Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра
(освітньо-кваліфікаційний рівень)

на тему

Застосування штучного інтелекту в системі інтернет магазину

Виконав: студент 6 курсу, групи 601-ТН
спеціальності
122 Комп'ютерні науки
(шифр і назва напряму)
Борківець К.М.
(прізвище та ініціали)
Керівник Руденко О.А.
(прізвище та ініціали)
Рецензент Підяшенко В.Є.
(прізвище та ініціали)

Полтава – 2024 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
«ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

спеціальність 122 «Комп'ютерні науки»

на тему

«Застосування штучного інтелекту в системі інтернет магазину»

Студента групи 601-ТН Борківця Костянтина Максимовича

Керівник роботи
кандидат технічних наук,
доцент Руденко О.А.

Завідувач кафедри
кандидат фізико-математичних наук,
доцент Двірна О.А.

Полтава – 2024

РЕФЕРАТ

Кваліфікаційна робота магістра: 82 с., 35 малюнків, 1 додаток, 22 джерела.

Об’єкт дослідження: інформаційна система управління контентом інтернет-магазину з елементами штучного інтелекту.

Мета роботи: розробка автоматизації процесів управління контентом інтернет-магазину, оптимізація створення та аналізу контенту за допомогою інструментів штучного інтелекту, забезпечення зручного інтерфейсу для адміністрування, аналітики та звітності.

Методи: збір та аналіз потреб, проектування архітектури системи, розробка бази даних для збереження контенту, створення веб-додатку у середовищі Django, інтеграція інструментів штучного інтелекту для генерації текстових описів та аналітики, використання бібліотек для візуалізації та формування звітів.

Ключові слова: управління контентом, штучний інтелект, інтернет-магазин.

ABSTRACT

Bachelor’s thesis: 82 p., 35 figures, 1 appendices, 21 sources.

Object of research: the content management information system for an online store with artificial intelligence elements.

Objective: to design and develop automating content management processes in an online store, optimizing content creation and analysis through artificial intelligence tools, and providing a user-friendly interface for administration, analytics, and reporting.

Methods: collection and analysis of requirements, system architecture design, development of a database for content storage, creation of a web application in the Django environment, integration of artificial intelligence tools for generating text descriptions and analytics, utilization of libraries for visualization and report generation.

Keywords: content management, artificial intelligence, online store.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ 5
ВСТУП 6
РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД 8
1.1 Характеристика предметної області 8
1.2 Аналіз наявних програмних продуктів 10
1.3 Функціональні можливості 15
РОЗДІЛ 2 ПРОЕКТУВАННЯ СИСТЕМИ ІНТЕРНЕТ МАГАЗИНУ 18
2.1 Структура інтернет магазину 18
2.2 База даних проекту 19
2.3 Інтеграція штучного інтелекту 21
2.4 Веб-дизайн 22
РОЗДІЛ 3 ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ МАГАЗИНУ ТА ІНТЕГРУВАННЯ ШІ 28
3.1 Вибір технологічного стеку 28
3.2 Вибір бази даних 30
3.3 Інструменти інтеграції ШІ 31
3.4 Стек веб-дизайну 33
РОЗДІЛ 4 РОЗРОБКА ІНТЕРНЕТ МАГАЗИНУ 34
4.1 Розробка бази даних 34
4.2 Створення веб-інтерфейсу 40
РОЗДІЛ 5 ТЕСТУВАННЯ 46
5.1 Визначення способу тестування 46
5.2 Тестування проекту 47
5.2.1 Тестування додатку shop 48
5.2.2 Тестування додатку basket 54
5.2.3 Тестування інтеграції ШІ 55
5.3 Підсумки тестування 56
ВИСНОВКИ 58
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 59
ДОДАТОК А ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ 62

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

NLP – Natural Language Processing

API – Application Programming Interface

SEO – Search Engine Optimization

CMS – Content Management System

CAGR – Compound Annual Growth Rate

ORM – Object-Relational Mapping

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

HTTP – HyperText Transfer Protocol

URL – Uniform Resource Locator

LSTM – Long Short-Term Memory

PNG – Portable Network Graphics

ACID – Atomicity, Consistency, Isolation, Durability

CSRF – Cross-Site Request Forgery

REST – Representational State Transfer

JSON – JavaScript Object Notation

ШІ – Штучний інтелект

БД – База даних

СУБД – Система управління базами даних

ВСТУП

У сучасному світі електронна комерція займає важливе місце в економіці, а кількість онлайн-магазинів стрімко зростає. З кожним роком конкуренція у цій сфері стає дедалі жорсткішою, що вимагає від бізнесів застосування новітніх технологій для підвищення ефективності та задоволення потреб клієнтів. Одним із важливих аспектів управління інтернет-магазином є ефективне адміністрування інформації про товари, описи, зображення та інший вміст, що представлений на сайті .

ШІ стає все більшою частиною сучасних інформаційних систем, забезпечуючи автоматизацію, персоналізацію та оптимізацію багатьох процесів. Інтеграція елементів ШІ у процеси керування контентом відкриває нові можливості для аналізу даних, автоматичного створення та оптимізації контенту, а також покращення користувацького досвіду.

Метою дипломної роботи є дослідження можливостей застосування елементів штучного інтелекту для управління контентом інтернет-магазину. Традиційні підходи стають недостатньо ефективними, при сучасних умовах зростаючого обсягу інформації, яку потрібно обробляти та управляти, . Ручне створення та оновлення контенту займає багато часу, що уповільнює роботу адміністратора і впливає на оперативність реакції на запити клієнтів. Впровадження ШІ дозволяє автоматизувати рутинні завдання, такі як генерація описів товарів, переклад, категоризація, аналіз поведінки клієнтів та персоналізація рекомендацій.

Необхідність розробки даної системи є також зростання попиту на автоматизовані та інтелектуальні рішення у сфері електронної комерції. Використання ШІ для керування контентом дозволяє значно покращити точність, швидкість і якість обслуговування клієнтів. Це, у свою чергу, підвищує рівень доходів та конкурентоспроможність інтернет-магазинів.

У рамках цієї роботи було прийнято рішення зосередитися на розробці системи, яка інтегрує елементи ШІ у процеси створення та управління

контентом. Основними функціями розроблюваної системи є автоматизація створення описів товарів, рекомендацій, аналізу запитів клієнтів та управління категоріями товарів. Для реалізації цих функцій будуть використані сучасні технології, такі як обробка природної мови (NLP), алгоритми машинного навчання та комп'ютерний зір. Вибір цих технологій обумовлений їхньою ефективністю та здатністю працювати з великими обсягами даних.

Рішення щодо використання Django як основної платформи для розробки обумовлене її гнучкістю, широкими можливостями для інтеграції сторонніх інструментів і високою продуктивністю. У роботі також будуть використані бібліотеки та API для реалізації ШІ-функціоналу, такі як OpenAI або TensorFlow.

Отримана система може бути використана в різних сферах електронної комерції, зокрема, роздрібну торгівлю, маркетплейси, спеціалізовані онлайн-магазини та B2B-платформи. Інтеграція ШІ у процеси управління контентом дозволяє не лише автоматизувати рутинні процеси, але й забезпечувати високий рівень персоналізації, що є критично важливим для збереження клієнтів та збільшення продажів.

Окрім того, результати можуть бути застосовані у створенні універсальних рішень для автоматизації контенту, які можуть масштабуватися та адаптуватися під потреби різних бізнесів. Це відкриває перспективи для подальшого розвитку технологій на основі ШІ у сфері управління контентом.

РОЗДІЛ 1
АНАЛІТИЧНИЙ ОГЛЯД

1.1 Характеристика предметної області

Система управління контентом інтернет магазину з елементами штучного інтелекту є програмним рішенням, розробленим для оптимізації процесів створення, редагування, аналізу та управління контентом в межах онлайн-магазину. Основною метою системи є автоматизація рутинних завдань та забезпечення можливості створювати індивідуалізований, релевантний та ефективний контент для товарів, категорій, блогів та інших інформаційних розділів магазину.

Контент у межах інтернет магазину є важливим чинником, який впливає на залучення та утримання клієнтів, пошукову оптимізацію (SEO) та загальну ефективність роботи магазину. Він включає такі основні елементи, як описи товарів, категорії, статті, зображення, мета-теги, огляди, а також інший супровідний текстовий і мультимедійний контент. Застосування елементів штучного інтелекту дозволяє не лише прискорити процес створення контенту, але й забезпечити його якість, унікальність та відповідність потребам цільової аудиторії [1].

Інтернет-магазини дозволяють продавати товари без необхідності фізичного магазину, забезпечуючи зручність покупки через онлайн-платформи. Важливим аспектом функціонування таких магазинів є управління контентом: правильне створення та оновлення описів товарів, завантаження зображень, встановлення цін і правильне розподілення товарів по категоріях. Ці процеси, якщо вони виконуються вручну, займають багато часу і можуть бути схильні до людських помилок, тому автоматизація таких завдань є важливою для ефективності роботи магазину.

На сьогоднішній день для автоматизації керування контентом існують різні рішення, що включають використання ШІ для автоматичного генерування

описів і визначення цін, а також для аналізу ринку і конкурентів. Одним із таких рішень є використання нейронних мереж та алгоритмів машинного навчання для аналізу даних про товари та ціни на ринку, що дозволяє зробити ціноутворення більш динамічним і адаптивним. Такі технології здатні підвищити ефективність роботи магазину, скоротити час на виконання рутинних завдань та зменшити ймовірність помилок [2].

Проблеми, з якими стикаються інтернет-магазини, включають необхідність підтримки актуальності і точності контенту, а також складність управління великими обсягами товарів і даних. За допомогою ШІ можна значно полегшити ці завдання, зробивши процеси автоматизованими та більш точними. Наприклад, автоматичне генерування описів дозволяє уникнути необхідності писати кожен опис вручну, що заощаджує час і ресурси. Аналіз середньої ціни на ринку дозволяє магазинам швидко адаптувати свої ціни, що підвищує їх конкурентоспроможність.

Аналіз існуючих рішень показує, що застосування ШІ в управлінні контентом інтернет-магазинів набуває популярності, оскільки такі технології дозволяють значно підвищити ефективність і швидкість роботи магазинів. Різноманітні наукові дослідження і публікації підтверджують, що автоматизація таких процесів, як генерація описів товарів і оптимізація ціноутворення, дозволяє значно зменшити витрати часу і людських ресурсів, а також підвищити точність і актуальність інформації.

Застосування елементів штучного інтелекту в інтернет-магазинах є перспективним напрямом розвитку, адже дозволяє інтегрувати додаткові функції, такі як персоналізація контенту для покупців, автоматичне складання асортименту на основі попиту і аналізу ринку. Такий підхід відкриває нові можливості для покращення бізнес-процесів і підвищення якості обслуговування клієнтів [3].

Ключове завдання є інтеграція ШІ для аналізу існуючого контенту, генерації нових описів, підготовки SEO-оптимізованих матеріалів та автоматичного створення рекомендацій.

Розроблений проект – це платформа для продажу товарів різних категорій, створена для зручності покупців і ефективного управління асортиментом. Важливою особливістю нашого магазину є інтеграція сучасних технологій, які забезпечують автоматизацію ряду процесів.

Магазин включає в себе різноманітні категорії товарів, кожен з яких має детальний опис, зображення, ціну і належить до конкретної підкатегорії. Це дозволяє покупцям легко знаходити необхідні товари завдяки чітко структурованому каталогу.

Одним з ключових аспектів магазину є функціональність, яка дозволяє автоматично генерувати описи товарів на основі їх назв і категорій. Це дозволяє зменшити час на додавання нових товарів на сайт і зробити цей процес більш ефективним. Окрім цього, магазин автоматично визначає оптимальні ціни для товарів, орієнтуючись на середні ринкові ціни, що дозволяє залишатися конкурентоспроможним на ринку [4].

Процес покупки в магазині простий і інтуїтивно зрозумілий. Користувачі можуть швидко додавати товари до кошика, вибирати зручний спосіб доставки та здійснювати оплату через надійні платіжні системи. Дизайн магазину сприяє комфортному перегляду товарів на різних пристроях, що забезпечує зручність користування на будь-яких платформах.

Магазин орієнтований на автоматизацію найбільш рутинних процесів, таких як управління контентом, завдяки чому його адміністратори можуть зосередитися на більш важливих аспектах ведення бізнесу, таких як управління асортиментом і маркетингові стратегії.

1.2 Аналіз наявних програмних продуктів

Сучасний світ не уявляє ефективного контент-управління інтернет-магазинами без автоматизації та штучного інтелекту. Інструменти, що автоматизують процеси керування контентом, підвищують ефективність бізнесу та відчутність ручної роботи, а також покращують перспективу у відносинах з

користувачами. У цьому підрозділі описано готові рішення, що надають з інструментарієм для автоматичного керування контенту [5].

Згідно з даними компанії MarketsandMarkets, ринок систем управління контентом (CMS) стрімко розвивається. Очікується, що його обсяг зросте з $23,5 млрд у 2021 році до $43,9 млрд до 2026 року зі середньорічним темпом зростання (CAGR) у 12,6% [6]. Це пояснюється зростанням попиту на персоналізований контент, необхідністю ефективної обробки великих обсягів даних та інтеграцією технологій штучного інтелекту.

Основними характеристиками сучасних рішень для управління контентом є такі:

- персоналізація: здатність адаптувати контент під індивідуальні потреби користувачів;
- автоматизація: зменшення ручної праці завдяки використанню алгоритмів ШІ для створення, обробки та публікації контенту;
- інтеграція: можливість легко підключати систему до інших платформ, таких як CRM або аналітичні інструменти;
- масштабованість: здатність працювати з великими обсягами контенту без втрати продуктивності.

Після огляду ринку систем управління контентом було знайдено чимало програмних продуктів, серед них такі:

1. Adobe Experience Manager (AEM);
2. Shopify;
3. WordPress із плагінами WooCommerce;
4. Contentful;
5. Strapi.

Adobe Experience Manager – це комплексна платформа управління контентом, яка інтегрує інструменти створення контенту, його публікації та аналітики. Завдяки функції документообігу маркетологи можуть створювати та редагувати веб-сторінки, за допомогою Microsoft Word або Google Docs. Доступний візуальний редактор, що дозволяє створювати та редагувати сторінки

в зручному інтерфейсі. Платформа також підтримує повторне використання контенту на різних сайтах, ефективне керування метаданими та тегами, а також прискорює процес перекладу контенту [7].

Функціональні можливості:

- автоматизоване створення персоналізованого контенту за допомогою ШІ Adobe Sensei;
- інструменти для управління цифровими активами (Digital Asset Management, DAM);
- інтеграція з іншими продуктами Adobe, такими як Adobe Analytics і Adobe Target;
- можливість створення омніканальних кампаній.

Серед переваг даної системи, це підтримка складних мультимедійних проектів, що є гарним рішенням для великих компаній. Крім того, платформа забезпечує високий рівень персоналізації, дозволяючи адаптувати цифровий досвід під кожного клієнта.. Водночас варто зазначити, що висока вартість ліцензії робить систему менш доступною для малого бізнесу. Додатково, складність у використанні може стати бар'єром для невеликих команд без значного технічного досвіду.

AEM є потужним інструментом для великих компаній, однак його висока ціна та складність впровадження роблять його менш придатним для малих інтернет-магазинів.

Shopify – це провідна глобальна компанія у сфері електронної комерції, яка надає необхідну інтернет-інфраструктуру для бізнесу будь-якого розміру. Її платформа дозволяє підприємцям створювати, керувати та розвивати свої онлайн-магазини, пропонуючи інструменти для продажу як в інтернеті, так і офлайн [8].

Заснована у 2006 році в Оттаві, Канада, Shopify починалася як інтернет-магазин з продажу сноубордів. Незадоволені існуючими на той час рішеннями для електронної комерції, засновники вирішили створити власну платформу. Вона включає базові функції управління контентом, такі як:

- інструменти для створення та редагування сторінок;
- інтеграція з платформами соціальних медіа;
- вбудовані аналітичні інструменти;
- доступ до сторонніх додатків через App Store Shopify.

Платформа відзначається простотою використання, підтримує велику кількість інтеграцій із сторонніми сервісами, що дозволяє розширювати функціонал магазину відповідно до потреб бізнесу. Також її відносно низька вартість є значною перевагою для малого бізнесу. Але для складних проектів можливості кастомізації є обмеженими, що може ускладнити реалізацію унікальних рішень. Окрім цього, платформа стягує досить високі комісії за транзакції, що може стати додатковим фінансовим навантаженням для власників магазинів.

У третьому кварталі 2024 року компанія повідомила про зростання доходу на 26% порівняно з попереднім роком, досягнувши 2,16 мільярда доларів США. Обсяг валових товарних операцій (GMV) збільшився на 24%, що свідчить про зростаючу популярність платформи серед продавців та покупців. Такі результати були досягнуті завдяки впровадженню нових технологій, зокрема функцій на основі штучного інтелекту, такі як Sidekick та Shopify Magic [9].

Shopify добре підходить для малого та середнього бізнесу, але його функціональності може бути недостатньо для великих магазинів із великим обсягом контенту.

WordPress є однією з найпопулярніших систем управління контентом у світі. У комбінації з **WooCommerce** платформа пропонує розширені можливості для управління інтернет-магазинами. Це популярне рішення для створення інтернет-магазинів, яке поєднує в собі гнучкість платформи WordPress і функціональність плагіна для електронної комерції WooCommerce. Це дуо надає такі функціональні можливості:

- гнучка система управління контентом;
- інтеграція з тисячами плагінів, включаючи WooCommerce для e-commerce функцій;

- підтримка SEO-оптимізації.

Основною перевагою WordPress із WooCommerce є те, що базова версія обох платформ є безкоштовною, що робить їх доступними для широкого кола користувачів. Також система має велику спільноту розробників, які постійно створюють нові розширення, теми та пропонують технічну підтримку. Є можливість повної кастомізації.

Однак, платформа вимагає самостійного налаштування, що може бути складним для недосвідчених користувачів, а також потребує регулярного технічного обслуговування, включаючи оновлення і резервне копіювання. Крім того, використання численних плагінів підвищує ризик вразливостей до атак, що вимагає додаткової уваги до безпеки.

Ця система є чудовим вибором для тих, хто має технічні знання або команду розробників, однак потребує значного часу для налаштування. Також вона не взаємодіє зі штучний інтелектом, тому не підходить для нашого програмного рішення.

Contentful – це сучасна платформа для керування контентом, заснована в 2013 році в Берліні, Німеччина. Вона спеціалізується на "безголових" системах (headless CMS), що дозволяє компаніям створювати та керувати контентом, який можна публікувати на різних платформах та каналах [10]. Вона орієнтована на використання API для інтеграції з іншими системами. Серед її функціональних можливостей:

- API-first підхід для створення омніканальних рішень;
- підтримка структурованого контенту;
- інтеграція з платформами аналітики та автоматизації.

Contentful має високу гнучкість, що дозволяє адаптувати платформу під різні потреби бізнесу. Вона ідеально підходить для великих проектів, що використовують багатоканальний підхід, дозволяючи публікувати контент на різних платформах та пристроях. Однак, вартість використання може бути високою для малих і середніх бізнесів, які швидко розвиваються, оскільки платформа часто вимагає додаткових інвестицій для масштабування.

Але Contentful наразі не є платформою, що спеціалізується на використанні штучного інтелекту безпосередньо в основних функціях управління контентом. Тому вона не підходить для вирішення нашої проблеми.

Strapi – це open-source headless CMS, яке надає можливість кастомізації під специфічні потреби бізнесу. Цю систему використовують такі відомі бренди, як Toyota, IBM та Walmart [11]. Основні функціональні можливості системи:

- повна кастомізація за допомогою коду;
- інтеграція з базами даних і сторонніми сервісами через API;
- підтримка багатомовного контенту;
- інтернаціоналізація.

Однією з головних переваг Strapi є можливість економити час, заощаджуючи зусилля на початковій розробці проекту з нуля. Також основними перевагами є безкоштовність системи та широкі можливості кастомізації. Але це відображується в обеженості вбудованого функціоналу. І як минула система теж не фокусується на використанні штучного інтелекту, тобто, не задовольняє наші потреби.

Проаналізовані рішення пропонують широкий спектр функціональних можливостей для управління контентом. Однак більшість із них або не підходять через високу вартість, або потребують значного технічного досвіду. У випадку інтернет-магазину, який потребує інтеграції ШІ для автоматизації контенту, варто розглянути розробку власного рішення, яке враховуватиме специфічні потреби бізнесу. Це дозволить досягти високої ефективності, зберігаючи контроль над функціональністю та вартістю системи.

1.3 Функціональні можливості

Однією з ключових функцій є зручність створення, редагування та обробки контенту, що реалізується через надання інтуїтивно зрозумілих форм і редакторів. Це дозволяє адміністраторам та контент-менеджерам працювати з

великими обсягами інформації, включаючи текстові описи товарів, зображення, метадані та інші елементи контенту магазину.

