МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПОЛІСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних
технологій, обліку та фінансів
Кафедра комп'ютерних технологій
і моделювання систем

Кваліфікаційна робота
На правах рукопису

Кирилюка Богдана Михайловича
(прізвище, ім'я, по батькові здобувача освіти)

УДК_____

КВАЛІФІКАЦІЙНА РОБОТА

Інформаційна система для інтернет-магазину мобільних телефонів з використанням штучного інтелекту
(тема роботи)

122 – комп'ютерні науки
(шифр і назва спеціальності)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____
(підпис, ініціали та прізвище здобувача вищої освіти)

Керівник роботи

Гіваргізов Інвія Геннадійович
(прізвище, ім'я, по батькові)

кандидат економічних наук
(науковий ступінь, вчене звання)

Житомир – 2025

Висновок кафедри _____
за результатами попереднього захисту: _____

Протокол засідання кафедри _____
№ ____ від «_____» _____ 20____ р.

Завідувач кафедри _____

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)
«_____» _____ 20____ р.

Результати захисту кваліфікаційної роботи

Здобувач вищої освіти _____ захистив (ла)
(прізвище ,ім'я, по батькові)
кваліфікаційну роботу з оцінкою:

сума балів за 100-бальною шкалою _____
за шкалою ECTS _____
за національною шкалою _____

Секретар ЕК

_____ _____ _____
(науковий ступінь, вчене звання) (підпис) (прізвище, ім'я, по батькові)

АНОТАЦІЯ

Кирилюк Б. М. Інформаційна система для інтернет-магазину мобільних телефонів з використанням штучного інтелекту – Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття освітнього ступеня магістра за спеціальністю 122 – комп'ютерні науки. – Поліський національний університет, Житомир, 2025.

У роботі представлено розробку веб-орієнтованої інформаційної системи для інтернет-магазину мобільних телефонів, яка інтегрує компоненти штучного інтелекту для покращення користувацького досвіду. Основна увага приділена реалізації функцій інтелектуального текстового пошуку з використанням методів обробки природної мови (NLP) та системи рекомендацій товарів на основі алгоритмів пошуку подібностей (FAISS). Проведено аналіз предметної області, розроблено архітектуру та інтерфейс системи, описано технічну реалізацію з використанням сучасних веб-технологій. Практична цінність роботи полягає у створенні інструменту, здатного адаптуватися до потреб користувача та забезпечувати персоналізовану взаємодію, що є актуальним для електронної комерції.

Ключові слова: FAISS, NLP, інтернет-магазин, мобільні телефони, інформаційна система.

SUMMARY

Kyryliuk B. M. Information System for an Online Mobile Phone Store with the Use of Artificial Intelligence – Qualification thesis (manuscript format).

A thesis submitted in partial fulfillment of the requirements for the Master's degree in specialty 122 – Computer Science. – Polissia National University, Zhytomyr, 2025.

The thesis presents the development of a web-oriented information system for an online mobile phone store, which integrates artificial intelligence components to enhance user experience. The main focus is on implementing intelligent text search functionality using natural language processing (NLP) methods and a product recommendation system based on similarity search algorithms (FAISS). The subject area is analyzed, the system's architecture and interface are designed, and the technical implementation using modern web technologies is described. The practical value of the work lies in creating a tool capable of adapting to user needs and ensuring personalized interaction, which is highly relevant for e-commerce.

Keywords: FAISS, NLP, online store, mobile phones, information system.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ШІ (Штучний інтелект) – це галузь комп’ютерної науки, що займається створенням систем, здатних виконувати завдання, які зазвичай потребують людського інтелекту.

UML – Unified Modeling Language (Уніфікована мова моделювання) – це стандартизована мова для візуалізації, специфікації, конструювання та документування компонентів програмного забезпечення.

IDEF0 (Integration Definition for Function Modeling) - це методика моделювання функціональних процесів, яка дозволяє зобразити систему у вигляді блоків (функцій) з входами, виходами, механізмами та управлінням.

API – Application Programming Interface (Інтерфейс прикладного програмування) – це набір визначень і протоколів, за допомогою яких різні програмні додатки взаємодіють між собою.

Frontend (Фронтенд) – це частина веб-застосунку або сайту, яку бачить користувач і з якою він взаємодіє.

Backend (Бекенд) – це внутрішня частина застосунку, що відповідає за логіку, обробку даних, взаємодію з базами даних і API.

NLP - Natural Language Processing (Обробка природної мови) – це підгалузь штучного інтелекту, що займається аналізом, розумінням та генерацією людської мови комп’ютерами.

FAISS (Facebook AI Similarity Search) – це бібліотека, розроблена Facebook AI, призначена для ефективного пошуку подібних елементів у великих наборах векторних даних.

JSON (JavaScript Object Notation) – це текстовий формат обміну даними між комп'ютерами.

ВСТУП

Актуальність теми дослідження. Сучасні інтернет-магазини вже давно вийшли за межі простих каталогів товарів, перетворившись на інтелектуальні системи, що адаптуються до поведінки користувача. В умовах високої конкуренції у сфері електронної комерції особливої актуальності набуває впровадження технологій штучного інтелекту для підвищення якості обслуговування, персоналізації взаємодії та оптимізації внутрішніх процесів. Зокрема, використання рекомендаційних систем, інтелектуального пошуку, аналізу поведінкових даних користувачів та автоматизованої обробки контенту відкриває нові можливості для покращення досвіду покупця.

Мета і завдання дослідження. Метою дипломної роботи є розробка інтернет-магазину мобільних телефонів із вбудованими елементами штучного інтелекту. Основна увага приділяється впровадженню рекомендаційної системи, яка базується на обробці векторних представленнях товарів, а також інтелектуальному пошуку, що заснований на обробці природної мови.

Для досягнення цієї мети потрібно виконати такі завдання:

– Дослідити сучасні підходи щодо використання ШІ в електронній комерції;

– Розробити загальну архітектуру всіх модулів розроблюваної системи;

– Реалізувати базову функціональність інтернет-магазину;

– Реалізувати систему рекомендацій та інтелектуальний пошук товарів;

– Забезпечити зручний користувацький інтерфейс для взаємодії з усіма функціями системи.

Об’єкт дослідження. Процеси функціонування та розвитку електронної комерції в сегменті онлайн-продажу мобільних телефонів із використанням технологій штучного інтелекту.

Предмет дослідження. Сукупність методів інтеграції, а також моделей штучного інтелекту в архітектуру інтернет-магазину: зокрема, застосування систем рекомендацій та інтелектуального пошуку.

Методи дослідження. Застосування аналізу та синтезу дало змогу виявити особливості функціонування сучасних інтернет-магазинів і визначити вимоги до архітектури майбутньої інформаційної системи (підрозділи 1.1, 1.2). Метод моделювання дозволив побудувати структурну модель системи, що охоплює її функціональні блоки та взаємозв'язки між ними (підрозділ 2.1). За допомогою індуктивного та дедуктивного методів досліджено можливості застосування штучного інтелекту, зокрема векторного порівняння та інтелектуального пошуку, у функціональності інтернет-магазину (підрозділ 2.2). Емпіричний метод застосовано під час реалізації інформаційної системи (підрозділ 3.1). Метод порівняння забезпечив можливість зіставлення інтерфейсних рішень та функціональних можливостей розробленої системи (підрозділ 3.2).

Перелік публікацій автора за темою дослідження.

Кирилюк Б. М. Інтелектуальні технології в архітектурі сучасних інтернет-магазинів. Інформаційні технології та моделювання систем: матеріали Всеукраїнської науково-практичної конференції. — 2025. — С. 18–21.

Кирилюк Б. М. Рекомендаційні системи в e-commerce: сучасні підходи та інструменти. Наукові читання – 2025: матеріали науково-практичної конференції науково-педагогічних працівників, докторантів, аспірантів та молодих вчених до Дня науки в Україні. — 2025.

Практичне значення. Розроблена система може бути використана як готове рішення для створення інтелектуальних онлайн-магазинів. Завдяки використанню ШІ-функціоналу платформа дозволяє підвищити якість сервісу, збільшити конверсію та забезпечити гнучкість у масштабуванні під різні товарні категорії. Запропоновані підходи можуть бути застосовані також в інших сегментах ринку електронної комерції.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох основних розділів, висновків, списку використаних джерел із 40 найменувань, 40 додатків. Загальний обсяг роботи становить 76 сторінки, з яких 22 - основний текст.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИМОГ ДО СИСТЕМИ

1.1 Характеристика предметної області та огляд існуючих рішень

Розвиток електронної комерції супроводжується впровадженням передових інформаційних технологій, серед яких важливу роль відіграє штучний інтелект. Інтернет-магазини використовують ШІ для автоматизації бізнес-процесів, підвищення ефективності роботи з клієнтами та оптимізації управління товарним асортиментом. Це дозволяє не лише покращити користувацький досвід, а й забезпечити конкурентну перевагу на ринку.

Одним із ключових напрямів застосування ШІ в інтернет-магазинах є персоналізація взаємодії з покупцем. Завдяки аналізу поведінки користувачів та їхніх попередніх покупок, система може формувати індивідуальні рекомендації, пропонуючи товари, що найбільш відповідають інтересам клієнта. Найпоширеніші методи реалізації таких систем включають колаборативну фільтрацію, контентно-орієнтовані алгоритми та гібридні моделі [39].

Колаборативна фільтрація працює за принципом пошуку схожих товарів. Наприклад, якщо користувач А купив телефон певної моделі, а користувач В придбав таку ж модель та аксесуари до неї, то система може рекомендувати аксесуари користувачу А. Основна перевага цього підходу – здатність знаходити релевантні пропозиції без необхідності глибокого аналізу самих товарів. Однак проблема виникає у випадку нових користувачів чи нових товарів – коли немає достатньої історії для формування якісних рекомендацій (проблема "холодного старту") [28].

Контентно-орієнтовані алгоритми аналізують характеристики товарів і порівнюють їх із уподобаннями користувача. Якщо покупець раніше цікавився смартфонами з великим екраном і потужною батареєю, система запропонує йому інші моделі з подібними характеристиками. Цей метод не страждає від проблеми "холодного старту" для користувачів, але має свої недоліки: рекомендації можуть бути занадто однотипними, що обмежує різноманітність пропозицій [28].

Гібридні системи поєднують обидва підходи, усуваючи їхні слабкі сторони. Наприклад, Netflix і Amazon використовують такі моделі для покращення персоналізованих рекомендацій. У випадку інтернет-магазинів мобільних телефонів це може виглядати так: система аналізує технічні характеристики телефонів, але також враховує поведінкові фактори інших користувачів, які робили схожі покупки [28].

Популярні рішення на ринку включають Google Recommendations AI, Amazon Personalize, Microsoft Azure Personalizer. Google Recommendations AI використовує глибоке навчання для передбачення потреб клієнтів і дозволяє впроваджувати рекомендації у реальному часі, однак вимагає значних обчислювальних ресурсів. Amazon Personalize працює за принципом колаборативної фільтрації з використанням машинного навчання, але для ефективної роботи потребує значної кількості даних. Microsoft Azure Personalizer орієнтований на навчання моделей в процесі взаємодії з користувачами, що підвищує адаптивність, але потребує тривалого налаштування [13].

Окрім "готових рішень" існує багато інших методів та підходів щодо створення системи рекомендацій. Подібні системи реалізуються за допомогою різних алгоритмів та бібліотек, які дозволяють ефективно аналізувати дані та формувати індивідуальні пропозиції для користувачів.

Для створення системи рекомендацій використовуються методи матричної факторизації, такі як Singular Value Decomposition (SVD) або Alternating Least Squares (ALS). Вони дозволяють представляти взаємозв’язки між користувачами та товарами у вигляді матриці, де значення позначають рівень інтересу чи оцінки. Наприклад, метод SVD розкладає цю матрицю на кілька простіших, виділяючи приховані закономірності. Хоча такий підхід добре працює при наявності великої кількості даних, він потребує періодичного перевчання, що може бути ресурсозатратним [9].

Інший підхід до рекомендацій – використання моделі LightFM, яка поєднує матричну факторизацію та контентний аналіз. LightFM враховує як історію взаємодії користувачів, так і характеристики товарів (наприклад, категорію,

ціновий сегмент, бренд), що дозволяє краще підлаштовувати рекомендації під конкретного покупця.

Більш сучасним варіантом є нейронні мережі, зокрема моделі Neural Collaborative Filtering (NCF), які використовують багатошарові нейромережі для виявлення глибших зв'язків між товарами та користувачами. Такі системи можуть адаптуватися до змін у поведінці користувачів, але вимагають значних обчислювальних ресурсів [27].

Також, в системі рекомендацій, високу ефективність демонструє бібліотека FAISS, бібліотека для швидкого пошуку схожих векторів у великих наборах даних. Вона використовується для пошуку схожих товарів на основі їх векторних представлень. Наприклад, якщо мобільний телефон представлений у вигляді вектора характеристик (ціна, бренд, технічні параметри, тощо), FAISS дозволяє швидко знаходити найбільш схожі товари за цими параметрами. Подібний підхід використовується в семантичному пошуку, коли товари та запити користувачів конвертуються у вектори за допомогою моделей, таких як Sentence-BERT (SBERT) або Universal Sentence Encoder (USE) [4].

Крім цього, важливу роль відіграє пошук товарів на основі обробки природної мови (NLP). Завдяки цьому користувачі можуть здійснювати пошук не лише за ключовими словами, а й за описами, подібними товарами чи навіть неструктурованими запитами, що схожі на людську мову. Наприклад, запит "дешевий смартфон з хорошою камерою" буде інтерпретований системою, яка знайде відповідні моделі, навіть якщо в їх описах немає точного збігу всіх слів [10].

Для реалізації пошуку товарів на базі NLP часто застосовують методи векторизації тексту. Одним із найпростіших є TF-IDF, який оцінює важливість слів у тексті товарних описів. Проте цей підхід не враховує контекст і не завжди ефективний для складних запитів [12].

Більш точні результати забезпечує BM25 – алгоритм ранжування документів, який використовується в пошукових системах, таких як Elasticsearch.

Він враховує довжину тексту, частоту слів у ньому та розподіл термінів у всій колекції документів [12].

Сучасні пошукові системи все частіше використовують трансформерні моделі, такі як BERT або Sentence-BERT (SBERT), які дозволяють аналізувати запити користувачів з урахуванням контексту. SBERT представляє текстові запити та описи товарів у вигляді векторів у багатовимірному просторі, що дозволяє знаходити семантично схожі елементи [26].

1.2 Визначення вимог до інформаційної системи

Процес розробки інформаційної системи потребує чіткого визначення вимог, які вона повинна задовольняти. Вимоги виступають основою для проєктування архітектури, вибору технологій та реалізації функціоналу. В цьому підрозділі наведено перелік функціональних та нефункціональних вимог до інформаційної системи для інтернет-магазину мобільних телефонів з використанням штучного інтелекту.

Функціональні вимоги визначають основні можливості, які повинна забезпечувати інформаційна система, вони охоплюють як стандартну e-commerce-функціональність, так і компоненти, пов’язані з використанням штучного інтелекту. Система повинна забезпечувати зручну взаємодію користувача з онлайн-магазином, дозволяючи переглядати товари, здійснювати пошук, додавати товари до кошика, оформлювати замовлення та відстежувати його статус. Також передбачається реалізація особистого кабінету користувача, де покупець зможе переглядати історію своїх замовлень, змінювати персональні дані, а також керувати списками бажань [32].

Особливу роль у системі відіграє модуль персоналізованих рекомендацій. На основі історії переглядів, замовлень і взаємодій із товарами користувача система повинна формувати добірки рекомендованих моделей мобільних телефонів. Для цього повинні використовуються алгоритми машинного навчання,

які можуть бути реалізовані з використанням векторного пошуку за допомогою FAISS або за допомогою гібридного підходу, що комбінує факторизацію матриць і контентний аналіз.

Окрім рекомендацій, система повинна підтримувати інтелектуальний пошук товарів. Замість точного збігу ключових слів, пошуковий механізм має враховувати семантичне значення запитів, що досягається за допомогою NLP-моделей. Це дозволить користувачам знаходити потрібні товари навіть за загальними або описовими запитами.

Управління асортиментом товарів має бути доступним через адміністративну панель. Адміністратор повинен мати можливість додавати, змінювати або видаляти товари, встановлювати ціни, додавати характеристики, зображення, а також відстежувати статистику продажів і поведінку користувачів.

Нефункціональні вимоги до інформаційної системи охоплюють низку важливих характеристик, які впливають на якість її роботи. Однією з ключових вимог є продуктивність — система повинна забезпечувати швидку обробку запитів навіть у разі значного навантаження з боку користувачів. Завантаження основних сторінок повинно бути швидким при середньому навантаженні, а пошук товарів і побудова рекомендацій мають виконуватися практично миттєво.

З огляду на потенційне зростання обсягів даних і кількості користувачів, архітектура системи повинна бути масштабованою, тобто мати можливість горизонтального розширення без втрати продуктивності. Це особливо актуально для компонентів на базі штучного інтелекту, таких як модулі рекомендацій або пошуку, які можуть вимагати додаткових обчислювальних ресурсів із часом.

Безпека є ще одним критично важливим аспектом. Система повинна гарантувати захист персональних даних користувачів, використовуючи автентифікацію для доступу до адміністративної панелі та особистого кабінету. Усі взаємодії з клієнтом мають здійснюватися через захищений протокол HTTPS.

Інтерфейс магазину повинен бути інтуїтивним, зрозумілим, та адаптивним до різних пристроїв — від комп’ютерів до мобільних телефонів. Особливу увагу

слід приділити простоті процесу оформлення замовлення, пошуку товарів та перегляду персоналізованих рекомендацій [22].

Останнім елементом є сумісність. Розроблений вебзастосунок повинен коректно працювати у всіх сучасних браузерах, таких як Google Chrome, Mozilla Firefox, Microsoft Edge та Safari. Також система має бути доступною для користувачів мобільних пристроїв на базі операційних систем Android та iOS, що забезпечить максимальне охоплення аудиторії [40].

Висновки до розділу 1

Таким чином, визначено що, штучний інтелект відіграє ключову роль у розвитку сучасних інтернет-магазинів, забезпечуючи ефективну роботу системи, підвищуючи рівень обслуговування клієнтів і оптимізуючи бізнес-процеси. Система персоналізованих рекомендацій та інтелектуального пошуку в інтернет-магазині може функціонувати шляхом впровадження готових рішень, або ж поєднувати різні методи та алгоритми. Використання матричної факторизації або нейромереж для аналізу взаємодії користувачів із товарами дозволяє формувати точні рекомендації, а застосування NLP-моделей, таких як SBERT у комбінації з FAISS, забезпечує ефективний пошук за змістом. Це дозволяє не тільки покращити досвід користувачів, а й підвищити рівень конверсії та продажів у магазині.

Були сформульовані чіткі функціональні та нефункціональні вимоги для розробки інформаційної системи інтернет-магазину мобільних телефонів із використанням штучного інтелекту. Вони визначають як набір основних можливостей, що забезпечують повноцінну комерційну діяльність онлайн-магазину, так і важливі технічні характеристики, які гарантують високу продуктивність, надійність, безпеку та зручність використання системи. Чітке дотримання цих вимог дозволить створити масштабовану, сучасну та ефективну

платформу, орієнтовану на покращення взаємодії з користувачем і забезпечення конкурентоспроможності на ринку.

