs? [Electronic resource]. – Access mode: <https://startadam.com/blog/telegram-vs-sms-messaging-comparison/>.

ДОДАТОК А
Тексти програм

A.1 Текст файлу «index.html»

```

<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Inventory Manager</title>
    <link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600&disp
lay=swap" rel="stylesheet">
    <link                                rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.5.1/css/all.min.css">
    <link rel="stylesheet" href="./static/css/styles.css">
    <script src="./static/js/app.js" defer></script>
</head>
<body>
    <!-- Loading Overlay -->
    <div id="loader" class="loading-overlay">
        <div class="loader-spinner"></div>
    </div>

    <!-- Navigation Toggle Button -->
    <button id="openNavBtn" class="nav-toggle-btn" hidden><i
class="fas fa-bars"></i></button>

    <!-- Main Layout -->
    <div class="main-layout">
        <!-- Navigation Panel -->
        <nav class="nav-panel nav-panel--open" id="navPanel">
            <div class="nav-header">
                <h2>Inventory Manager</h2>
                <button id="closeNavBtn" class="nav-collapse-
btn"><i class="fas fa-times"></i></button>
            </div>
            <div class="nav-controls">
                <div class="control-section">
                    <label for="searchField">Search</label>

```

```

        <input          type="text"          id="searchField"
placeholder="Search by title..." class="control-input">
    </div>
    <div class="control-section">
        <label>Sort By</label>
        <select      id="sortDropdown"      class="control-
dropdown">
            <option value="item_title">Title</option>
            <option value="stock_level">Stock</option>
        </select>
        <div class="filter-options">
            <label><input type="radio" name="sortOrder"
value="asc" checked> Ascending</label>
            <label><input type="radio" name="sortOrder"
value="desc"> Descending</label>
        </div>
    </div>
    <button  id="createItemBtn"  class="create-btn">Add
Item</button>

    <div      class="record-count">Total      Items:      <span
id="totalItems"></span></div>
    </div>
</nav>

<!-- Item Grid -->
<section class="items-grid" id="itemGrid">
    <!-- Items populated via JavaScript -->
</section>
</div>

<!-- Edit Dialog -->
<div id="editDialog" class="dialog" hidden>
    <div class="dialog-box">
        <button  class="dialog-close"><i  class="fas fa-times-
circle"></i></button>
        <form id="editItemForm" class="dialog-form">
            <h3>Edit Item</h3>
            <div class="input-group">
                <input          type="text"          id="editItemTitle"
name="title" required>
                <label for="editItemTitle">Title</label>

```

```

        </div>
        <div class="input-group">
            <input      type="number"      id="editItemStock"
name="stock" required>
            <label for="editItemStock">Stock</label>
        </div>
        <div class="input-group">
            <input      type="text"      id="editItemImage"
name="image">
            <label for="editItemImage">Image URL</label>
        </div>
        <div class="input-group">
            <input      type="text"      id="editItemStoreId"
name="storeId">
            <label for="editItemStoreId">Store ID</label>
        </div>
        <div class="input-group">
            <input      type="text"      id="editItemPlatformId"
name="platformId">
            <label      for="editItemPlatformId">Platform
ID</label>
        </div>
        <input type="hidden" id="editItemId" name="itemId">
        <button      type="submit"      class="submit-
btn">Save</button>
    </form>
</div>
</div>

<!-- Delete Dialog -->
<div id="deleteDialog" class="dialog" hidden>
    <div class="dialog-box">
        <button class="dialog-close"><i class="fas fa-times-
circle"></i></button>
        <h3>Delete Item?</h3>
        <p>This action cannot be undone.</p>
        <div class="dialog-actions">
            <button id="deleteConfirmBtn" class="action-btn--
confirm">Delete</button>
            <button class="action-btn--cancel">Cancel</button>
        </div>
    </div>

```

```

        </div>
    </div>

    <!-- Create Dialog -->
    <div id="createDialog" class="dialog" hidden>
        <div class="dialog-box">
            <button class="dialog-close"><i class="fas fa-times-
circle"></i></button>
            <form id="createItemForm" class="dialog-form">
                <h3>Add Item</h3>
                <div class="input-group">
                    <input type="text" id="createItemTitle"
name="title" required>
                    <label for="createItemTitle">Title</label>
                </div>
                <div class="input-group">
                    <input type="number" id="createItemStock"
name="stock" required>
                    <label for="createItemStock">Stock</label>
                </div>
                <div class="input-group">
                    <input type="text" id="createItemImage"
name="image">
                    <label for="createItemImage">Image URL</label>
                </div>
                <div class="input-group">
                    <input type="text" id="createItemStoreId"
name="storeId">
                    <label for="createItemStoreId">Store ID</label>
                </div>
                <div class="input-group">
                    <input type="text" id="createItemPlatformId"
name="platformId">
                    <label for="createItemPlatformId">Platform
ID</label>
                </div>
                <button type="submit" class="submit-
btn">Create</button>
            </form>
        </div>
    </div>

```

```

        <!-- Notification -->
        <div id="notification" class="notification"></div>
    </body>
</html>

```

A.2 Текст файла «script.js»

```

let inventoryItems = [];
let paginatedItems = [];
let displayedItems = [];

const itemGrid = document.querySelector('#itemGrid');
const sortDropdown = document.querySelector('#sortDropdown');
const searchField = document.querySelector('#searchField');
const editDialog = document.querySelector('#editDialog');
const deleteDialog = document.querySelector('#deleteDialog');
const
    deleteConfirmBtn
=
document.querySelector('#deleteConfirmBtn');
    let itemToDelete = null;
const createItemBtn = document.querySelector('#createItemBtn');
const createDialog = document.querySelector('#createDialog');
const
    sortOrderInputs
=
document.querySelectorAll('input[name="sortOrder"]');
const navPanel = document.querySelector('#navPanel');
const openNavBtn = document.querySelector('#openNavBtn');
const closeNavBtn = document.querySelector('#closeNavBtn');

// Display notification
function displayNotification(message) {
    const notification = document.querySelector('#notification');
    notification.textContent = message;
    notification.classList.add('notification--visible');
    setTimeout(() => notification.classList.remove('notification--
visible'), 4000);
}

// Render inventory cards
const renderInventory = () => {

```

```

itemGrid.innerHTML = '';
paginatedItems.forEach(item => {
    const card = document.createElement('div');
    card.className = 'item-card';
    card.dataset.itemId = item.item_id;
    card.innerHTML = `
        <div class="item-image-container">
            
        </div>
        <div class="item-info">
            <p><strong>ID:</strong> ${item.item_id}</p>
            <p><strong>Title:</strong> <span
class="${!item.item_title ? 'error-message' : ''}">${item.item_title ||
'Missing'}</span></p>
            <p><strong>Stock:</strong> <span
class="${item.stock_level == null ? 'error-message' :
''}">${item.stock_level >= 0 ? item.stock_level : 0}</span></p>
            <p><strong>Store:</strong> <span
class="${!item.store_id ? 'error-message' : ''}">${item.store_id ||
'N/A'}</span></p>
            <p><strong>Market:</strong> <span
class="${!item.platform_id ? 'error-message' : ''}">${item.platform_id ||
'N/A'}</span></p>
        </div>
        <div class="item-actions">
            <button class="action-btn--edit" data-item-
id="${item.item_id}"><i class="fas fa-pencil-alt"></i></button>
            <button class="action-btn--delete" data-item-
id="${item.item_id}"><i class="fas fa-trash-alt"></i></button>
        </div>
    `;
    itemGrid.appendChild(card);
});

// Refresh grid
function updateGrid() {
    paginatedItems = cloneObject(displayedItems);
    renderInventory();
}