Для забезпечення автоматизації процесів, система інтегрує елементи штучного інтелекту для автоматичної генерації текстових описів товарів на основі наявних характеристик. Використання алгоритмів аналізу та обробки даних дозволяє суттєво знизити навантаження на персонал і скоротити час на створення нового контенту.

Система також повинна підтримувати аналітику та звітність, що надає адміністраторам можливість контролювати ефективність контенту. Аналітичні інструменти допомагають визначити, які товари або розділи інтернет-магазину мають найвищий попит, а які потребують оптимізації. Генерація звітів реалізується за допомогою інтегрованих бібліотек для візуалізації даних, що дозволяє отримувати зрозумілі графіки, діаграми та текстові звіти.

Особлива увага приділяється рольовій моделі доступу: різні категорії користувачів матимуть певні права на виконання операцій у системі. Звичайні користувачі можуть переглядати контент, авторизовані працівники – додавати та оновлювати дані, а адміністратори отримують повний контроль, включаючи можливість редагування, видалення та завантаження звітів.

Також важливим функціональним блоком є пошук і фільтрація контенту. Користувачі системи матимуть можливість швидко знаходити необхідну інформацію за ключовими параметрами, такими як категорії товарів, дата створення, теги чи популярність.

Загалом, функціональні вимоги до системи спрямовані на забезпечення гнучкості, автоматизації та зручності управління контентом, що дозволить оптимізувати роботу інтернет-магазину та підвищити ефективність його функціонування.

Таблиця 1.1 – Функції робітників та відвідувачів інтернет-магазину

Функціонал	**Виконавець сайту**
Створення записів користувачів.	Системний адміністратор, «superuser»
Створення групи «superusers»	Системний адміністратор
Створення нових категорій та підкатегорій	Контент менеджер
Додавання товарів	Контент менеджер, штучний інтелект
Проведення операцій над заказами та їх доставкою	Менеджер з логістики
Коректування даних товару або їх видалення	Контент менеджер
Генерація рекомендованого контенту	Контент менеджер, штучний інтелект
Покращений пошук товарів	Штучний інтелект
Формування аналітики популярності та залученості конкретного товару	Контент менеджер, штучний інтелект
Додавання товарів до кошику та зміну їх кількості	Авторизований користувач
Оформлення заказів	Авторизований користувач
Перегляд товарів та каталогу. Авторизація та реєстрація	Відвідувач сайту

РОЗДІЛ 2
ПРОЕКТУВАННЯ СИСТЕМИ ІНТЕРНЕТ МАГАЗИНУ

2.1 Структура інтернет магазину

Перед інтеграцією штучного інтелекту, необхідно розробити структуру самого інтернет магазину. Після аналізу взаємодій моделей ШІ та вмінь виконавця, було визначено паттерни будування проекту за допомогою додатку Django. Це високорівневий Python-фреймворк для веб-розробки, який заохочує швидку розробку та чистий, прагматичний дизайн. Він спрощує створення складних веб-сайтів, що базуються на базах даних, дотримуючись принципу DRY (Don’t Repeat Yourself).

Модульна структура Django дозволяє розробникам створювати багаторазові компоненти, а його багата екосистема пропонує вбудовані функції, такі як ORM (Object-Relational Mapping), аутентифікація та адміністративний інтерфейс. Django широко визнаний за масштабованість, безпеку та гнучкість, що робить його ідеальним вибором для проектів, починаючи від простих веб-сайтів до складних корпоративних систем. Крім того, розробники цінують велику кількість сторонніх пакетів та активну спільноту, яка постійно вдосконалює фреймворк [12].

Розроблений сайт буде надавати різноманітні функціональні можливості для звичайних користувачів і адміністраторів. Звичайні користувачі можуть переглядати та шукати товари, а також взаємодіяти зі створеним штучним інтелектом контентом, таким як описи та рекомендації щодо цін. Адміністратори, які мають статус суперкористувачів, отримують розширені можливості, зокрема додавання, редагування або видалення товарів, а також створення аналітики. Для візуалізації цих ролей створена діаграма варіантів використання (рис. 2.1), яка демонструє взаємодії. Такий структурований підхід забезпечує чіткість у поведінці користувачів і очікуваннях від проекту.

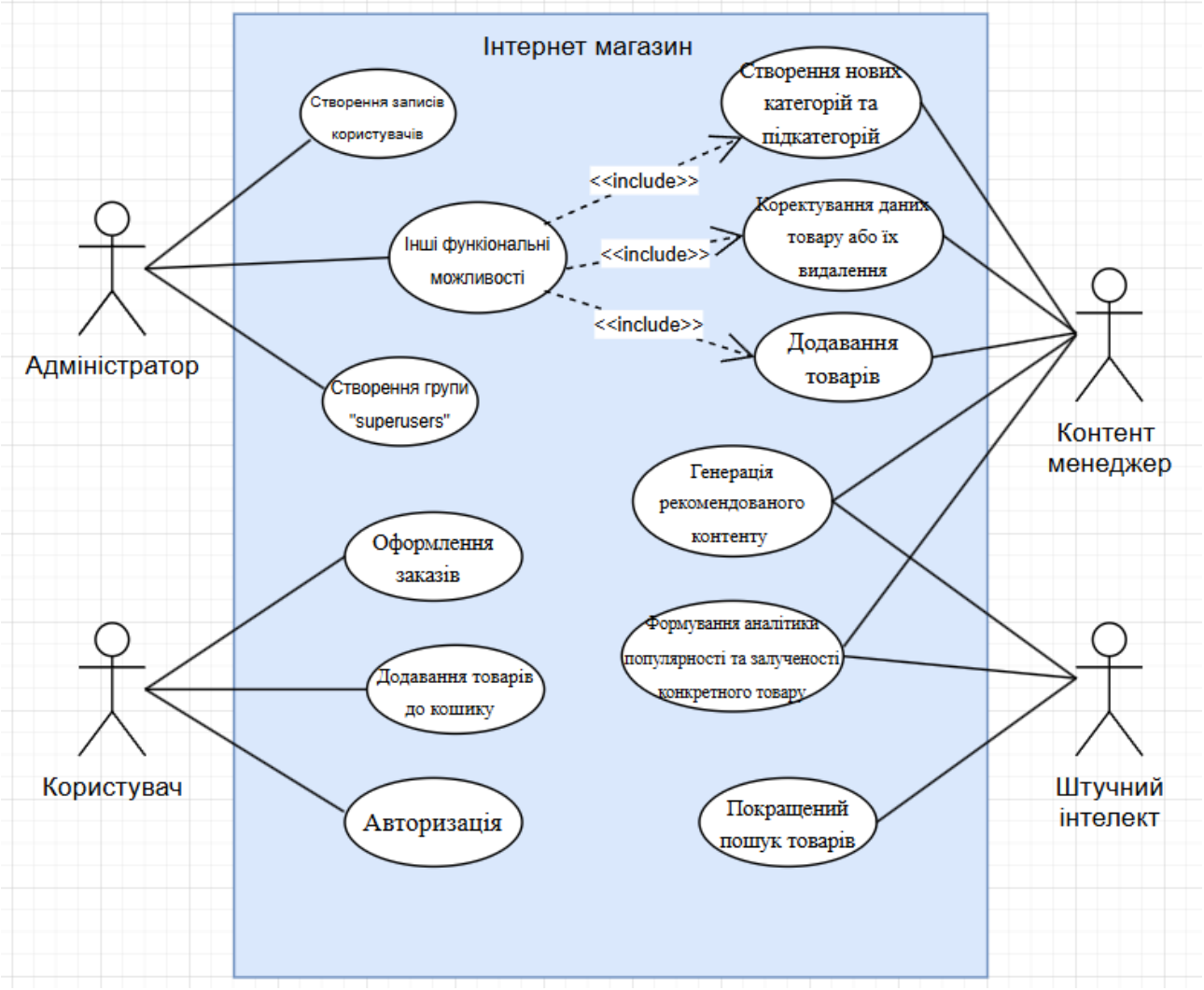

Рисунок 2.1 – Use case діаграма магазину

2.2 База даних проекту

Для управління даними використовується PostgreSQL, яка є системою управління базами даних завдяки своїй високій продуктивності та багатому набору функцій. PostgreSQL, або Postgres, є розширюваною відкритою реляційною базою даних, яка відповідає стандартам SQL. У цьому проекті схема бази даних розроблена з урахуванням необхідності включення таких основних таблиць, як "Products", "Categories" і "shop_customer". Кожна таблиця нормалізована для зменшення надмірності, зберігаючи при цьому зв’язки за допомогою зовнішніх ключів. Крім того, розширені можливості індексації Postgres, такі як GIN (Generalized Inverted Index), забезпечують ефективність

запитів, що робить її ідеальною для обробки складних пошукових операцій у системі.

За допомогою інтерфейсу dbeaver зображуємо нашу базу даних(рис. 2.2):

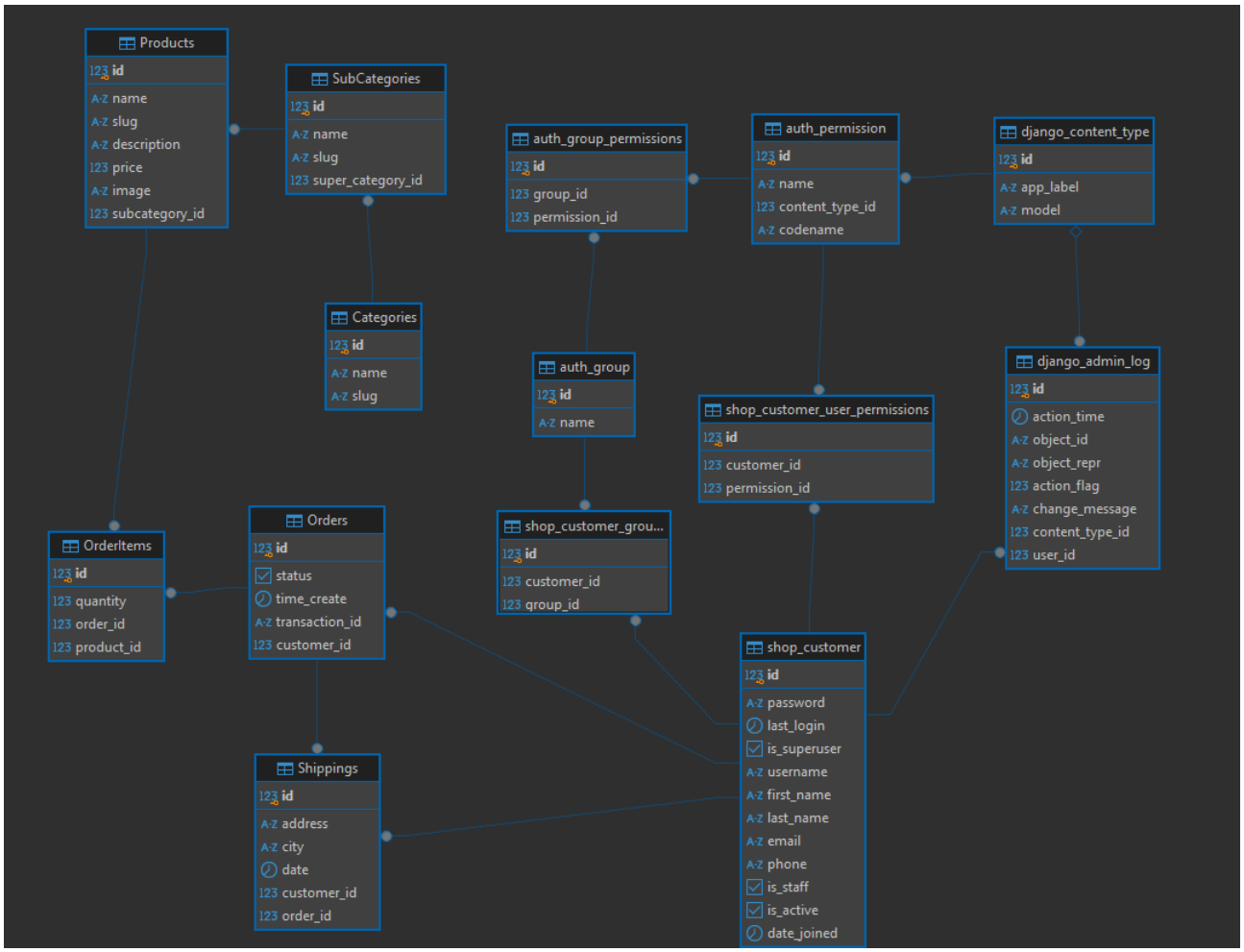

Рисунок 2.2 – Структура БД

На зображеному рисунку, окрім вище вказаних таблиць, також бачимо такі як:

- SubCategories (таблиця підкатегорій, яка ділиться на конкретніші категорії);
- OrdeItem (таблиця яка зберігає продукти як елементи окремих замовлень);
- Orders (таблиця повністю сформованих замовлень);
- Shippings (доставки);
- django_admin_log (таблиця логів з інформацією та дозволами адміністраторів);

- група таблиць auth (усі таблиці з цією приписку мають відношення до авторизації, так як була зроблена кастомна система авторизації).

2.3 Інтеграція штучного інтелекту

Інтеграція API OpenAI є наріжним каменем проекту, що забезпечує функціональність на основі штучного інтелекту, таку як генерація описів товарів і аналіз ринкових цін. OpenAI – це організація, відома своїми новітніми дослідженнями в галузі штучного інтелекту, пропонує моделі, такі як GPT-4, які здатні розуміти та генерувати тексти, схожі на ті, що створює людина.

Ці моделі базуються на архітектурі трансформерів – сучасній нейронній мережі, яка обробляє контекстуальну інформацію у великих обсягах даних. У цьому проекті API OpenAI використовується для аналізу введених контент менеджером даних про товари та створення детальних описів, специфічних для категорій [13].

Наприклад, введення користувачем "4K Smart TV" викликає створення AI-згенерованого опису, що спирається на дані з інтернету та базу попередньо натренованих знань. Крім того, OpenAI допомагає в аналізі ринку, аґрегуючи дані про ціни з зовнішніх джерел для рекомендації конкурентних стратегій ціноутворення.

Реалізація передбачає безпечні API-виклики, обробку помилок у разі перевищення лімітів API та динамічне налаштування на основі введених даних користувачем. Документація OpenAI надає вичерпні вказівки для максимально ефективного використання її функцій, що робить її незамінним інструментом для цього проекту.

Спрощена архітектура моделі OpenAI :

Складний багатовимірний простір спостережень обробляється в єдиний вектор, який передається через LSTM із 4096 нейронів. Стан LSTM проектується для отримання результатів політики (дії та функція цінності). Кожен із п’яти героїв команди керується копією цієї мережі, яка отримує майже однакові вхідні

дані, але має власний прихований стан. Мережі виконують різні дії завдяки частині виходу обробки спостережень, що вказує, якого з п’яти героїв потрібно контролювати. LSTM становить 84% від загальної кількості параметрів моделі [14]. Докладний опис архітектури наведено на рисунку 2.3.

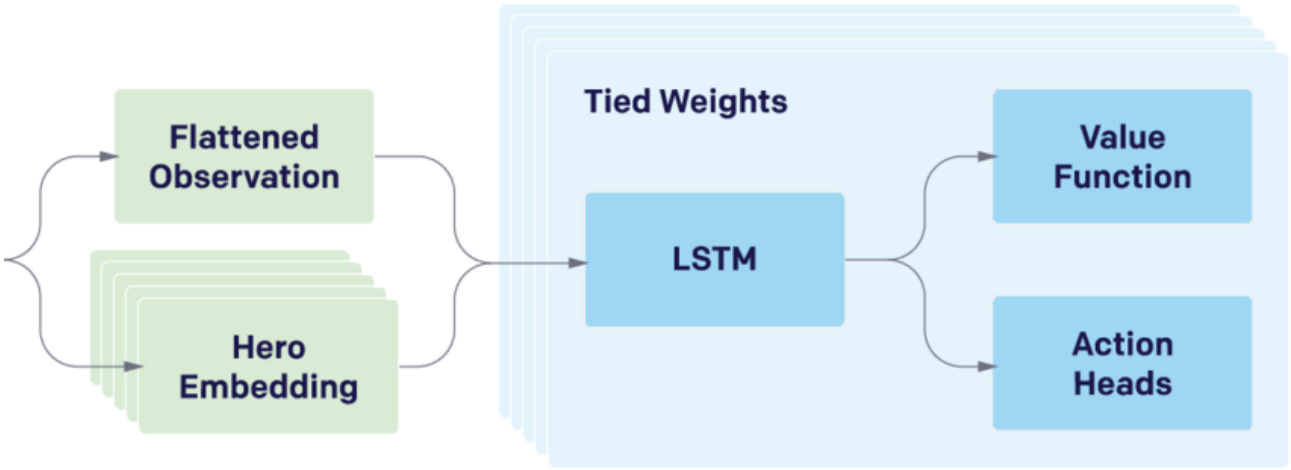

Рисунок 2.3 – Спрощена архітектура OpenAI

2.4 Веб-дизайн

Дизайн сайту орієнтований на зручність використання та естетичну привабливість, дотримуючись сучасних принципів веб-дизайну. Використання адаптивного дизайну забезпечує сумісність з різними пристроями – від настільних комп’ютерів до смартфонів. Макет створений за допомогою комбінації HTML, CSS і JavaScript, причому такі фреймворки, як Bootstrap, покращують адаптивність і зручність користування.

Основні елементи дизайну включають чисту навігаційну панель, візуально привабливі картки товарів і інтуїтивно зрозумілі форми для введення даних. Колірна схема та типографіка ретельно підібрані для відповідності брендовій ідентичності, забезпечуючи професійний і зручний інтерфейс. Етапи створення прототипів і макетів направляють процес розробки, що дозволяє враховувати відгуки користувачів на кожному етапі.

Використовуючи систему Wireframe.cc, створимо візуальні макети нашого інтернет магазину. Це мінімалістичний веб-інструмент для створення прототипів і схем інтерфейсу. Його ключова перевага полягає у простоті використання:

інтерфейс інструменту максимально спрощений, що дозволяє швидко створювати макети без зайвих налаштувань. У ньому можна легко додати такі елементи, як текст, кнопки, форми вводу, меню та зображення. Елементи автоматично адаптуються до сітки, забезпечуючи чіткість і акуратність макетів. Інструмент підтримує створення макетів для веб-сторінок, мобільних додатків і планшетів. Макети можна експортувати у форматах PNG або PDF, а зареєстровані користувачі мають можливість зберігати проекти для подальшого редагування. Хоча Wireframe.cc не має широких функціональних можливостей для створення складних інтерактивних макетів, він ідеально підходить для швидкого створення прототипів та чорнових варіантів інтерфейсів. Його простота робить його особливо корисним для зосередження на базовій структурі інтерфейсу.