РОЗДІЛ 2 ПРОЕКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ МОБІЛЬНИХ ТЕЛЕФОНІВ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

2.1 Моделювання структури інтернет-магазину

Розробка інтернет-магазину потребує побудови узгодженої та структурованої моделі, яка відображає всі ключові компоненти системи й забезпечує ефективну взаємодію користувача з платформою. Архітектура такої системи охоплює клієнтський інтерфейс, серверну частину, базу даних і допоміжні модулі. Для моделювання структури та логіки роботи інтернет-магазину використані сучасні підходи - зокрема, діаграми UML та функціональні схеми у нотації IDEF0 [33].

Початковим етапом моделювання є аналіз функціональних процесів, які реалізуються в системі. Для цього застосовується нотація IDEF0, яка дозволяє візуалізувати потоки інформації, задіяні ресурси та основні бізнес-процеси. На діаграмі (додаток. А) зображено узагальнену модель, що демонструє інформаційні потоки, ресурси та процеси, які забезпечують роботу магазину. Умовно модель можна поділити на два типи компонентів: ресурси та процеси. До ресурсів належать всі користувачі платформи, інформація про товари, а також програмні й апаратні засоби, що забезпечують функціонування системи. Процеси охоплюють перегляд каталогу, пошук та фільтрацію товарів, оформленням замовлень, а також збереження і обробку даних.

Взаємодія користувача із системою розпочинається з ознайомлення з каталогом товарів, де представлено асортимент мобільних телефонів. Користувач має змогу переглядати список доступних моделей, а також здійснювати пошук за заданими параметрами відповідно до власних потреб. Щоб наочно показати, як саме реалізується процес взаємодії, було розроблено контекстну діаграму (додаток Б). Діаграма складається з логічно впорядкованих етапів, які демонструють послідовність дій з боку користувача та адміністратора, а також їхню взаємодію із системними модулями. Початковим етапом є публікація товару

в інтернет-магазині, яка здійснюється адміністратором. До системи надходить інформація про товари, після чого ці дані публікуються у каталозі. Список товарів, які опубліковані адміністратором є доступним для покупців через інтерфейс інтернет-магазину.

У випадку, якщо користувач бажає зберегти товар до списку обраного або придбати товар, система пропонує пройти процедуру реєстрації, яку схематично демонструє додаток В. Процес реєстрації передбачає введення особистих даних, таких як: ім’я користувача, електронна пошта та пароль. Після заповнення необхідних даних, здійснюється валідація. Якщо були виявлені помилки, користувач отримає відповідні сповіщення та повернеться на попередній крок. Якщо помилки відсутні система авторизовує користувача. Після авторизації користувач буде мати можливість додати товар до списку бажаного або кошика. Завершальним етапом взаємодії користувача з системою є оформлення замовлення та оплата, які включають передачу даних про замовлення, підтвердження покупки та надходження грошових коштів.

Також після авторизації користувачу відкривається доступ до "Профілю", де користувач може переглядати та змінювати персональну інформацію, моніторити збережені товари, стан кошика, а також історію оформлених замовлень. Реєстрація є необхідною умовою для повноцінної участі у взаємодії з платформою

Щоб продемонструвати можливості програмного модуля з точки зору взаємодії з користувачами, була створена діаграма прецедентів (додаток Г). Основними елементами діаграми є актори, прецеденти та зв’язки між ними. На зображеній діаграмі показано основні сценарії використання функціоналу інтернет-магазину. Вона охоплює дії як з боку звичайного користувача, так і адміністратора системи. Користувач починає взаємодію з перегляду каталогу товарів, або з введення пошукових запитів та фільтрування, після чого користувач отримує результати та має змогу переглядати товари. Залежно від зацікавленості, він може додати товар до списку бажаного або до кошика для подальшого оформлення замовлення. Після завершення покупки існує можливість залишити

відгук щодо товару чи якості обслуговування. Адміністратор, у свою чергу, відповідає за створення товарів, керування користувачами та перегляд статистики, що забезпечує контроль за процесами всередині платформи.

Щоб відобразити часову логіку взаємодій в системі, створено діаграму послідовності (додаток Д). На діаграмі зображено чотири основних об’єкти: адміністратор, програмний модуль, покупець та платіжний сервіс. Процес починається з дій адміністратора, який ініціює створення товару в системі. Після цього програмний модуль обробляє відповідний запит і дозволяє адміністратору опублікувати товар, що робить його доступним для перегляду покупцями.

Покупець розпочинає взаємодію із системою з проходження процедури реєстрації. Далі користувач виконує пошук потрібного оголошення, після чого здійснюється перегляд результатів. У разі зацікавленості конкретним товаром, покупець додає його до кошика. Далі здійснюється оформлення замовлення та оплата покупки, при цьому активується взаємодія з платіжним сервісом.

На фінальному етапі моделювання, для того щоб продемонструвати загальну логіку функціонування інтернет-магазину мобільних телефонів та взаємодію всіх компонентів системи, було побудовано функціональну діаграму IDEF0 (додаток Е). Вона наочно відображає ключові процеси, учасників системи, інформаційні потоки та взаємозв’язки між підсистемами [33].

Процес починається з адміністрування, де адміністратор додає або оновлює дані про мобільні телефони. Ці дані потрапляють у базу даних, з якої надалі передаються для відображення в інтерфейсі користувача. Користувач взаємодіє з системою через веб-інтерфейс, де спочатку відкриває "Каталог", переглядає товари, після чого може перейти до сторінки конкретного товару. За потреби обраний товар можна додати до кошика, після чого відбувається перегляд вмісту кошика й оформлення замовлення. Кожен із функціональних блоків пов’язаний із трьома ключовими компонентами системи — базою даних, веб-інтерфейсом та серверною логікою. Вони відповідають за обробку запитів, збереження та передачу даних між користувачем, адміністратором і системними модулями.

2.2 Інтеграція штучного інтелекту в функціональність інтернет-магазину

Сучасні інтернет-магазини потребують високого рівня персоналізації та зручності для користувача. Саме тому одним із ключових етапів у розробці платформи є інтеграція інструментів штучного інтелекту (ШІ), зокрема для реалізації інтелектуального текстового пошуку та системи персональних рекомендацій.

Першою функцією, що інтегруються у дану систему інтернет-магазину, є інтелектуальний текстовий пошук, що реалізований за допомогою інструментів обробки природної мови. Його основною метою є забезпечення точного, зручного та контекстно обґрунтованого пошуку товарів за ключовими словами.

Процес розпізнавання сутностей зазвичай включає кілька послідовних етапів. Спочатку вхідний текст розбивається на окремі речення (sentence segmentation), а потім кожне речення — на токени (tokenization), тобто окремі слова та розділові знаки. Після цього кожному токену присвоюється частина мови (part-of-speech tagging), що допомагає в подальшому аналізі [26].

Наступним кроком є виявлення потенційних сутностей за допомогою техніки, відомої як "chunking". Це процес, який об'єднує послідовності токенів у більші структурні одиниці, що можуть представляти сутності. Наприклад, послідовність "10 тисяч гривень" може бути розпізнана як грошова сума [26].

Після виявлення сутностей, система може також визначати відносини між ними. Наприклад, у фразі "Смартфон Samsung з 8 ГБ оперативної пам’яті до 10 тисяч гривень" можна встановити, що "Samsung" є брендом, "8 ГБ оперативної пам’яті" — характеристикою, а "до 10 тисяч гривень" — ціновим обмеженням.

Процес пошуку розпочинається з того, що користувач вводить текстовий запит у пошуковий рядок, який є доступним на всіх сторінках магазину. Цей запит, надсилається на сервер для обробки. На сервері працює модуль, що приймає запит та передає його на обробку NLP. Завдяки механізмам аналізу тексту модель визначає, які саме частини тексту відповідають конкретним

атрибутам товарів. Результатом роботи моделі буде структурований список виявлених сутностей, який далі конвертується в об'єкт у форматі JSON. Цей JSON передається в пошуковий модуль, де використовується для формування запиту до бази даних за допомогою ORM-бібліотеки. На цьому етапі система виконує фільтрацію товарів відповідно до знайдених параметрів та повертає результати користувачеві на сторінці результатів пошуку.

Додаток Ж демонструє архітектуру процесу інтелектуального пошуку: від моменту введення запиту користувачем до отримання результатів з бази даних. Ця діаграма ілюструє послідовність етапів, які проходить текст перед тим, як перетворюється на набір параметрів, за якими відбувається пошук товарів.

Наступною функцією, що інтегрується в систему інтернет-магазину є система рекомендацій. В її основі лежить застосування методів машинного навчання, зокрема - векторного представлення об’єктів та використання бібліотеки FAISS [4]. Технічно процес функціонування системи рекомендацій починається ще на етапі створення товару. Коли адміністратор додає новий товар до бази даних, одночасно з цим для нього генерується векторне представлення - embedding. Це числовий вектор, який кодує ключові характеристики товару, такі як бренд, технічні параметри, ціна тощо. Створені вектори зберігаються у спеціальній таблиці бази даних.

На боці користувача система фіксує кожну взаємодію з товаром. Коли користувач переглядає сторінку певного товару, ця інформація зберігається в базі даних в полі переглянутих товарів. Таким чином формується історія переглядів, яка служить основою для генерації рекомендацій. Система не одразу починає формувати персоналізовану стрічку - вона активується після того, як назбирається певна кількість переглядів, достатня для обчислення подібності між товарами.

У момент, коли користувач відкриває сторінку каталогу або конкретного товару, система рекомендацій виконує пошук схожих позицій на основі історії переглядів. Далі, на сервері, викликається спеціальний метод, який отримує всі наявні векторні представлення товарів з бази даних, а також відповідні вектори

товарів, які користувач вже переглядав. Бібліотека FAISS обчислює схожість між векторами переглянутих товарів і всіма іншими.

Результатом цього процесу є впорядкований список товарів - від найбільш схожих до найменш схожих. Цей список передається на фронтенд, де відображається у вигляді блоку "Рекомендовані товари" або інтегрується безпосередньо в загальний список товарів, формуючи персоналізовану стрічку. Таким чином, користувач отримує релевантні рекомендації на основі власних уподобань. На додатку И подано діаграму послідовності, яка демонструє, як реалізовано взаємодію між компонентами системи для формування рекомендацій на основі перегляду товарів.

Висновки до розділу 2

У процесі проєктування інтернет-магазину мобільних телефонів було створено чітку архітектуру системи, що охоплює клієнтську частину, серверну логіку, базу даних і допоміжні модулі. За допомогою UML-діаграм вдалося візуалізувати основні бізнес-процеси платформи: від перегляду каталогу й реєстрації до оформлення замовлень і оплати.

Важливим доповненням стала інтеграція інструментів штучного інтелекту. Було розроблено алгоритми інтелектуального пошук із використанням NLP. Система буде здатна інтерпретувати запити, виділяти ключові сутності, трансформувати їх у структуровані критерії пошуку та формувати відповідні запити до бази даних.

Крім того, був створений проект системи персоналізованих рекомендацій на основі векторного представлення товарів і технології FAISS. Система основана на тому, що всі товари будуть мати вектори з урахуванням їхніх характеристик, а потім збережені у векторному індексі. Це дасть змогу швидко знаходити подібні товари на основі історії переглядів користувача, формуючи динамічні списки рекомендацій.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНТЕРНЕТ-МАГАЗИНУ МОБІЛЬНИХ ТЕЛЕФОНІВ З ВИКОРИСТАННЯМ ШТУЧНОГО ІНТЕЛЕКТУ

3.1 Обрані інструменти розробки та процес технічної реалізації

У процесі розробки інформаційної системи інтернет-магазину мобільних телефонів було прийнято рішення використовувати клієнт-серверну архітектуру. Такий підхід дає змогу розділити логіку додатка на дві незалежні частини — клієнтську (інтерфейс користувача) та серверну (обробка запитів, взаємодія з базою даних та моделями ШІ). Це забезпечує масштабованість, гнучкість у підтримці системи, а також дозволяє оптимізувати навантаження на сервер і зменшити час відгуку для користувача.

Розробка клієнтської частини інформаційної системи інтернет-магазину мобільних телефонів розпочалася зі створення зручного, інтуїтивного та адаптивного інтерфейсу. За основу було взято сучасний фреймворк Next.js [15], який забезпечує серверний рендеринг, оптимізовану маршрутизацію та чудово підходить для SEO [3]. Структуру проєкту наведено в додатку К.

Проєктування інтерфейсу почалося в сервісі Figma, де були створені макети основних сторінок: головної, списку товарів, картки товару, кошика, оформлення замовлення, авторизації, кабінету користувача та адмін-панелі. Після цього розпочалася розробка інтерфейсних компонентів [22].

Для побудови інтерфейсу було використано бібліотеку shadcn-ui, яка надає набір готових до використання, але повністю кастомізованих компонентів [1]. Це дозволило уникнути перевантаження інтерфейсу зайвими стилями, зберігаючи водночас естетику та функціональність. Усі компоненти було стилізовано за допомогою Tailwind CSS — утилітарного фреймворку для стилізації, що дозволяє швидко створювати адаптивні інтерфейси без необхідності написання окремих CSS-файлів [20]. Tailwind чудово інтегрується з Next.js і забезпечує повний контроль над зовнішнім виглядом елементів.

Управління станом у клієнтському додатку реалізовано за допомогою Zustand — мінімалістичної, але дуже потужної бібліотеки керування станом, яка не вимагає великої кількості шаблонного коду, як-от Redux [23]. Наприклад, стан кошика, користувача, списку товарів чи повідомлень зберігається у глобальному сторі та може бути використаний у будь-якому компоненті без зайвих залежностей.

Обмін даними з сервером відбувається через GraphQL, що дозволяє надсилати лише ті дані, які потрібні на конкретній сторінці [6]. Для інтеграції GraphQL у клієнтську частину використовувався Apollo Client, який автоматично кешує результати запитів та забезпечує реактивність інтерфейсу. Наприклад, при додаванні товару в кошик або оновленні замовлення, Apollo миттєво оновлює дані без потреби ручного перезавантаження компонентів (додаток Л).

На головній сторінці виводиться список популярних товарів, які отримуються через GraphQL-запит до відповідного резолвера на сервері. На сторінці каталогу виводиться список всіх товарів, наявних в інтернет-магазині. Користувач може фільтрувати товари за категорією, ціною та технічними характеристиками — ці фільтри передаються у вигляді змінних до GraphQL-запиту, а сервер повертає лише релевантні результати. Ця логіка реалізована у вигляді окремого компонента CatalogFilters (додаток М), який використовує zustand для зберігання активних фільтрів.

Сторінка окремого товару включає слайдер із зображеннями, блок відгуків, можливість додати товар до кошика, а також блок із рекомендованими товарами, що реалізовано за допомогою окремого запиту до системи рекомендацій.

Функціонал кошика винесено в окремий стор (додаток Н) — при додаванні товару до кошика компонент CartButton викликає відповідну мутацію GraphQL, а у zustand-сховищі оновлюється локальний стан. Завдяки цьому, користувач одразу бачить зміну кількості товарів у кошику, навіть без перезавантаження сторінки. Усі зміни в кошику синхронізуються із сервером, що дозволяє зберігати його вміст навіть після виходу з системи або зміни пристрою.

Щоб додати товар до кошику користувач має бути авторизований. Якщо ні — йому пропонується пройти процес авторизації (додаток П). Після авторизації користувач може підтвердити замовлення, і воно створюється на сервері за допомогою мутації createOrder. Після цього користувач отримує повідомлення про успішне оформлення.

Авторизація реалізована за допомогою сесій — після логіну запит надсилається до відповідного GraphQL-резолвера, і при успішній відповіді сесія зберігається, а клієнт отримує доступ до закритих сторінок, таких як кабінет користувача чи адмін-панель. На клієнтському боці стан автентифікації зберігається у zustand (додаток Р), а дані користувача витягуються з окремого GraphQL-запиту, для зручності роботи з яким створено окремий хук (додаток С).

У кабінеті користувача реалізовано перегляд замовлень, відгуків і можливість змінити особисті дані. Окремо реалізована функція управління активними сесіями користувача. Адміністративна панель дає змогу додавати нові товари, редагувати існуючі, змінювати статуси замовлень та керувати категоріями. Усі ці дії реалізовані через відповідні мутації GraphQL та забезпечені захистом на рівні клієнтської логіки.

Кожна сторінка додатку розроблялася як окремий компонент з власним станом і стилями. У процесі розробки активно використовувався підхід code splitting, що дозволило значно зменшити розмір початкового пакету і пришвидшити завантаження сторінок.

На серверну частину інтернет-магазину було покладено наступні функції: обробка запитів, взаємодія з базою даних та моделями ШІ, авторизація та збереження сесій, завантаження файлів і надання API для клієнтської частини. Тому для реалізації серверу в якості основного фреймворка було обрано NestJS [14]. Це сучасний Node.js-фреймворк, що побудований на принципах модульності, контролерів і сервісів. NestJS побудований на базі Express, але забезпечує чітку структуру, зручну підтримку декораторів та інтеграцію з GraphQL [6], що робить його ідеальним вибором для великих та масштабованих застосунків. Для зручності та масштабованості проєкт побудовано з дотриманням

принципів модульності, які пропонує NestJS. Основні директорії мають структуру, що наведена в додатку Т. При цьому кожен модуль має свою власну структуру (додаток У).

Розробку було вирішено почати з визначення основних сутностей: користувач, товар, замовлення, відгук, кошик та збережений товар. Для зберігання даних було обрано PostgreSQL — реляційну базу даних, яка ідеально підходить для структурованої інформації, підтримує складні зв'язки та транзакції [16]. Схема розробленої бази даних наведена в додатку Ф. Для взаємодії з нею використовувалась ORM Prisma, що дозволяє описувати моделі у вигляді декларативної схеми, генерувати типи для TypeScript і виконувати міграції бази даних. Також на етапі налагодження використовувалась Prisma Studio, для перегляду і редагування вмісту таблиць.

Після побудови схеми бази даних було реалізовано GraphQL API. GraphQL дає змогу клієнту самостійно визначати, які саме поля необхідні в кожному запиті, що особливо зручно при роботі зі складно вкладеними структурами. NestJS має офіційний модуль @nestjs/graphql, що дозволяє інтегрувати GraphQL із мінімальними зусиллями. Для кожної сутності було створено окремі резолвери, input-класи та типи.

Наприклад, у ProductResolver (додаток Х) реалізовано запити для отримання списку товарів із фільтрацією за технічними характеристиками, назвою або діапазоном цін. Також реалізовані мутації для створення, оновлення та видалення товарів. У межах цих мутацій викликаються відповідні сервіси, які, у свою чергу, працюють із Prisma.

Окрему увагу було приділено механізму авторизації. Для цього створено модуль автентифікації, який базується на сесіях. Після успішного входу в систему дані про користувача зберігаються через бібліотеку express-session, а сесії кешуються в Redis. Redis було обрано як високошвидкісне in-memory сховище, яке ідеально підходить для зберігання сесій у масштабованих застосунках. NestJS дозволяє легко інтегрувати сесійну авторизацію, а також використовувати власні guard-и для обмеження доступу до певних API залежно від ролі користувача [36].