```

```

// Fetch, sort, and filter items
const fetchAndProcessItems = async () => {
  const loader = document.querySelector('#loader');
  loader.style.display = 'flex';
  try {
    const response = await fetch('/items', {
      method: 'GET',
      headers: { 'Content-Type': 'application/json' }
    });
    if (!response.ok) throw new Error('Failed to fetch items');
    const data = await response.json();

    console.log('Data received:', data);
    inventoryItems = Array.isArray(data) ? data : [];
    displayedItems = Array.isArray(data) ? data : [];
    document.querySelector('#totalItems').textContent =
inventoryItems.length;

    // Filter by search
    const query = searchField.value.trim().toLowerCase();
    if (query) {
      displayedItems = displayedItems.filter(item =>
        item.item_title?.toLowerCase().includes(query)
      );
    }

    // Sort items
    const order =
document.querySelector('input[name="sortOrder"]:checked').value || 'asc';
    const sortKey = sortDropdown.value;
    console.log('Sort key:', sortKey, 'Order:', order);
    displayedItems.sort((a, b) => {
      let valA, valB;
      if (sortKey === 'item_title') {
        valA = a.item_title?.toLowerCase() || '';
        valB = b.item_title?.toLowerCase() || '';
        console.log('Sorting by title:', { idA: a.item_id,
valA, idB: b.item_id, valB });
        return order === 'asc' ? valA.localeCompare(valB) :
valB.localeCompare(valA);

```

```

        } else if (sortKey === 'stock_level') {
            valA = a.stock_level ?? 0;
            valB = b.stock_level ?? 0;
            console.log('Sorting by stock:', { idA: a.item_id,
valA, idB: b.item_id, valB });
            return order === 'asc' ? valA - valB : valB - valA;
        }
        return 0;
    });

    console.log('Processed items:', displayedItems);
    updateGrid();
} catch (error) {
    console.error('Fetch error:', error);
    displayNotification('Error loading inventory');
} finally {
    loader.style.display = 'none';
}
};

// Edit item
itemGrid.addEventListener('click', e => {
    const editBtn = e.target.closest('.action-btn--edit');
    if (!editBtn) return;
    const itemId = editBtn.dataset.itemId;
    const item = inventoryItems.find(i => i.item_id === itemId);
    if (item) {
        document.querySelector('#editItemId').value = item.item_id;
        document.querySelector('#editItemTitle').value =
item.item_title || '';
        document.querySelector('#editItemStock').value =
item.stock_level >= 0 ? item.stock_level : 0;
        document.querySelector('#editItemImage').value =
item.photo_url || '';
        document.querySelector('#editItemStoreId').value =
item.store_id || '';
        document.querySelector('#editItemPlatformId').value =
item.platform_id || '';
        editDialog.classList.add('dialog--active');
    }
});

```

```

// Delete item
itemGrid.addEventListener('click', e => {
  const deleteBtn = e.target.closest('.action-btn--delete');
  if (!deleteBtn) return;
  itemToDelete = deleteBtn.dataset.itemId;
  deleteDialog.classList.add('dialog--active');
});

// Close dialogs
document.querySelectorAll('.dialog-close, .action-btn--cancel').forEach(btn => {
  btn.addEventListener('click', () => {
    editDialog.classList.remove('dialog--active');
    createDialog.classList.remove('dialog--active');
    deleteDialog.classList.remove('dialog--active');
    itemToDelete = null;
  });
});

// Close dialogs on overlay click
document.querySelectorAll('.dialog').forEach(dialog => {
  dialog.addEventListener('click', e => {
    if (e.target === dialog) {
      dialog.classList.remove('dialog--active');
      itemToDelete = null;
    }
  });
});

// Submit edit form
document.querySelector('#editItemForm').addEventListener('submit',
async e => {
  e.preventDefault();
  const data = {
    item_id: document.querySelector('#editItemId').value,
    item_title: document.querySelector('#editItemTitle').value,
    stock_level:
parseInt(document.querySelector('#editItemStock').value, 10) || 0,
    photo_url: document.querySelector('#editItemImage').value,
    store_id: document.querySelector('#editItemStoreId').value,

```

```

        platform_id:
document.querySelector('#editItemPlatformId').value
    };
    try {
        const response = await fetch('/modify_item', {
            method: 'PUT',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(data)
        });
        const result = await response.json();
        if (response.ok) {
            displayNotification('Item updated successfully');
            editDialog.classList.remove('dialog--active');
            await fetchAndProcessItems();
        } else {
            console.error('Edit error:', result.status);
            displayNotification('Failed to update item');
        }
    } catch (error) {
        console.error('Edit error:', error);
        displayNotification('Error updating item');
    }
});

// Confirm deletion
deleteConfirmBtn.addEventListener('click', async () => {
    try {
        const response = await fetch('/remove_item', {
            method: 'DELETE',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({ item_id: itemToDelete })
        });
        const result = await response.json();
        if (response.ok) {
            displayNotification('Item removed');
            deleteDialog.classList.remove('dialog--active');
            itemToDelete = null;
            await fetchAndProcessItems();
        } else {
            console.error('Delete error:', result.status);
            displayNotification('Failed to remove item');
        }
    }
});

```

```

    }
  } catch (error) {
    console.error('Delete error:', error);
    displayNotification('Error removing item');
  }
});

// Open create form
createItemBtn.addEventListener('click', () => {
  document.querySelector('#createItemTitle').value = '';
  document.querySelector('#createItemStock').value = 0;
  document.querySelector('#createItemImage').value = '';
  document.querySelector('#createItemStoreId').value = '';
  document.querySelector('#createItemPlatformId').value = '';
  createDialog.classList.add('dialog--active');
});

// Submit create form
document.querySelector('#createItemForm').addEventListener('submit'
, async e => {
  e.preventDefault();
  const data = {
    item_title:
document.querySelector('#createItemTitle').value,
    stock_level:
parseInt(document.querySelector('#createItemStock').value, 10) || 0,
    photo_url:
document.querySelector('#createItemImage').value,
    store_id:
document.querySelector('#createItemStoreId').value,
    platform_id:
document.querySelector('#createItemPlatformId').value
  };
  try {
    const response = await fetch('/add_item', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(data)
    });
    const result = await response.json();
    if (response.ok) {