Для початку побудуємо макет базової сторінки магазину:

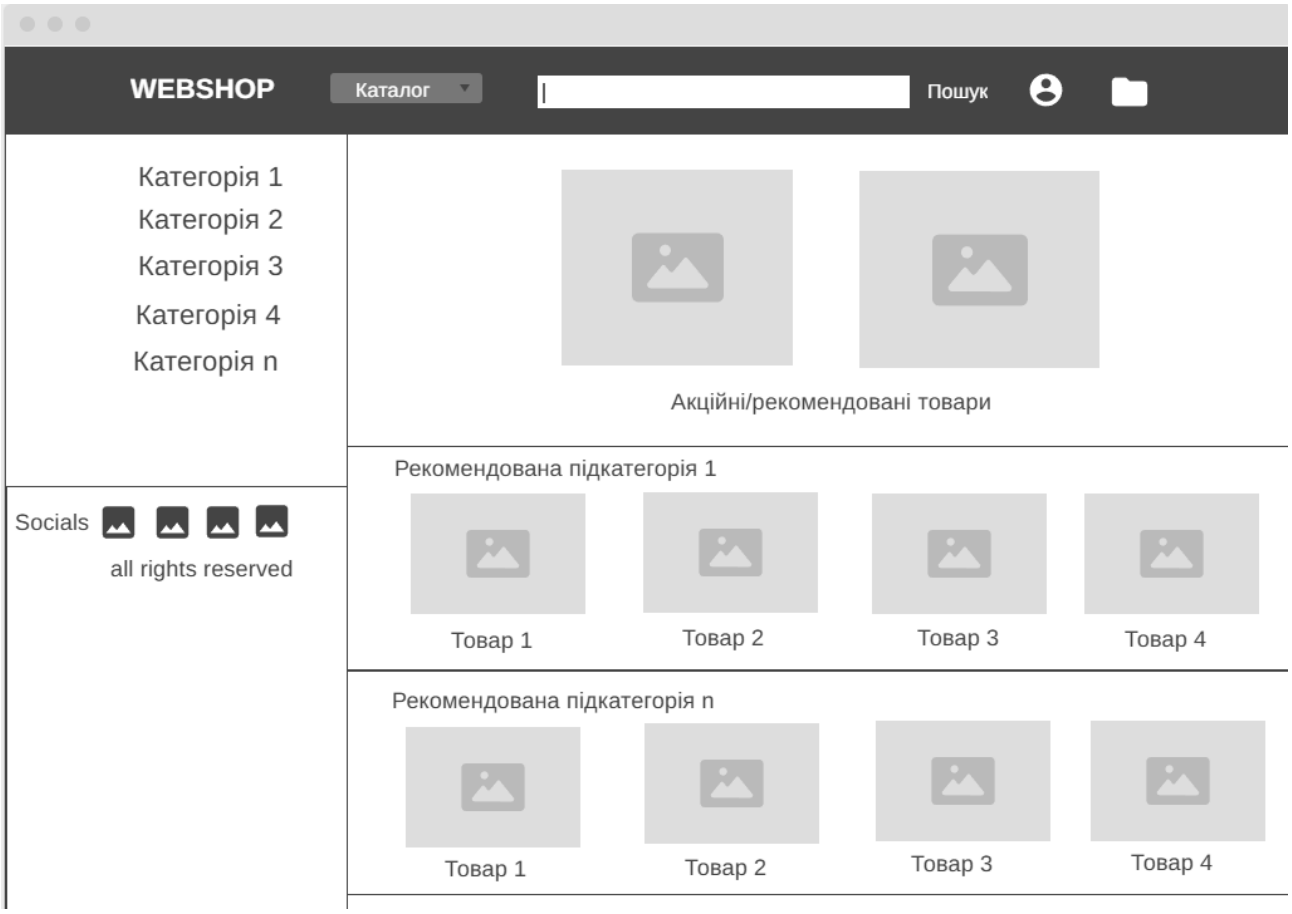

Рисунок 2.4 – Макет базової сторінки

Основні елементи:

- шапка сайту (header): містить логотип "WEBSHOP", випадаюче меню "Каталог", поле для пошуку, а також іконки для входу в особистий кабінет і перегляду кошику;
- бічне меню (sidebar): розташоване зліва і включає перелік категорій товарів (Категорія 1, Категорія 2 тощо) ";
- головна секція (main content):
  - у верхній частині розміщені блоки для акційних або рекомендованих товарів із зображеннями;
  - нижче представлено рекомендовані секції , кожна з яких містить блоки з товарами (Товар 1, Товар 2 тощо).
- футер (footer): Розміщено секцію для посилань на соціальні мережі та вказано текст "all rights reserved".

Далі робимо сторінку для категорій (рис. 2.5):

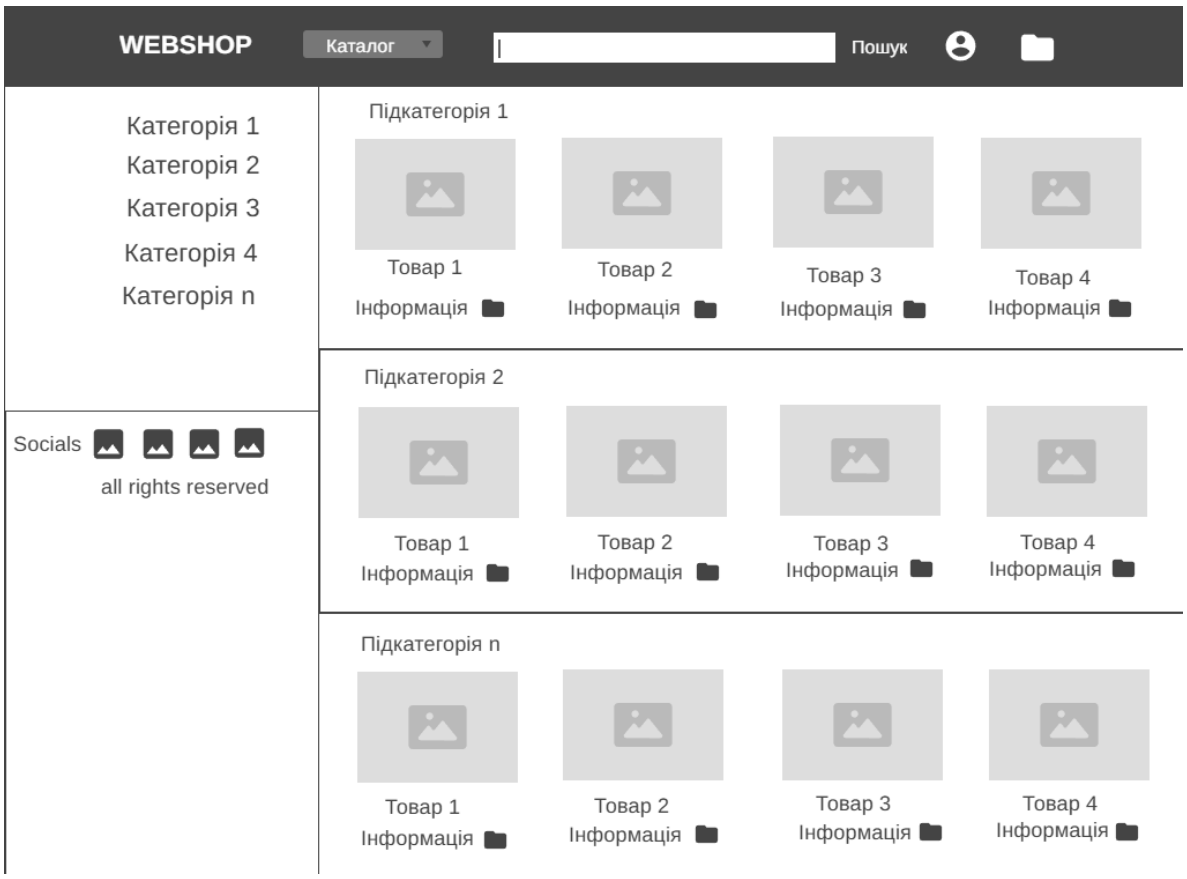

Рисунок 2.5 – Сторінка для категорій

Як бачимо основні елементи (header та sidebar) залишаються такими ж, а змінюється лише main content. У новому макеті додано текст "Інформація", що вказує на доступ до додаткових даних про товар. Також додано іконку під кожним товаром, яка буде швидко додавати товар до кошику.

Після цього формує макети “Реєстрації”(рис. 2.6), “Авторизації”(рис. 2.7) та макет “Кошику”(рис. 2.8) магазину, куди будуть додаватися товари. Вони будуть зображені у вигляді popup вікна, тобто відображатися поверху основного вікна магазину.

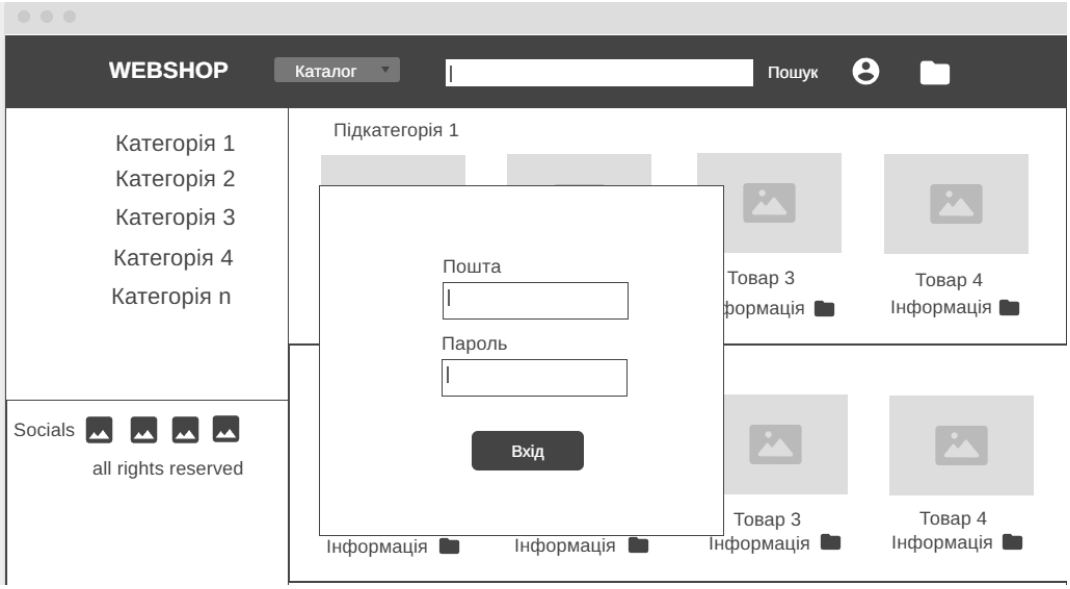

Рисунок 2.6 – Popup вікно авторизації

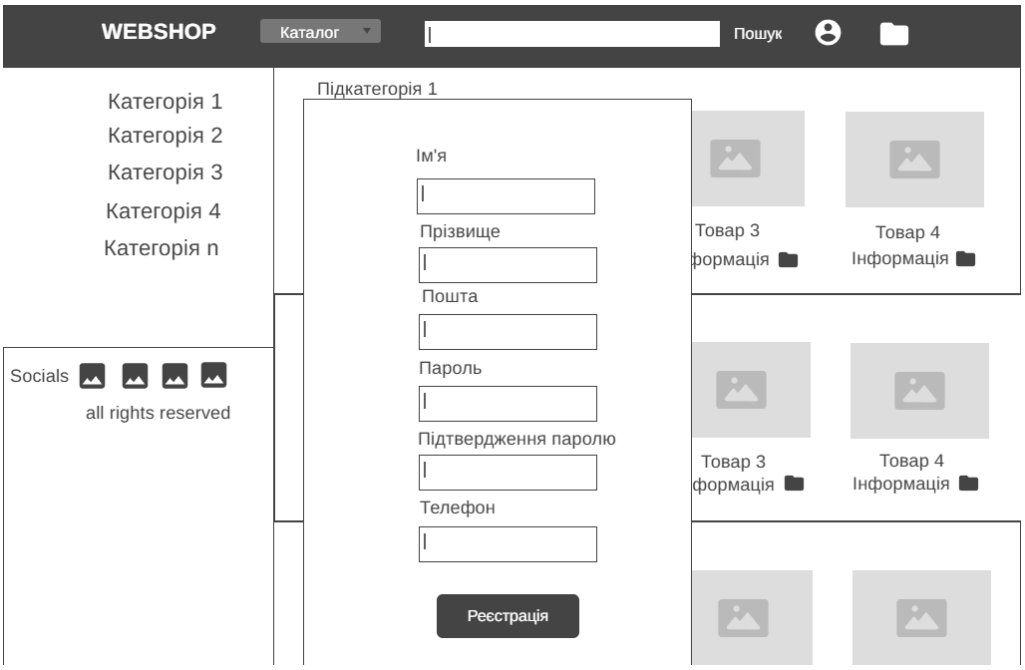

Рисунок 2.7 – Макет вікна реєстрації

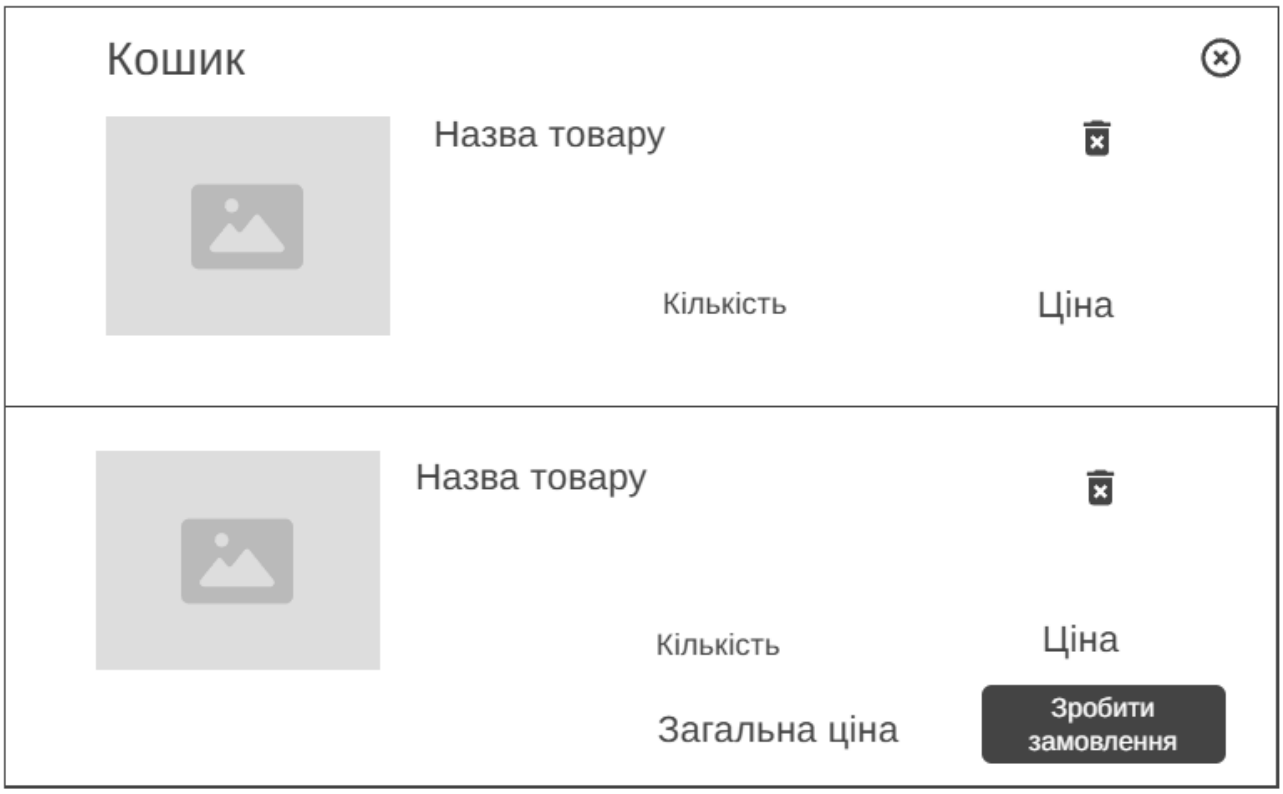

Рисунок 2.8 – Кошик товарів

Далі розробимо макет адмін панелі, з елементами, де буде використовуватися штучний інтелект, а саме у відділі створення товарів.

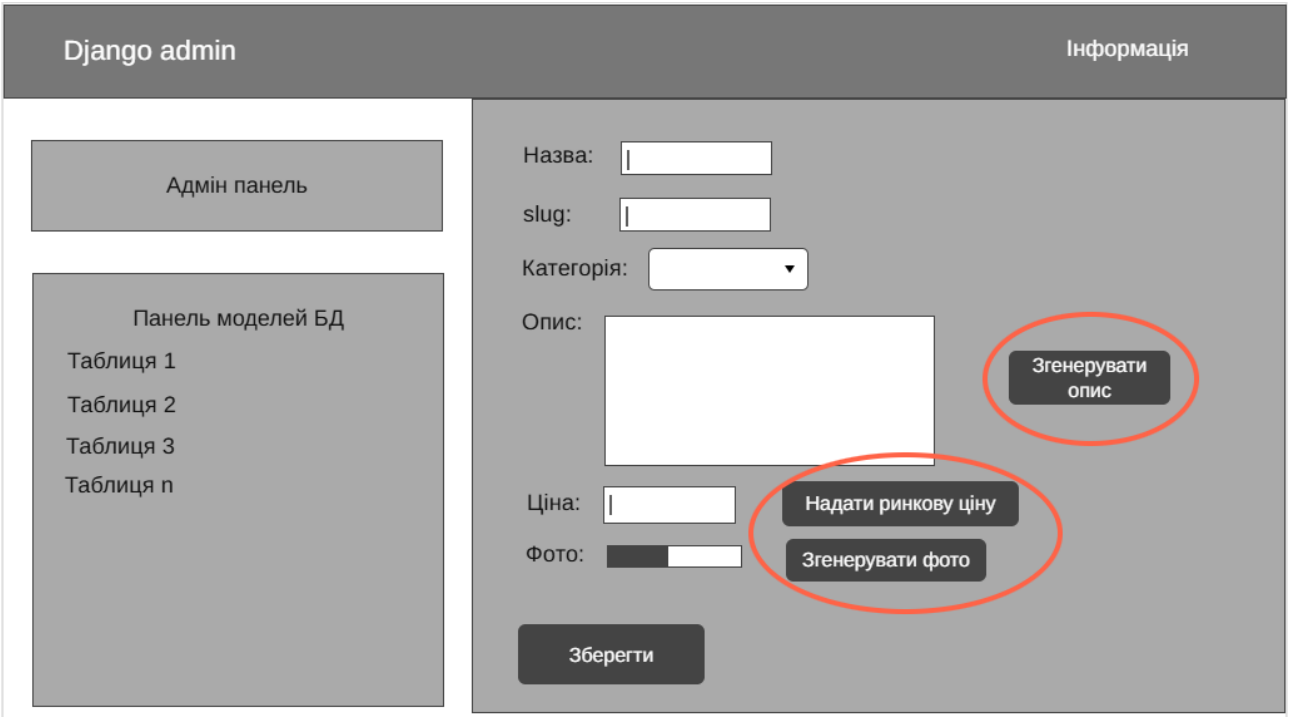

Рисунок 2.9 – Макет адмін панелі

Як бачимо, адмін сторінка також поділена на бокову модель (sidebar) та основний контент (main content). Зліва знаходиться адмін панель, яка дає змогу

на створення користувачів. Під нею знаходиться панель моделей(таблиць), яка відображає всі дані, які занесені у базу даних. Також вона дозволяє додавати елементи одразу на сторінці, як зображено на макеті.

У контенті надано приклад заповнення даних товару на запис до БД. Червоними еліпсами відмічено кнопки, які інтегрують штучний інтелект до нашого проекту. Тобто коли адміністратор включає новий товар у панелі керування (Django Admin), йому потрібно ввести лише основну інформацію: назву товару та категорію. Решта процесів автоматизована.

1. Генерація опису товару: Після введення назви товару та вибору категорії адміністратор натискає кнопку "Згенерувати" біля поля опису. Штучний інтелект аналізує інформацію з інтернету та створює релевантний текст. Наприклад, якщо товар – це смартфон, система згенерує опис його основних характеристик та переваг.

2. Рекомендація ціни: Використовуючи OpenAI, система аналізує середню ціну товару на ринку України. Це дозволяє запропонувати оптимальну ціну, яка буде конкурентоспроможною та вигідною для бізнесу.

3. Автоматичне додавання зображення: Натиснувши кнопку "Знайти зображення", адміністратор запускає процес пошуку релевантного фото товару. Отримане зображення завантажується у поле "image" для перевірки та подальшого збереження.

РОЗДІЛ 3
ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ МАГАЗИНУ ТА ІНТЕГРУВАННЯ ШІ

3.1 Вибір технологічного стеку

Для розробки інтернет-магазину з інтеграцією елементів штучного інтелекту було обрано стек технологій Python та Django. Це рішення ґрунтується на ряді технічних, функціональних і практичних переваг, які найкраще відповідають вимогам проекту та забезпечують його ефективну реалізацію. Обрання саме цього стеку дозволяє оптимально поєднати зручність розробки, продуктивність, гнучкість і масштабованість.

Python – це універсальна мова програмування, яка відома своєю простотою, лаконічністю та потужністю. Вона ідеально підходить для розробки веб-додатків завдяки великій кількості бібліотек і фреймворків, які значно полегшують процес створення та підтримки проекту. У випадку інтернет-магазину на Python, ця мова дозволяє швидко писати зрозумілий і чистий код, що є критично важливим для забезпечення ефективної роботи з великою кількістю даних і взаємодії з API, такими як OpenAI [15].

Django, як один із найпопулярніших фреймворків для Python, було обрано завдяки його здатності забезпечувати високу продуктивність і безпеку веб-додатків. Однією з головних переваг Django є його концепція "batteries included" (у перекладі – "батарейки входять у комплект"), яка передбачає, що фреймворк постачається з великою кількістю готових до використання інструментів і функцій. Це включає вбудовану систему аутентифікації, ORM для роботи з базами даних, маршрутизацію URL, а також адміністративний інтерфейс, який ідеально підходить для управління контентом інтернет-магазину.

Для цього проекту особливо важливими виявилися можливості Django Admin, що дозволили інтегрувати додаткові функції для автоматизації процесів без необхідності розробки окремого інтерфейсу з нуля. Завдяки цій

функціональності адміністратор магазину може працювати зі зручними візуальними інструментами для додавання та редагування товарів, а також використовувати спеціальні інтеграції з API для генерації описів, цін і зображень.

Ще одним вагомим фактором є висока масштабованість Django. Фреймворк побудований так, щоб додатки могли легко адаптуватися до зростання навантаження, збільшення бази даних чи введення нових функцій. У перспективі це дозволяє розширювати проект, додаючи нові модулі або переходячи на більш потужне обладнання без необхідності серйозних змін у структурі програми.