Завдяки такому підходу, резолвери, які відповідають за керування товарами, замовленнями чи категоріями, обмежуються лише для авторизованих адміністраторів. Наприклад, створення товару можливо лише після перевірки сесії користувача та його ролі.

Далі було розроблено функціонал для завантаження зображень товарів та аватарів користувачів. Для цього було налаштовано FileService (додаток Ц), який приймає multipart/form-data запити за допомогою Multer, зберігає файли на сервері у відповідну директорію (uploads/products або uploads/users) та повертає URL до файлу, який потім зберігається у базі даних.

Для того, щоб користувачі інтернет-магазину могли робити замовлення, було розроблено відповідний модуль, що реалізовує повний цикл роботи з покупкою: від створення замовлення до зміни його статусів. Контроллер paymentController приймає список товарів з кошика, id покупця та загальну суму. Далі передає ці дані до сервісу PaymentService, в якому є метод createPayment (додаток Ш), що налаштований на інтеграцію з платіжною системою Fondy. Після успішної оплати Fondy іповертає відповідь, яка обробляється іншим методом, confirmPayment (додаток Щ), який створює відповідний запис про замовлення у базі даних. Роль адміністратора дозволяє переглядати усі замовлення. Для демонстрації роботи автоматизації також було реалізовано cron-задачу, яка щогодини змінює статус усіх замовлень зі статусом "Оплачено" на "Доставлено", що імітує роботу логістичної служби.

Додатково реалізована можливість для користувачів залишати відгуки та оцінки для товарів. Для цього був створений відповідний модуль. Відгук може створити лише авторизований користувач.

Також був реалізований модуль кошика та збережених товарів. Ці дані зберігаються на сервері у зв’язку з користувачем, що дозволяє не втрачати вміст навіть при зміні пристрою або перезавантаженні сторінки. У кошику користувач може збільшувати або зменшувати кількість одиниць товару, видаляти позиції або переходити до оформлення замовлення.

Пошукова система та система рекомендацій були розроблені з метою зробити користувацький досвід у інтернет-магазині максимально інтуїтивним та персоналізованим. Особливу увагу приділено саме обробці природної мови користувача та пошуку за змістом, а не лише за жорстко заданими параметрами. Цей функціонал було реалізовано за допомогою сучасних методів обробки тексту та векторного представлення даних.

Робота над інтелектуальним пошуком почалася з побудови спеціалізованої моделі для виявлення сутностей у текстових запитах. Для цього було використано бібліотеку spaCy, яка є однією з найпопулярніших для задач обробки природної мови. Як базову модель було обрано uk_core_news_sm - попередньо натреновану мовну модель для української мови, яка забезпечує високу якість токенізації, лематизації та морфологічного аналізу.

Проте базової функціональності моделі було недостатньо для виявлення спеціалізованих сутностей, характерних саме для мобільних телефонів. Тому було прийнято рішення донавчити модель. Для цього спочатку було реалізовано Python-алгоритм генерації прикладів можливих пошукових запитів користувачів (додаток Ю). Генератор враховував типові запити на кшталт "айфон до 20 тисяч", "андроїд з гарною камерою", "дешевий самсунг чорного кольору" тощо.

Завдяки цьому підходу було згенеровано датасет із 100 000 прикладів текстових запитів, кожен із яких мав чітко позначені сутності, необхідні для фільтрації товарів у базі даних. Зразок такого прикладу з розміткою наведено в додатку Я.

Після генерації датасету було налаштовано середовище Google Colab, додаток АА, де встановлено бібліотеку spaCy, завантажено початкову модель uk_core_news_sm і розпочато процес донавчання моделі на зібраних даних. Завдяки великій кількості якісних прикладів, модель успішно навчилась розпізнавати необхідні сутності у вхідних запитах.

Навчену модель було завантажено локально в проєкт — безпосередньо до директорії з бекендом. Далі було реалізовано окремий Python-модуль, який відповідає за обробку запитів (додаток АБ): він завантажує модель, приймає текст

на вхід, проводить розпізнавання сутностей і повертає список знайдених параметрів.

На стороні NestJS було створено окремий сервіс NLP, який приймає текстовий запит користувача, надсилає його до Python-скрипту, за допомогою бібліотеки python-shell, отримує у відповідь список сутностей (додаток АВ) і передає його іншій функції для трансформації у формат, придатний для фільтрації. Цей формат — це JSON-об’єкт, який далі використовується для побудови запиту до бази даних через Prisma.

Цей процес забезпечує надзвичайно гнучкий пошук: користувач вводить природну мову, система автоматично визначає наміри та повертає лише релевантні результати, що значно покращує користувацький досвід порівняно зі стандартною формою пошуку.

Паралельно було реалізовано систему рекомендацій, яка базується на векторному представленні тексту. Реалізація системи рекомендацій відбувалася поетапно з урахуванням вимог до точності, масштабованості та ефективності.

Першим етапом стало визначення моделі ШІ, здатної ефективно генерувати змістовні векторні представлення товарів. Було обрано модель bge-small-en-v1.5 з родини sentence-transformers, оптимізованої для української мови [8]. Ця модель здатна враховувати контекст товару, що надає можливість будувати глибше семантичне представлення кожного товару.

Наступним кроком було створення Python-скрипта (додаток АГ), який отримує текстовий опис товару, обробляє його за допомогою обраної моделі та повертає векторне представлення у вигляді списку чисел фіксованої довжини.

Далі реалізовано NestJS-сервіс (додаток АД), який інтегрується з Python-скриптом за допомогою бібліотеки python-shell. При створенні нового товару адміністратором, NestJS формує запит до Python-скрипта, передаючи опис товару. У відповідь він отримує векторне представлення товару. Отриманий вектор зберігається в окремій таблиці бази даних, прив’язаній до відповідного товару.

Далі був створений Python-скрипт (додаток АЕ), який відповідає за обчислення схожості між товарами. Він приймає два набори даних: векторні

представлення товарів, які переглядав користувач, та вектори всіх товарів з бази даних. Далі, з використанням FAISS, виконується швидкий пошук найближчих за змістом товарів, які повертаються у відсортованому порядку — від найбільш схожого до найменш схожого. Для інтеграції з цим скриптом реалізовано окремий NestJS-сервіс (додаток АД), який отримує ідентифікатор користувача, за допомогою якого, через Prisma ORM, отримує історію його переглядів і список всіх товарів з їхніми ембедінгами, після чого формує запит до Python-скрипта. У відповідь отримується впорядкований список рекомендованих товарів.

На завершальному етапі релевантний список товарів повертається на клієнт через GraphQL API. Коли користувач відкриває каталог товарів, на сервер надсилається запит на побудову рекомендацій на основі його попередніх взаємодій. Отримані товари динамічно відображаються на сторінці.

3.2 Інтерфейс системи та його функціональні можливості

Для забезпечення зручного та ефективного використання створеного інтернет-магазину мобільних телефонів, було розроблено зручний графічний інтерфейс, що відповідає сучасним вимогам до онлайн-комерції. Після відкриття головної сторінки користувача зустрічає привабливий і простий у навігації інтерфейс, який дозволяє одразу перейти до пошуку товарів, перегляду каталогу або авторизації в особистому кабінеті. Інтерфейс головної сторінки показано в додатку АЖ.

Залежно від ролі користувача, система пропонує різний набір функцій. Звичайні користувачі можуть переглядати каталог мобільних телефонів, здійснювати пошук за допомогою інтелектуальної системи, додавати товари до кошика, оформлювати замовлення та переглядати їхній статус. Адміністратори системи, після авторизації (додаток АИ), отримують доступ до панелі управління, де можуть додавати нові товари, керувати замовленнями, переглядати статистику продажів, а також редагувати опис товарів і їх характеристики.

Однією з ключових функціональних можливостей системи є інтелектуальний пошук товарів. Інтерфейс пошуку (додаток АК) побудований таким чином, щоб користувач міг вводити запити у звичайній розмовній. Система автоматично обробляє та розпізнає сутності в запиті та показує користувачу найбільш релевантні дані. Додатково, користувачеві надаються підказки з прикладами того, як можна сформулювати запит.

Повний список товарів з можливістю сортування за ціною, рейтингом, популярністю та іншими критеріями присутній на сторінці каталогу. Інтерфейс сторінки (додаток АЛ) надає детальну інформацію про кожен пристрій, включаючи ціну, технічні характеристики, фотографії, та кнопки для збереження або додавання в кошик.

Функція системи рекомендацій реалізована безпосередньо на сторінці товару. Після перегляду пристрою, запис про це заноситься до бази даних. З часом, коли список переглянутих товарів буде достатнім для формування рекомендацій, користувач почне бачити товари в порядку релевантності, від найбільш схожих на раніше переглянуті товари до менш схожих.

Особистий кабінет користувача містить розділи для перегляду історії замовлень (додаток АН), зміни особистих даних (додаток АМ), корзини (додаток АП) та збережених товарів (додаток АР). Всі сторінки адаптовані для мобільних пристроїв, що дозволяє користувачу зручно здійснювати покупки з будь-якого пристрою.

Для адміністратора реалізовано окремий інтерфейс управління контентом (додаток АС). Тут передбачено можливість додавання нових моделей телефонів або редагування вже наявних. Також у панелі адміністратора доступна аналітика замовлень (додаток АТ), що дозволяє оцінювати ефективність продажів, виявляти найпопулярніші товари та контролювати процес обробки замовлень.

Для поліпшення взаємодії з користувачем було реалізовано сповіщення - наприклад, при успішному оформленні замовлення, додавання товару до корзини або виникненні помилки. Усі сповіщення реалізовано у вигляді ненав’язливих

вікон, що не переривають роботу користувача, але своєчасно інформують його про зміни.

Крім того, було реалізовано повну адаптацію інтерфейсу для мобільних телефонів та планшетів. Додатково впроваджено можливість перемикання між світлою та темною темою оформлення. Також реалізовано підтримку багатомовності — інтерфейс доступний українською та англійською мовами.

Висновки до розділу 3

У розділі детально розглянуто процес програмної реалізації інтернет-магазину мобільних телефонів із вбудованими елементами штучного інтелекту. Розробка системи виконувалась у межах клієнт-серверної архітектури, що забезпечує гнучкість, масштабованість та легку підтримку проєкту. Реалізація базується на архітектурі клієнт-сервер із використанням Next.js, Tailwind CSS, Zustand та GraphQL для фронтенду, а також NestJS, Prisma і PostgreSQL для бекенду

Особливу увагу приділено впровадженню інтелектуального пошуку та системи рекомендацій. Розроблено систему, яка з використанням spaCy і донавченої моделі розпізнає сутності у запитах користувача й перетворює їх на параметри фільтрації. Рекомендації реалізовано шляхом векторного порівняння товарів за допомогою FAISS. Товарні дані перетворюються на embedding-и, що зберігаються у векторному індексі. Пошук здійснюється на основі семантичної близькості, що дозволяє знаходити релевантні рекомендації навіть без точних збігів у назві чи категорії.

Інтерфейс користувача реалізовано відповідно до сучасних стандартів e-commerce. Особливу увагу приділено темізації, логічній структурі розміщення елементів, швидкій навігації та інтуїтивній взаємодії з користувачем. Інтерфейс включає головну сторінку з категоріями, каталог, систему фільтрів і сортування,

сторінки товару з характеристиками, систему оформлення замовлення, авторизацію, а також профіль користувача та сторінку адміністратора.

ВИСНОВОК

У результаті виконання дипломної роботи було створено веб-систему інтернет-магазину мобільних телефонів із підтримкою інтелектуальних функцій, що відповідає поставленій меті та сформульованим завданням дослідження. Реалізоване рішення забезпечує зручну взаємодію користувача з системою, високу швидкодію та адаптивність.

У процесі роботи досягнуто наступних результатів:

– Проведено огляд існуючих підходів до реалізації систем пошуку та рекомендацій, що дало змогу визначити ефективні інструменти для впровадження інтелектуальних систем в архітектуру веб-додатка.

– Розроблено загальну архітектуру інформаційної системи, яка охоплює основні модулі та забезпечує ефективну взаємодію між ними.

– Реалізовано базову функціональність інтернет-магазину, яка включає авторизацію та реєстрацію користувачів, каталогізацію товарів, пошук, додавання товарів у кошик, оформлення замовлення, а також адміністративну панель для управління контентом.

– Розроблено систему рекомендацій на основі векторного подання товарів, за допомогою бібліотеки FAISS для швидкого пошуку подібностей у векторному просторі. Також розроблено систему інтелектуального пошуку, на основі обробки природної мови. Для цього застосовано модель обробки природної мови spaCy, донавчену на спеціалізованому наборі даних.

– Створено адаптивний та інтуїтивно зрозумілий інтерфейс, який забезпечує зручну взаємодію користувача з усіма основними можливостями системи.

Практична цінність реалізованої розробки полягає в можливості її безпосереднього використання для автоматизації електронної торгівлі мобільними телефонами, з урахуванням особливостей ринку та потреб користувачів. Вбудований модуль NLP-пошуку та системи рекомендацій

підвищують рівень персоналізованої взаємодії, сприяючи зростанню лояльності клієнтів та збільшенню конверсій.

Система, створена в межах дослідження, вже є функціонально завершеною і здатною до розгортання в реальних умовах. Водночас, перспективними напрямами подальшого розвитку є інтеграція рекомендаційного модуля на основі історії переглядів і дій користувача, а також розширення можливостей системи аналітики на основі пошукових запитів.

Узагальнюючи результати, можна стверджувати, що розроблена система повністю відповідає сучасним вимогам до електронної комерції та демонструє ефективність впровадження технологій обробки природної мови в інтерфейс пошуку товарів. Отримані результати можуть бути масштабовані й адаптовані для інших сфер онлайн-продажу, де ключовим є швидкий і точний пошук за текстовими запитами.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Build your component library – shadcn/ui: веб-сайт. URL: https://ui.shadcn.com/ (дата звернення: 14.02.2025).
2. Codesido I. What is front-end development? веб-сайт. URL: https://www.theguardian.com/help/insideguardian/2009/sep/28/blogpost. (дата звернення: 11.01.2025);
3. Difference Between Static and Dynamic Web Pages. веб-сайт. URL: https://techdifferences.com/difference-between-static-and-dynamic-web-pages.html (дата звернення: 10.02.2025);
4. FAISS (Facebook AI Similarity Search). Веб-сайт. URL: https://github.com/facebookresearch/faiss (дата звернення: 03.02.2025).
5. GitHub веб-сайт. URL: https://github.com/ (дата звернення: 10.02.2025);
6. GraphQL | A query language for your API: веб-сайт. URL: https://graphql.org (дата звернення: 20.02.2025)
7. Honnibal, M., & Montani, I. (2017). spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. Веб-сайт. URL: https://spacy.io (дата звернення: 04.01.2025).
8. Hugging Face – Transformers, Datasets, and NLP Models. Веб-сайт. URL: https://huggingface.co (дата звернення: 18.01.2025).
9. Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. IEEE Transactions on Big Data. Веб-сайт. URL: https://arxiv.org/abs/1702.08734 (дата звернення: 01.02.2025).
10. Jurafsky, D., & Martin, J. H. (2021). Speech and Language Processing (3rd ed.). Draft. Веб-сайт. URL: https://web.stanford.edu/~jurafsky/slp3/ (дата звернення: 03.02.2025).
11. Lodash Documentation. Веб-сайт. URL: https://lodash.com/docs/4.17.15 (дата звернення: 12.03.2024).
12. Manning, C. D., Raghavan, P., & Schütze, H. (2018). Introduction to Information Retrieval. Cambridge University Press. Веб-сайт. URL: https://nlp.stanford.edu/IR-

book/ (дата звернення: 27.02.2025).

13. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. Веб-сайт. URL: https://arxiv.org/abs/1301.3781 (дата звернення: 22.02.2025).

14. NestJS - A progressive Node.js framework. Веб-сайт. URL: https://nestjs.com/ (дата звернення: 11.03.2025).

15. Next.js by Vercel - The React Framework: веб-сайт. URL: https://nextjs.org/ (дата звернення: 27.02.2025).

16. PostgreSQL: The World’s Most Advanced Open Source Relational Database веб-сайт. URL: https://www.elephantsql.com/ (дата звернення: 12.03.2025);

17. scikit-learn: Machine Learning in Python. Веб-сайт. URL: https://scikit-learn.org/stable/ (дата звернення: 19.02.2025).

18. spaCy: Industrial-Strength Natural Language Processing in Python. Веб-сайт. URL: https://spacy.io3781 (дата звернення: 22.02.2025).

19. Stack Overflow - Where Developers Learn, Share & Build Careers. URL: https://stackoverflow.com/ (дата звернення: 26.01.2025);

20. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML: веб-сайт. URL: https://tailwindcss.com/ (дата звернення: 14.02.2025).

21. TypeScript is JavaScript with syntax for types. Веб-сайт. URL: https://www.typescriptlang.org/ (дата звернення: 03.02.2025).

22. User Experience (UX): Process and Methodology. URL: https://uiuxtrend.com/user-experience-ux-process/ (дата звернення: 16.01.2025);

23. Zustand Docs: веб-сайт. URL: https://zustand.docs.pmnd.rs/ (дата звернення: 20.02.2025)

24. Анісімов А. В. Інформаційні системи та бази даних: навч. посіб. Для студентів факультету комп’ютерних наук та кібернетики. Київ. 2023. – 110 с.

25. Бази даних веб-сайт. URL: https://shpora.me/Pedro/database-520 (дата звернення: 11.03.2025);

26. Гапон М. І., Колісник М. В. Використання штучного інтелекту для аналізу природної мови в інформаційних системах. Сучасні проблеми моделювання,

прогнозування та оптимізації : зб. наук. пр. – Львів : ЛНУ ім. Івана Франка, 2023. – С. 91–97.

27. Гапон М. І., Колісник М. В. Інтелектуальний аналіз даних у завданнях машинного навчання. Львів : ЛНУ ім. Івана Франка, 2020. 244 с.

28. Демченко А. В. Побудова систем рекомендацій на основі векторного представлення даних. Системи управління, навігації та зв'язку. – 2022. – Вип. 4 (70). – С. 45–51.

29. Інститут штучного інтелекту НАН України: веб-сайт. URL: https://www.iai.dn.ua (дата звернення: 10.02.2025).

30. Коннолли Т. Бегг К. Бази данних. Проектування, реалізація та супроводження. 3 видання: Пер. С англ.: Видавничий дім «Вільямс», 2019. 844с.

31. Краус К.М., Краус Н.М., Манжура О.В. К 78 Електронна комерція та Інтернет торгівля: навчально-методичний посібник. – Київ: Аграр Медіа Груп, 2021. – 454 с.

32. Морзе Н.В. Інформаційні системи. Навч. посіб. Івано-Франківськ, «ЛілеяНВ», – 2022. – 384 с.

33. Навчальний посібник з UML – вивчення уніфікованої діаграми мови моделювання. веб-сайт. URL: https://www.guru99.com/uk/uml-tutorial.html (дата звернення: 15.01.2025).

34. Основи розробки баз даних. веб-сайт. URL: https://support.microsoft.com/uk-ua/office/основи-розробки-баз-даних-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5 (дата звернення: 17.02.2025);

35. Основні етапи створення сайту. веб-сайт. URL: https://pbb.lviv.ua/statti-inovyny/statti-shchodo-stvorennia-saitu/osnovni-etapy-stvorennia-saitu (дата звернення: 26.01.2025);

36. Павлиш В. А., Гліненко Л. К. Основи інформаційних технологій і систем: Навч. посіб. Львів: Видавництво Львівської політехніки, 2021. − 500 с.