```

```

        displayNotification('Item added successfully');
        createDialog.classList.remove('dialog--active');
        await fetchAndProcessItems();
    } else {
        console.error('Create error:', result.status);
        displayNotification('Failed to add item');
    }
} catch (error) {
    console.error('Create error:', error);
    displayNotification('Error adding item');
}
});

// Clone object
function cloneObject(obj) {
    if (obj == null || typeof obj !== 'object') return obj;
    const cloned = Array.isArray(obj) ? [] : {};
    for (const key in obj) {
        cloned[key] = cloneObject(obj[key]);
    }
    return cloned;
}

// Toggle navigation panel
openNavBtn.addEventListener('click', () => {
    navPanel.classList.add('nav-panel--open');
    openNavBtn.hidden = true;
});

closeNavBtn.addEventListener('click', () => {
    navPanel.classList.remove('nav-panel--open');
    openNavBtn.hidden = false;
});

// Event listeners for sorting and filtering
sortDropdown.addEventListener('change', () => {
    console.log('Sort dropdown changed:', sortDropdown.value);
    fetchAndProcessItems();
});

searchField.addEventListener('input', () => {

```

```

        console.log('Search query:', searchField.value);
        fetchAndProcessItems();
    });

    sortOrderInputs.forEach(input => {
        input.addEventListener('change', () => {
            console.log('Sort order changed:', input.value);
            fetchAndProcessItems();
        });
    });

    // Initialize
    fetchAndProcessItems();

```

A.3 Текст файла «style.css»

```

/* General Styles */
html {
    font-size: 16px;
}

body {
    margin: 0;
    padding: 0;
    font-family: 'Inter', sans-serif;
    background: #121212;
    color: #e0e0e0;
    line-height: 1.5;
    overflow-x: hidden;
}

*,
*::before,
*::after {
    box-sizing: border-box;
}

/* Loading Overlay */
.loading-overlay {
    position: fixed;

```

```

    inset: 0;
    background: rgba(0, 0, 0, 0.85);
    z-index: 3000;
    display: flex;
    align-items: center;
    justify-content: center;
}

.loader-spinner {
    width: 60px;
    height: 60px;
    border: 6px solid #4CAF50;
    border-right-color: transparent;
    border-radius: 50%;
    animation: rotate 0.8s linear infinite;
}

@keyframes rotate {
    100% {
        transform: rotate(360deg);
    }
}

/* Layout */
.main-layout {
    display: flex;
    min-height: 100vh;
}

/* Navigation Panel */
.nav-panel {
    width: 280px;
    background: #1e1e1e;
    padding: 24px;
    transition: transform 0.4s ease-in-out;
    z-index: 1000;
}

.nav-panel--open {
    transform: translateX(0);
}

```

```
.nav-panel--closed {  
  transform: translateX(-100%);  
}  
  
.nav-header {  
  display: flex;  
  align-items: center;  
  justify-content: space-between;  
  margin-bottom: 24px;  
}  
  
.nav-collapse-btn {  
  background: transparent;  
  border: none;  
  color: #4CAF50;  
  font-size: 1.5rem;  
  cursor: pointer;  
  transition: color 0.2s;  
}  
  
.nav-collapse-btn:hover {  
  color: #45a049;  
}  
  
.nav-toggle-btn {  
  position: fixed;  
  top: 16px;  
  left: 16px;  
  background: #4CAF50;  
  color: #fff;  
  border: none;  
  padding: 8px 12px;  
  border-radius: 4px;  
  cursor: pointer;  
  z-index: 1100;  
  transition: background 0.2s;  
}  
  
.nav-toggle-btn:hover {  
  background: #45a049;
```

```
}

.nav-controls {
  display: flex;
  flex-direction: column;
  gap: 16px;
}

.control-section {
  display: flex;
  flex-direction: column;
  gap: 8px;
}

.control-label {
  font-size: 0.875rem;
  color: #a0a0a0;
}

.control-input,
.control-dropdown {
  padding: 10px;
  border: 1px solid #555;
  border-radius: 4px;
  background: #2a2a2a;
  color: #e0e0e0;
  font-size: 0.875rem;
}

.filter-options {
  display: flex;
  flex-direction: column;
  gap: 6px;
  color: #a0a0a0;
  font-size: 0.875rem;
}

.create-btn {
  background: #4CAF50;
  border: none;
  padding: 12px;
```

```

border-radius: 4px;
color: #fff;
font-weight: 600;
cursor: pointer;
transition: background 0.2s, transform 0.2s;
}

.create-btn:hover {
background: #45a049;
transform: translateY(-2px);
}

.record-count {
font-size: 0.875rem;
color: #a0a0a0;
}

/* Item Grid */
.items-grid {
flex: 1;
padding: 24px;
display: grid;
grid-template-columns: repeat(auto-fill, minmax(280px, 1fr));
gap: 16px;
}

.item-card {
background: #1e1e1e;
border-radius: 6px;
box-shadow: 0 3px 10px rgba(0, 0, 0, 0.4);
display: flex;
flex-direction: column;
overflow: hidden;
}

.item-image-container {
padding: 12px;
display: flex;
justify-content: center;
}

```

```
.item-image {
  width: 100px;
  height: 100px;
  object-fit: contain;
  border-radius: 4px;
}

.item-info {
  padding: 12px;
  flex-grow: 1;
}

.item-info p {
  margin: 4px 0;
  font-size: 0.875rem;
}

.item-actions {
  padding: 12px;
  display: flex;
  gap: 8px;
}

.action-btn--edit,
.action-btn--delete {
  background: none;
  border: none;
  font-size: 1rem;
  cursor: pointer;
  transition: color 0.2s;
}

.action-btn--edit {
  color: #4CAF50;
}

.action-btn--edit:hover {
  color: #45a049;
}

.action-btn--delete {
```

```

    color: #F44336;
}

.action-btn--delete:hover {
    color: #D32F2F;
}

.error-message {
    color: #F44336;
    font-weight: 500;
}

/* Dialog Windows */
.dialog {
    position: fixed;
    inset: 0;
    background: rgba(0, 0, 0, 0.75);
    z-index: 2000;
    display: none;
    align-items: center;
    justify-content: center;
}

.dialog--active {
    display: flex;
}

.dialog-box {
    background: linear-gradient(145deg, #2a2a2a, #1e1e1e);
    padding: 24px;
    border-radius: 8px;
    max-width: 400px;
    width: 90%;
    box-shadow: 0 5px 15px rgba(76, 175, 80, 0.2);
    animation: slideIn 0.3s ease;
}

@keyframes slideIn {
    from {
        transform: translateY(-20px);
        opacity: 0;
    }

```

```

    }
    to {
        transform: translateY(0);
        opacity: 1;
    }
}

```

```

.dialog-close {
    position: absolute;
    top: -12px;
    right: -12px;
    background: #F44336;
    color: #fff;
    border: none;
    border-radius: 50%;
    width: 32px;
    height: 32px;
    display: flex;
    align-items: center;
    justify-content: center;
    cursor: pointer;
    transition: transform 0.2s;
}