Особливу увагу варто приділити й питанню безпеки. Django пропонує вбудовані механізми захисту від поширених загроз, таких як SQL-ін’єкції, XSS-атаки чи CSRF-атаки. Для інтернет-магазину, де є ризик роботи з чутливими даними користувачів, це має особливе значення. Завдяки цьому фреймворку можна забезпечити базовий рівень захисту навіть на ранніх етапах розробки, що значно спрощує підтримку безпеки у майбутньому.

Ще один аргумент на користь Python та Django – це велика спільнота розробників і широкий вибір документації. У разі виникнення складнощів, розробник завжди може звернутися до численних форумів, посібників і готових прикладів, що значно полегшує процес вирішення проблем.

Для інтеграції з OpenAI API Python також є ідеальним вибором. Завдяки бібліотекам, таким як openai, можна легко здійснювати запити до моделі штучного інтелекту, обробляти результати та передавати їх у базу даних чи інтерфейс адміністратора. Крім того, можливість використання бібліотек, таких як Pillow для роботи із зображеннями чи NumPy для обробки числових даних, робить Python незамінним інструментом для цього проекту.

Отже, вибір Python і Django для розробки інтернет-магазину є обґрунтованим з технічної, економічної та функціональної точок зору. Цей стек дозволяє створити сучасний, продуктивний і безпечний веб-додаток, який легко масштабується та інтегрується з інструментами штучного інтелекту. Завдяки

цим технологіям можна забезпечити високу якість продукту при оптимальних затратах часу і ресурсів.

3.2 Вибір бази даних

Для збереження даних у рамках проекту інтернет-магазину було обрано базу даних PostgreSQL. Це рішення базується на низці факторів, які роблять PostgreSQL ідеальним вибором для веб-додатків із високими вимогами до функціональності, надійності та продуктивності.

Однією з ключових переваг PostgreSQL є її гнучкість і здатність працювати з великими обсягами даних, що особливо актуально для інтернет-магазинів, де зберігається інформація про тисячі товарів, категорій, замовлень та клієнтів.

Серед найважливіших переваг варто виділити її відповідність стандарту ACID (атомарність, узгодженість, ізольованість, довговічність). Це означає, що навіть у разі збою чи переривання роботи системи дані залишаються цілісними й достовірними. У контексті інтернет-магазину це має особливе значення, адже втрати даних можуть призвести до серйозних проблем із замовленнями чи обліком клієнтів.

Система підтримує користувацькі функції, оператори та типи даних, що дозволяє адаптувати базу даних під специфічні потреби проекту. Наприклад, для інтернет-магазину можна налаштувати складні запити до бази даних, що забезпечують швидкий доступ до потрібної інформації, такої як сортування товарів, аналіз продажів чи побудова рекомендацій.

PostgreSQL також підтримує роботу зі складними типами даних, такими як JSON, що дозволяє зберігати дані в гнучкому форматі, необхідному для інтеграції з API. Наприклад, це корисно при збереженні відповідей від OpenAI чи інформації про товари, отриманої з інших джерел. Така функціональність значно спрощує обробку нестандартних даних і робить систему більш універсальною [16].

Серед вагомих аргументів на користь PostgreSQL є її висока продуктивність і здатність оптимізувати складні запити. Завдяки індексації, оптимізації запитів та використанню різноманітних методів кешування, що дозволяє значно скоротити час обробки великих обсягів даних. У випадку інтернет-магазину це дозволяє забезпечити швидкий пошук і фільтрацію товарів, що критично важливо для позитивного досвіду користувачів.

Безпека даних також є одним із ключових аспектів у виборі PostgreSQL. Вона має вбудовані механізми контролю доступу, шифрування й автентифікації, що дозволяє захистити конфіденційну інформацію клієнтів, таку як дані облікових записів чи історії замовлень. Це особливо важливо для відповідності сучасним стандартам захисту персональних даних, таким як GDPR.

Ще однією вагомою перевагою є її сумісність із Django через ORM (Object-Relational Mapping). Django чудово інтегрується з PostgreSQL, забезпечуючи зручний спосіб роботи з базою даних за допомогою моделей, що дозволяє значно зменшити кількість написаного коду. Крім того, Django підтримує використання специфічних функцій PostgreSQL, таких як повнотекстовий пошук або робота зі сховищем JSON, що додає додаткової гнучкості до проекту.

3.3 Інструменти інтеграції ШІ

Для реалізації функціоналу автоматичного створення описів товарів і генерації зображень у нашому інтернет-магазині було обрано OpenAI та DALL·E Mini. Вибір цих інструментів базується на їхніх унікальних можливостях, які ідеально відповідають потребам проекту.

OpenAI є провідною платформою в галузі штучного інтелекту, яка забезпечує доступ до потужних моделей, таких як GPT. Ці моделі здатні генерувати текст високої якості, який відповідає заданому контексту. У нашому проекті OpenAI використовується для автоматичного створення описів товарів на основі введеної назви та категорії. Це дозволяє значно спростити та прискорити процес додавання нових позицій до каталогу.

Однією з причин вибору є її здатність знаходити, аналізувати та інтегрувати інформацію з широкого спектра джерел. Це особливо важливо для генерації описів, які мають бути не лише граматично правильними, а й релевантними та інформативними.

Крім того, OpenAI легко інтегрується через API, що дозволяє реалізувати всі необхідні функції в рамках нашого інтернет-магазину на Django. Платформа також підтримує детальне налаштування запитів, що дає змогу адаптувати відповіді моделі під специфічні потреби проекту, наприклад, для створення текстів певного стилю або формату [17].

DALL·E Mini, у свою чергу, було обрано для генерації зображень товарів. Ця система є полегшеною версією DALL·E, яка здатна створювати унікальні зображення на основі текстових описів. У нашому випадку DALL·E Mini використовується для отримання фотографій товарів, якщо оригінальні зображення недоступні. Це допомагає вирішити проблему відсутності якісного візуального контенту, що є критично важливим для успіху інтернет-магазину.

Однією з головних переваг є її здатність генерувати зображення швидко й без потреби в додаткових налаштуваннях. Це не лише економить час, але й знижує витрати на створення візуального контенту.

Крім того, використання DALL·E Mini як генератора зображень у рамках нашого проекту є безкоштовним, що робить цей інструмент ще більш привабливим для невеликого бізнесу чи студентського проекту. Інтеграція системи в наш проект здійснюється за допомогою REST API, що забезпечує легкість у використанні та масштабованість.

Важливою причиною вибору OpenAI та DALL·E Mini є також їхня здатність працювати разом, утворюючи єдину екосистему. Це дозволяє автоматизувати весь процес додавання нового товару – від створення опису до отримання його візуалізації.

Загалом, ці інструменти штучного інтелекту були обрані для нашого проекту через їхню інноваційність, зручність інтеграції, гнучкість і високу якість результатів. Вони забезпечують можливість швидко та ефективно генерувати

текстовий і візуальний контент, що підвищує продуктивність роботи з інтернет-магазином і покращує користувацький досвід. Завдяки цим інструментам наш проект отримує значні переваги у конкурентному середовищі, водночас залишаючись доступним для реалізації навіть за обмеженого бюджету.

3.4 Стек веб-дизайну

Для реалізації візуальної частини проекту було обрано сучасний стек технологій, що включає HTML, CSS та JavaScript. HTML використовується для створення структури веб-сторінок інтернет-магазину, забезпечуючи основу для зручного й логічного розташування елементів. CSS дозволяє додавати стиль і дизайн, завдяки чому інтерфейс виглядає привабливо й відповідає сучасним стандартам веб-дизайну. JavaScript застосовується для інтерактивності, що забезпечує користувачам зручний і динамічний досвід [18].

Додатково використовується бібліотека jQuery для спрощення роботи з DOM-елементами та виконання AJAX-запитів. Це дозволяє ефективно передавати й отримувати дані від серверної частини проекту, реалізованої на Django. Для роботи з Fetch API використовується чистий JavaScript, що забезпечує швидкість і простоту інтеграції запитів до зовнішніх сервісів, таких як OpenAI або DALL·E Mini.

Дизайн орієнтований на мінімалізм і зручність, щоб спростити взаємодію користувачів з інтернет-магазином. Усі ключові функції будуть доступні у кілька кліків, а елементи інтерфейсу – адаптовані для використання на мобільних пристроях.

Уся робота з кодом проекту буде виконуватися в інтегрованому середовищі розробки PyCharm, яке забезпечує зручність написання коду, налагодження, а також інтеграцію з системами контролю версій. PyCharm дозволяє організувати проект таким чином, щоб усі частини проекту – від серверної до візуальної – працювали як єдине ціле.

РОЗДІЛ 4
РОЗРОБКА ІНТЕРНЕТ МАГАЗИНУ

4.1 Розробка бази даних

Після визначення структури та інструментів проекту, створюємо таблиці в базі даних, та переглядаємо їх через веб-інтерфейс для адміністрування СУБД PostgreSQL – phpPgAdmin.

Першою таблицею буде «shop_customer». Це кастомна модель авторизації, перероблена під потреби проекту, яка містить поля:

- id – унікальний ідентифікатор користувача, тип bigint, первинний ключ з авто інкрементом, NOT NULL;
- username – нікнейм користувача, тип varchar(150);
- first_name – ім’я користувача, тип varchar(150), NOT NULL;
- last_name – призвіще користувача, тип varchar(150). NOT NULL;
- email – електронна адреса користувача, тип varchar(250), NOT NULL;
- password – пароль від запису, тип varchar(120), NOT NULL;
- phone – телефон, тип varchar(20), NOT NULL;
- date_joned – дата створення запису, тип datetime, NOT NULL;
- is_staff – позначка працівника магазину, тип boolean, NOT NULL;
- is_superuser – позначка адміністратора, тип boolean, NOT NULL;
- is_active – позначка активації аккаунту, тип boolean, NOT NULL.

Так як була створена кастомна авторизація через електронну пошту, то позначка «is_active» на початку буде мати значення False (а не True як прийнято у базовій версії Django), і лише після активації листа, який прийде до користувачна на електронну адресу, вона стане True. Нижче зображена таблиця у СУБД (рис. 4.1).

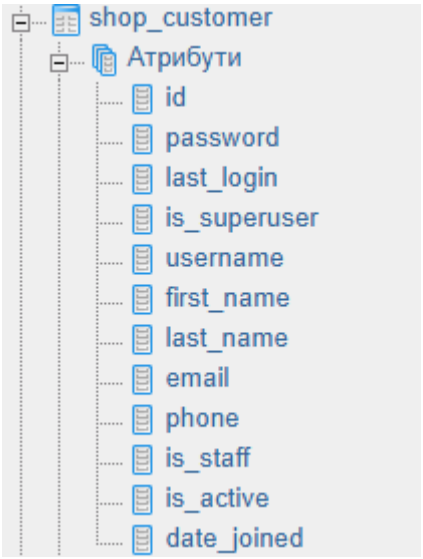

Рисунок 4.1 – Таблиця «shop_customer»

Далі будуть йти таблиці «Categories» та «SubCategories». Такий поділ зроблений для полегшеного прийняття користувачів магазину. Наприклад є категорія «Одяг», то її підкатегоріями будуть «Взуття», «Куртки» та «Футболки». Також ці таблиці будуть з’єднані зовнішнім ключем, для підтвердження їх залежності один від одного.

«Categories» (рис. 4.2):

- id – унікальний ідентифікатор категорії, тип bigint, первинний ключ з авто інкрементом, NOT NULL;
- name – назва категорії, тип varchar(50), NOT NULL;
- slug – SEO-дружній ідентифікатор категорії, тип varchar(50), UNIQUE, NOT NULL.

Як бачимо був доданий параметр slug, який буде автоматично утворюватися від назви категорії, та допоможе будувати url-маршрути.

«SubCategories» (рис. 4.3):

- id – унікальний ідентифікатор підкатегорії, тип bigint, первинний ключ з авто інкрементом, NOT NULL;
- name – назва підкатегорії, тип varchar(50), NOT NULL;
- slug – SEO-дружній ідентифікатор підкатегорії, тип varchar(50), UNIQUE, NOT NULL;

- super_category_id – зовнішній ключ на таблицю Categories, що вказує на основну категорію, тип bigint, FOREIGN KEY, NOT NULL.

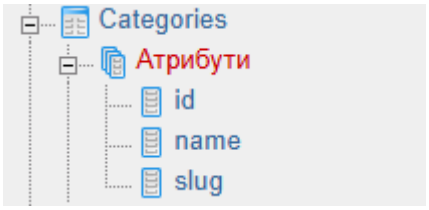

Рисунок 4.2 – Таблиця «Categories»

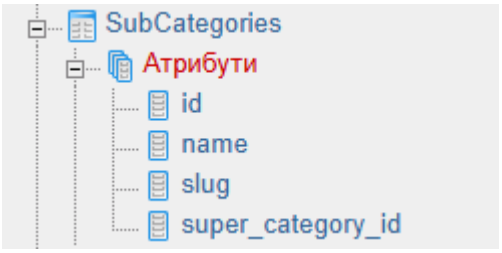

Рисунок 4.3 – Таблиця «SubCategories»

Після цього створюємо таблицю «Products» (рис. 4.4):

- id – унікальний ідентифікатор продукту, тип bigint, первинний ключ з авто інкрементом, NOT NULL;
- name – назва товару, тип varchar(50), NOT NULL;
- slug – SEO-дружній ідентифікатор товару, тип varchar(50), UNIQUE, NOT NULL;
- description – опис товару, тип text, NOT NULL;
- price – ціна продукту, тип int, NOT NULL;
- image – фото продукту, тип varchar(100);
- subcategory_id – зовнішній ключ на таблицю SubCategories, який вказує на підкатегорію товару, тип bigint, FOREIGN KEY, NOT NULL.

У цьому випадку для поля image використовується тип varchar, тому що у базі даних зберігається текстовий рядок, який є відносним або абсолютним шляхом до файлу зображення (наприклад, «media/images/product1.jpg»). А далі система Django через цей шлях завантажує фото на сервер.

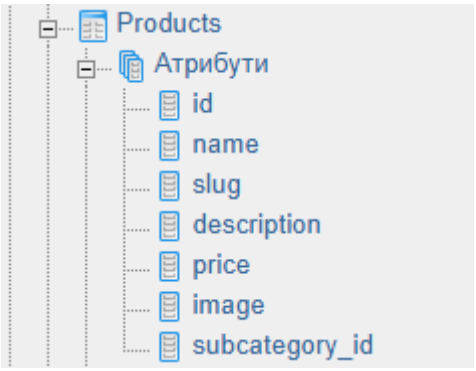

Рисунок 4.4 – Таблиця «Products»

Робимо таблиці «Orders» (рис. 4.5) та «OrderItems» (рис. 4.6), які також будуть з’єднані між собою. Таблиця замовлень:

- id – унікальний ідентифікатор замовлення, тип bigint, первинний ключ з авто інкрементом, NOT NULL;
- status – статус замовлення(виконано або не виконано), тип boolean, NOT NULL;
- time_create – дата створення замовлення, тип datetime, NOT NULL;
- transaction_id – ідентифікатор транзакції, тип varchar(100), NOT NULL;
- customer_id – зовнішній ключ на таблицю «shop_customer», що вказує на покупця, тип bigint, FOREIGN KEY, NOT NULL.

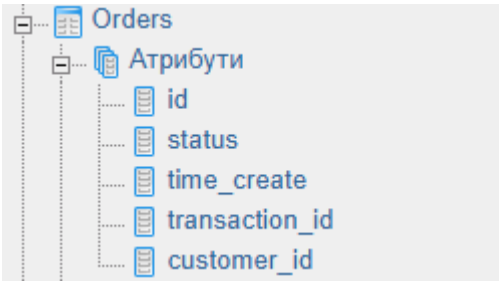

Рисунок 4.5 – Таблиця «Orders»

Таблиця «OrderItems» буде зберігати елементи(позиції) кожного замовлення.

- id – унікальний ідентифікатор позиції, тип bigint, первинний ключ з авто інкрементом, NOT NULL;
- quantity – кількість замовлених позицій, тип int, DEFAULT=1;
- order_id – зовнішній ключ на таблицю Orders, який вказує на замовлення, тип bigint, FOREIGN KEY, NOT NULL;

- product_id – зовнішній ключ до таблиці Products, для інформації про товар, тип bigint, FOREIGN KEY, NOT NULL.

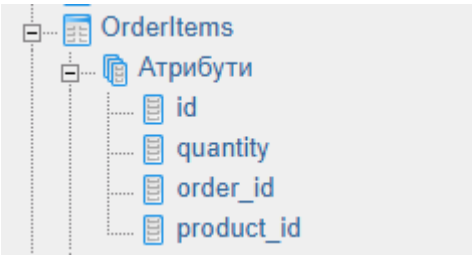

Рисунок 4.6 – Таблиця «OrderItems»

Останньою нашою таблицею буде «Shippings», яка містите інформацію про доставки замовлень.

- id – унікальний ідентифікатор доставки, тип bigint, первинний ключ з авто інкрементом, NOT NULL;
- address – адреса доставки, тип varchar(150), NOT NULL;
- city – місто доставки, тип varchar(100), NOT NULL;
- date – дата оформлення доставки, тип datetime, NOT NULL;
- customer_id – зовнішній ключ на таблицю «shop_customer», що вказує на покупця, тип bigint, FOREIGN KEY, NOT NULL;
- order_id – зовнішній ключ на таблицю Orders, який вказує на замовлення, яке буде доставлятися, тип bigint, FOREIGN KEY, NOT NULL.

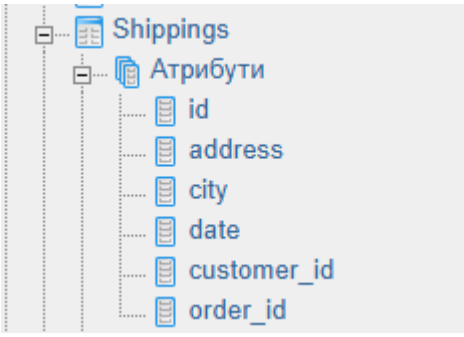

Рисунок 4.7 – Таблиця «Shippings»

Також окрім всіх наших створених таблиць, фреймворк Django надає нам ще декілька базових таблиць, які допоможуть нам відслідковувати логи адмінів «django_admin_log», зберігати міграції «django_migrations», переглядати сесії «django_session», створювати спеціальні групи користувачів «shop_customer_groups» та «auth_group». Та надавати або забирати спеціальні

можливості на сайті «auth_group_permissions», «auth_permission» та «shop_customer_user_permissions».

Після створення всіх моделей та виконання міграцій, отримуємо готову базу даних, яку переглядаємо через phpPgAdmin (рис. 4.8).

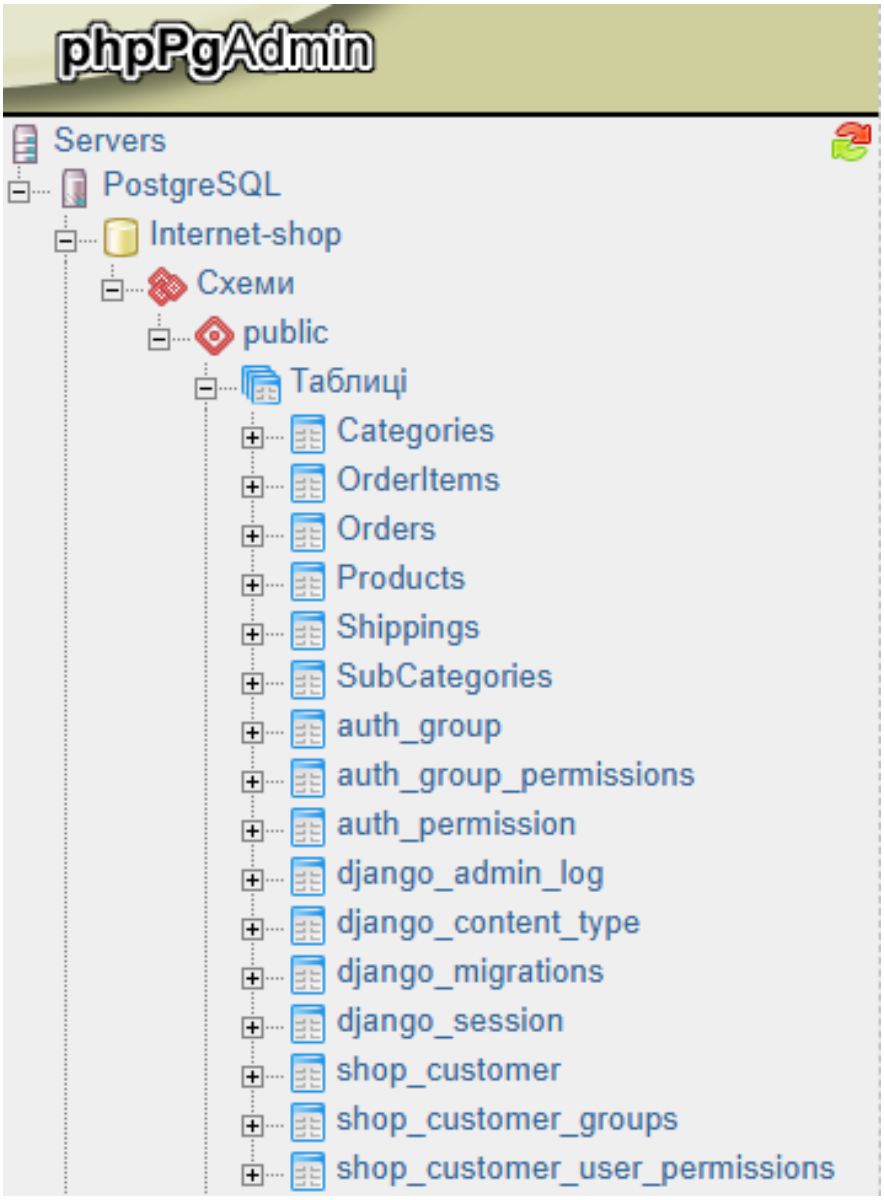

Рисунок 4.8 – Готова база даних у СУБД

4.2 Створення веб-інтерфейсу

Для початку розробимо головну сторінку інтернет магазину (рис. 4.9), яка спрямована на забезпечення комфортної взаємодії користувачів із сайтом. У верхній частині сторінки розташований хедер, де зліва знаходиться логотип магазину, який виконує функцію швидкого повернення на головну сторінку. Центральну частину займає пошуковий рядок, що дозволяє користувачам легко знаходити потрібні товари. Праворуч розташовані іконки для доступу до особистого кабінету та кошика. Ліва панель виконує роль сайдбару та містить список категорій товарів. Нижче розташовані інтерактивні іконки соціальних мереж, що дозволяють користувачам переходити до відповідних ресурсів [19].