37. Скибицький О. М. Методи обробки природної мови в інформаційних системах. Наукові записки Університету "КРОК". 2019. № 3(55). С. 34–40.

38. Степаненко В. О. Практичне застосування технологій машинного навчання в інформаційних системах. Інформаційні технології і системи. – 2023. – № 1 (31). – С. 22–30.

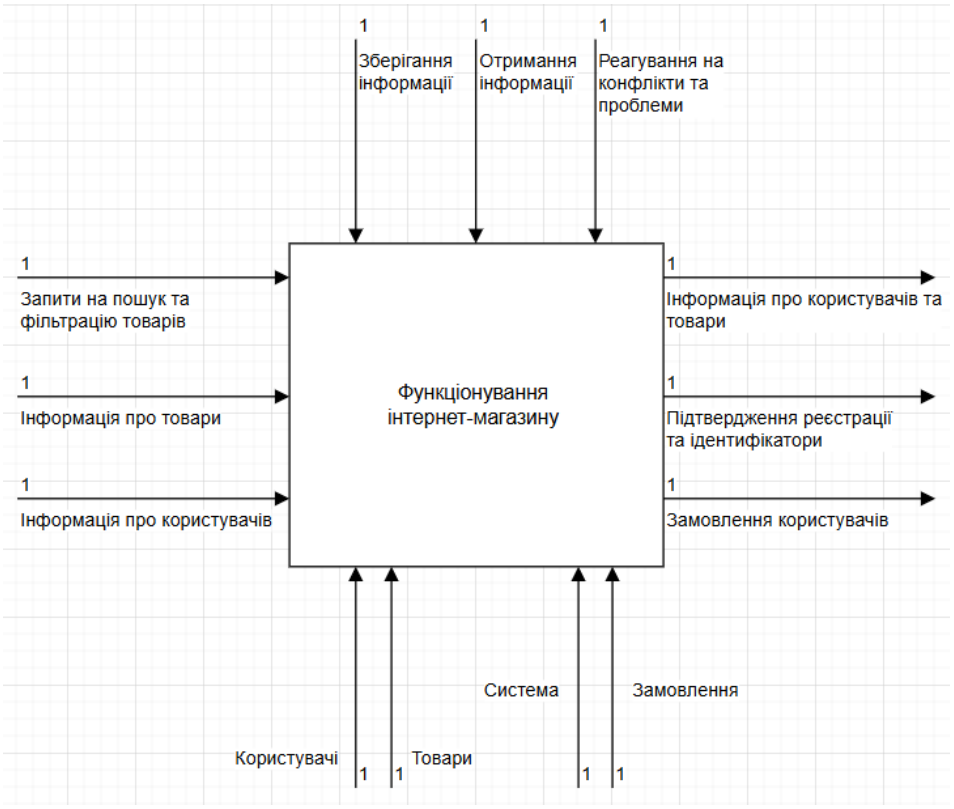
39. Шевченко О. А. Основи побудови систем рекомендацій з використанням штучного інтелекту. Сучасні інформаційні технології та комп’ютерні системи. 2021. № 3. С. 56–62.

40. Шило С. Г. Щербак Г.В. Огурцова К.В. Інформаційні системи та технології: навч. посіб. Вид. ХНЕУ, 2021. – 220 с.