```

```

.dialog-close:hover {
    transform: scale(1.1);
}

```

```

.dialog-form {
    display: flex;
    flex-direction: column;
    gap: 16px;
}

```

```

.dialog-form h3 {
    margin: 0;
    font-size: 1.25rem;
    color: #4CAF50;
}

```

```

.input-group {

```

```
    position: relative;
}

.input-group input {
    width: 100%;
    padding: 10px;
    border: 1px solid #555;
    border-radius: 4px;
    background: #2a2a2a;
    color: #e0e0e0;
    font-size: 0.875rem;
}

.input-group input:focus {
    outline: none;
    border-color: #4CAF50;
}

.input-group label {
    position: absolute;
    top: -8px;
    left: 8px;
    background: #2a2a2a;
    padding: 0 4px;
    font-size: 0.75rem;
    color: #a0a0a0;
}

.submit-btn {
    background: #4CAF50;
    border: none;
    padding: 10px;
    border-radius: 4px;
    color: #fff;
    font-weight: 500;
    cursor: pointer;
    transition: background 0.2s, transform 0.2s;
}

.submit-btn:hover {
    background: #45a049;
```

```

    transform: scale(1.02);
}

.dialog-actions {
    display: flex;
    gap: 8px;
    justify-content: flex-end;
}

.action-btn--confirm {
    background: #F44336;
    border: none;
    padding: 8px 16px;
    border-radius: 4px;
    color: #fff;
    cursor: pointer;
    transition: background 0.2s;
}

.action-btn--confirm:hover {
    background: #D32F2F;
}

.action-btn--cancel {
    background: #555;
    border: none;
    padding: 8px 16px;
    border-radius: 4px;
    color: #fff;
    cursor: pointer;
}

.action-btn--cancel:hover {
    background: #666;
}

/* Notification Toast */
.notification {
    position: fixed;
    bottom: 16px;
    right: 16px;

```

```

background: #4CAF50;
color: #fff;
padding: 8px 16px;
border-radius: 4px;
opacity: 0;
transition: opacity 0.4s;
}

.notification--visible {
  opacity: 1;
}

/* Responsive Design */
@media (max-width: 768px) {
  .main-layout {
    flex-direction: column;
  }

  .nav-panel {
    width: 100%;
    max-width: none;
    position: fixed;
    height: 100%;
    top: 0;
    left: 0;
  }

  .items-grid {
    grid-template-columns: 1fr;
  }

  .dialog-box {
    width: 95%;
  }
}

```

A.4 Текст файла «ecommerce.py»

```

import aiohttp
import asyncio

```

```

from typing import List, Dict, Optional
from inventory_manager import InventoryManager

class StoreService:
    def __init__(self, api_token: str):
        self._token = api_token
        self._timeout = 20
        self._payment_options =

    asyncio.run(self._load_payment_options())

    async def _load_payment_options(self) -> Dict[str, str]:
        """Fetch and map available payment methods."""
        data = await self._execute_request("GET",
"https://my.prom.ua/api/v1/payment_options/list")
        if not isinstance(data, dict) or "payment_options" not in
data:
            print("Warning: Failed to load payment options. Using
defaults.")

        return {
            "cash_on_delivery": "",
            "wire_transfer": "",
            "credit_card": "",
            "store_payment": ""
        }

    options = {
        "cash_on_delivery": "Наложенный платеж",
        "wire_transfer": "Оплата за реквизитами",
        "credit_card": "Оплата картой Visa, Mastercard -
WayForPay",
        "store_payment": "Пром-оплата"
    }
    result = options.copy()
    for key, name in options.items():
        for option in data["payment_options"]:
            if option["name"] == name:
                result[key] = option["id"]
                break
    return result

```

```

    async def _execute_request(self, method: str, url: str,
payload: Optional[Dict] = None, params: Dict = None) -> Optional[Dict]:
        """Perform an API request with error handling."""
        headers = {
            "Authorization": f"Bearer {self._token}",
            "Content-Type": "application/json"
        }
        async with aiohttp.ClientSession() as session:
            try:
                async with session.request(
                    method, url, json=payload, params=params,
headers=headers, timeout=self._timeout
                ) as response:
                    try:
                        return await response.json()
                    except aiohttp.ContentTypeError:
                        print(f"Error: Invalid JSON response from
{url}. Received: {await response.text()[:100]}...")
                        return None
            except aiohttp.ClientError as e:
                print(f"Request error for {url}: {e}")
                return None

    async def sync_inventory(self, db_path: str) -> None:
        """Synchronize product data with local database."""
        manager = InventoryManager(db_path)
        items = await self._fetch_all_items()
        for item in items:
            manager.insert_item({
                "photo_url": item.get("main_image", ""),
                "item_title": item.get("name", ""),
                "stock_level": item.get("quantity_in_stock", 0),
                "store_id": item.get("sku", ""),
                "platform_id": ""
            })

    async def _fetch_all_items(self) -> List[Dict]:
        """Retrieve all products in batches."""
        batch_size = 100
        last_id = ""
        items = []

```

```

        while True:
            data = await self._execute_request(
                "GET",
                f"https://my.prom.ua/api/v1/products/list",
                params={"limit": batch_size, "last_id": last_id}
            )
            if not data or not isinstance(data, dict) or "products"
not in data:
                break
            products = data["products"]
            if not products or products[-1]["id"] == last_id:
                break
            items.extend(products)
            last_id = products[-1]["id"]
        return items

    async def _fetch_item_info(self, item_id: str) ->
Optional[Dict]:
        """Get details for a specific product."""
        return await self._execute_request("GET",
f"https://my.prom.ua/api/v1/products/{item_id}")

    async def _modify_item(self, item_data: Dict) ->
Optional[Dict]:
        """Update product information."""
        return await self._execute_request("POST",
"https://my.prom.ua/api/v1/products/edit", payload=[item_data])

    async def _fetch_orders(self, status: str = "", start_date: str
= "") -> List[Dict]:
        """Retrieve orders based on status and date."""
        params = {"status": status, "date_from": start_date} if
status or start_date else {}
        data = await self._execute_request("GET",
"https://my.prom.ua/api/v1/orders/list", params=params)
        return data.get("orders", []) if isinstance(data, dict)
else []

    async def _update_order_status(self, order_info: Dict) ->
Optional[Dict]:
        """Change the status of an order."""