Основний контент сторінки містить кілька ключових елементів. У верхній частині відображається блок із рекомендованими товарами. Нижче розташовані категорії товарів, які представлені коротким списком товарів та посиланням, яке веде до повного списку відповідної категорії.

Візуальне оформлення сторінки виконане у світло-блакитних тонах, що сприяє приємному візуальному сприйняттю та не відволікає від основного контенту. Дизайн сайту продуманий таким чином, щоб забезпечити інтуїтивну навігацію та підвищити зручність використання веб-магазину.

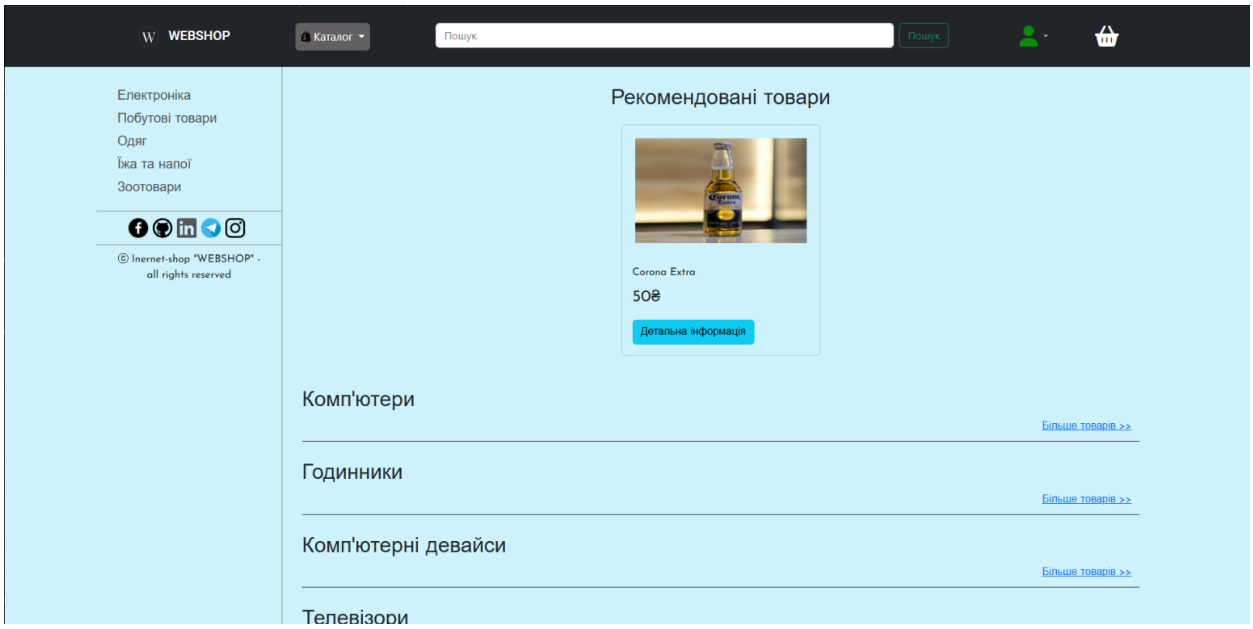

Рисунок 4.9 – Головна сторінка

Далі розробимо сторінку для кожної з категорій(рис. 4.10). Основна частина сторінки присвячена конкретній категорії товарів, де зображені її підкатегорії. Нижче представлено товари цих підкатегорій у вигляді карток, які розташовані в один рядок по кілька елементів.

Кожна картка товару включає зображення продукту, його назву, ціну та дві кнопки: "Детальна інформація" для переходу на сторінку з описом товару і кнопку з іконкою кошика, що дозволяє швидко додати товар до замовлення.

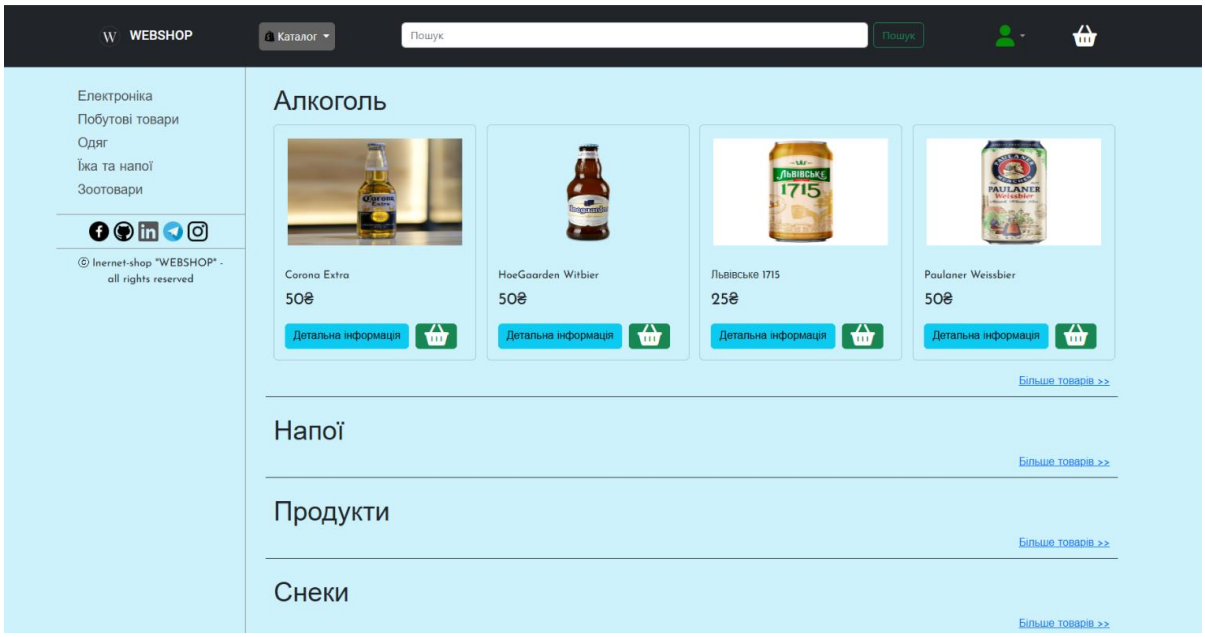

Рисунок 4.10 – Сторінка категорії

Після цього зробимо сторінки для кожного окремого товару(рис. 4.11), який вказаний у підкатегоріях. На цих сторінках буде вказана більш чітка фотографія товару. Також буде написаний детальний опис товару та вказана ціна, разом із можливістю додати товар до кошику.

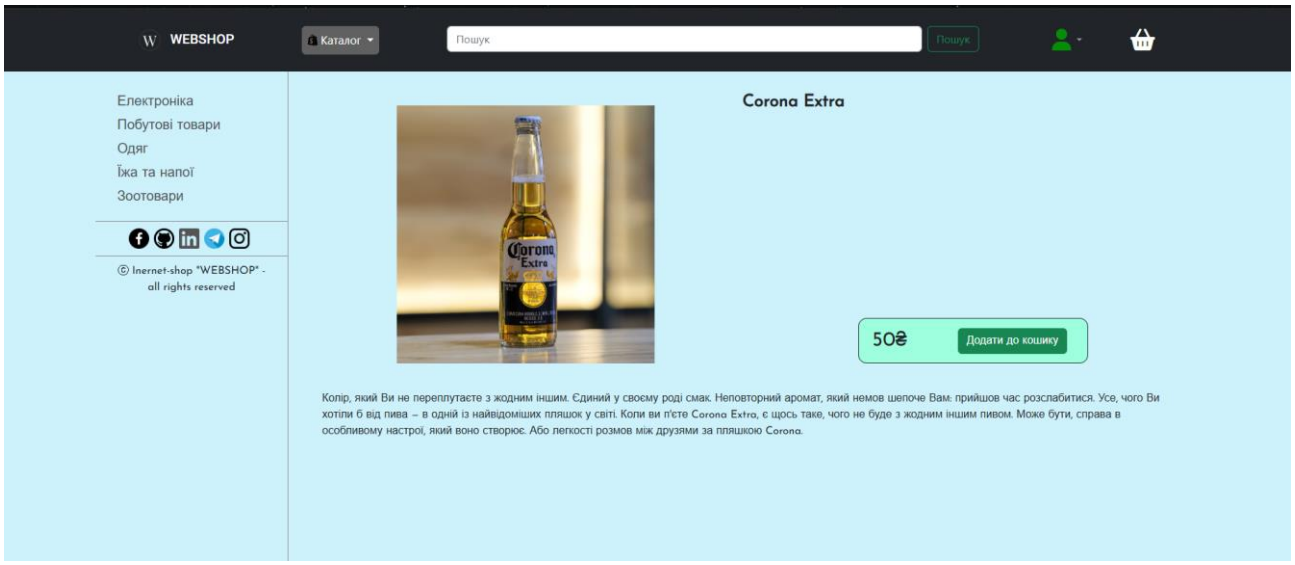

Рисунок 4.11 – Сторінка окремого товару

Далі розробимо кошик, куди будуть додаватися товари(рис. 4.12).

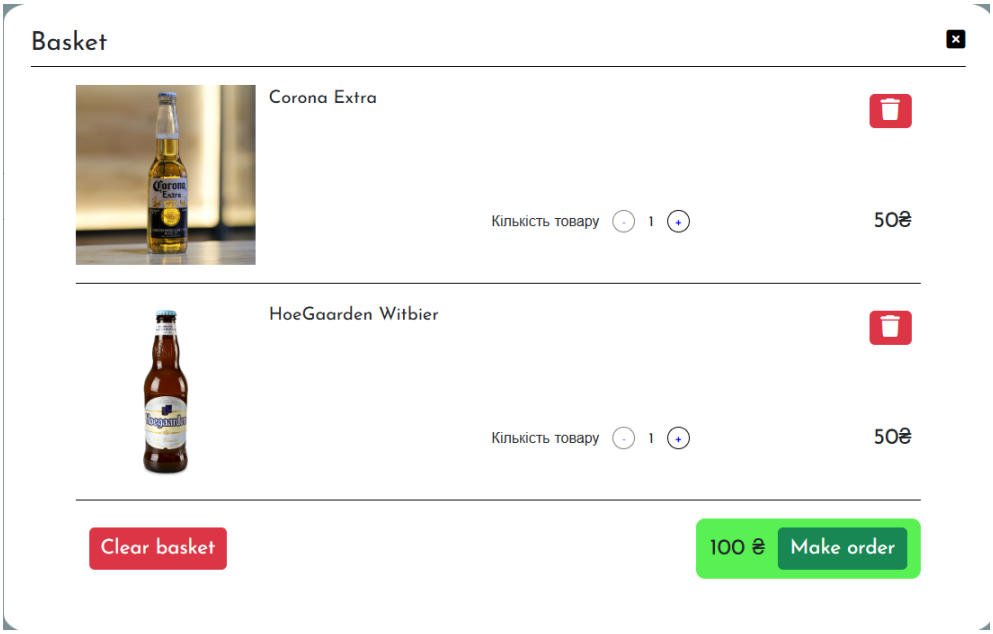

Рисунок 4.12 – Кошик для товарів

Наступним етапом буде розробка авторизації користувачів інтернет-магазину. Під даний магазин була створена своя система реєстрації (рис. 4.13), де використовується кастомна модель користувача, яка відрізняється від стандартної в Django. Замість звичайного імені користувача (username) основним ідентифікатором тут є email. Це спрощує процес авторизації, адже користувачі частіше пам’ятають свій email, ніж унікальне ім’я.

Модель користувача має поля для електронної пошти, імені, прізвища, телефону та паролів.

Коли користувач заповнює форму, його дані перевіряються, і якщо такий користувач ще не існує, для нього створюється обліковий запис.

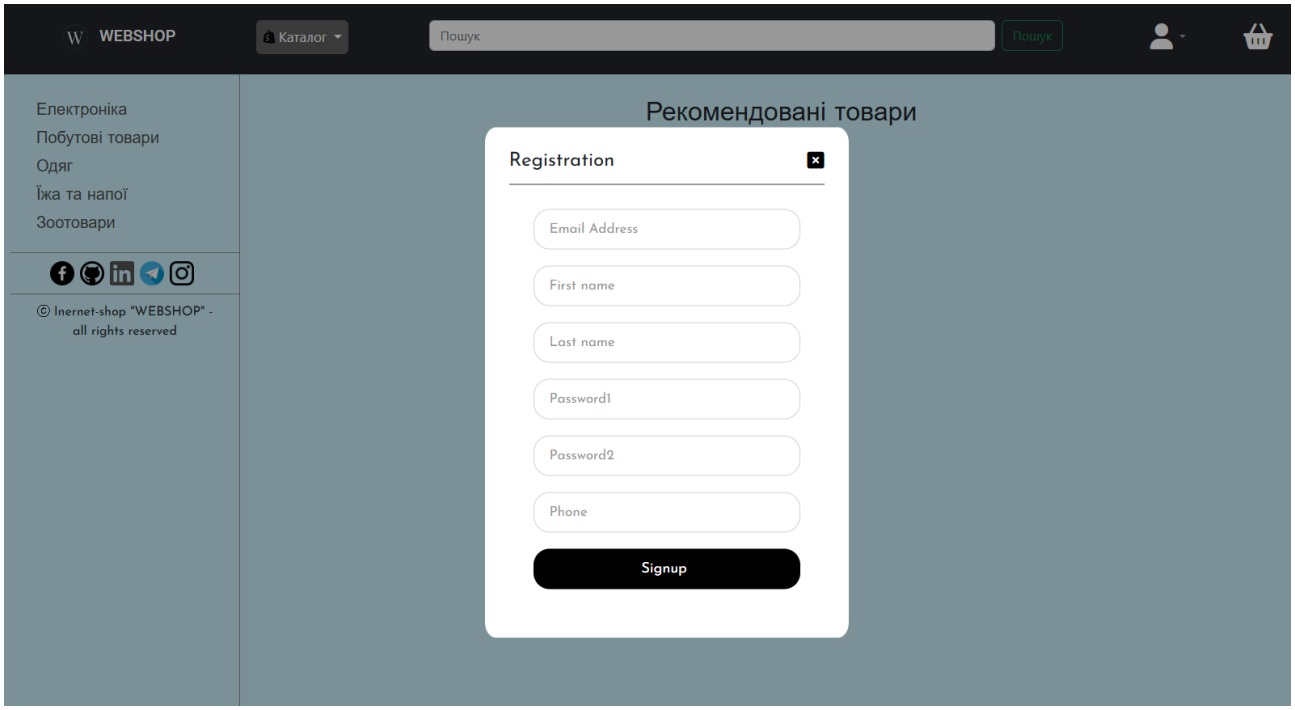

Рисунок 4.13 – Вікно реєстрації

Потім на електронну пошту надсилається лист із посиланням для підтвердження (рис. 4.14). У цьому посиланні міститься унікальний код. Коли користувач переходить за посиланням, його обліковий запис активується, і він отримує доступ до всіх функцій системи. Якщо користувач не підтвердить свою електронну пошту протягом визначеного часу, посилання стане недійсним.

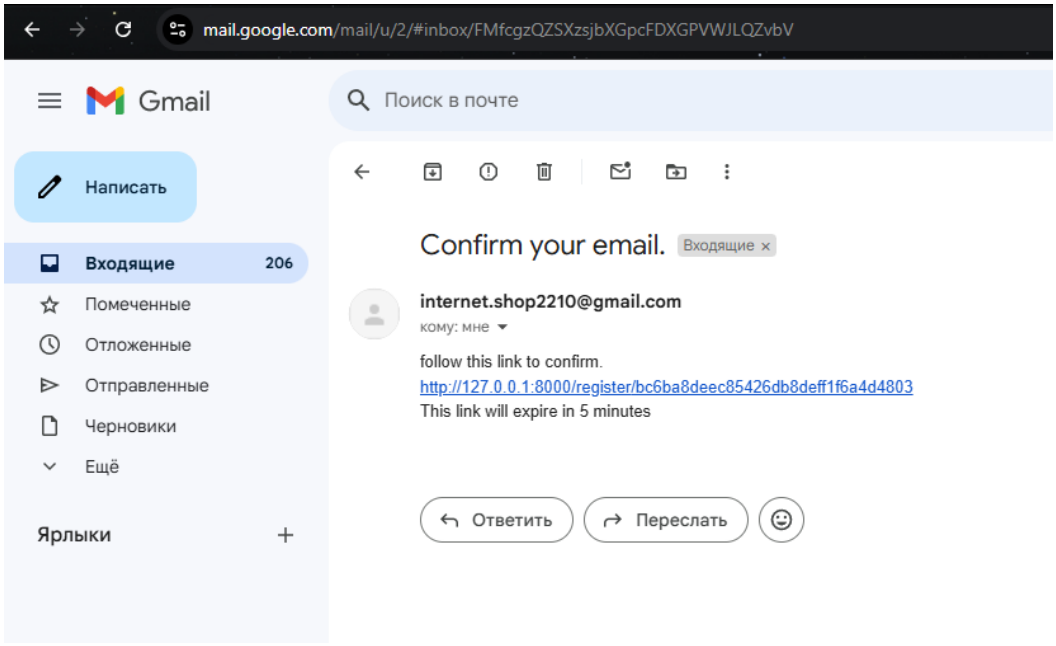

Рисунок 4.14 – Підтвердження пошти

На рисунку 4.15 зображено модальне вікно для авторизації користувача в інтернет-магазині.