ДОДАТКИ

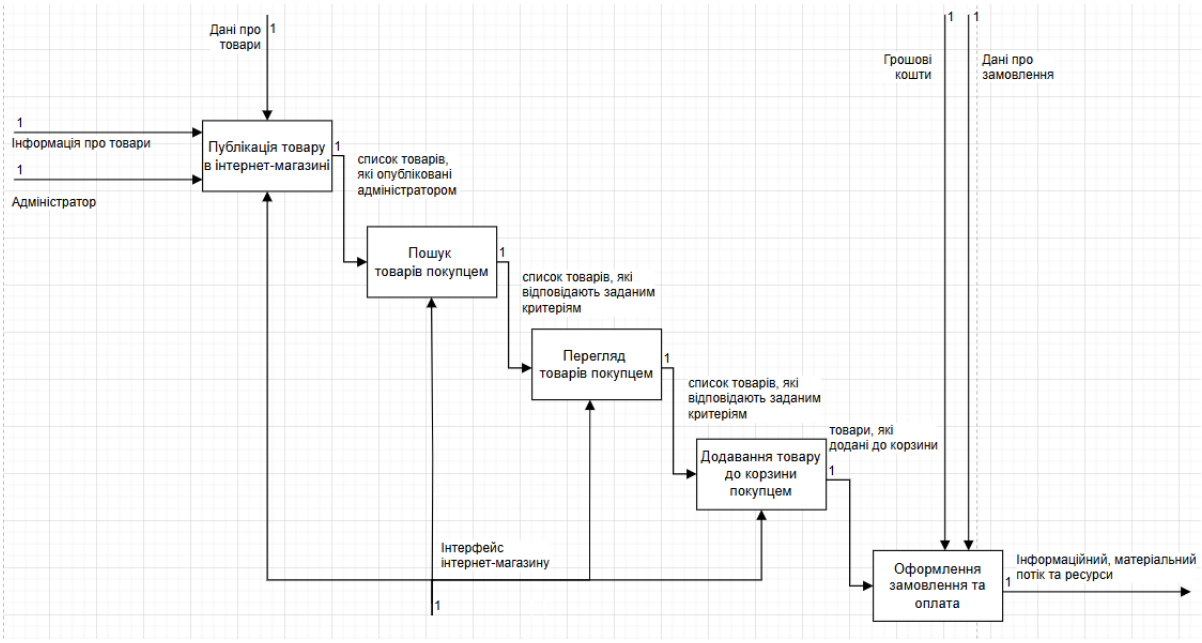
ДОДАТОК А

Основна структурно-функціональна діаграма IDEF0



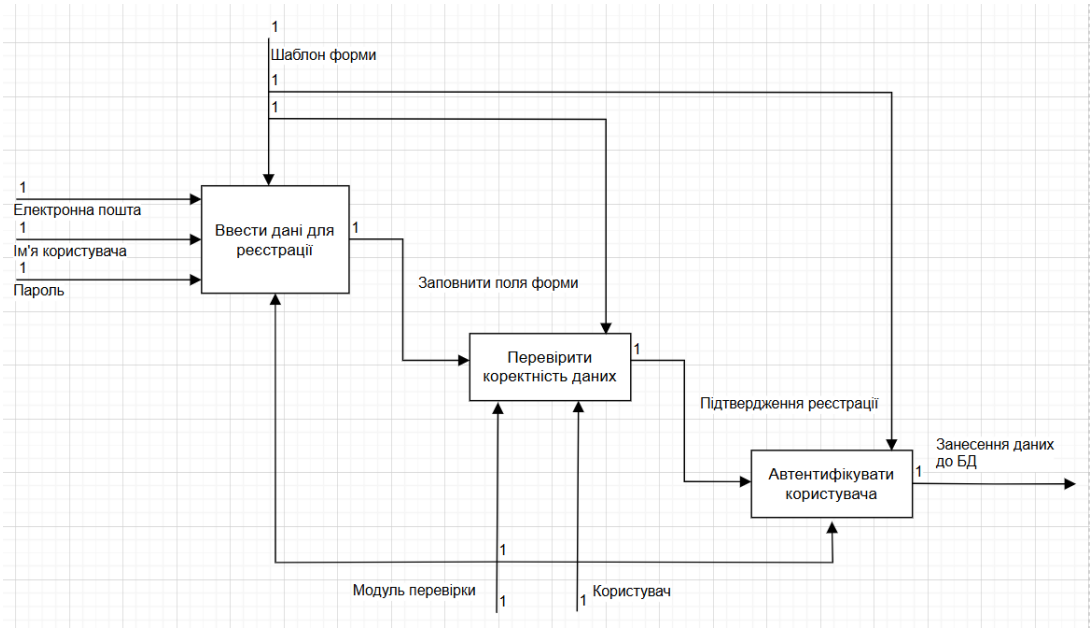
ДОДАТОК Б

Контекстна діаграма А1 – процес взаємодії користувачів із системою



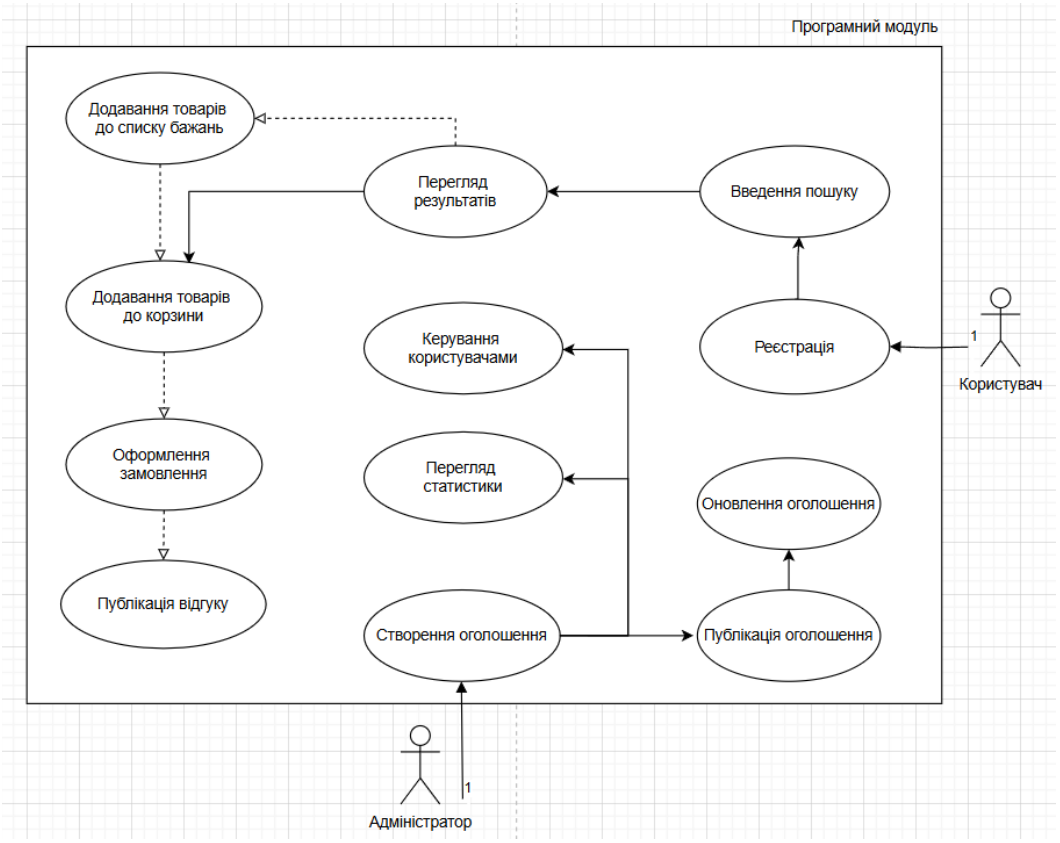
ДОДАТОК В

Контекстна діаграма А1 – процес реєстрації користувачів



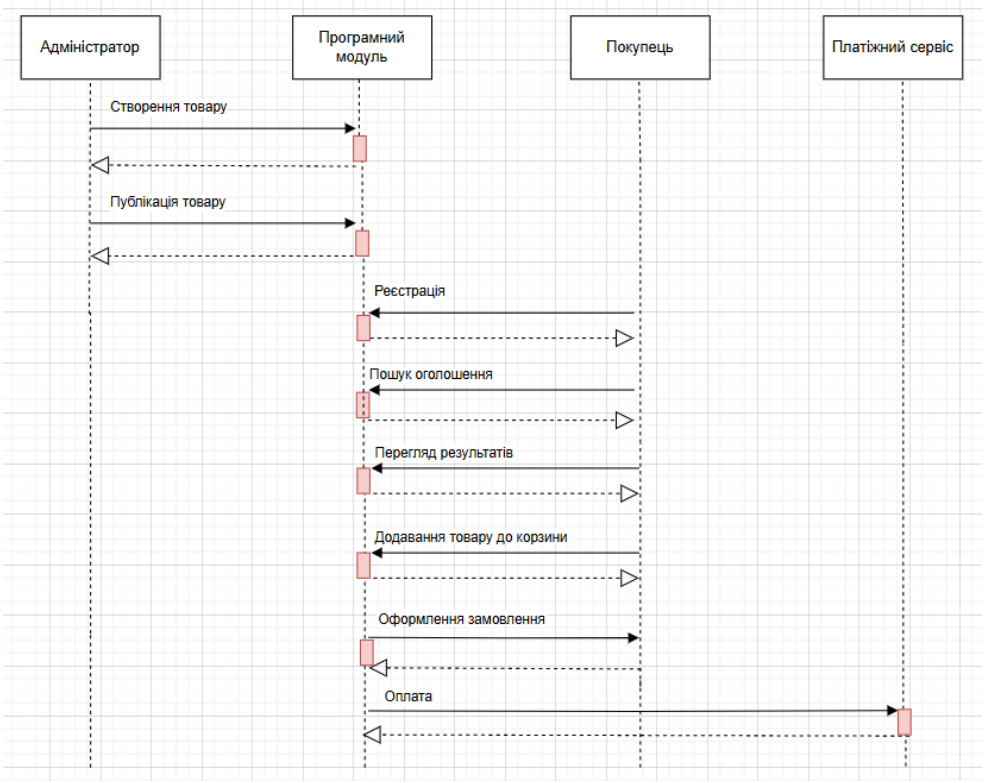
ДОДАТОК Г

UML діаграма прецендентів



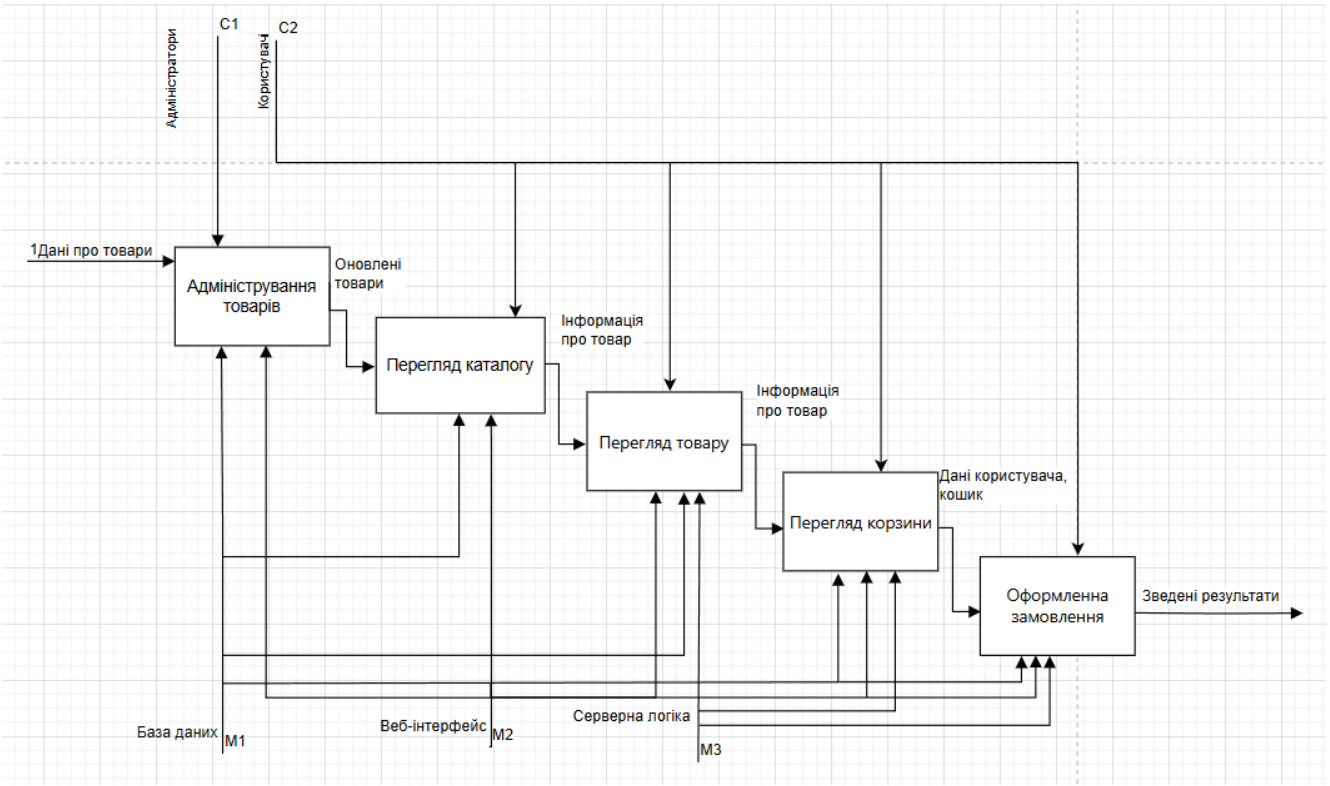
ДОДАТОК Д

Діаграма послідовності



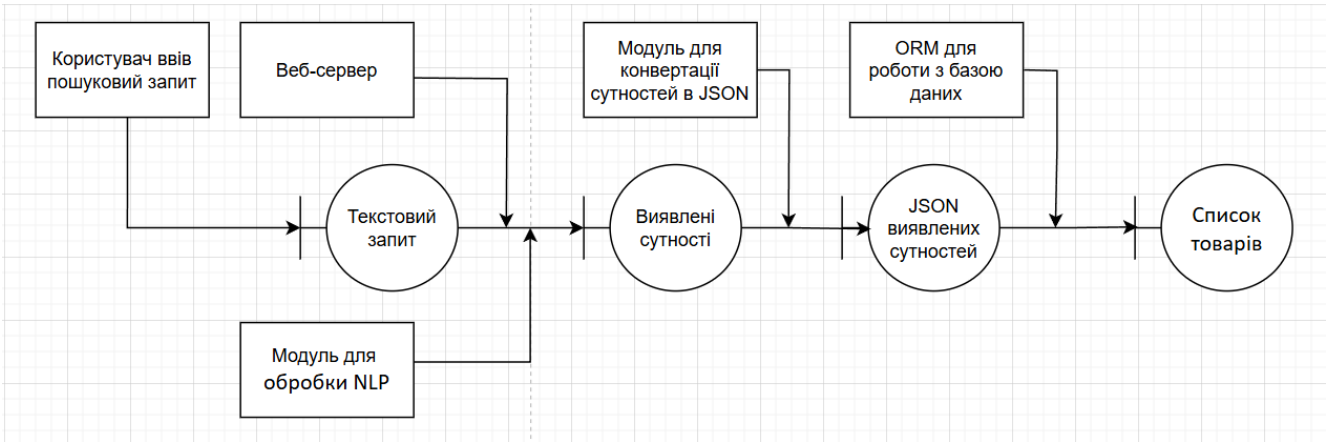
ДОДАТОК Е

Функціональна діаграма IDEF0



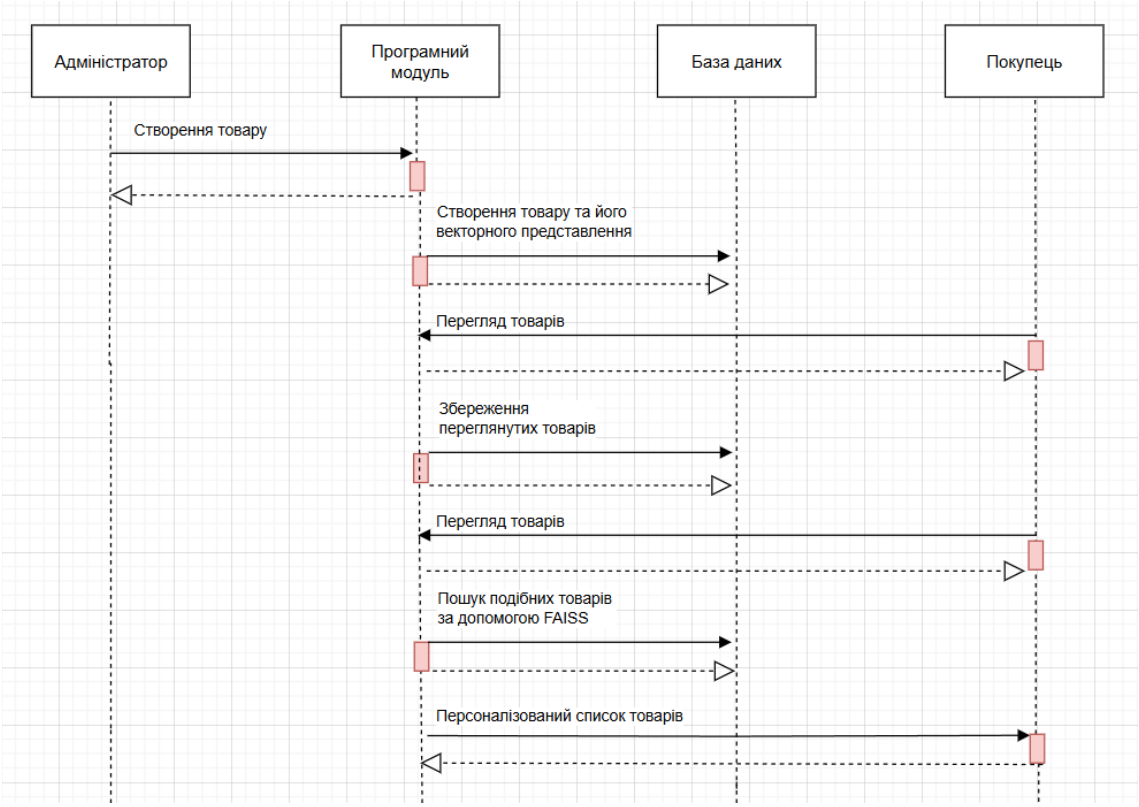
ДОДАТОК Ж

OSTN діаграма інтелектуального пошуку товарів



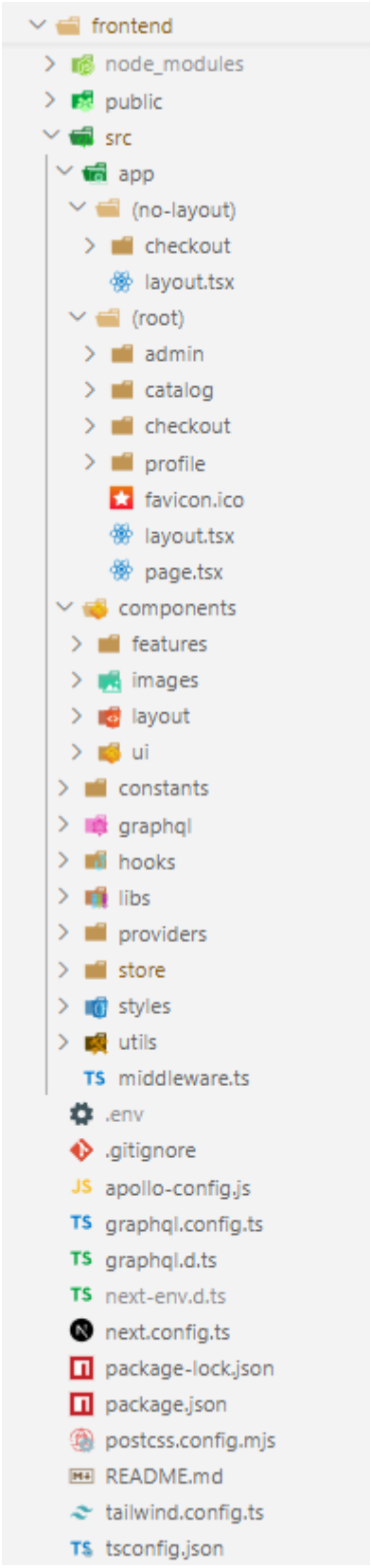
ДОДАТОК И

Діаграма послідовності взаємодій між компонентами системи для формування рекомендацій на основі перегляду товарів.



ДОДАТОК К

Структура фронтенду



ДОДАТОК Л

Використання методу refetch() для автоматичного оновлення даних

```
const [addToCart, { loading: isCartLoading }] = useToggleCartMutation({
  onCompleted() {
    refetch();
    toast.success("Товар було додано до вашої корзини");
  },
  onError(error, clientOptions) {
    toast.error("Помилка при видаленні товару з корзини");
    console.log(error, clientOptions);
  },
});
```

ДОДАТОК М

Компонент CatalogFilters

```tsx
const CatalogFilters: React.FC<ICatalogFiltersProps> = ({
  filter,
  viewType,
  maxPrice,
  setFilter,
  setViewType,
  fetchFilteredData,
  handleChangeFilter,
}) => {
  const t = useTranslations("catalog");

  return (
    <div className="flex items-center justify-between gap-[50px] w-full flex-col md:flex-row">
      <div className="flex items-center gap-[10px]">
        <Drawer>
          <DrawerTrigger>
            <Button className="block xl:hidden">Фільтри</Button>
          </DrawerTrigger>

          <DrawerContent>
            <DrawerHeader>
              <div className="flex justify-between items-center">
                <DrawerTitle>Фільтри</DrawerTitle>

                <DrawerClose>
                  <Button variant="secondary" size="icon" className="">
                    X
                  </Button>
                </DrawerClose>
              </div>

              <DrawerDescription>
                <ProductFilter
                  filter={filter}
                  setFilter={setFilter}
                  maxPrice={maxPrice}
                  handleChangeFilter={handleChangeFilter}
                  fetchFilteredData={fetchFilteredData}
                />
              </DrawerDescription>
            </DrawerHeader>
          </DrawerContent>
        </Drawer>

        <p className="text-nowrap hidden 2xl:block">{t("filter.sort.title")}</p>

        <Select onValueChange={(value) => fetchFilteredData({ sortBy: value })}>
          <SelectTrigger>
            <SelectValue placeholder={t("filter.sort.byDefault")} />
          </SelectTrigger>
          <SelectContent>
            <SelectGroup>
              <SelectItem value="default">{t("filter.sort.byDefault")}</SelectItem>
              <SelectItem value="rating">{t("filter.sort.byRating")}</SelectItem>
              <SelectItem value="new">{t("filter.sort.byNew")}</SelectItem>
              <SelectItem value="price:asc">{t("filter.sort.byPrice:asc")}</SelectItem>
              <SelectItem value="price:desc">{t("filter.sort.byPrice:desc")}</SelectItem>
            </SelectGroup>
          </SelectContent>
        </Select>
      </div>

      <div className="flex items-center gap-[30px]">
        <div className="flex items-center gap-[10px]">
          <div className="">{t("filter.itemsPerPage")}</div>
          <Select onValueChange={(value) => fetchFilteredData({ limit: Number(value) })}>
            <SelectTrigger className="w-[75px]">
              <SelectValue placeholder="24" />
            </SelectTrigger>
            <SelectContent>
              <SelectGroup>
                <SelectItem value="24">24</SelectItem>
                <SelectItem value="48">48</SelectItem>
                <SelectItem value="72">72</SelectItem>
                <SelectItem value="94">94</SelectItem>
              </SelectGroup>
            </SelectContent>
          </Select>
        </div>

        <div>
          <Button
            size="icon"
            variant="icon"
            className="border-none w-[44px] h-[44px]"
            onClick={() => setViewType("cards")}
          >
            <ViewCardIcon className={viewType === "cards" ? "fill-primary" : "fill-accent-foreground"} />
          </Button>

          <Button
            size="icon"
            variant="icon"
            className="border-none w-[44px] h-[44px]"
            onClick={() => setViewType("rows")}
          >
            <ViewRowsIcon className={viewType === "rows" ? "fill-primary" : "fill-accent-foreground"} />
          </Button>
        </div>
      </div>
    </div>
  );
};
```

ДОДАТОК Н

Zustand-стор корзини товарів: cartStore

```
export const cartStore = create(
  persist<ICartStore>(
    (set, get) => ({
      cartItems: [],
      setCartItems: (items: ICartItem[]) => {
        set({ cartItems: items });
      },
      changeCartItemsCount: (id: string, count: number) => {
        const cartItems = get().cartItems.map((el) => {
          if (el.id === id) {
            return { ...el, count };
          }
          return el;
        });
        set({ cartItems });
      },
      addItemToCart: (item: ICartItem) => {
        set({ cartItems: [...get().cartItems, item] });
      },
      removeItemFromCart: (id: string) => {
        const cartItems = get().cartItems.filter((el) => el.product.id !== id);
        set({ cartItems });
      },
      //
      selectedCartItems: [],
      changeSelectedCartItemsCount: (id: string, count: number) => {
        const selectedCartItems = get().selectedCartItems.map((el) => {
          if (el.id === id) {
            return { ...el, count };
          }
          return el;
        });
        set({ selectedCartItems });
      },
      toggleSelectedCartItems: (id: string, count: number, product: ProductModel) => {
        const isAdded = get().selectedCartItems.some((el) => el.product.id === id);
        if (isAdded) {
          const selectedCartItems = get().selectedCartItems.filter((el) => el.product.id !== id);
          set({ selectedCartItems });
        } else {
          const selectedCartItems = [...get().selectedCartItems, { id, count, product }];
          set({ selectedCartItems });
        }
      },
      clearSelectedItems: () => {
        set({ selectedCartItems: [] });
      },
    }),
    {
      name: "cart",
      storage: createJSONStorage(() => localStorage),
    }
  )
);
```

ДОДАТОК П

Форма для авторизації: LoginForm

```tsx
const LoginForm: React.FC<ILoginFormProps> = ({ setFromType }) => {
  const t = useTranslations('header')

  const { auth } = useAuth()

  const [login, { loading: isLoading }] = useLoginMutation({
    onCompleted() {
      toast.success('Ви успішно увійшли')
      auth()
      setAuthCookie()
    },
    onError(error) {
      toast.error(error.message)
    },
  })

  const form = useForm<z.infer<typeof formSchema>>({
    resolver: zodResolver(formSchema),
    defaultValues: {
      login: '',
      password: '',
    },
  })

  const onSubmit = (data: z.infer<typeof formSchema>) => {
    login({ variables: { data } })
  }

  const handleChangeFormType = () => {
    setFromType('register')
  }

  return (
    <Form {...form}>
      <form onSubmit={form.handleSubmit(onSubmit)}>
        <FormField
          control={form.control}
          name="login"
          render={({ field }) => (
            <FormItem className="pt-[30px] pb-[20px]">
              <FormLabel>{t('auth.loginForm.loginLabel')}</FormLabel>
              <FormControl>
                <Input
                  {...field}
                  className="h-[50px] px-[20px] w-full"
                  placeholder={t('auth.loginForm.loginPlaceholder')}
                />
              </FormControl>
              <FormMessage />
            </FormItem>
          )}
        />

        <FormField
          control={form.control}
          name="password"
          render={({ field }) => (
            <FormItem className="pb-[30px]">
              <FormLabel>{t('auth.loginForm.passLabel')}</FormLabel>
              <FormControl>
                <Input
                  placeholder={t('auth.loginForm.passPlaceholder')}
                  className="h-[50px] px-[20px] w-full"
                  type="password"
                  {...field}
                />
              </FormControl>
              <FormMessage />
            </FormItem>
          )}
        />

        <div className="flex items-center space-x-2 mb-[20px]">
          <Checkbox id="remember" className="border border-primary border-muted-foreground" />
          <label
            htmlFor="remember"
            className="text-sm font-medium leading-none cursor-pointer peer-disabled:cursor-not-allowed peer-disabled:opacity-70"
          >
            {t('auth.loginForm.rememberBtn')}
          </label>
        </div>

        <Button type="submit" className="w-full" disabled={isLoading}>
          {t('auth.loginBtn')}
        </Button>

        <div className="mt-[20px] flex">
          <Button variant="link" type="button" className="w-full text-primary opacity-100">
            {t('auth.loginForm.forgotPassBtn')}
          </Button>

          <Button
            variant="link"
            type="button"
            onClick={handleChangeFormType}
            className="w-full text-primary opacity-100"
          >
            {t('auth.registerBtn')}
          </Button>
        </div>
      </form>
    </Form>
  )
}
```

ДОДАТОК Р

Zustand-стор стану автентифікації: authStore

```
import { create } from "zustand";
import { createJSONStorage, persist } from "zustand/middleware";

import { AuthStore } from "./auth.types";

export const authStore = create(
  persist<AuthStore>(
    (set) => ({
      isAuthentificated: false,
      setIsAuthentificated: (value: boolean) =>
        set({ isAuthentificated: value }),
    }),
    {
      name: "auth",
      storage: createJSONStorage(() => localStorage),
    }
  )
);
```

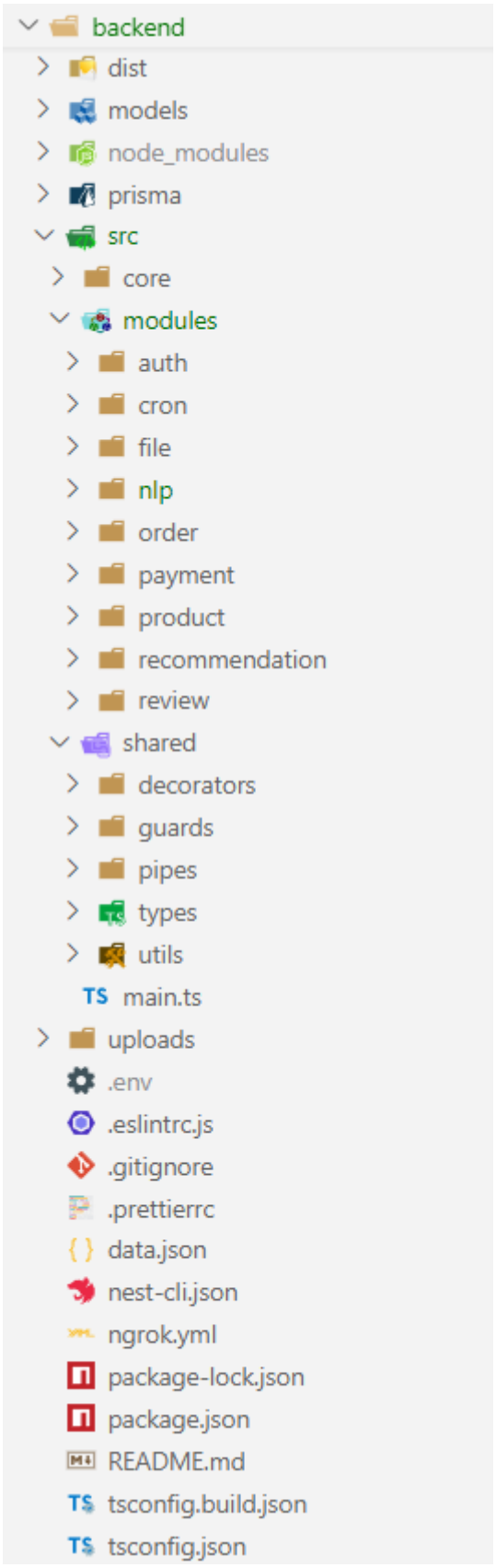
ДОДАТОК С

Хук для перевірки та отримання даних про користувача

```
export const useCurrent = () => {
  const { isAuthentificated, exit } = useAuth()

  const { data, loading, refetch, error } = useFindProfileQuery({
    skip: !isAuthentificated,
  })

  const [clear] = useClearSessionCookieMutation()

  React.useEffect(() => {
    if (error) {
      if (isAuthentificated) {
        clear()
        removeAuthCookie()
      }
      exit()
    }
  }, [isAuthentificated, exit, clear])

  return {
    user: data?.findProfile,
    isLoadingProfile: loading,
    refetch,
  }
}
```