```

```

        return await self._execute_request("POST",
"https://my.prom.ua/api/v1/orders/set_status", payload=order_info)

    async def update_stock_by_sku(self, sku: str, stock: int) ->
Optional[Dict]:
        """Update inventory for a product by SKU."""
        items = await self._fetch_all_items()
        for item in items:
            if item["sku"] == sku:
                update_data = {
                    "id": item["id"],
                    "quantity_in_stock": max(0, stock),
                    "presence": "available" if stock > 0 else
"not_available"
                }
                return await self._modify_item(update_data)
        return None

    async def retrieve_active_orders(self) -> List[Dict]:
        """Fetch and process active orders."""
        orders = []
        # Fetch paid and pending orders
        paid_orders = await self._fetch_orders(status="paid")
        orders.extend(paid_orders)
        pending_orders = await self._fetch_orders(status="pending")

        # Filter pending orders by payment method
        for order in pending_orders:
            payment_id = order["payment_option"]["id"]
            if payment_id in
(self._payment_options["cash_on_delivery"],
self._payment_options["wire_transfer"]):
                orders.append(order)

        # Update order status to received
        for order in orders:
            status_update = {"status": "received", "ids":
[order["id"]]}
            await self._update_order_status(status_update)

        # Format order data

```

```

formatted_orders = []
for order in orders:
    order_data = {
        "order_number": order["id"],
        "payment_type": order["payment_option"]["name"],
        "amount": order["price"],
        "comments": order["client_notes"],
        "products": []
    }
    for product in order["products"]:
        product_details = await
self._fetch_item_info(product["id"])
        product_info = {
            "id": product["id"],
            "stock":
product_details["product"]["quantity_in_stock"] if product_details else 0,
            "quantity": product["quantity"],
            "name": product_details["product"]["name"] if
product_details else "",
            "sku": product_details["product"]["sku"] if
product_details else ""
        }
        order_data["products"].append(product_info)
    formatted_orders.append(order_data)

return formatted_orders

```

A.5 Текст файла «app.py»

```

import asyncio
import threading
from flask import Flask, request, render_template, jsonify
import aiohttp
from typing import List, Dict, Optional

from inventory_manager import InventoryManager
from ecommerce_service import ECommercePlatform
from message_service import NotificationService

app = Flask(__name__)

```

```

# Database configuration
db_path = "stock.db"
inventory = InventoryManager(db_path)

# Track inventory changes
modified_items: List[Dict] = []

def validate_stock_change(item: Dict) -> None:
    """Check if item stock has changed and queue for update."""
    stored_item = inventory.find_item(item["item_id"])
    if stored_item and stored_item["stock_level"] !=
item["stock_level"]:
        modified_items.append(item)

def prompt_for_input(message: str, allow_blank: bool = False,
retries: int = 3) -> str:
    """Safely collect user input with validation."""
    for _ in range(retries):
        response = input(message).strip()
        if response or allow_blank:
            return response
        print(f"Error: Input cannot be empty. {retries - _ - 1}
attempts left.")
    raise ValueError("Maximum input attempts exceeded.")

def setup_connections() -> tuple:
    """Configure API connections with user-provided credentials."""
    print("\n=== Configuring Notification Service ===")
    bot_token = prompt_for_input("Enter Telegram bot token (starts
with 'bot'): ")
    channel_id = prompt_for_input("Enter notification channel ID
(e.g., -123456789): ")
    notifier = NotificationService(bot_token, channel_id)
    print("Notification service configured successfully!\n")

    print("=== Configuring E-commerce Stores ===")
    while True:
        try:
            store_count = int(prompt_for_input("How many Prom.ua
stores to connect? (0 to skip): "))

```

```

        if store_count >= 0:
            break
        print("Error: Number of stores must be 0 or positive.")
    except ValueError:
        print("Error: Please enter a valid number.")

    store_apis = []
    for i in range(store_count):
        print(f"\nConfiguring store #{i + 1}")
        while True:
            store_key = prompt_for_input(f"Enter API key for
Prom.ua store #{i + 1}: ")
            if len(store_key) > 5:
                break
            print("Error: API key seems invalid (too short). Try
again.")

        store_apis.append(ECommercePlatform(store_key, ""))
        print(f"Store #{i + 1} connected successfully!")

    print("\n=== Configuring Marketplace ===")
    market_user = prompt_for_input("Enter Rozetka user ID: ")
    market_pass = prompt_for_input("Enter Rozetka password: ")
    market_api = ECommercePlatform(market_user, market_pass)
    print("Marketplace configured successfully!\n")

    return notifier, store_apis, market_api

@app.route("/")
def dashboard():
    """Render the main dashboard page."""
    return render_template("index.html")

@app.route("/items", methods=["GET"])
async def list_items():
    """Return all inventory items."""
    return jsonify(await
asyncio.get_event_loop().run_in_executor(None, inventory.fetch_all_items))

@app.route("/add_item", methods=["POST"])
async def add_item():
    """Create a new item in the inventory."""

```

```

        item_data = request.json
        await asyncio.get_event_loop().run_in_executor(None,
inventory.insert_item, {
            "photo_url": item_data["photo_url"],
            "item_title": item_data["item_title"],
            "stock_level": item_data["stock_level"],
            "store_id": item_data["store_id"],
            "platform_id": item_data["platform_id"]
        })
        return jsonify({"status": "Item created"}), 201

@app.route("/modify_item", methods=["PUT"])
async def modify_item():
    """Update an existing item in the inventory."""
    item_data = request.json
    validate_stock_change(item_data)
    await asyncio.get_event_loop().run_in_executor(None,
inventory.modify_item, {
        "item_id": item_data["item_id"],
        "photo_url": item_data["photo_url"],
        "item_title": item_data["item_title"],
        "stock_level": item_data["stock_level"],
        "store_id": item_data["store_id"],
        "platform_id": item_data["platform_id"]
    })
    return jsonify({"status": "Item updated"}), 200

@app.route("/remove_item", methods=["DELETE"])
async def remove_item():
    """Delete an item from the inventory."""
    item_data = request.json
    await asyncio.get_event_loop().run_in_executor(None,
inventory.remove_item, int(item_data["item_id"]))
    return jsonify({"status": "Item removed"}), 200

@app.route("/upload_image", methods=["POST"])
async def upload_image():
    """Handle image uploads to a remote service."""
    if "image" not in request.files:
        return jsonify({"error": "No image provided"}), 400

```

```

image_file = request.files["image"]
async with aiohttp.ClientSession() as session:
    try:
        async with session.post(
            "https://telegra.ph/upload",
            data={"file": image_file}
        ) as response:
            result = await response.json()
            if isinstance(result, list) and
result[0].get("src"):
                return jsonify({
                    "status": "Image uploaded",
                    "url":
f"https://telegra.ph{result[0]['src']}"
                }), 200
            return jsonify({"error": "Upload failed"}), 500
    except aiohttp.ClientError as e:
        print(f"Image upload error: {e}")
        return jsonify({"error": "Server error"}), 500

def update_inventory(updates: List[Dict]) -> None:
    """Synchronize inventory changes across platforms."""
    items = inventory.fetch_all_items()
    for change in updates:
        target_item = None
        if change["source"] == "store":
            target_item = next((item for item in items if
item["store_id"] == change["identifier"]), None)
        else:
            target_item = next((item for item in items if
item["platform_id"] == change["identifier"]), None)

        if target_item:
            target_item["stock_level"] = change["stock_level"]
            validate_stock_change(target_item)
            inventory.modify_item(target_item)

    async def monitor_stock(store_apis: List[ECommercePlatform],
market_api: ECommercePlatform):
        """Periodically process stock updates."""
        while True:

```

```

        await asyncio.sleep(10)
        if modified_items:
            item = modified_items.pop(0)
            for store_api in store_apis:
                await store_api.update_stock(item["store_id"],
item["stock_level"])
            await market_api.update_stock(item["platform_id"],
item["stock_level"])

    def format_order_notification(orders: List[Dict], source: str) ->
None:
        """Send formatted order notifications."""
        for order in orders:
            notification = f"Order    #{order['order_number']}    for
{order['amount']}\n"
            notification += f"Payment: {order['payment_type']}\n"
            notification += f"Comments: {order['comments']}\n"
            notification += "Items:\n"
            for product in order["products"]:
                notification += f"- {product['name']}\n"
                notification += f"  Qty: {product['quantity']}\n"
                notification += f"  Stock: {product['stock']}\n"
                notification += f"  ID: {product['id']}\n"
            notifier._broadcast(notification)

    async def monitor_orders(store_apis: List[ECommercePlatform],
market_api: ECommercePlatform):
        """Track and process new orders."""
        await asyncio.sleep(2)
        sync_prompt = prompt_for_input("Sync data from first Prom.ua
store? (1 = yes, 0 = no): ")
        if sync_prompt == "1" and store_apis:
            await store_apis[0].check_service_status()    # Placeholder
for sync logic
            print("Data synchronized!")

        while True:
            await asyncio.sleep(600)    # 10 minutes
            stock_updates = []

            for store_api in store_apis:

```

```

orders = await store_api.retrieve_active_orders()
if orders:
    format_order_notification(orders, "store")
    for order in orders:
        for product in order["products"]:
            stock_updates.append({
                "source": "store",
                "identifier": product["id"],
                "stock_level": product["stock"] or 0
            })

market_orders = await market_api.retrieve_active_orders()
if market_orders:
    format_order_notification(market_orders, "market")
    for order in market_orders:
        for product in order["products"]:
            stock_updates.append({
                "source": "market",
                "identifier": product["id"],
                "stock_level": product["stock"] or 0
            })

if stock_updates:
    await asyncio.get_event_loop().run_in_executor(None,
update_inventory, stock_updates)

def start_background_task(coroutine, *args):
    """Run an async task in a separate thread."""
    loop = asyncio.new_event_loop()
    threading.Thread(target=lambda:
loop.run_until_complete(coroutine(*args)), daemon=True).start()

if __name__ == "__main__":
    # Initialize services
    notifier, store_apis, market_api = setup_connections()

    # Start background tasks
    start_background_task(monitor_stock, store_apis, market_api)
    start_background_task(monitor_orders, store_apis, market_api)

    # Launch Flask server

```

```
app.run(host="0.0.0.0", port=8080, debug=False)
```

A.6 Текст файла «storage_manager.py»

```
import sqlite3
from typing import List, Dict, Optional

class InventoryManager:
    def __init__(self, database_path: str):
        self._db_path = database_path
        self._initialize_database()

    def _initialize_database(self) -> None:
        """Set up the inventory database schema."""
        with sqlite3.connect(self._db_path, timeout=10.0) as conn:
            cursor = conn.cursor()
            schema = """
            CREATE TABLE IF NOT EXISTS inventory (
                item_id INTEGER PRIMARY KEY AUTOINCREMENT,
                photo_url TEXT NOT NULL,
                item_title TEXT NOT NULL,
                store_id TEXT NOT NULL,
                platform_id TEXT NOT NULL,
                stock_level INTEGER CHECK (stock_level >= 0)
            )
            """
            cursor.execute(schema)
            conn.commit()

    def insert_item(self, item_data: Dict) -> None:
        """Add a new item to the inventory."""
        with sqlite3.connect(self._db_path, timeout=10.0) as conn:
            cursor = conn.cursor()
            query = """
            INSERT INTO inventory (photo_url, item_title,
stock_level, store_id, platform_id)
            VALUES (?, ?, ?, ?, ?)
            """
            stock
            =
self._validate_stock(item_data.get("stock_level", 0))
```

```

        cursor.execute(query, (
            item_data["photo_url"],
            item_data["item_title"],
            stock,
            item_data["store_id"],
            item_data["platform_id"]
        ))
        conn.commit()

def _validate_stock(self, stock: any) -> int:
    """Ensure stock level is valid and non-negative."""
    try:
        stock_value = int(stock)
        return max(0, stock_value)
    except (ValueError, TypeError):
        return 0

def fetch_all_items(self) -> List[Dict]:
    """Retrieve all items from the inventory."""
    with sqlite3.connect(self._db_path, timeout=10.0) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM inventory")
        rows = cursor.fetchall()
        return [{
            "item_id": row["item_id"],
            "photo_url": row["photo_url"],
            "item_title": row["item_title"],
            "store_id": row["store_id"],
            "platform_id": row["platform_id"],
            "stock_level": row["stock_level"]
        } for row in rows]

def find_item(self, item_id: int) -> Optional[Dict]:
    """Locate an item by its ID."""
    with sqlite3.connect(self._db_path, timeout=10.0) as conn:
        conn.row_factory = sqlite3.Row
        cursor = conn.cursor()
        cursor.execute("SELECT * FROM inventory WHERE item_id =
?", (item_id,))

        row = cursor.fetchone()

```

```

        if row:
            return {
                "item_id": row["item_id"],
                "photo_url": row["photo_url"],
                "item_title": row["item_title"],
                "store_id": row["store_id"],
                "platform_id": row["platform_id"],
                "stock_level": row["stock_level"]
            }
        return None

def modify_item(self, item_data: Dict) -> None:
    """Update an existing item's details."""
    with sqlite3.connect(self._db_path, timeout=10.0) as conn:
        cursor = conn.cursor()
        query = """
            UPDATE inventory
            SET photo_url = ?, item_title = ?, stock_level = ?,
store_id = ?, platform_id = ?
            WHERE item_id = ?
        """
        stock =
self._validate_stock(item_data.get("stock_level", 0))
        cursor.execute(query, (
            item_data["photo_url"],
            item_data["item_title"],
            stock,
            item_data["store_id"],
            item_data["platform_id"],
            item_data["item_id"]
        ))
        conn.commit()

def remove_item(self, item_id: int) -> None:
    """Delete an item from the inventory."""
    with sqlite3.connect(self._db_path, timeout=10.0) as conn:
        cursor = conn.cursor()
        cursor.execute("DELETE FROM inventory WHERE item_id =
?", (item_id,))
        conn.commit()