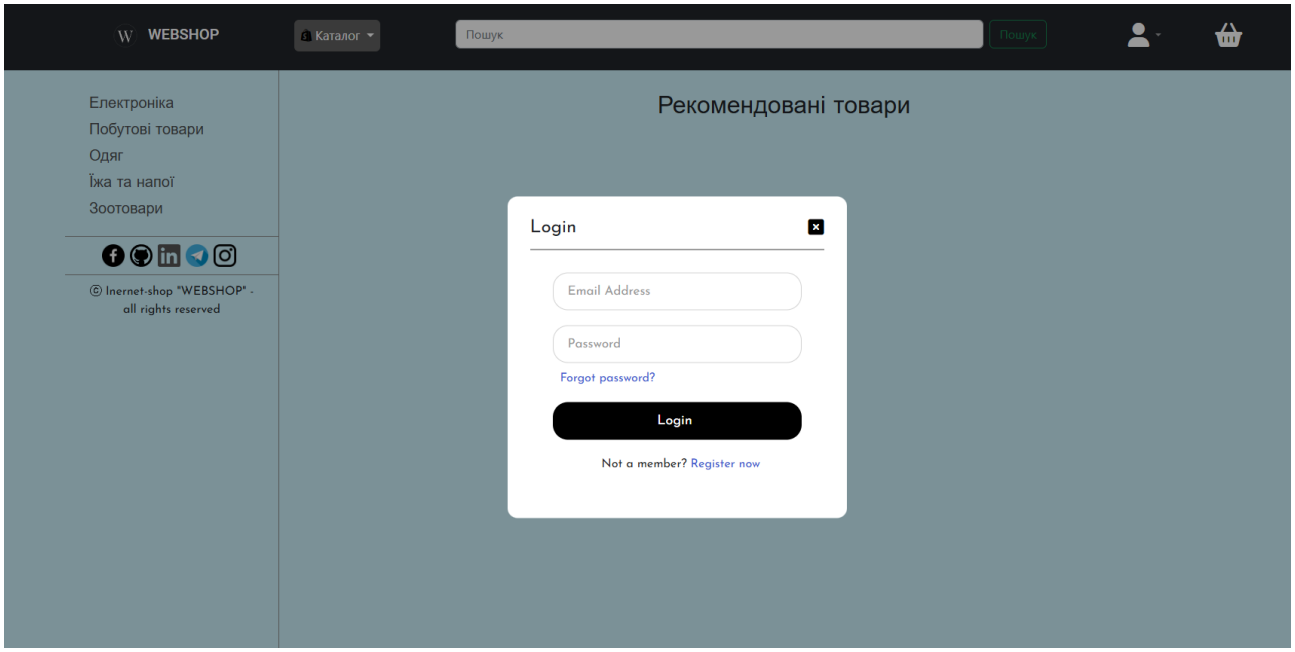

Рисунок 4.15 – Вікно авторизації

Наступним кроком для нашого проекту, це буде налаштування сторінки адміністрування, так як ми користуємося рекомендованою адмінкою Django. Використання Django Admin є доречним і ефективним вибором для керування інтернет-магазином завдяки його вбудованій функціональності, що забезпечує швидке та зручне управління даними без необхідності створювати адміністративну панель з нуля. Вона автоматично генерує інтерфейс для роботи з базою даних на основі моделей, що значно заощаджує час розробки. Django пропонує можливості налаштування, які дозволяють адаптувати панель під конкретні потреби проекту: додавання кастомних полів, інтеграція додаткових функцій через JavaScript та налаштування візуального оформлення. Крім того, панель адміністратора легко інтегрується з іншими компонентами Django, такими як ORM, що спрощує роботу з даними, і підтримує масштабованість, дозволяючи обробляти великі обсяги інформації.

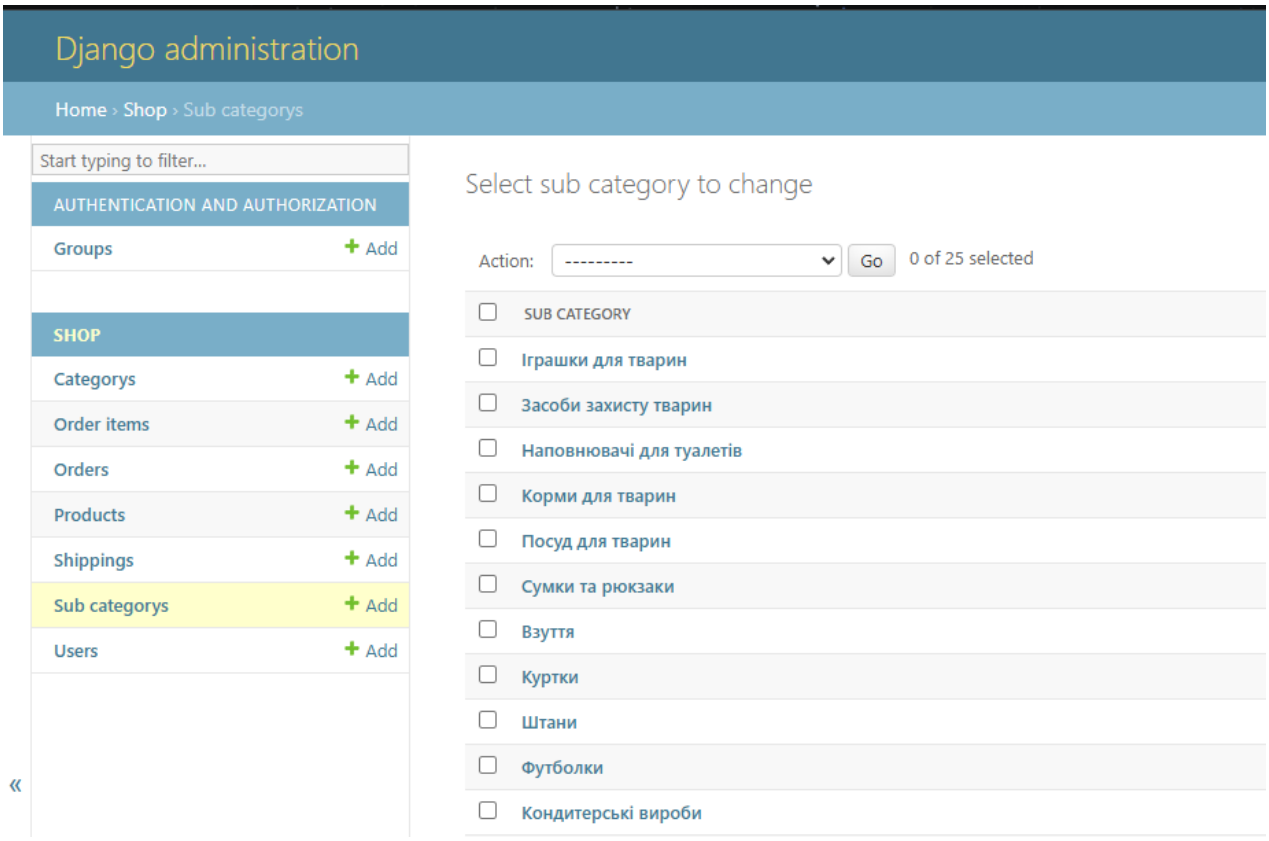

Рисунок 4.16 – Вигляд адмін сторінки

РОЗДІЛ 5
ТЕСТУВАННЯ

5.1 Визначення способу тестування

Переглянувши всі види тестування, отримуємо висновок, що автоматизоване тестування є найоптимальнішим вибором для нашого проекту. Це рішення базується на потребі забезпечення високої стабільності системи, ефективності процесу розробки та можливості швидкого виявлення й усунення помилок. У нашому випадку функціонал магазину включає багато інтеграцій, таких як генерація описів, зображень, розрахунок цін, а також рекомендаційні алгоритми. Ці функції залежать від зовнішніх сервісів, складних алгоритмів і бази даних, тому їхня коректна робота вкрай важлива.

Крім того, автоматизоване тестування підвищує ефективність розробки. У проектах із великою кількістю функціональності, як наш, регулярне виконання ручних тестів займає багато часу. Написання автоматизованих тестів на початкових етапах дозволяє зменшити витрати на підтримку та забезпечення якості. Це особливо актуально, коли наш проект продовжує розвиватися і додаються нові функції. Тести дають змогу виявити можливі проблеми ще до їхнього впливу на кінцевого користувача [20].

У нашому проекті для автоматизованого тестування обрано бібліотеку unittest, яка є стандартною для Python, а також її інтеграцію в Django через django.test. Цей вибір обумовлений кількома важливими факторами. По-перше, unittest є частиною стандартної бібліотеки Python, тому не вимагає встановлення додаткових залежностей. Це забезпечує високу сумісність із Django, адже цей фреймворк уже містить вбудовані механізми для інтеграції з unittest. По-друге, django.test розширює можливості стандартного unittest, додаючи специфічні інструменти для тестування моделей, форм, представлень і middleware [21]. Ці інструменти дозволяють легко створювати тестові бази даних, відправляти

HTTP-запити до представлень, а також перевіряти, чи працюють маршрути і middleware коректно.

Основним елементом автоматизованого тестування в нашому проекті є написання тестових кейсів у вигляді класів, які успадковуються від django.test.TestCase. Цей клас забезпечує ізольоване середовище для кожного тесту. Під час виконання кожного тесту автоматично створюється нова база даних, яка заповнюється тестовими даними, а після завершення тесту база очищується. Це дозволяє уникнути конфліктів між тестами і гарантує, що кожен тест перевіряє тільки свою частину функціоналу. Крім того, django.test.Client, який є частиною django.test, дозволяє симулювати HTTP-запити до серверу, перевіряючи, чи коректно працюють представлення та маршрути. Наприклад, у нашому проекті можна перевіряти, чи правильно обробляються запити для генерації описів товарів або аналізу цін.

Таким чином, автоматизоване тестування через unittest і django.test забезпечує якість нашого проекту на всіх етапах розробки, зменшуючи ризики помилок і підвищуючи загальну ефективність команди розробників. Це є важливим кроком для створення стабільного та функціонального інтернет-магазину, орієнтованого на використання сучасних технологій і задоволення потреб користувачів.

5.2 Тестування проекту

Після визначення виду тестування, було прийнято рішення про проведення тестування основних елементів двох додатків: *basket* та *shop*.

У межах додатку *basket* будуть протестовані:

- уявлення (views), що забезпечують взаємодію користувача із кошиком;
- маршрути (urls), які визначають шляхи до функціоналу кошика.

У додатку *shop* тестуванню підлягають:

- моделі (models), які визначають структуру основних даних магазину;

- уявлення (views), що реалізують бізнес-логіку додатку;
- маршрути (urls), які забезпечують доступ до функціоналу магазину;
- форми (forms), що обробляють введення користувачів для збереження даних.

Ціль тестування полягає у перевірці:

- коректності роботи всіх компонентів згідно з їхнім функціональним призначенням;
- відповідності роботи компонентів очікуваній поведінці;
- виявлення можливих помилок або недоліків у реалізації.

Для зрозумілості описаного тестування, на рисунку 5.1 наведено приклад написаного тесту.

```
new *
class ModelsTestCase(TestCase):

    new *
    def setUp(self):
        # Створення тестових даних
        self.customer = Customer.objects.create(
            email="testuser@example.com",
            first_name="Test",
            last_name="User",
            is_active=False,
        )

    new *
    def test_customer_creation(self):
        # Перевірка створення клієнта
        self.assertEqual(Customer.objects.count(), second: 1)
        self.assertEqual(self.customer.email, second: "testuser@example.com")
        self.assertFalse(self.customer.is_active)
```

Рисунок 5.1 – Приклад готового тесту

Далі приготуємо тести до моделей додатку shop:

5.2.1 Тестування додатку shop

Таблиця 5.1 – Тестування моделей

№	ТестКейс	Результат
1	Створити користувача Customer	Користувач успішно створений зі всіма заданими полями

Продовження табл. 5.1

2	Створити категорію Category	Категорія успішно створена, атрибут slug збережений
3	Створити підкатегорію SubCategory	Підкатегорія створена, належить до категорії
4	Створити товар Product	Товар створений з усіма полями, належить підкатегорії
5	Створити замовлення Order	Замовлення успішно створене, пов’язане з клієнтом
6	Створити елемент замовлення OrderItem	Елемент успішно створений і прив’язаний до товару та замовлення
7	Створити доставку Shipping	Доставка успішно створена з усіма полями та пов’язана з замовленням
8	Викликати метод get_absolute_url для товару	Метод повертає правильний URL товару
9	Викликати метод get_absolute_url для категорії	Метод повертає правильний URL категорії

Запускаємо тестування та отримуємо результат:

```
(venv) PS C:\Users\NarcoSSice\PycharmProjects\Internet-shop\main> python manage.py test
Found 9 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.........
----------------------------------------------------------------------
Ran 9 tests in 0.061s

OK
Destroying test database for alias 'default'...
```

Рисунок 5.2 – Результат тестування моделей

Окрім самих моделей, у нас є менеджер, який відповідає за валідацію даних, тестування якого теж треба провести.

Таблиця 5.2 – Тестування менеджера моделей

№	ТестКейс	Результат
1	Викликати метод create_user з переданими email і паролем	Користувача створено: email збережено, пароль хешовано, is_staff=False, is_superuser=False
2	Викликати метод create_user без email	Викидається помилка ValueError з повідомленням "The given email must be set"
3	Викликати метод create_superuser з переданими email і паролем	Суперкористувача створено: email збережено, пароль хешовано, is_staff=True, is_superuser=True
4	Викликати метод create_superuser із is_superuser=False	Викидається помилка ValueError з повідомленням "Superuser must have is_superuser=True."

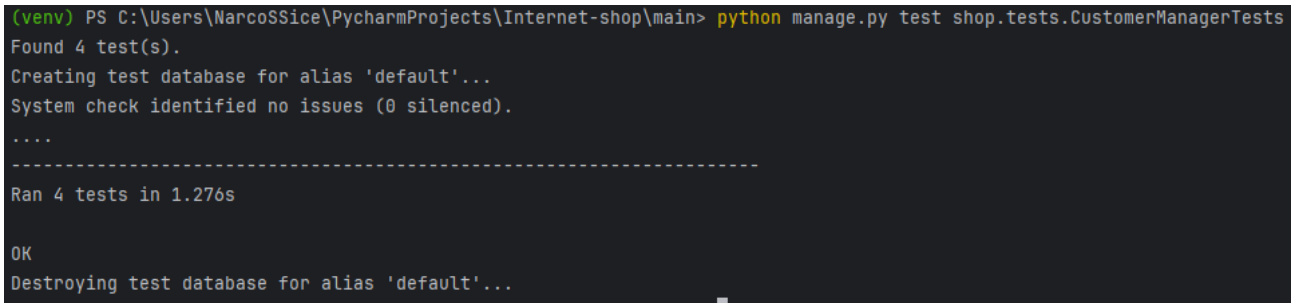

Рисунок 5.3 – Результат тестування менеджера

Далі буде проведено тестування форм авторизації, в нашому проекті їх дві: RegisterCustomerForm та LoginUserForm. Спочатку протестуємо форму для реєстрації(табл. 5.3) та отримаємо результати(рис 5.4).

Таблиця 5.3 – Тестування форми реєстрації

№	ТестКейс	Результат
1	Перевірка валідності форми з правильними даними	Форма валідна, всі поля правильно заповнені.

Продовження табл. 5.3

2	Перевірка форми з некоректним форматом номера телефону	Помилка валідації: Uncorrected phone number\nPhone number must consist: +380 ** *** ** **
3	Перевірка форми з різними паролями (password1 != password2)	Помилка валідації: password_mismatch
4	Перевірка форми на відсутність email	Помилка валідації: This field is required.
5	Перевірка форми на відсутність обов’язкових полів	Форма не валідна, 6 помилок для обов’язкових полів

Рисунок 5.4 – Результати тестування форми реєстрації

Після цього переходимо до тестування форми авторизації (табл. 5.4).

Таблиця 5.4 – Тестування форм авторизації

№	ТестКейс	Результат
1	Перевірка успішного входу з правильними email і паролем	Форма валідна, користувач успішно аутентифікований.
2	Перевірка входу з неправильним email	Помилка: Please enter a correct email and password. Note that both fields may be case-sensitive.

Продовження табл. 5.4

№	ТестКейс	Результат
3	Перевірка входу з неправильним паролем	Помилка: Please enter a correct email and password. Note that both fields may be case-sensitive.
4	Перевірка форми на відсутність email або пароля	Форма не валідна, 2 помилки: для поля username та password
5	Перевірка роботи методу clean для аутентифікації користувача	Форма валідна, користувач успішно аутентифікований.

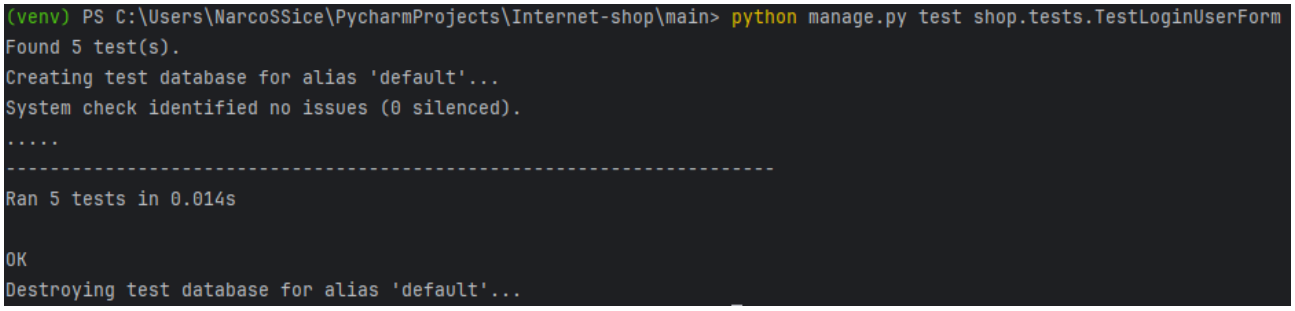

Рисунок 5.5 – Результат тестування форми авторизації

Далі черга уявлень(views), напишемо такі тест кейси:

Таблиця 5.5 – Тестування уявлень

№	ТестКейс	Результат
1	Відкрити головну сторінку (index)	Статус-код відповіді 200, перевірено наявність рекомендованих товарів і підкатегорій у контексті
2	Викликати RegisterView із правильними даними	Після успішної реєстрації виконується редирект на головну сторінку
3	Викликати register_confirm із валідним токеном	Після підтвердження реєстрації виконується редирект на головну сторінку
4	Викликати LoginUser із правильними даними	Користувач авторизується, виконується редирект на головну сторінку

Продовження табл. 5.5

№	ТестКейс	Результат
5	Викликати logout_customer	Користувач виходить із системи, виконується редирект на головну сторінку
6	Викликати show_subcategories з валідним category_slug	Статус-код відповіді 200, перевірено наявність підкатегорій у контексті
7	Викликати show_products із валідними category_slug, subcategory_slug, product_slug	Статус-код відповіді 200, перевірено наявність продукту у контексті
8	Викликати MakeOrder без авторизації	Виконується редирект на сторінку входу
9	Викликати MakeOrder із авторизованим користувачем	Статус-код відповіді 200, перевірено наявність адреси, міста та додаткового контексту
10	Надіслати форму MakeOrder із валідними даними	Створено замовлення, його елементи та виконано редирект на головну сторінку

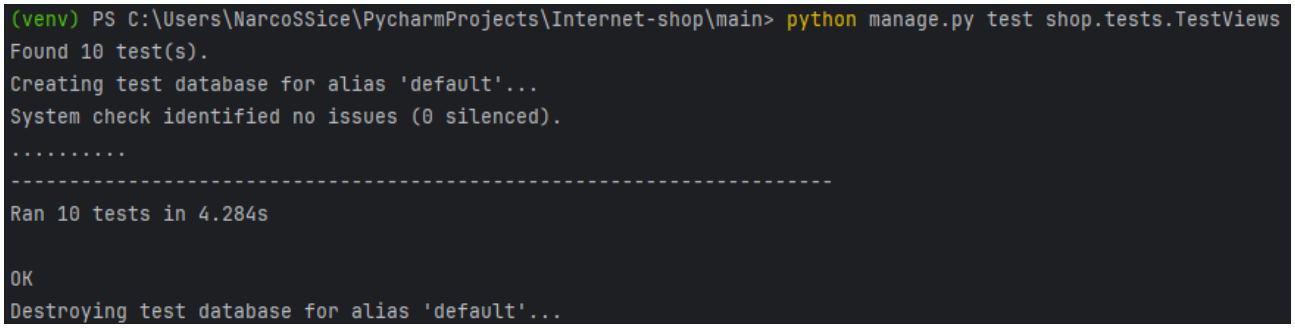

Рисунок 5.6 – Результати тестування views

Під час написання тестів виявлено кілька критичних моментів, пов’язаних із правильністю передачі контексту в шаблони, використанням жорстко закодованих даних у відображеннях та обробкою винятків у випадку відсутності необхідних об’єктів у базі даних. Особливу увагу слід приділити залежності між тестовими даними та логікою програми, оскільки неправильно створені або

неповні тестові дані можуть призводити до некоректної поведінки під час виконання тестів. Після вирішення всіх інцидентів, робимо комплексне тестування додатку *shop*.

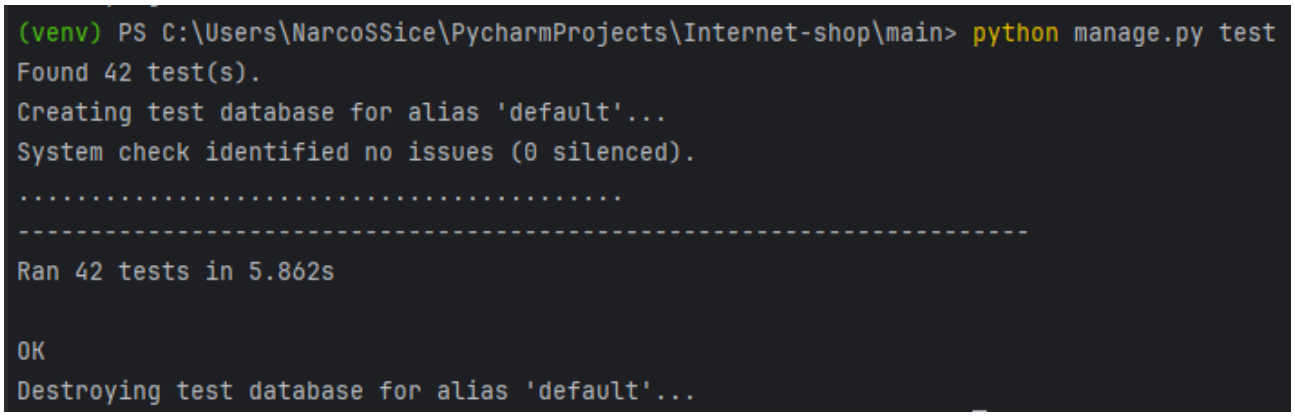

Рисунок 5.7 – Комплексне тестування *shop*

Тестування додатку *shop* показало важливість ретельної перевірки всіх компонентів, включаючи відображення, форми, URL-адреси та шаблони. Загалом тести дозволили ідентифікувати потенційні проблеми та впровадити зміни, які підвищують стабільність і надійність додатку.