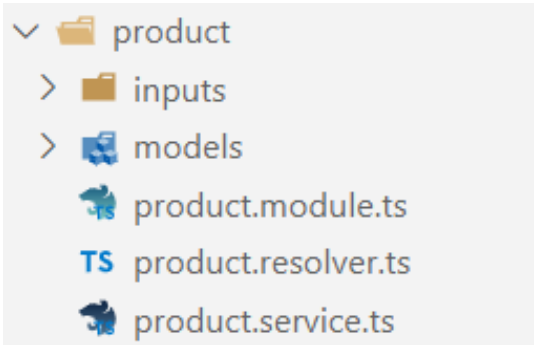
ДОДАТОК Т

Структура бекенду



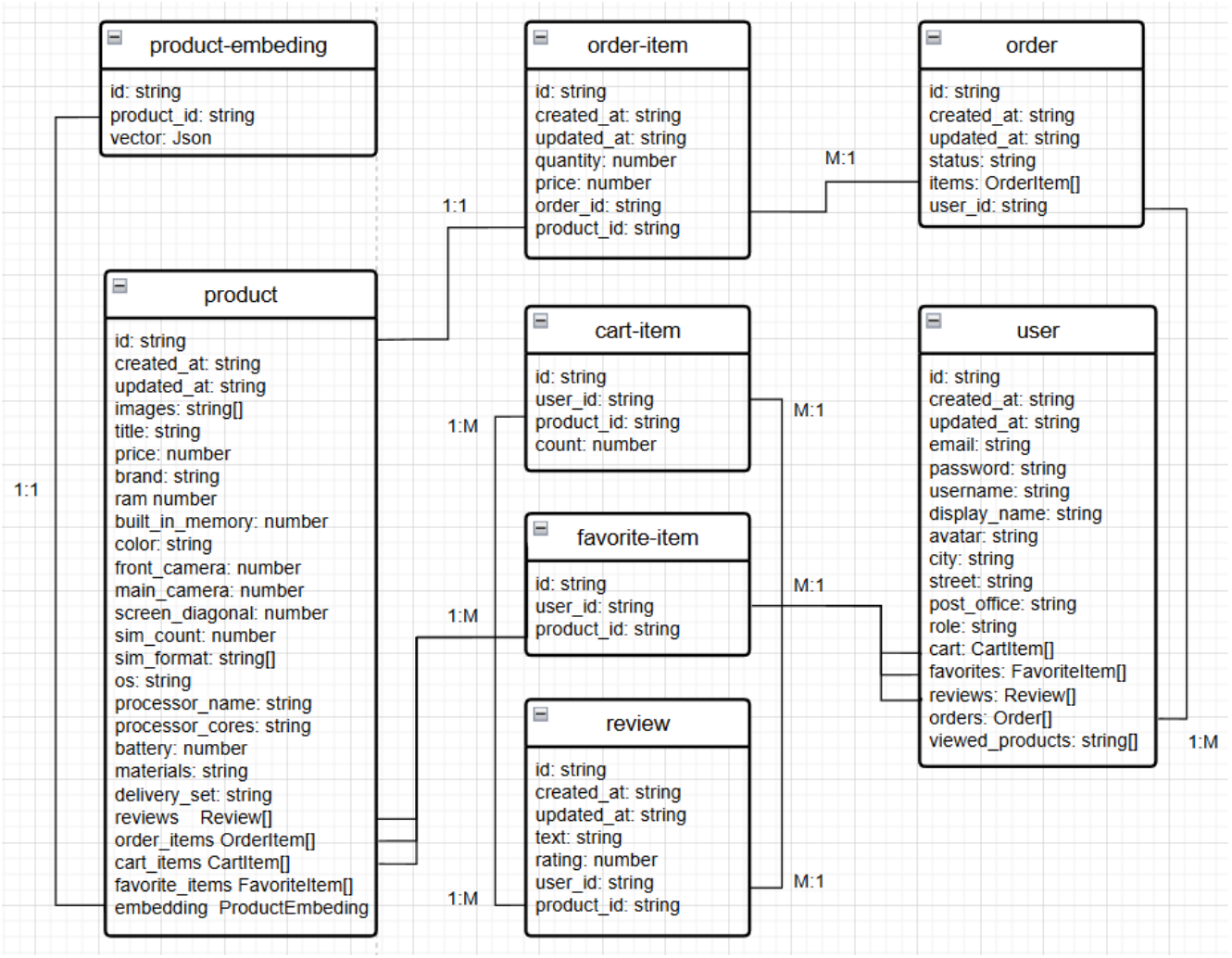
ДОДАТОК У

Приклад структури модулів



ДОДАТОК Ф

Структура бази даних



ДОДАТОК Х

Реалізація productsResolver для роботи з товарами

```
@Resolver('Product')
export class ProductResolver {
  constructor(private readonly productService: ProductService) {}

  @Query(() => ProductsAndTotalModel, { name: 'getAllProducts' })
  async getAll() {
    return this.productService.getAll();
  }

  @Query(() => Number, { name: 'getAllProductsCount' })
  async getTotalCount() {
    return this.productService.getTotalCount();
  }

  @Query(() => ProductsAndTotalModel, { name: 'paginateAndFilter' })
  async paginateAndFilter(@Args('data') query: PaginateAndFilterInput) {
    return this.productService.paginateAndFilter(query);
  }

  @Query(() => ProductModel, { name: 'getProductById' })
  async getById(@Args('productId') productId: string) {
    return this.productService.getById(productId);
  }

  @Query(() => [ProductModel], { name: 'searchProduct' })
  async search(@Args('data') input: string) {
    return this.productService.search(input);
  }

  @Query(() => [ProductModel], { name: 'getMostPopularProducts' })
  async getMostPopular() {
    return this.productService.getMostPopular();
  }

  @Query(() => ProductModel, { name: 'getSimilarProducts' })
  async getSimilar(@Args('productId') productId: string) {
    return this.productService.getSimilar(productId);
  }

  @Authorization()
  @Mutation(() => ProductModel, { name: 'createProduct' })
  async create(@Args('data') input: CreateProductInput) {
    return this.productService.create(input);
  }

  @Authorization()
  @Mutation(() => Boolean, { name: 'addProductPhoto' })
  async addPhoto(@Args('productId') productId: string, @Args({ name: 'file', type: () => GraphQLUpload }) file: any) {
    return this.productService.addPhoto(productId, file);
  }

  @Authorization()
  @Mutation(() => Boolean, { name: 'removeProductPhoto' })
  async removePhotos(@Args('productId') productId: string, @Args('filename') filename: string) {
    return this.productService.removePhotos(productId, filename);
  }

  @Mutation(() => Boolean, { name: 'createManyProducts' })
  async createMany() {
    return this.productService.createMany();
  }

  @Authorization()
  @Mutation(() => ProductModel, { name: 'updateProduct' })
  async update(@Args('data') input: UpdateProductInput) {
    return this.productService.update(input);
  }

  @Authorization()
  @Mutation(() => Boolean, { name: 'deleteProduct' })
  async delete(@Args('productId') productId: string) {
    return this.productService.delete(productId);
  }
}
```

ДОДАТОК Ц

Реалізація FileService для роботи з файлами

```
@Injectable()
export class FileService {
  constructor() {}

  generateId = () => {
    return [...Array(6)].map(() => Math.round(Math.random() * 6).toString(6)).join('');
  };

  async upload(file: Upload, folderName: 'products' | 'users' = 'products') {
    const { createReadStream, filename } = await file;

    const newFilename = `${this.generateId()}_${filename}`;
    const filePath = path.join(process.cwd(), `uploads/${folderName}`, newFilename);

    await new Promise((resolve, reject) => {
      createReadStream()
        .pipe(createWriteStream(filePath))
        .on('finish', () => resolve(`Файл завантажений: ${newFilename}`))
        .on('error', reject);
    });

    return newFilename;
  }

  async removeFile(filename: string, folderName: string) {
    const filePath = path.join(process.cwd(), `uploads/${folderName}`, filename);

    if (fs.existsSync(filePath)) {
      fs.unlinkSync(filePath);
    } else {
      throw new NotFoundException('Файл не знайдено');
    }

    return true;
  }
}
```

ДОДАТОК Ш

Реалізація оплати за допомогою платіжного сервісу Fondy

```
async createPayment(dto: CreatePaymentDto) {
  const FONDY_MERCHANT_ID = this.configService.getOrThrow<string>('FONDY_MERCHANT_ID');
  const FONDY_MARCHANT_PASSWORD = this.configService.getOrThrow<string>('FONDY_MERCHANT_PASSWORD');
  const FRONTEND_URL = this.configService.getOrThrow<string>('FRONTEND_URL');
  const APPLICATION_URL = this.configService.getOrThrow<string>('APPLICATION_URL');
  const NGROCK_FORWARDING_URL = this.configService.getOrThrow<string>('NGROCK_FORWARDING_URL');
  const ENVIRONMENT = this.configService.getOrThrow<string>('NODE_ENV');

  const BASE_URL = ENVIRONMENT === 'development' ? NGROCK_FORWARDING_URL : APPLICATION_URL;

  const orderedItemsString = JSON.stringify(dto.items);
  const order_id = `name=${dto.name}//price=${dto.price}//userId=${dto.userId}//items=${orderedItemsString}//createdAt=${Date.now()}`;

  console.log('CREATE PAYMENT', FRONTEND_URL, BASE_URL);

  const orderBody = {
    response_url: `${FRONTEND_URL}/checkout/thank-you`,
    server_callback_url: `${BASE_URL}/payment/confirmation`,
    order_id: order_id,
    merchant_id: FONDY_MERCHANT_ID,
    order_desc: dto.name,
    amount: dto.price * 100,
    currency: 'UAH',
  };

  const orderKeys = Object.keys(orderBody).sort((a, b) => {
    if (a < b) return -1;
    if (a > b) return 1;
    return 0;
  });

  const signatureRaw = orderKeys.map((v) => orderBody[v]).join('|');
  const signature = crypto.createHash('sha1');

  signature.update(`${FONDY_MARCHANT_PASSWORD}|${signatureRaw}`);

  const json = JSON.stringify({
    request: {
      ...orderBody,
      signature: signature.digest('hex'),
    },
  });

  const response = await fetch('https://pay.fondy.eu/api/checkout/url/', {
    body: json,
    headers: {
      'Content-Type': 'application/json',
    },
    method: 'POST',
  });

  const data = await response.json();

  return data;
}
```

ДОДАТОК Щ

Збереження замовлення в базі даних

```
async confirmPayment(dto: FondyCallbackResponseDto) {
  try {
    const isCurrentPayment = this.checkSignature(dto);

    if (dto.order_status === 'approved' && isCurrentPayment) {
      const orderDataString = dto.order_id;

      const ordersFieldsArray = orderDataString.split('//').map((el: string) => {
        const substr = el.split('=');
        if (substr[0] === 'price') {
          return { [substr[0]]: Number(substr[1]) };
        } else if (substr[0] === 'items') {
          return { [substr[0]]: JSON.parse(substr[1]) };
        } else {
          return { [substr[0]]: substr[1] };
        }
      });

      const orderData: CreatePaymentDto = ordersFieldsArray.reduce(
        (obj, item) => {
          const key = Object.keys(item)[0];
          if (key && typeof item[key] !== 'undefined') {
            obj[key] = item[key];
          }
          return obj;
        },
        { name: '', userId: '', price: 0, items: [] },
      );

      const { userId, items } = orderData;

      const isOrderExist = await this.orderService.checkIsExist(dto.order_id);
      console.log('isOrderExist:', isOrderExist);
      if (!isOrderExist) {
        const order = await this.orderService.create({ userId, orderId: dto.order_id, items });

        await Promise.all(
          items.map(async (el) => {
            await this.accountService.toggleCart(userId, { productId: el.productId, count: el.quantity });
          }),
        );

        return order;
      }
    }
  } catch (error) {
    throw new Error('Сталась помилка з платіжним сервісом. Спробуйте пізніше');
  }
}
```

ДОДАТОК Ю

Python-алгоритм для генерації датасету

```python
def generate_text_query(product, index):
    query = ""
    keys = ""
    chance_of_adding = 5
    spacy_entities = []

    # Фильтрація по бренду
    if 'Назва' in product:
        random_number = random.randint(1, chance_of_adding)
        if random_number == 1:
            words_list = product['Назва'].lower().split()
            for word in words_list:
                if word in PHONE_BRANDS_NAMES:
                        all_brand_names = PHONE_BRANDS_NAMES[word].split('/')
                        random_brand_name = random.choice(all_brand_names)
                        query_len_before = len(query)
                        query += f"{random_brand_name} "
                        # SPACY
                        is_has_brand = any('BRAND' in tpl for tpl in spacy_entities)
                        if not is_has_brand:
                            spacy_entities.append((query_len_before, query_len_before + len(random_brand_name) + 1, 'BRAND'))

        elif random_number == 2 or random_number == 3:
            symbol_index = product['Назва'].find("/")
            phone_string = product['Назва'][:symbol_index]
            phone_list = phone_string.split()
            phone_model_list = phone_list[2:len(phone_list) - 1][:5]
            phone_model_str = ' '.join(phone_model_list)
            query_len_before = len(query)
            query += f"{phone_model_str} "
            spacy_entities.append((query_len_before, query_len_before + len(phone_model_str) + 1, 'TITLE'))


    # Фильтрація по ціні
    if 'Ціна' in product:
        price_random_number = random.randint(1, chance_of_adding - 3)
        if price_random_number == 1:
            price = str(re.sub(r'\D', '', product['Ціна']))
            prices_list = get_price_query(price)
            random_index = random.randint(0, len(prices_list) - 1)
            random_price_words = prices_list[random_index]
            query_len_before = len(query)
            query += f"{random_price_words} "

            if random_index <= 2:
                start_from = query_len_before + 3
                spacy_entities.append((start_from, start_from + len(price) + 1, 'PRICE_LT'))
            elif random_index <= 7:
                start_from = query_len_before + 4
                spacy_entities.append((start_from, start_from + len(price) + 1, 'PRICE_GT'))
            elif random_index <= 14:
                start_from = query_len_before + 5
                spacy_entities.append((start_from, start_from + len(price) + 1, 'PRICE'))
            elif random_index <= 17:
                start_from = query_len_before + 6
                spacy_entities.append((start_from, start_from + len(price) + 1, 'PRICE_LT'))
            elif random_index <= 20:
                start_from = query_len_before + 7
                spacy_entities.append((start_from, start_from + len(price) + 1, 'PRICE_GT'))
            elif random_index == 21:
                start_from = query_len_before + 9
                spacy_entities.append((start_from, start_from + len(price) + 1, 'PRICE_GT'))
            elif random_index <= 24:
                start_from = query_len_before + 12
                spacy_entities.append((start_from, start_from + len(price) + 1, 'PRICE_GT'))
            elif random_index == 25:
                price_from = int(price) - 2000
                if price_from < 0:
                    price_from = 0
                price_from_len = len(str(price_from))
                start_from = query_len_before + 9
                spacy_entities.append((start_from, start_from + price_from_len + 1, 'PRICE_GT'))
                start_from = query_len_before + 13
                spacy_entities.append((start_from + price_from_len, start_from + price_from_len + len(price) + 1, 'PRICE_LT'))
            elif random_index == 26:
                price_from = int(price) - 2000
                if price_from < 0:
                    price_from = 0
                price_from_len = len(str(price_from))
                start_from = query_len_before + 12
                spacy_entities.append((start_from, start_from + price_from_len + 1, 'PRICE_GT'))
                start_from = query_len_before + 19
                spacy_entities.append((start_from + price_from_len, start_from + price_from_len + len(price) + 1, 'PRICE_LT'))
            elif random_index == 27:
                start_from = query_len_before + 4
                spacy_entities.append((start_from, start_from + len(price) + 1, 'PRICE_GT'))
                start_from = query_len_before + 8
                price_to_len = len(str(int(price) + 5000))
                spacy_entities.append((start_from + len(price), start_from + len(price) + price_to_len + 1, 'PRICE_LT'))

    # Фильтрація по оперативной памяти
    if 'Оперативна пам\'ять' in product:
        random_number = random.randint(1, chance_of_adding)
        if random_number == 1:
            ram_value = str(product['Оперативна пам\'ять'].split(' ')[0])
            ram = [
                f"ОЗУ {ram_value}",
                f"оперативна память {ram_value}",
                f"ОЗУ {ram_value} гб",
                f"оперативна память {ram_value} гігабайт",
                f"ОЗУ {ram_value} гігабайт",
                f"оперативна память {ram_value} гіг",
            ]
            random_index = random.randint(0, len(ram) - 1)
            random_ram = ram[random_index]
            query_len_before = len(query)
            query += f"{random_ram} "
            if random_index == 0 or random_index == 2 or random_index == 4:
                symbols_to_ram_val = query_len_before + 4
                spacy_entities.append((symbols_to_ram_val, symbols_to_ram_val + len(ram_value) + 1, 'RAM'))
            else:
                symbols_to_ram_val = query_len_before + 18
                spacy_entities.append((symbols_to_ram_val, symbols_to_ram_val + len(ram_value) + 1, 'RAM'))

    # Фильтрація по кольору
    if 'Колір' in product:
        random_number = random.randint(1, chance_of_adding)
        if random_number == 1:
            if product['Колір'] in COLORS_DICT:
                color = COLORS_DICT[product['Колір']]
                query_len_before = len(query)
                query += f"{color}{random.choice(['', ' колір'])} "
                keys += 'Колір'
                spacy_entities.append((query_len_before, query_len_before + len(color) + 1, 'COLOR'))


    # Фильтрація по камері (фронтальній)
    if 'Фронтальна камера' in product:
        random_number = random.randint(1, chance_of_adding)
        if random_number == 1:
            front_camera_mp = product['Фронтальна камера'].split(' ')[0]
            if len(front_camera_mp) > 0:
                front_camera_names = ['фронтальна камера ', 'фронтальна ', 'фронтальна ', 'камера ', 'camera ']
                mp_names = [' мп', ' MP', ' мегапікселей', ' мега пікселей', '', '']
                query_len_before = len(query)
                random_front_camera_names = random.choice(front_camera_names)
                query += f"{random_front_camera_names}{front_camera_mp}{random.choice(mp_names)} "
                start_from = query_len_before + len(random_front_camera_names)
                spacy_entities.append((start_from, start_from + len(front_camera_mp) + 1, 'FRONT_CAMERA'))


    cleaned_value = re.sub(r'[^\x20-\x7E]', '', query)
    if len(cleaned_value) == 0:
        data = generate_text_query(product, index)
        return (data[0], data[1])

    return [query, spacy_entities]

dataset = []


for i in range(NUMBER_OF_CYCLE_ITERATIONS):
    with open(CSV_FILE_NAME, mode="r", newline="", encoding="utf-8") as csv_file:
        reader = csv.DictReader(csv_file)
        for index, row in enumerate(reader):
            data = generate_text_query(row, index)
            dataset.append({
                "text": data[0],
                "entities": data[1]
            })

    bar_length = 30
    progress = int((i / NUMBER_OF_CYCLE_ITERATIONS) * bar_length)
    bar = "#" * progress + "-" * (bar_length - progress)
    percent = (i / NUMBER_OF_CYCLE_ITERATIONS) * 100
    sys.stdout.write(f"\r[{bar}] {percent:.2f}%")
    sys.stdout.flush()


with open(OUTPUT_FILE_NAME, "w", encoding='utf-8') as file:
    json.dump(dataset, file, ensure_ascii=False, indent=3)
```

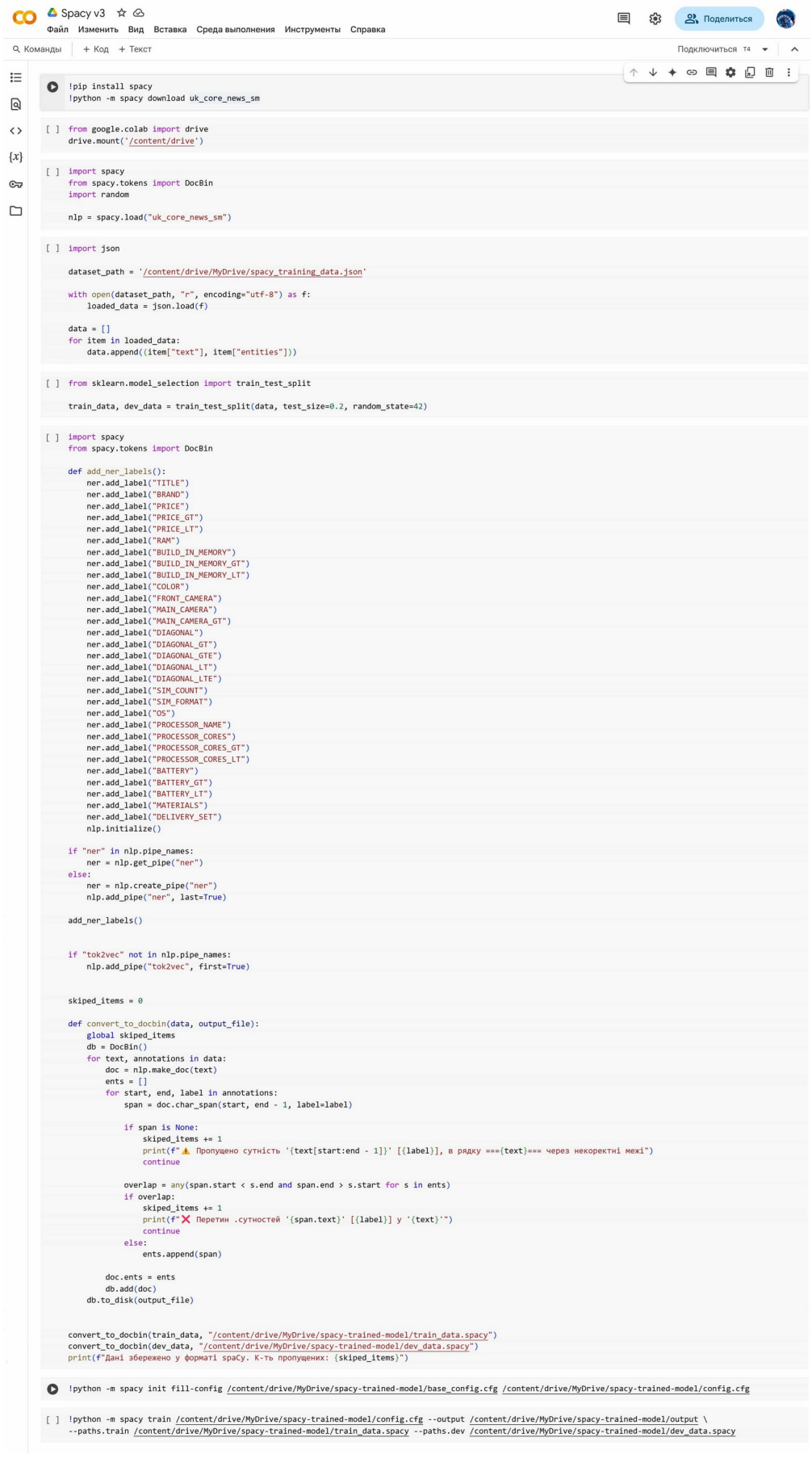
ДОДАТОК Я

Приклад згенерованого текстового запиту та позначених сутностей

```
{
  "text": "самсунг ціна менше 15000 грн озу 6 гб",
  "entities": [
    [1, 7, "BRAND"],
    [20, 25, "PRICE_LT"],
    [34, 35, "RAM"]
  ]
}
```