```

A.7 Текст файлу «marketplace.py»

```

import httpx
import aiohttp
import base64
import asyncio
from typing import Dict, List, Optional

class ECommercePlatform:
    def __init__(self, client_id: str, secret_key: str):
        self._timeout = 20
        self._session = None
        self._access_key = None
        asyncio.run(self._initialize_session(client_id,
secret_key))

    async def _initialize_session(self, client_id: str, secret_key:
str):
        """Establish authenticated session with the platform."""
        credentials = {
            "client_id": client_id,
            "secret": base64.b64encode(secret_key.encode("utf-
8")).decode("utf-8")
        }
        async with aiohttp.ClientSession() as session:
            try:
                async with session.post(
                    "https://api-seller.rozetka.com.ua/auth/login",
                    json=credentials,
                    headers={"Content-Type": "application/json"},
                    timeout=self._timeout
                ) as response:
                    data = await response.json()
                    if data.get("success"):
                        self._access_key = data["content"]["token"]
                        print(f"Connected with token:
{self._access_key}")
                    else:
                        print("Authentication failed")
            except aiohttp.ClientError as e:

```

```

        print(f"Connection error: {e}")

    async def _perform_api_call(self, method: str, url: str, data:
Optional[Dict] = None, params: Dict = None, custom_headers: Dict = None) -
> Optional[Dict]:
        """Execute an API request with proper headers and error
handling."""
        headers = custom_headers or {
            "Authorization": f"Bearer {self._access_key}",
            "Accept": "application/json"
        }
        async with aiohttp.ClientSession() as session:
            try:
                async with session.request(
                    method, url, json=data, params=params,
headers=headers, timeout=self._timeout
                ) as response:
                    result = await response.json()
                    return result["content"] if
result.get("success") else None
            except aiohttp.ClientError as e:
                print(f"API call failed: {e}")
                return None

    async def fetch_order_statuses(self) -> Dict:
        """Retrieve available order status options."""
        headers = {"Authorization": f"Bearer {self._access_key}"}
        result = await self._perform_api_call("GET", "https://api-
seller.rozetka.com.ua/order-statuses/list", custom_headers=headers)
        return result or {"status": "failed"}

    async def get_product_catalog(self, limit: int = 5000) -> List:
        """Fetch all products from the catalog."""
        headers = {
            "Authorization": f"Bearer {self._access_key}",
            "Accept-Language": "ua"
        }
        params = {"limit": limit}
        result = await self._perform_api_call("GET", "https://api-
seller.rozetka.com.ua/products", params=params, custom_headers=headers)
        return result.get("products", []) if result else []

```

```

        async def modify_product_data(self, product_info: Dict) ->
Optional[Dict]:
            """Update product information in bulk."""
            result = await self._perform_api_call("PATCH",
"https://api-seller.rozetka.com.ua/items/bulk", data=product_info)
            return result

        async def update_stock(self, item_id: str, stock: int) ->
Optional[Dict]:
            """Adjust inventory for a specific product."""
            payload = {"products": [{"id": item_id, "stock": stock}]}
            return await self.modify_product_data(payload)

    async def retrieve_active_orders(self) -> List[Dict]:
        """Fetch and format pending orders."""
        params = {
            "status": "pending",
            "include": "items,pricing,delivery"
        }
        headers = {
            "Authorization": f"Bearer {self._access_key}",
            "Accept-Language": "ua"
        }
        result = await self._perform_api_call("GET", "https://api-
seller.rozetka.com.ua/orders", params=params, custom_headers=headers)
        if not result:
            return []

        formatted_orders = []
        for order in result.get("orders", []):
            order_details = {
                "order_number": order["order_id"],
                "amount": order["total_price"],
                "comments": order.get("notes", ""),
                "payment_type": order["payment"]["method"],
                "products": []
            }
            for item in order.get("items", []):
                product = {
                    "id": item["product_id"],

```

```

        "name": item["title"],
        "quantity": item["count"],
        "stock": item["available"]
    }
    order_details["products"].append(product)
    formatted_orders.append(order_details)
    return formatted_orders

    async def change_order_status(self, order_id: str, new_status:
str) -> Optional[Dict]:
        """Update the status of a specific order."""
        payload = {"state": new_status}
        return await self._perform_api_call("PATCH", f"https://api-
seller.rozetka.com.ua/orders/{order_id}/status", data=payload)

```

A.8 Текст файлу «messenger.py»

```

import httpx
from loguru import logger
from time import strftime

class NotificationService:
    def __init__(self, token, channel):
        self._auth = token
        self._endpoint = "sendMessage"
        self._target = channel
        self._configure_logging()
        self._broadcast(f"Service                initialized                at
{strftime('%H:%M:%S')}")

    def _configure_logging(self):
        """Initialize logging system with custom configuration."""
        logger.add(
            f"service_{strftime('%d%m%Y')}.log",
            format="{time} | {level} | {message}",
            level="DEBUG",
            rotation="1 MB"
        )
        logger.debug("Logging system ready")

```

```

async def _broadcast(self, content):
    """Send notification to the designated channel."""
    async with httpx.AsyncClient() as client:
        try:
            payload = {"chat_id": self._target, "text":
content}

            response = await client.post(

f"https://api.telegram.org/bot{self._auth}/{self._endpoint}",
                json=payload,
                timeout=10
            )
            if response.is_success:
                logger.info(f"Notification sent: {content}")
            else:
                logger.error(f"Failed to send:
{response.text}")
        except httpx.HTTPError as err:
            logger.error(f"Network issue: {err}")

async def check_service_status(self):
    """Validate connection to the notification API."""
    async with httpx.AsyncClient() as client:
        try:
            response = await client.get(

f"https://api.telegram.org/bot{self._auth}/getMe",
                timeout=8
            )
            if response.status_code == 200:
                data = response.json()
                logger.info(f"Service active:
{data['result']['username']}")
                return True
            logger.error(f"Status check failed:
{response.text}")
            return False
        except httpx.HTTPError as err:
            logger.error(f"Status check error: {err}")
            return False

```

ДОДАТОК Б
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

РОЗРОБКА ПРОГРАМНОГО ЗАСТОСУНКУ "СИСТЕМА УПРАВЛІННЯ ТОВАРАМИ ДЛЯ УКРАЇНСЬКИХ МАРКЕТПЛЕЙСІВ"

Виконав
студент групи КНТ-151

Віталій ТРОЦ

Керівник
к.т.н. доцент

Євгеній ГОФМАН

Рисунок Б.1 – Слайд 1

Мета та задачі дипломної роботи

Мета роботи – розробка та програмна реалізація панелі керування товарами, що автоматизує процес менеджменту товарів, замовлень та їх обліку для інтеграції з українськими маркетплейсами.

Задачі роботи:

- аналіз вимог до системи управління товарами;
- проектування архітектури програмного забезпечення;
- вибір технологій та інструментів розробки;
- реалізація програмного продукту;
- тестування та оптимізація системи;

2

Рисунок Б.2 – Слайд 2

Аналіз вимог до роботи

Для успішної реалізації мети дипломного проєкту було сформульовано наступний перелік функціональних і нефункціональних вимог до системи.

Функціональні вимоги:

- Інтеграція з API Prom.ua та Rozetka через введення ключів.
- Підключення кількох магазинів Prom.ua.
- Автоматичне відстеження та приймання нових замовлень.
- Сповіщення через Telegram-бот із деталями замовлення.
- Синхронізація залишків товарів у БД та магазинах.
- Імпорт товарів із маркетплейсів для ініціалізації БД.
- Додавання, редагування, видалення товарів.
- Зручний веб-інтерфейс для віддаленого доступу.