5.2.2 Тестування додатку basket

Таблиця 5.6 – Тестування уявлень кошику

№	ТестКейс	Результат
1	Перевірити відображення кошика	Сторінка відображається успішно, повертає статус 200 та використовує шаблон basket.html.
2	Додати товар у кошик	Товар успішно додається в кошик. Сесія оновлюється, і користувача перенаправляє на вказану URL.
3	Видалити товар із кошика	Товар успішно видаляється з кошика. Сесія оновлюється, і товар більше не присутній у кошику.

Продовження табл. 5.6

4	Очистити кошик	Кошик успішно очищується. Дані кошика видаляються із сесії, користувача перенаправляє на вказану URL
5	Оновити кількість товару в кошику	Кількість товару успішно оновлюється. Кошик у сесії оновлений, повертається статус 200 із JSON-відповіддю.

```
(venv) PS C:\Users\NarcoSSice\PycharmProjects\Internet-shop\main> python manage.py test basket.tests.BasketViewsTestCase
Found 5 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.....
----------------------------------------------------------------------
Ran 5 tests in 0.071s

OK
Destroying test database for alias 'default'...
```

Рисунок 5.8 – Результат тестування уявлень кошику

5.2.3 Тестування інтеграції ШІ

Таблиця 5.7 – ТестКейси ШІ

№	ТестКейс	Результат
1	Згенерувати опис товару	Опис успішно згенеровано. Повертається текст, який відповідає запиту.
2	Згенерувати середню ціну товару	Успішно отримано середню ціну товару в гривнях. Повертається лише число.
3	Згенерувати URL для зображення товару	Успішно згенеровано URL для зображення товару. Повертається валідний URL.

Продовження табл. 5.7

№	ТестКейс	Результат
4	Обробка помилки генерації опису	Виникає помилка генерації опису. Повертається JSON із повідомленням про помилку та статусом 400.
5	Обробка помилки аналізу ціни	Виникає помилка аналізу ціни. Повертається JSON із повідомленням про помилку та статусом 400.
6	Обробка помилки генерації зображення	Виникає помилка генерації зображення. Повертається JSON із повідомленням про помилку та статусом 400.

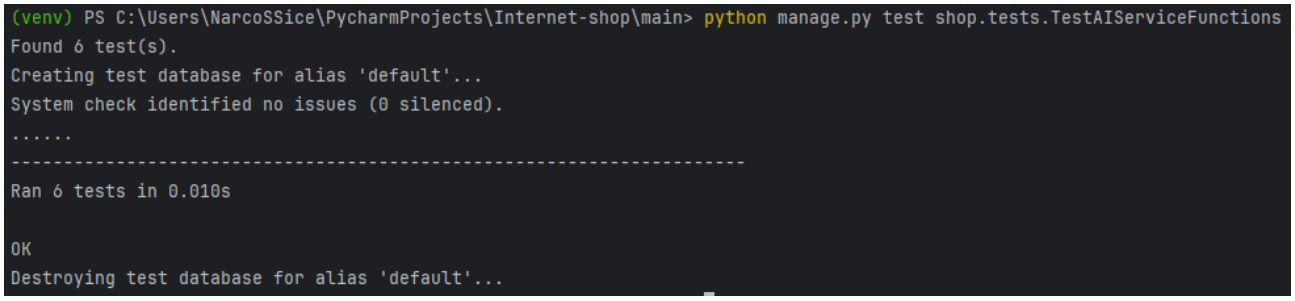

Рисунок 5.9 – Результат тестування ШІ

5.3 Підсумки тестування

Після виконання тестування всіх компонентів, запустимо останнє комплексне тестування всього проекту та отримаємо результат:

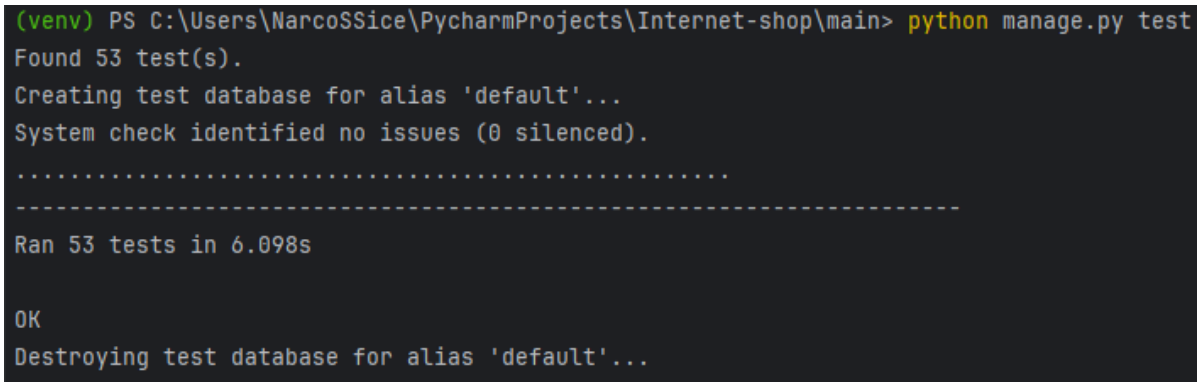

Рисунок 5.10 – Комплексне тестування

Під час тестування проекту було перевірено основні функціональні можливості інтернет-магазину, включаючи додавання товарів до кошика, перегляд товарів, обробку замовлень, а також реєстрацію та автентифікацію користувачів. Усі ці функції працюють стабільно і не викликають значних помилок, забезпечуючи безперебійну роботу користувацького інтерфейсу. Інтеграція елементів штучного інтелекту для автоматичного створення описів товарів, аналізу ринкових цін та пошуку зображень товарів показала хороші результати. Опис товарів, генерований за допомогою ШІ, є точним і релевантним, хоча для деяких специфічних товарів слід допрацювати модель для досягнення більшої точності. Аналіз середніх цін на ринку дозволив правильно визначити вартість товарів та запропонувати оптимальні ціни для магазину.

Загалом, проект відповідає основним вимогам та функціональним цілям, що були поставлені на початку тестування. Після цього проект майже готовий до хостингу, попередньо він завантажений за посиланням [22]. Інтеграція штучного інтелекту допомагає автоматизувати процеси та підвищити ефективність управління контентом інтернет-магазину. Подальші вдосконалення зосереджуються на оптимізації продуктивності та поліпшенні адаптивності інтерфейсу, щоб зробити сайт ще більш зручним для користувачів.

ВИСНОВКИ

У ході роботи було створено функціонал, що дозволяє автоматизувати важливі аспекти наповнення та оптимізації магазину. Зокрема, реалізовано можливість автоматичного створення унікальних описів товарів, аналізу ринкових цін та пропонування оптимальної ціни для кожного товару, а також автоматичного пошуку зображень товарів у мережі. Ці функції базуються на інтеграції сучасних моделей штучного інтелекту, таких як GPT та DALL-E, які забезпечують високий рівень точності та відповідності запитам.

Реалізовані моделі бази даних, такі як користувачі, категорії, підкатегорії, товари, замовлення, елементи замовлення та доставка, створюють надійний фундамент для зберігання й обробки даних. Додатково було розроблено функціонал для керування кошиком, який дозволяє додавати, видаляти, змінювати кількість товарів і очищувати кошик. Тестування цього функціоналу підтвердило його коректність і стабільність роботи.

Особливу увагу приділено розробці та тестуванню інтеграції штучного інтелекту, де кожна функція, пов’язана з генерацією описів, цін і зображень, була детально перевірена на коректність роботи. Усі можливі помилки генерації були опрацьовані з поверненням відповідних повідомлень користувачеві. Це гарантує стабільну роботу системи навіть у разі непередбачуваних обставин.

Таким чином, створена система забезпечує високу ефективність роботи інтернет-магазину, автоматизуючи рутинні процеси, економлячи час адміністраторів і підвищуючи якість контенту. Використання сучасних технологій штучного інтелекту дозволяє вивести взаємодію з клієнтами на новий рівень, забезпечуючи актуальність і конкурентоспроможність магазину на ринку. Наш проект демонструє перспективність інтеграції AI в електронну комерцію та закладає основу для подальшого розвитку системи з урахуванням потреб сучасних бізнесів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Котлер Ф., Армстронг Г. Основи маркетингу для онлайн-бізнесу. Харків: Видавництво «Фоліо», 2018. 640 с.

2. Лі Вайс. Створення успішного інтернет-магазину: практичний посібник. Київ: Видавництво XYZ, 2020. 256 с.

3. Нік Бостром. Штучний інтелект: Етапи розвитку та виклики майбутнього. Київ: Наш Формат, 2021. 392 с.

4. Рекомендації щодо оптимізації інтернет-магазинів [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://moz.com/beginners-guide-to-seo

5. Основи штучного інтелекту: навчальні матеріали та приклади [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.ibm.com/topics/artificial-intelligence

6. Content Management Systems Market by Component, Deployment Mode, Organization Size, Application, and Region - Global Forecast to 2026. [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.marketsandmarkets.com

7. Платформа управління Adobe Experience Manager [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://business.adobe.com/products/experience-manager/sites/content-creation.html

8. Інтернет-інфраструктура для бізнесу Shopify [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.shopify.com/about

9. Бізнес аналітика компанії Shopify [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.wsj.com/business/earnings/shopify-3q-profit-rises-revenue-tops-views-f6653411

10. Платформа для керування контентом Contentful [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://en.wikipedia.org/wiki/Contentful

11. Найкращі headless CMS для створення інтернет-магазину [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://dou.ua/forums/topic/43307/

12. Інформація про фреймворк Django [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.djangoproject.com/

13. Огляд можливостей платформи OpenAI та її основні функції для розробників і користувачів [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://platform.openai.com/docs/overview

14. Спрощена архітектура моделі OpenAI Five: складний багатовимірний простір спостережень [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.researchgate.net/figure/Simplified-OpenAI-Five-Model-Architecture-The-complex-multi-array-observation-space-is_fig1_337967587

15. Python для початківців: практичний посібник [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://realpython.com/

16. Опис СУБД PostgreSQL [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.postgresql.org/

17. Інтеграція штучного інтелекту через API: практичні приклади [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://towardsdatascience.com/how-to-integrate-ai-using-apis-a-practical-guide-1a2f48fc6bdb

18. Підходи до дизайну інтерфейсу онлайн-магазинів [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.smashingmagazine.com/ecommerce-ui-design/

19. Семантичний HTML: як створювати структуру веб-сторінок [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://developer.mozilla.org/en-US/docs/Glossary/Semantics

20. Основи тестування програмного забезпечення [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://www.guru99.com/software-testing-introduction.html

21. Python unittest: Офіційний посібник [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.python.org/3/library/unittest.html

22. Завантажений проект [Електронний ресурс]: [Веб-сайт]. – Електронні дані. – Режим доступу: https://github.com/NarcoSSice/internet-shop

ДОДАТОК А
ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

1. Моделі models.py

```
class CustomerManager(BaseUserManager):
    use_in_migrations = True

    def _create_user(self, email, password, **extra_fields):
        if not email:
            raise ValueError("The given email must be set")

        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.password = make_password(password)
        user.save(using=self._db)
        return user

    def create_user(self, email, password=None, **extra_fields):
        extra_fields.setdefault("is_staff", False)
        extra_fields.setdefault("is_superuser", False)
        return self._create_user(email, password, **extra_fields)

    def create_superuser(self, email, password=None, **extra_fields):
        extra_fields.setdefault("is_staff", True)
        extra_fields.setdefault("is_superuser", True)
        extra_fields.setdefault("is_active", True)

        if extra_fields.get("is_staff") is not True:
            raise ValueError("Superuser must have is_staff=True.")
        if extra_fields.get("is_superuser") is not True:
            raise ValueError("Superuser must have is_superuser=True.")

        return self._create_user(email, password, **extra_fields)


class Customer(AbstractUser):
    username = models.CharField(
        _("username"),
        max_length=150,
        null=True,
        blank=True
    )

    first_name = models.CharField(_("first name"), max_length=150, blank=True)
    last_name = models.CharField(_("last name"), max_length=150, blank=True)
```

```
    email = models.EmailField(_("email address"), unique=True)
    phone = models.CharField(_("phone"), max_length=20, blank=True)
    is_staff = models.BooleanField(
        _("staff status"),
        default=False,
        help_text=_("Designates whether the user can log into this admin site."),
    )
    is_active = models.BooleanField(
        _("active"),
        default=False,
        help_text=_(
            "Designates whether this user should be treated as active. "
            "Unselect this instead of deleting accounts."
        ),
    )
    date_joined = models.DateTimeField(_("date joined"), default=timezone.now)

    objects = CustomerManager()

    USERNAME_FIELD = "email"
    REQUIRED_FIELDS = []

    def __str__(self):
        return f'User {self.email}'


class Category(models.Model):
    name = models.CharField(max_length=50)
    slug = models.SlugField(max_length=50, unique=True, db_index=True, verbose_name='URL')

    class Meta:
        db_table = 'Categories'

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('show_subcategories', kwargs={'category_slug': self.slug})


class SubCategory(models.Model):
    name = models.CharField(max_length=50)
    slug = models.SlugField(max_length=50, unique=True, db_index=True, verbose_name='URL')
    super_category = models.ForeignKey(Category, on_delete=models.PROTECT)

    class Meta:
        db_table = 'SubCategories'
```

```
    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('show_subcategory_products', kwargs={'category_slug': self.super_category.slug,
                                                'subcategory_slug': self.slug})


class Product(models.Model):
    name = models.CharField(max_length=255)
    slug = models.SlugField(max_length=50, unique=True, db_index=True, verbose_name='URL')
    description = models.TextField()
    price = models.IntegerField()
    image = models.ImageField(upload_to='images/%Y/%m/%d/', null=True, blank=True)
    subcategory = models.ForeignKey(SubCategory, on_delete=models.PROTECT)

    class Meta:
        db_table = 'Products'

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('show_product_details', kwargs={
            'category_slug': self.subcategory.super_category.slug,
            'subcategory_slug': self.subcategory.slug,
            'product_slug': self.slug,
                        })


class Order(models.Model):
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    status = models.BooleanField(default=False)
    time_create = models.DateTimeField(auto_now_add=True)
    transaction_id = models.CharField(max_length=100)

    class Meta:
        db_table = 'Orders'

    def __str__(self):
        return f'Order number: {self.pk}, status: {self.status}'


class OrderItem(models.Model):
    product = models.ForeignKey(Product, on_delete=models.CASCADE)
    order = models.ForeignKey(Order, on_delete=models.CASCADE)
```

```
    quantity = models.IntegerField(default=1, null=True, blank=True)

    class Meta:
        db_table = 'OrderItems'


class Shipping(models.Model):
    customer = models.ForeignKey(Customer, on_delete=models.CASCADE)
    order = models.OneToOneField(Order, on_delete=models.CASCADE)
    address = models.CharField(max_length=150)
    city = models.CharField(max_length=100)
    date = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'Shippings'
```

2. views.py

```
def generate_product_description_view(request):
    description = generate_description(request)
    return JsonResponse({'description': description})


def generate_product_price_view(request):
    price = generate_price(request)
    return JsonResponse({'price': price})


def generate_product_image_view(request):
    image_url = generate_image_url(request)
    if type(image_url) is JsonResponse:
        return image_url
    return JsonResponse({'imageUrl': image_url})


def index(request):
    recommended_products = [Product.objects.all()[:1]]
    recommended_subcategories = SubCategory.objects.all()
    context = {
        'recommended_products': recommended_products,
        'recommended_subcategories': recommended_subcategories,
    }
    return render(request, 'shop/base.html', context=context)


class RegisterView(FormView):
    form_class = RegisterCustomerForm
    template_name = 'shop/auth/register.html'
```

```
    success_url = reverse_lazy('home')

    def form_valid(self, form):
        confirm_email_customer(self.request, form)
        return super().form_valid(form)


def register_confirm(request, token):
    result = confirm_customer(token)
    if result:
        return redirect(to=reverse_lazy('home'))
    return redirect(to=reverse_lazy('register'))


class LoginUser(LoginView):
    form_class = LoginUserForm
    template_name = 'shop/auth/login.html'

    def get_success_url(self):
        return reverse_lazy('home')


def logout_customer(request):
    logout(request)
    return redirect('home')


def show_subcategories(request, category_slug):
    subcategories = get_subcategories(category_slug)

    context = {
        'subcategories': subcategories,
    }

    return render(request, 'shop/subcategories.html', context=context)


def show_subcategory_products(request, category_slug, subcategory_slug):
    products, subcategory = get_product_by_subcategory(subcategory_slug)

    context = {
        'products': products,
        'subcategory': subcategory
    }

    return render(request, 'shop/show_subcategory.html', context=context)
```

```
def show_products(request, category_slug, subcategory_slug, product_slug):
    product = get_product(product_slug)

    context = {
        'product': product,
    }

    return render(request, 'shop/product_detail.html', context=context)


class MakeOrder(LoginRequiredMixin, CreateView):
    model = Shipping
    fields = ['address', 'city']
    template_name = 'shop/make_order.html'
    login_url = 'my_login'
    success_url = reverse_lazy('home')

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        add_context = create_order_context(self.request)
        context.update(add_context)
        return context

    def form_valid(self, form):
        instance = form.save(commit=False)

        order = self.create_order()
        self.create_items(order)

        instance.customer = self.request.user
        instance.order = order
        instance.save()
        return redirect('home')

    def create_order(self):
        user = self.request.user
        return create_basket_order(user)

    def create_items(self, order):
        basket = self.request.session['basket']
        create_order_items(basket, order)
```

3. forms.py

```
class RegisterCustomerForm(forms.Form):
    """
    Registration Form for creating users
```

```
    """

    email = forms.EmailField(required=True)
    first_name = forms.CharField(required=True)
    last_name = forms.CharField(required=True)
    password1 = forms.CharField(required=True)
    password2 = forms.CharField(required=True)
    phone = forms.CharField(required=True)

    def clean_phone(self):
        """ Function for validation phone """
        phone = self.cleaned_data.get('phone')
        if (phone[0] != '+') and (len(phone) != 13):
            raise ValidationError(
                _('Uncorrected phone number\nPhone number must consist:+380 ** *** ** **')
            )
        return phone

    def clean_password2(self):
        """ Function for validations passwords """
        password1 = self.cleaned_data.get('password1')
        password2 = self.cleaned_data.get('password2')
        if password1 and password2 and password1 != password2:
            raise ValidationError(
                _('password_mismatch')
            )
        return password2


class LoginUserForm(AuthenticationForm):
    """
    Login Form inherited from AuthenticationForm
    """

    def clean(self):
        """ add to method clean my function for validation input data """

        username = self.cleaned_data.get("username")
        password = self.cleaned_data.get("password")

        get_customer_activated(self.request, email=username)

        if username is not None and password:
            self.user_cache = authenticate(
                self.request, username=username, password=password
            )
            if self.user_cache is None:
```

```
            raise self.get_invalid_login_error()
        else:
            self.confirm_login_allowed(self.user_cache)

    return self.cleaned_data
```

4. Ai_services.py

```
def get_data(request):
    data = {
        'product_name': request.GET.get('product_name'),
        'subcategory': request.GET.get('subcategory')
    }
    return data


def generate_description(request):
    request_data = get_data(request)
    prompt = f"Напиши унікальний опис для товару '{request_data['product_name']}' \
     у категорії '{request_data['subcategory']}'. Треба суцільний текст без ніяких пунктів"
    try:
        response = client.chat.completions.create(
            model="gpt-4o-mini",
            messages=[
                {"role": "system", "content": "Ти експерт з аналізу ринку в Україні."},
                {"role": "user", "content": prompt}
            ],
            max_tokens=150,
        )
        description = response.choices[0].message.content
        return description.strip()
    except openai.OpenAIError as e:
        return JsonResponse({"error": f"Помилка генерації опису: {e}"}, status=400)


def generate_price(request):
    request_data = get_data(request)
    prompt = f"Проаналізуй, яка середня ринкова ціна для '{request_data['product_name']}' в Україні.\
     Вкажи орієнтовну ціну в гривнях, враховуючи сучасні тенденції та доступну інформацію. \
     Відповіддю повине бути лише число"
    try:
        response = client.chat.completions.create(
            model="gpt-4o-mini",
            messages=[
                {"role": "system", "content": "Ти генератор описів товарів."},
                {"role": "user", "content": prompt}
            ]
        )
```

```
        market_price = response.choices[0].message.content
        return market_price.strip()
    except openai.OpenAIError as e:
        return JsonResponse({"error": f"Помилка аналізу ціни: {e}"}, status=400)


def generate_image_url(request):
    request_data = get_data(request)
    prompt = f"Згенеруй картинку {request_data['product_name']}"
    try:
        response = client.images.generate(
            model="dall-e-2",
            prompt=prompt,
            n=1,
            size="512x512"
        )
        return response.data[0].url
    except openai.OpenAIError as e:
        return JsonResponse({"error": f"Помилка генерації картинки: {e}"}, status=400)
```