ДОДАТОК АА

Налаштування середовища Google Colab для донавчання моделі



ДОДАТОК АБ

Python-модуль для обробки текстових запитів

```python
import os
import sys
import json
import spacy

MODEL_PATH = os.path.abspath(os.path.join(os.path.dirname(__file__), '..', '..', '..', '..', 'models', 'uk_core_news_sm'))
KEY_MAP = {
    "TITLE": "title",
    "BRAND": "brand",
    "PRICE": "price",
    "PRICE_GT": "price_gt",
    "PRICE_LT": "price_lt",
    "RAM": "ram",
    "BUILD_IN_MEMORY": "built_in_memory",
    "BUILD_IN_MEMORY_GT": "built_in_memory_gt",
    "BUILD_IN_MEMORY_LT": "built_in_memory_lt",
    "COLOR": "color",
    "FRONT_CAMERA": "front_camera",
    "MAIN_CAMERA": "main_camera",
    "MAIN_CAMERA_GT": "main_camera_gt",
    "DIAGONAL": "screen_diagonal",
    "DIAGONAL_GT": "screen_diagonal_gt",
    "DIAGONAL_LT": "screen_diagonal_lt",
    "SIM_COUNT": "sim_count",
    "SIM_FORMAT": "sim_format",
    "OS": "os",
    "PROCESSOR_NAME": "processor_name",
    "PROCESSOR_CORES": "processor_cores",
    "PROCESSOR_CORES_GT": "processor_cores_gt",
    "PROCESSOR_CORES_LT": "processor_cores_lt",
    "BATTERY": "battery",
    "BATTERY_GT": "battery_gt",
    "BATTERY_LT": "battery_lt",
    "MATERIALS": "materials",
    "DELIVERY_SET": "delivery_set",
}


nlp = spacy.load(MODEL_PATH)


def rename_keys(original_dict, key_map):
    nested_keys = ["price", "built_in_memory", "screen_diagonal", "sim_count"]
    keys_has = ["sim_format"]
    keys_contains = ["brand", "title", "color", "os", "processor_name", "processor_cores", "materials", "delivery_set"]
    keys_equals = ["ram", "front_camera", "main_camera", "battery"]
    keys_gt = ["price_gt", "built_in_memory_gt", "main_camera_gt", "screen_diagonal_gt", "processor_cores_gt"]
    keys_lt = ["price_lt", "built_in_memory_lt", "screen_diagonal_lt", "processor_cores_lt"]

    new_dict = {}

    for old_key, value in original_dict.items():
        new_key = key_map.get(old_key, old_key)

        if new_key in nested_keys:
            new_dict[new_key] = int(value)

        elif new_key in keys_has:
            new_dict[new_key] = {"has": value}

        elif new_key in keys_contains:
            new_dict[new_key] = {"contains": value, 'mode': 'insensitive'}

        elif new_key in keys_equals:
            try:
                new_dict[new_key] = {"equals": int(value.strip())}
            except ValueError:
                new_dict[new_key] = {"equals": value}

        elif new_key in keys_gt:
            new_current_key = new_key.split("_")
            new_current_key = "_".join(new_current_key[:-1])
            if new_current_key in new_dict:
                new_dict[new_current_key]['gt'] = int(value)
            else:
                new_dict[new_current_key] = {'gt': int(value)}

        elif new_key in keys_lt:
            new_current_key = new_key.split("_")
            new_current_key = "_".join(new_current_key[:-1])
            if new_current_key in new_dict:
                new_dict[new_current_key]['lt'] = int(value)
            else:
                new_dict[new_current_key] = {'lt': int(value)}


def analyze(text):
    doc = nlp(text)
    entities_dict = {ent.label_: ent.text for ent in doc.ents}
    print('entities_dict:', entities_dict)
    result = rename_keys(entities_dict, KEY_MAP)
    return json.dumps(result, ensure_ascii=False)


if __name__ == "__main__":
    try:
        if len(sys.argv) < 2:
            json.dumps({"error": "No text argument provided"})
            print(json.dumps({"error": "No text argument provided"}))

        text = sys.argv[1]
        result = analyze(text)
        json.dumps(result)
        print(json.dumps(result))

    except Exception as e:
        error_result = {"message": "error", "error": str(e)}
        print(json.dumps(error_result))
```

ДОДАТОК АВ

NestJS сервіс для обробки текстових запитів

```
const path = require('path');
import { PythonShell } from 'python-shell';
import { Injectable } from '@nestjs/common';

const PYTHON_PATH = path.join(process.cwd(), 'src/modules/nlp/python/venv/Scripts/python.exe');
const VENV_ACTIVATE = path.join(process.cwd(), 'src/modules/nlp/python/venv/Scripts/activate');
const SCRIPT_PATH = path.join(process.cwd(), 'src/modules/nlp/python/analyze.py');

@Injectable()
export class NlpProcessor {
  async analyzeText(text: string): Promise<any> {
    return new Promise(async (resolve, reject) => {
      try {
        const result = await PythonShell.run(SCRIPT_PATH, {
          args: [text],
          pythonPath: PYTHON_PATH,
          env: {
            ...process.env,
            PATH: PYTHON_PATH,
            VIRTUAL_ENV: VENV_ACTIVATE,
          },
        });
        console.log('result:', result);
        if (!result) return reject('No result from Python script');
        const embedding = JSON.parse(result[0]);
        resolve(embedding);
      } catch (e) {
        reject(e);
      }
    });
  }
}
```

ДОДАТОК АГ

Python-скрипт для створення векторного представлення товару

```python
import os
import sys
import json
import torch
from transformers import AutoTokenizer, AutoModel

MODEL_PATH = os.path.abspath(os.path.join(os.path.dirname(__file__), '..', '..', '..', '..', 'models', 'bge-small-en-v1.5'))
DIM = 384

tokenizer = AutoTokenizer.from_pretrained(MODEL_PATH)
model = AutoModel.from_pretrained(MODEL_PATH).to("cuda" if torch.cuda.is_available() else "cpu")


def get_embedding(text):
    inputs = tokenizer(text, return_tensors="pt").to(model.device)
    with torch.no_grad():
        outputs = model(**inputs)
    embedding = outputs.last_hidden_state.mean(dim=1).squeeze().cpu().numpy()
    return embedding


def resize_embedding(embedding, target_size=DIM):
    if len(embedding) < target_size:
        return embedding + [0.0] * (target_size - len(embedding))
    return embedding[:target_size]


def main():
    text = sys.argv[1]
    embedding = get_embedding(text)
    embedding = resize_embedding(embedding)
    print(json.dumps(embedding.tolist()))

if __name__ == '__main__':
    main()
```

NestJS сервіс для роботи з векторами товарів

```
const TMP_PATH = path.join(process.cwd(), 'src/modules/recommendation/python/tmp');
const MAIN_SCRIPT_PATH = path.join(process.cwd(), 'src/modules/recommendation/python/main.py');
const SEARCH_SCRIPT_PATH = path.join(process.cwd(), 'src/modules/recommendation/python/search.py');
const PYTHON_PATH = path.join(process.cwd(), 'src/modules/recommendation/python/venv/Scripts/python.exe');

@Injectable()
export class RecommendationService {
  constructor(private prisma: PrismaService) {}

  async createProduct(dto: CreateVectorInput) {
    const textRepresentation = this.createTextRepresentation(dto);
    const embedding = await this.generateEmbedding(textRepresentation);

    return this.prisma.productEmbeding.create({
      data: {
        product: { connect: { id: String(dto.id) } },
        vector: embedding,
      },
    });
  }

  private createTextRepresentation(product: any): string {
    return `
        title: ${product.title},
        price: ${product.price},
        brand: ${product.brand},
        ram: ${product.ram},
        builtInMemory: ${product.builtInMemory},
        color: ${product.color},
        frontCamera: ${product.frontCamera},
        mainCamera: ${product.mainCamera},
        screenDiagonal: ${product.screenDiagonal},
        simCount: ${product.simCount},
        simFormat: ${product.simFormat},
        os: ${product.os},
        processorName: ${product.processorName},
        processorCores: ${product.processorCores},
        battery: ${product.battery},
        materials: ${product.materials},
        deliverySet: ${product.deliverySet}
      `;
  }

  private async generateEmbedding(text: string): Promise<number[]> {
    return new Promise(async (resolve, reject) => {
      try {
        const result = await PythonShell.run(MAIN_SCRIPT_PATH, { args: [text], pythonPath: PYTHON_PATH });
        if (!result) return reject('No result from Python script');
        const embedding = JSON.parse(result[0]);
        resolve(embedding);
      } catch (e) {
        reject(e);
      }
    });
  }

  async findSimilarProducts(viewedProductIds: string[]) {
    if (viewedProductIds.length === 0) return [];
    const allVectors = await this.prisma.productEmbeding.findMany();
    const viewedProductsVectors = allVectors.filter((el) => viewedProductIds.includes(el.productId));
    if (!viewedProductsVectors.length) return [];
    const vectorsArray = viewedProductsVectors.map((p) => p.vector);
    const allVectorsArray = allVectors.map((p) => p.vector);
    const similarProductIds = await this.findNearestNeighbors(vectorsArray, allVectorsArray);

    const simProdIds = await Promise.all(
      // @ts-ignore
      similarProductIds.ids.map(async (el) => {
        const product = await this.prisma.productEmbeding.findUnique({ where: { id: el } });
        if (!product) {
          console.log(`Product embedding with ID ${el} is not exist`);
          return;
        }

        return product.productId;
      }),
    );

    const existedIds = simProdIds.filter((el) => el !== 'null' && !!el);

    const products = await this.prisma.product.findMany({
      where: { id: { in: existedIds } },
    });
    return products;
  }
  private async findNearestNeighbors(queryVector: any[], allVectors: any[]): Promise<string[]> {
    const filename = `input_${uuidv4()}.json`;
    const filepath = join(TMP_PATH, filename);

    return new Promise(async (resolve, reject) => {
      try {
        writeFileSync(filepath, JSON.stringify({ queryVector, allVectors }), 'utf-8');
        const result = await PythonShell.run(SEARCH_SCRIPT_PATH, {
          args: [filepath],
          pythonPath: PYTHON_PATH,
        });
        if (!result) return reject('No result from Python script');
        const embedding = JSON.parse(result[0]);
        resolve(embedding);
      } catch (e) {
        reject(e);
      } finally {
        unlinkSync(filepath);
      }
    });
  }
}
```

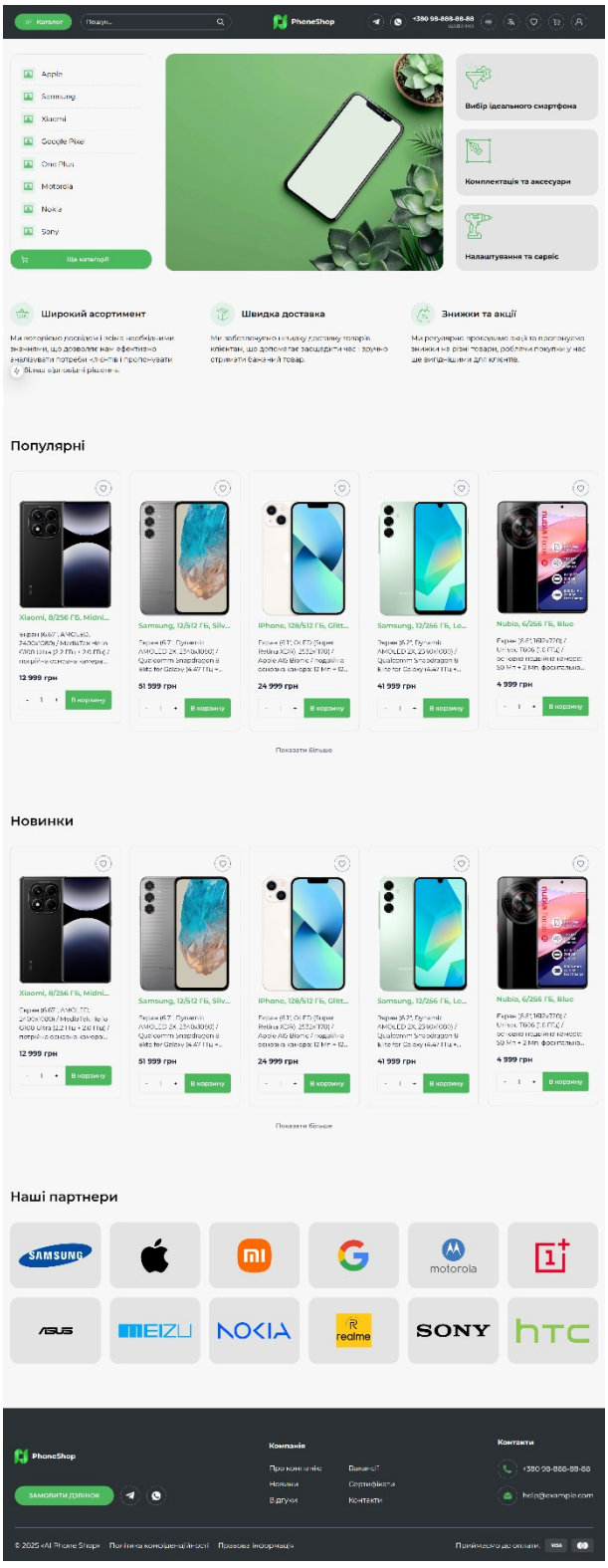
ДОДАТОК АЕ

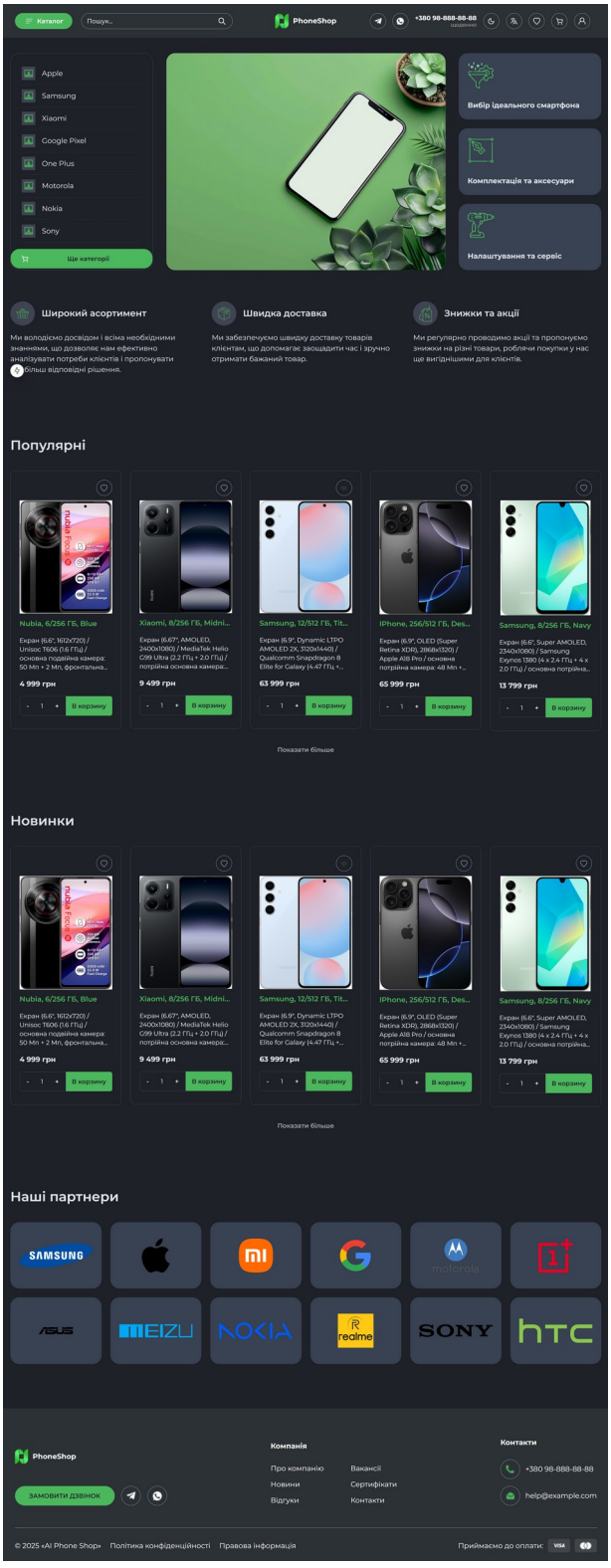
Python-скрипт для пошуку схожих векторів

```python
import sys
import json
import numpy
import faiss

json_file_path = sys.argv[1]

with open(json_file_path, 'r', encoding='utf-8') as f:
    data = json.load(f)


user_viewed_vectors = numpy.array(data['queryVector'], dtype='float32')
all_product_vectors = numpy.array(data['allVectors'], dtype='float32')

dimension = all_product_vectors.shape[1]
index = faiss.IndexFlatL2(dimension)
index.add(all_product_vectors)

user_viewed_vectors = user_viewed_vectors.reshape(user_viewed_vectors.shape[0], -1)

all_similar_products = []
for query_vector in user_viewed_vectors:
    query_vector = numpy.array(query_vector, dtype='float32').reshape(1, -1)
    distances, indices = index.search(query_vector, 50)

    for i in range(len(indices[0])):
        all_similar_products.append((int(indices[0][i]), float(distances[0][i])))


all_similar_products.sort(key=lambda x: x[1])

similar_product_ids = [product[0] for product in all_similar_products]
similar_distances = [product[1] for product in all_similar_products]

result = {
    'ids': similar_product_ids,
    'distances': similar_distances
}

print(json.dumps(result))
```