Нефункціональні вимоги:

- Надійність: стабільна робота, обробка помилок.
- Продуктивність: швидка синхронізація даних.
- Зручність: інтуїтивний інтерфейс для всіх користувачів.
- Безпека: захист ключів API та даних.
- Масштабованість: можливість додавання нових платформ.

3

Рисунок Б.3 – Слайд 3

Актуальність роботи



Приклади існуючих систем

4

Рисунок Б.4 – Слайд 4

Порівняльна характеристика CRM-систем

Назва	Переваги	Недоліки
KeyCRM	Інтеграція з Rozetka, Prom, зручний інтерфейс, єдиний тариф 19 у.о.	Обмежене налаштування, немає пошуку дублікатів.
KeepinCRM	Інтеграція з Prom, Rozetka, управління складом, сегментація клієнтів.	Складний інтерфейс, немає інтеграції з Telegram.
Модулі маркетплейсів	Безкоштовно, пряма синхронізація, простота використання.	Базовий функціонал, немає інтеграції з іншими платформами.
InSales	Інтеграція з маркетплейсами, SEO-оптимізація, хмарне рішення.	Обмежена автоматизація, додаткові витрати.
SalesDrive	Автоматизація замовлень, ТТН, інтеграція з Prom, Rozetka, комплексне управління.	Висока ціна, складне налаштування.

5

Рисунок Б.5 – Слайд 5

Порівняльна характеристика мов програмування

Мова	Переваги	Недоліки
C++	Висока продуктивність, прямий доступ до ресурсів, широке використання в системному програмуванні, велика спільнота.	Складний синтаксис, ручне керування пам'яттю, обмежена підтримка веб-розробки.
Java	Кросплатформність через JVM, об'єктно-орієнтований підхід, велика екосистема бібліотек.	Вища витрата ресурсів, складний синтаксис, менш гнучка для прототипів.
Python	Простий синтаксис, підтримка бібліотек для веб-розробки та API, легка інтеграція з Telegram Bot API, активна спільнота.	Нижча продуктивність, динамічна типізація, обмежена низькорівнева робота.
JavaScript	Ідеальна для веб-інтерфейсів, підтримка через Node.js, багато фреймворків (React, Vue.js).	Складність асинхронного програмування, низька ефективність для серверних задач, фрагментація бібліотек.

6

Рисунок Б.6 – Слайд 6

Схема архітектури системи



7

Рисунок Б.7 – Слайд 7

Характеристика компонентів системи

Компонент	Тип	Функція
Python	Мова програмування	Виконання основного функціоналу веб-сервера, автоматизований прийом замовлень, оновлення даних про товар, робота з БД, надсилення сповіщень
JavaScript	Мова програмування	Забезпечення взаємодії користувача з веб-інтерфейсом
Visual Studio Code	Середовище розробки	Середовище для написання коду програмного продукту
SQLite	Система керування БД	Керування записом, редагуванням та видаленням даних про товар у БД
Flask	Веб-сервер	Організація функцій веб-сервера, відображення інтерфейсу користувача.
Telegram-бот	Сповіщення продавця	Сповіщення продавця про наявність нових замовлень

8

Рисунок Б.8 – Слайд 8

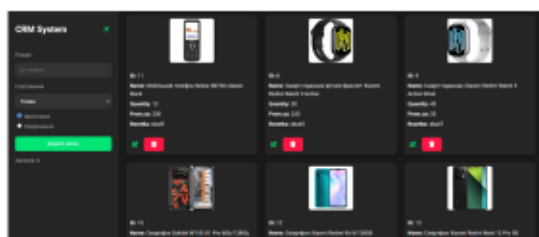
Алгоритм роботи системи управління товарами



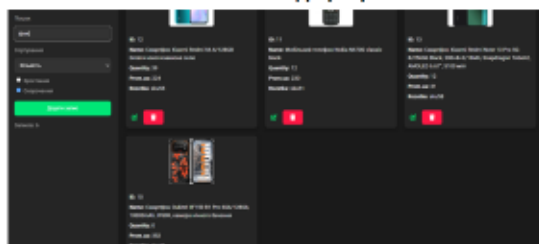
9

Рисунок Б.9 – Слайд 9

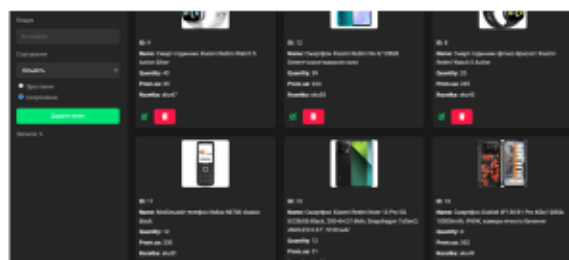
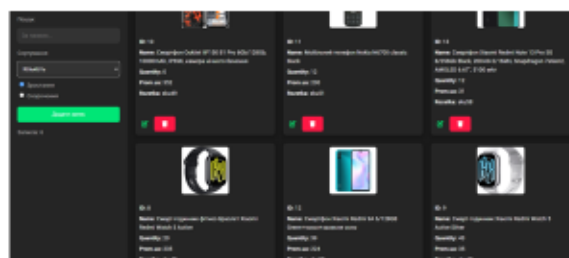
Приклади роботи системи



Зовнішній вигляд програми



Приклади пошуку по назві



Приклади сортування

10

Рисунок Б.10 – Слайд 10

Алгоритм роботи системи управління товарами



9

Рисунок Б.11 – Слайд 11

Висновки

- **Наукова новизна** роботи полягає в розробці гнучкої архітектури програмного забезпечення, яка забезпечує стабільну інтеграцію з API українських маркетплейсів та підтримує одночасну роботу з кількома магазинами. Запропонований підхід до асинхронної обробки запитів і синхронізації даних дозволяє мінімізувати навантаження на API та забезпечує високу надійність системи навіть у разі тимчасових збоїв.
- **Практична цінність** роботи полягає у створенні доступного інструменту для малого та середнього бізнесу, який автоматизує рутинні операції, зменшує час на обробку замовлень і оптимізує облік товарів. Веб-інтерфейс забезпечує зручне управління базою даних, а Telegram-бот дозволяє оперативно реагувати на нові замовлення, що сприяє підвищенню конкурентоспроможності продавців на маркетплейсах.
- **Перспективи подальшого розвитку** включають розширення функціоналу системи шляхом додавання інтеграції з іншими популярними українськими платформами, такими як Bigl.ua та OLX.ua. Планується впровадження функції експорту звітів про товари у формати CSV або Excel, а також створення мобільного додатку для зручнішого доступу до системи. Крім того, передбачається додавання аналітичного модуля для прогнозування попиту на товари на основі даних продажів.

12

Рисунок Б.12 – Слайд 12

Дякую за увагу

Рисунок Б.13 – Слайд 13