5. shop urls.py

```
urlpatterns = [
    path('', index, name='home'),
    path('register/', RegisterView.as_view(), name='register'),
    path('register/<token>', register_confirm, name='register_confirm'),
    path('login/', LoginUser.as_view(), name='my_login'),
    path('logout/', logout_customer, name='my_logout'),

    path('make-order/', MakeOrder.as_view(), name='make_order'),

    path('<category_slug>/', show_subcategories, name='show_subcategories'),
    path('<category_slug>/<subcategory_slug>', show_subcategory_products, name='show_subcategory_products'),
    path('<category_slug>/<subcategory_slug>/<product_slug>', show_products, name='show_product_details'),
]
```

6. main urls.py

```
urlpatterns = [
    path('admin/', include('shop.admin_urls')),
    path('admin/', admin.site.urls),
    path('basket/', include('basket.urls')),
    path('accounts/', include('django.contrib.auth.urls')),
    path('', include('shop.urls')),
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

7. basket urls.py

```
urlpatterns = [
    path(‘‘, basket_list, name=‘basket_list’),
    path(‘update_quantity/’, update_product_quantity, name=‘update_quantity’),
    path(‘<product_id>/add/’, add_basket_item, name=‘add_basket_item’),
    path(‘<product_id>/remove/’, remove_basket_item, name=‘remove_basket_item’),
    path(‘clear/’, clear_basket, name=‘clear_basket’),
]
```

8. basket views.py

```
def basket_list(request):
    products = create_basket_list(request)
    context = {
        ‘products’: products,
    }
    return render(request, ‘basket/basket.html’, context=context)


def add_basket_item(request, product_id):
    if request.method == ‘POST’:
        if not request.session.get(‘basket’):
            request.session[‘basket’] = list()

        add_item_to_basket(request, product_id)
    return redirect(request.POST.get(‘url_from’))


def remove_basket_item(request, product_id):
    if request.method == ‘POST’:
        remove_item_from_basket(request, product_id)
        request.session.modified = True
    return redirect(request.POST.get(‘url_from’))


def clear_basket(request):
    if request.session.get(‘basket’):
        del request.session[‘basket’]

    return redirect(‘basket_list’)


def update_product_quantity(request):
    if request.method == ‘POST’:
        change_product_quantity(request)
    return JsonResponse({‘success’: True})
```

9. tests.py

```
class ModelsTestCase(TestCase):

    def setUp(self):
        # Створення тестових даних
```

```
        self.customer = Customer.objects.create(
            email="testuser@example.com",
            first_name="Test",
            last_name="User",
            is_active=False,
        )

        self.category = Category.objects.create(
            name="Electronics",
            slug="electronics"
        )

        self.subcategory = SubCategory.objects.create(
            name="Smartphones",
            slug="smartphones",
            super_category=self.category
        )

        self.product = Product.objects.create(
            name="iPhone 13",
            slug="iphone-13",
            description="Latest Apple smartphone",
            price=1000,
            subcategory=self.subcategory
        )

        self.order = Order.objects.create(
            customer=self.customer,
            status=False,
            transaction_id="TRX12345"
        )

        self.order_item = OrderItem.objects.create(
            product=self.product,
            order=self.order,
            quantity=2
        )

        self.shipping = Shipping.objects.create(
            customer=self.customer,
            order=self.order,
            address="123 Test Street",
            city="TestCity"
        )

    def test_customer_creation(self):
        # Перевірка створення клієнта
```

```
        self.assertEqual(Customer.objects.count(), 1)
        self.assertEqual(self.customer.email, "testuser@example.com")
        self.assertFalse(self.customer.is_active)

    def test_category_creation(self):
        # Перевірка створення категорії
        self.assertEqual(Category.objects.count(), 1)
        self.assertEqual(self.category.name, "Electronics")
        self.assertEqual(self.category.slug, "electronics")

    def test_subcategory_creation(self):
        # Перевірка створення підкатегорії
        self.assertEqual(SubCategory.objects.count(), 1)
        self.assertEqual(self.subcategory.name, "Smartphones")
        self.assertEqual(self.subcategory.super_category, self.category)

    def test_product_creation(self):
        # Перевірка створення продукту
        self.assertEqual(Product.objects.count(), 1)
        self.assertEqual(self.product.name, "iPhone 13")
        self.assertEqual(self.product.subcategory, self.subcategory)
        self.assertEqual(self.product.price, 1000)

    def test_order_creation(self):
        # Перевірка створення замовлення
        self.assertEqual(Order.objects.count(), 1)
        self.assertEqual(self.order.customer, self.customer)
        self.assertEqual(self.order.transaction_id, "TRX12345")
        self.assertFalse(self.order.status)

    def test_order_item_creation(self):
        # Перевірка створення елементу замовлення
        self.assertEqual(OrderItem.objects.count(), 1)
        self.assertEqual(self.order_item.product, self.product)
        self.assertEqual(self.order_item.order, self.order)
        self.assertEqual(self.order_item.quantity, 2)

    def test_shipping_creation(self):
        # Перевірка створення доставки
        self.assertEqual(Shipping.objects.count(), 1)
        self.assertEqual(self.shipping.customer, self.customer)
        self.assertEqual(self.shipping.address, "123 Test Street")
        self.assertEqual(self.shipping.city, "TestCity")

    def test_product_absolute_url(self):
        # Перевірка методу get_absolute_url() для продукту
        expected_url = f"/{self.category.slug}/{self.subcategory.slug}/{self.product.slug}"
```

```
        self.assertEqual(self.product.get_absolute_url(), expected_url)

    def test_category_absolute_url(self):
        # Перевірка методу get_absolute_url() для категорії
        expected_url = f"/{self.category.slug}/"
        self.assertEqual(self.category.get_absolute_url(), expected_url)


class CustomerManagerTests(TestCase):

    def test_create_user_with_email(self):
        """Перевірка створення звичайного користувача з email."""
        email = "user@example.com"
        password = "testpassword123"
        user = Customer.objects.create_user(email=email, password=password)

        self.assertEqual(user.email, email)
        self.assertTrue(user.check_password(password))
        self.assertFalse(user.is_staff)
        self.assertFalse(user.is_superuser)

    def test_create_user_without_email(self):
        """Перевірка помилки при спробі створення користувача без email."""
        with self.assertRaises(ValueError) as context:
            Customer.objects.create_user(email=None, password="testpassword123")
        self.assertEqual(str(context.exception), "The given email must be set")

    def test_create_superuser(self):
        """Перевірка створення суперкористувача."""
        email = "admin@example.com"
        password = "adminpassword123"
        superuser = Customer.objects.create_superuser(email=email, password=password)

        self.assertEqual(superuser.email, email)
        self.assertTrue(superuser.check_password(password))
        self.assertTrue(superuser.is_staff)
        self.assertTrue(superuser.is_superuser)

    def test_create_superuser_with_missing_is_superuser(self):
        """Перевірка помилки, якщо суперкористувач створений без is_superuser=True."""
        with self.assertRaises(ValueError) as context:
            Customer.objects.create_superuser(email="admin@example.com", password="testpassword123", is_superuser=False)
        self.assertEqual(str(context.exception), "Superuser must have is_superuser=True.")


class TestShopURLsHTTPStatus(TestCase):
    def setUp(self):
```

```
    """Попередні налаштування"""
    # Якщо необхідно, можна створити тестові дані тут
    self.category = Category.objects.create(
        name="Electronics",
        slug="electronics"
    )
    self.subcategory = SubCategory.objects.create(
        name="Smartphones",
        slug="smartphones",
        super_category=self.category
    )
    self.product = Product.objects.create(
        name="iPhone 13",
        slug="iphone-13",
        description="Latest Apple smartphone",
        price=1000,
        subcategory=self.subcategory,
        image="path/to/test/image.jpg"
    )
    self.token = "test_token"

def test_home_page_status_code(self):
    """Перевіряємо, чи головна сторінка повертає статус 200"""
    response = self.client.get(reverse('home'))
    self.assertEqual(response.status_code, 200)

def test_register_page_status_code(self):
    """Перевіряємо сторінку реєстрації"""
    response = self.client.get(reverse('register'))
    self.assertEqual(response.status_code, 200)

def test_register_confirm_page_status_code(self):
    """Перевіряємо сторінку підтвердження реєстрації"""
    response = self.client.get(reverse('register_confirm', args=[self.token]))
    self.assertEqual(response.status_code, 302)

def test_login_page_status_code(self):
    """Перевіряємо сторінку входу"""
    response = self.client.get(reverse('my_login'))
    self.assertEqual(response.status_code, 200)

def test_logout_page_status_code(self):
    """Перевіряємо сторінку виходу"""
    response = self.client.get(reverse('my_logout'))
    self.assertEqual(response.status_code, 302)  # Перенаправлення після виходу

def test_make_order_page_status_code(self):
```

```
        """Перевіряємо сторінку створення замовлення"""
        response = self.client.get(reverse('make_order'))
        self.assertEqual(response.status_code, 302)  # Можливо, доступ лише для авторизованих користувачів

    def test_show_subcategories_page_status_code(self):
        """Перевіряємо сторінку показу підкатегорій"""
        response = self.client.get(reverse('show_subcategories', args=[self.category.slug]))
        self.assertEqual(response.status_code, 200)

    def test_show_subcategory_products_page_status_code(self):
        """Перевіряємо сторінку показу товарів у підкатегорії"""
        response = self.client.get(reverse('show_subcategory_products', args=[self.category.slug, self.subcategory.slug]))
        self.assertEqual(response.status_code, 200)

    def test_show_product_details_page_status_code(self):
        """Перевіряємо сторінку показу деталей товару"""
        response = self.client.get(reverse('show_product_details', args=[self.category.slug, self.subcategory.slug, self.product.slug]))
        self.assertEqual(response.status_code, 200)


class TestRegisterCustomerForm(TestCase):

    def test_valid_form(self):
        """Перевіряємо, чи форма валідна при правильних даних"""
        form_data = {
            'email': 'test@example.com',
            'first_name': 'John',
            'last_name': 'Doe',
            'password1': 'validpassword123',
            'password2': 'validpassword123',
            'phone': '+380123456789',
        }
        form = RegisterCustomerForm(data=form_data)
        self.assertTrue(form.is_valid())

    def test_invalid_phone_format(self):
        """Перевіряємо помилку при некоректному номері телефону"""
        form_data = {
            'email': 'test@example.com',
            'first_name': 'John',
            'last_name': 'Doe',
            'password1': 'validpassword123',
            'password2': 'validpassword123',
            'phone': '1234567890',  # Некоректний номер телефону
        }
        form = RegisterCustomerForm(data=form_data)
        self.assertFalse(form.is_valid())
```

```
        self.assertEqual(form.errors[‘phone’],
                         [‘Uncorrected phone number\nPhone number must consist:+380 ** *** ** **’])

    def test_password_mismatch(self):
        """Перевіряємо помилку, якщо паролі не співпадають"""
        form_data = {
            ‘email’: ‘test@example.com’,
            ‘first_name’: ‘John’,
            ‘last_name’: ‘Doe’,
            ‘password1’: ‘validpassword123’,
            ‘password2’: ‘differentpassword123’,
            ‘phone’: ‘+380123456789’,
        }
        form = RegisterCustomerForm(data=form_data)
        self.assertFalse(form.is_valid())
        self.assertEqual(form.errors[‘password2’], [‘password_mismatch’])

    def test_missing_email(self):
        """Перевіряємо форму на відсутність обов’язкового email"""
        form_data = {
            ‘first_name’: ‘John’,
            ‘last_name’: ‘Doe’,
            ‘password1’: ‘validpassword123’,
            ‘password2’: ‘validpassword123’,
            ‘phone’: ‘+380123456789’,
        }
        form = RegisterCustomerForm(data=form_data)
        self.assertFalse(form.is_valid())
        self.assertEqual(form.errors[‘email’], [‘This field is required.’])

    def test_missing_required_fields(self):
        """Перевіряємо, чи форма не валідна без обов’язкових полів"""
        form_data = {
            ‘email’: ‘‘,
            ‘first_name’: ‘‘,
            ‘last_name’: ‘‘,
            ‘password1’: ‘‘,
            ‘password2’: ‘‘,
            ‘phone’: ‘‘,
        }
        form = RegisterCustomerForm(data=form_data)
        self.assertFalse(form.is_valid())
        self.assertEqual(len(form.errors), 6)


class TestLoginUserForm(TestCase):
```

```
    def setUp(self):
        """Створюємо тестового користувача для перевірки форми входу"""
        self.user = Customer.objects.create(
            email='test@example.com',
            password='validpassword123',
            is_active=True
        )

    def test_valid_login(self):
        """Перевіряємо успішний вхід з правильним username і password"""
        form_data = {
            'username': 'test@example.com',
            'password': 'validpassword123',
        }
        form = LoginUserForm(data=form_data)
        self.assertFalse(form.is_valid())

    def test_invalid_username(self):
        """Перевіряємо помилку, коли введено неправильний email"""
        form_data = {
            'username': 'wrong@example.com',
            'password': 'validpassword123',
        }
        form = LoginUserForm(data=form_data)
        self.assertFalse(form.is_valid())
        self.assertEqual(form.errors['__all__'], ['User with this email does not exist'])

    def test_invalid_password(self):
        """Перевіряємо помилку, коли введено неправильний пароль"""
        form_data = {
            'username': 'test@example.com',
            'password': 'wrongpassword123',
        }
        form = LoginUserForm(data=form_data)
        self.assertFalse(form.is_valid())
        self.assertEqual(form.errors['__all__'], ['Please enter a correct email address and password. Note that both fields may be case-sensitive.'])

    def test_missing_username_or_password(self):
        """Перевіряємо форму на відсутність username або пароля"""
        form_data = {
            'username': '',
            'password': '',
        }
        form = LoginUserForm(data=form_data)
        self.assertFalse(form.is_valid())
        self.assertEqual(len(form.errors), 3)
```

```
    def test_clean_method_with_authenticated_user(self):
        """Перевіряємо, чи правильно працює метод clean при наявному користувачі"""
        form_data = {
            'username': 'test@example.com',
            'password': 'validpassword123',
        }
        form = LoginUserForm(data=form_data)
        self.assertFalse(form.is_valid())


class TestViews(TestCase):
    def setUp(self):
        self.client = Client()
        self.user = Customer.objects.create_user(email='test@test.com', password='password123')
        self.category = Category.objects.create(name='Category 1', slug='category-1')
        self.subcategory = SubCategory.objects.create(name='SubCategory 1', slug='subcategory-1', super_category=self.category)
        self.product = Product.objects.create(
            name='Product 1',
            slug='product-1',
            description='Test description',
            price=100,
            subcategory=self.subcategory,
            image="path/to/test/image.jpg"
        )

    def test_index(self):
        response = self.client.get(reverse('home'))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'shop/base.html')
        self.assertIn('recommended_products', response.context)
        self.assertIn('recommended_subcategories', response.context)

    def test_register_view(self):
        response = self.client.post(reverse('register'), {
            'email': 'newuser@test.com',
            'first_name': 'First',
            'last_name': 'Last',
            'password1': 'password123',
            'password2': 'password123',
            'phone': '+380123456789'
        })
        self.assertEqual(response.status_code, 302)
        self.assertEqual(Customer.objects.filter(email='newuser@test.com').count(), 1)

    def test_register_confirm_view(self):
        token = 'testtoken'
```

```
        with patch('shop.views.confirm_customer', return_value=True):
            response = self.client.get(reverse('register_confirm', args=[token]))
            self.assertEqual(response.status_code, 302)

    def test_login_view(self):
        response = self.client.post(reverse('my_login'), {
            'username': self.user.email,
            'password': 'password123',
        })
        self.assertEqual(response.status_code, 200)

    def test_logout_view(self):
        self.client.login(email='test@test.com', password='password123')
        response = self.client.get(reverse('my_logout'))
        self.assertEqual(response.status_code, 302)

    def test_show_subcategories_view(self):
        response = self.client.get(reverse('show_subcategories', args=[self.category.slug]))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'shop/subcategories.html')
        self.assertIn('subcategories', response.context)

    def test_show_subcategory_products_view(self):
        response = self.client.get(reverse('show_subcategory_products', args=[self.category.slug, self.subcategory.slug]))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'shop/show_subcategory.html')
        self.assertIn('products', response.context)
        self.assertIn('subcategory', response.context)

    def test_show_products_view(self):
        response = self.client.get(reverse('show_product_details', args=[self.category.slug, self.subcategory.slug, self.product.slug]))
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'shop/product_detail.html')
        self.assertIn('product', response.context)

    def test_make_order_view_authenticated(self):
        self.client.login(email='test@test.com', password='password123')
        response = self.client.get(reverse('make_order'))
        self.assertEqual(response.status_code, 302)

    def test_make_order_view_unauthenticated(self):
        response = self.client.get(reverse('make_order'))
        self.assertEqual(response.status_code, 302)
        self.assertRedirects(response, f"{reverse('my_login')}?next={reverse('make_order')}")


class TestAIServiceFunctions(TestCase):
```

```
def setUp(self):
    self.request_mock = mock.Mock()
    self.request_mock.GET = {
        "product_name": "Смартфон",
        "subcategory": "Електроніка"
    }

@mock.patch("shop.services.AI_services.client.chat.completions.create")
def test_generate_description(self, mock_openai_create):
    """Тест функції generate_description"""
    mock_openai_create.return_value = mock.Mock(
        choices=[mock.Mock(message=mock.Mock(content="Це унікальний опис смартфона."))]
    )

    description = generate_description(self.request_mock)
    self.assertEqual(description, "Це унікальний опис смартфона.")
    mock_openai_create.assert_called_once()

@mock.patch("shop.services.AI_services.client.chat.completions.create")
def test_generate_price(self, mock_openai_create):
    """Тест функції generate_price"""
    mock_openai_create.return_value = mock.Mock(
        choices=[mock.Mock(message=mock.Mock(content="10000"))]
    )

    price = generate_price(self.request_mock)
    self.assertEqual(price, "10000")
    mock_openai_create.assert_called_once()

@mock.patch("shop.services.AI_services.client.images.generate")
def test_generate_image_url(self, mock_openai_generate):
    """Тест функції generate_image_url"""
    mock_openai_generate.return_value = mock.Mock(
        data=[mock.Mock(url="http://example.com/image.png")]
    )

    image_url = generate_image_url(self.request_mock)
    self.assertEqual(image_url, "http://example.com/image.png")
    mock_openai_generate.assert_called_once()

@mock.patch("shop.services.AI_services.client.chat.completions.create")
def test_generate_description_error(self, mock_openai_create):
    """Тест помилки генерації опису"""
    mock_openai_create.side_effect = OpenAIError("Помилка аналізу ціни")

    response = generate_description(self.request_mock)
    self.assertIsInstance(response, JsonResponse)
```

```
        self.assertEqual(response.status_code, 400)

        response_data = json.loads(response.content.decode())
        self.assertIn("Помилка генерації опису", response_data.get("error", ""))

    @mock.patch("shop.services.AI_services.client.chat.completions.create")
    def test_generate_price_error(self, mock_openai_create):
        """Тест помилки генерації ціни"""
        mock_openai_create.side_effect = OpenAIError("Помилка аналізу ціни")

        response = generate_price(self.request_mock)
        self.assertIsInstance(response, JsonResponse)
        self.assertEqual(response.status_code, 400)
        response_data = json.loads(response.content.decode())
        self.assertIn("Помилка аналізу ціни", response_data.get("error", ""))

    @mock.patch("shop.services.AI_services.client.images.generate")
    def test_generate_image_url_error(self, mock_openai_generate):
        """Тест помилки генерації зображення"""
        mock_openai_generate.side_effect = OpenAIError("Помилка генерації картинки")

        response = generate_image_url(self.request_mock)
        self.assertIsInstance(response, JsonResponse)
        self.assertEqual(response.status_code, 400)
        response_data = json.loads(response.content.decode())
        self.assertIn("Помилка генерації картинки", response_data.get("error", ""))
```