ДОДАТОК АЖ

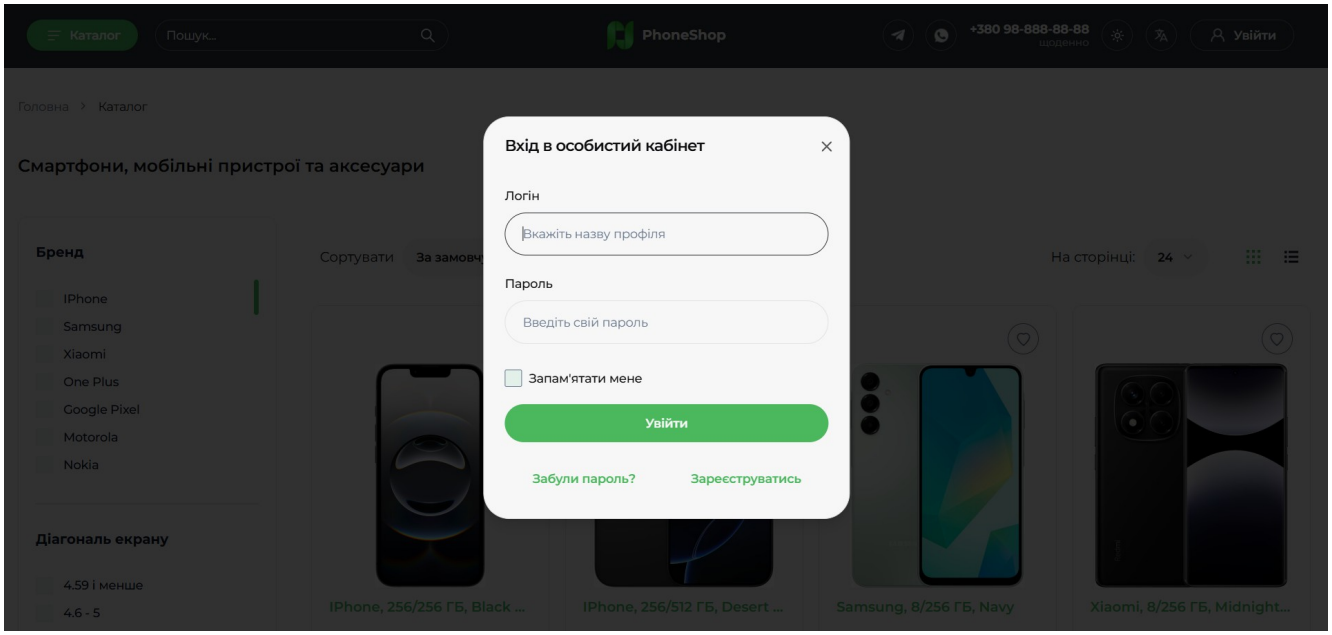
Інтерфейс головної сторінки





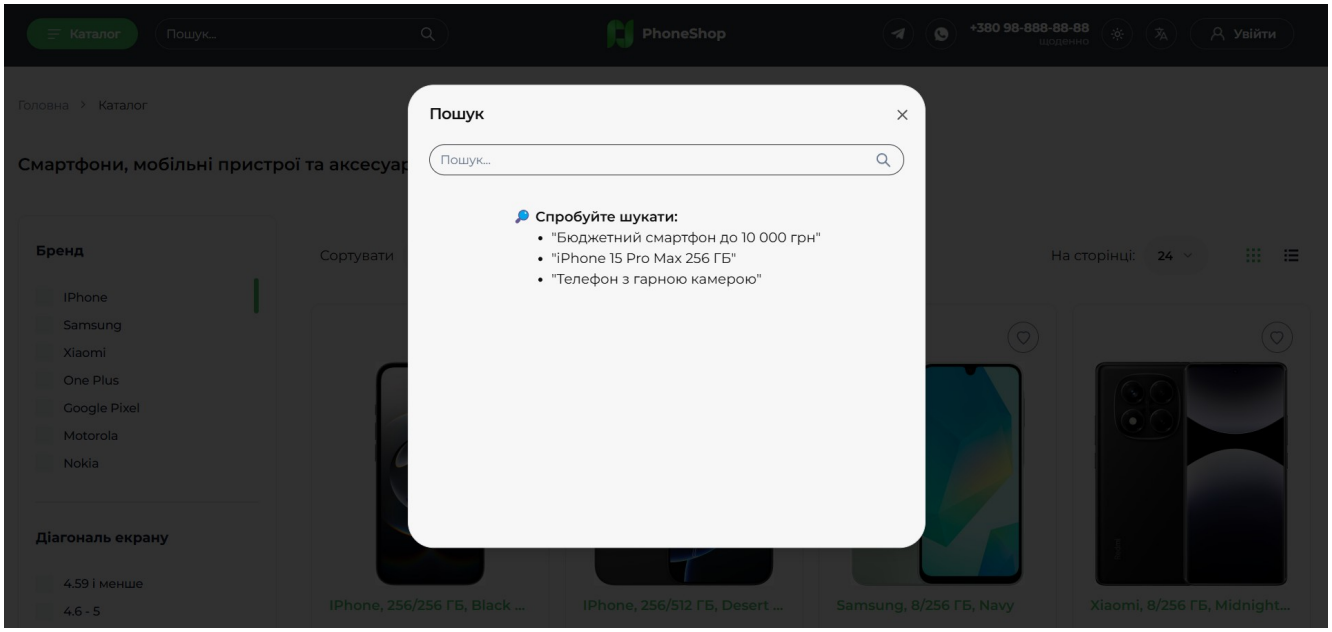
ДОДАТОК АИ

Інтерфейс форми авторизації



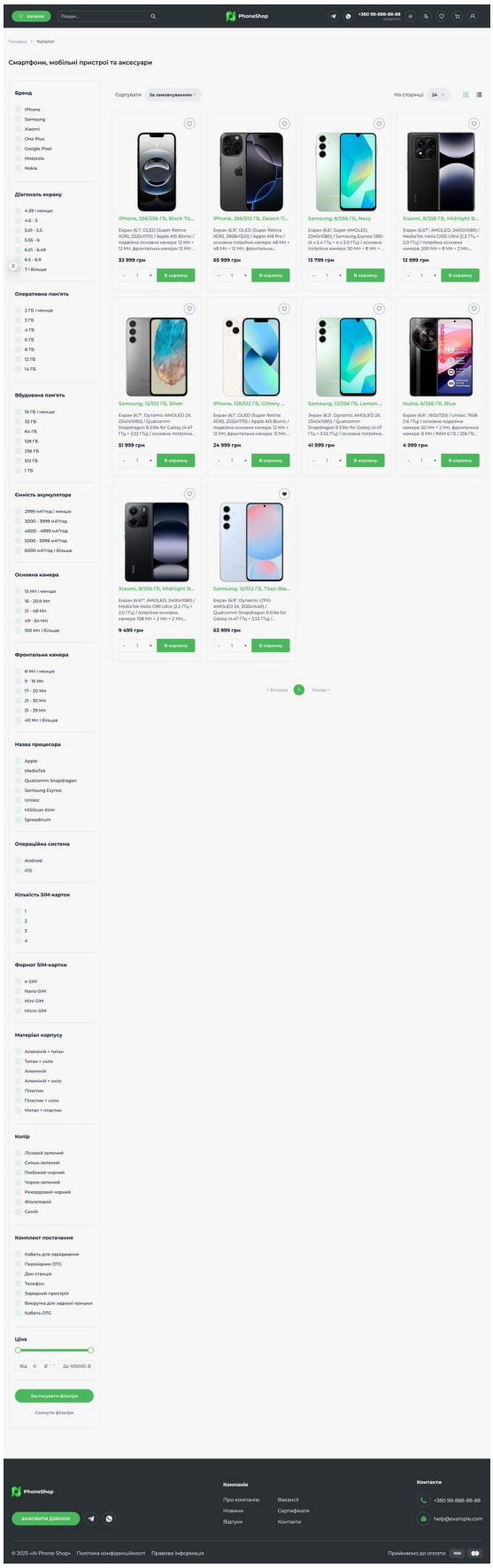
ДОДАТОК АК

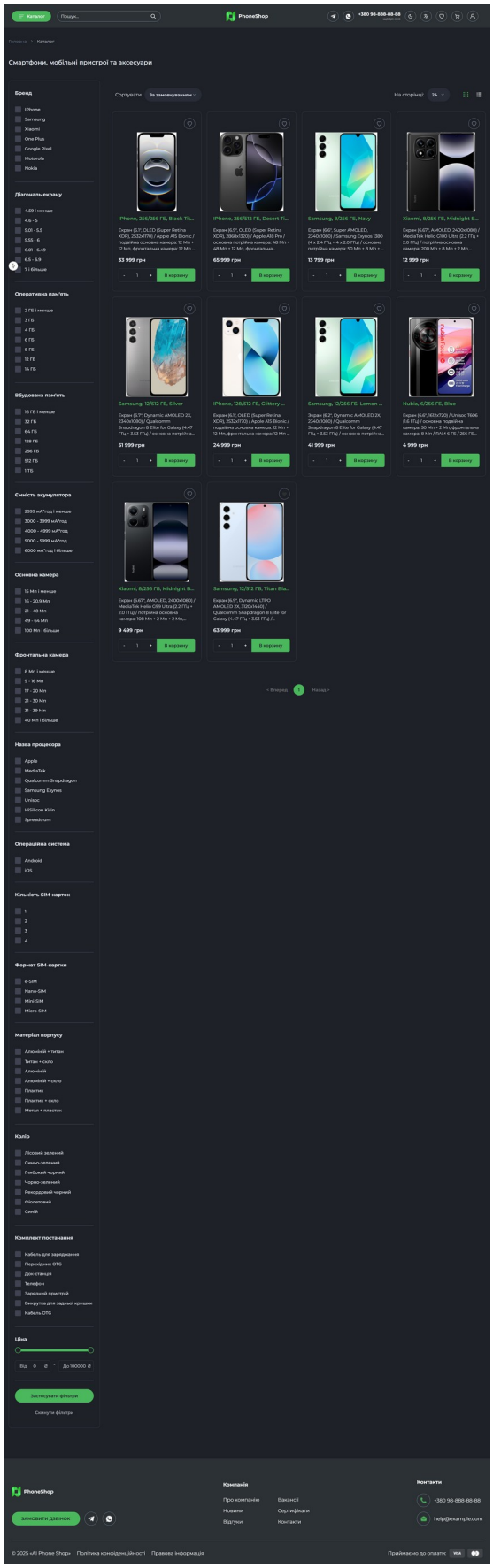
Інтерфейс форми для інтелектуального пошуку



ДОДАТОК АЛ

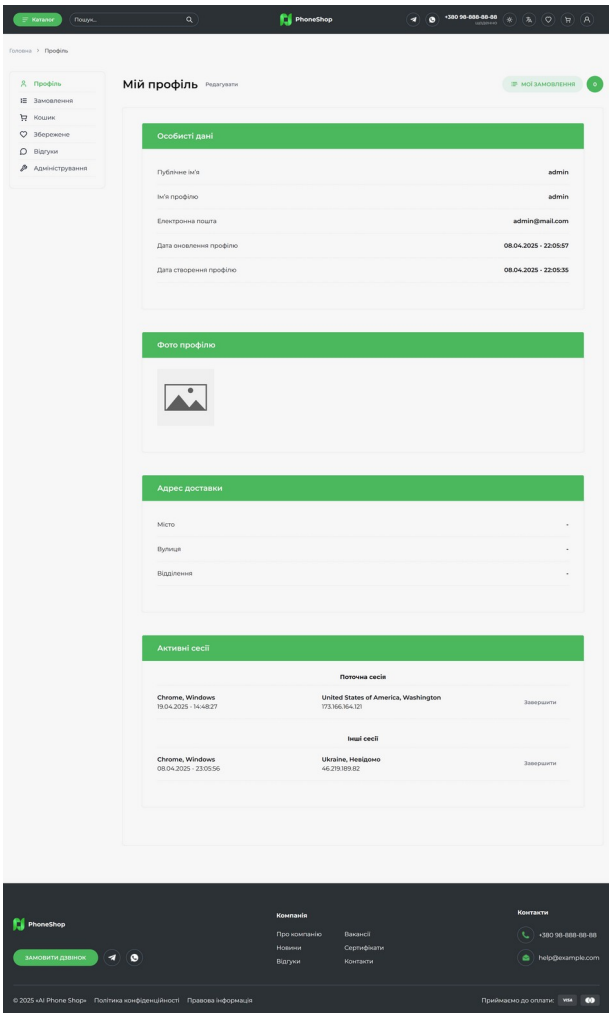
Інтерфейс сторінки "Каталог"

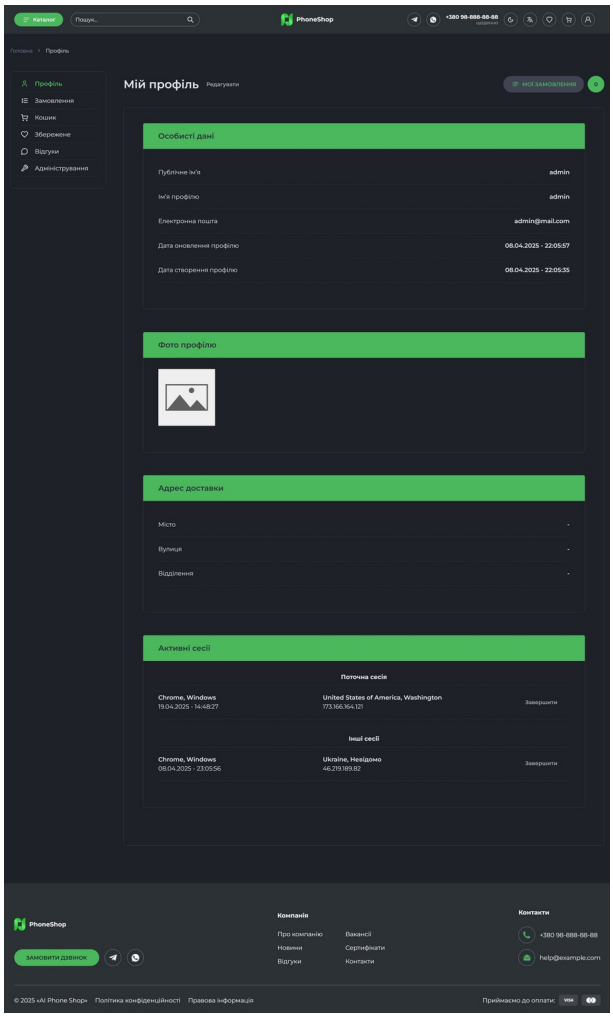




ДОДАТОК АМ

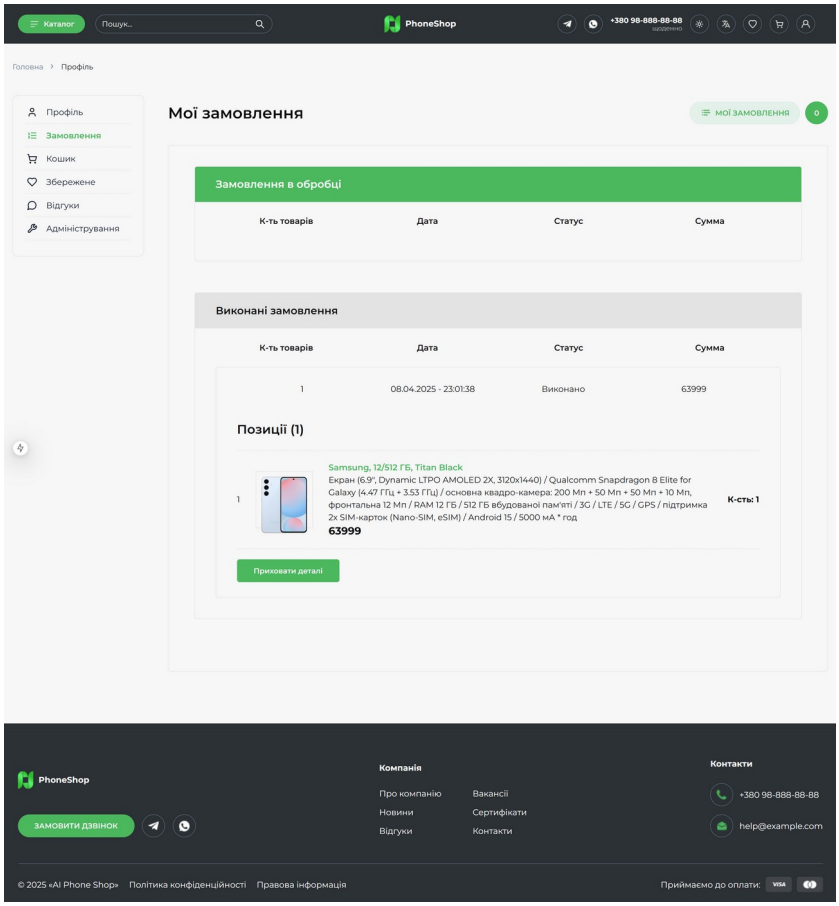
Інтерфейс сторінки "Профіль"

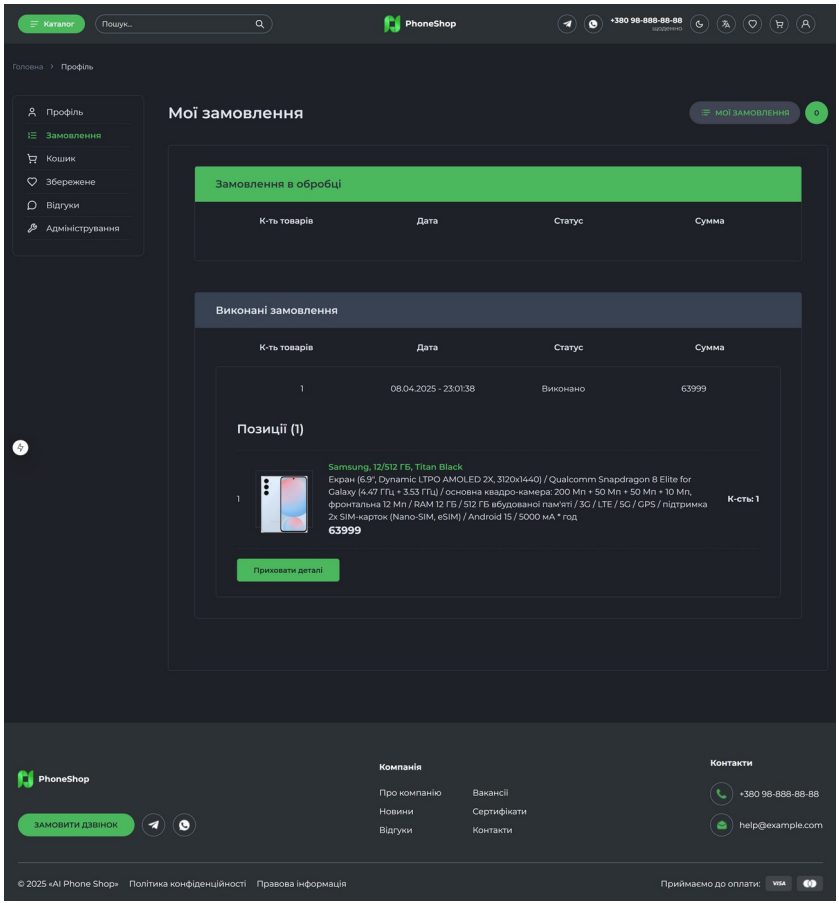




ДОДАТОК АН

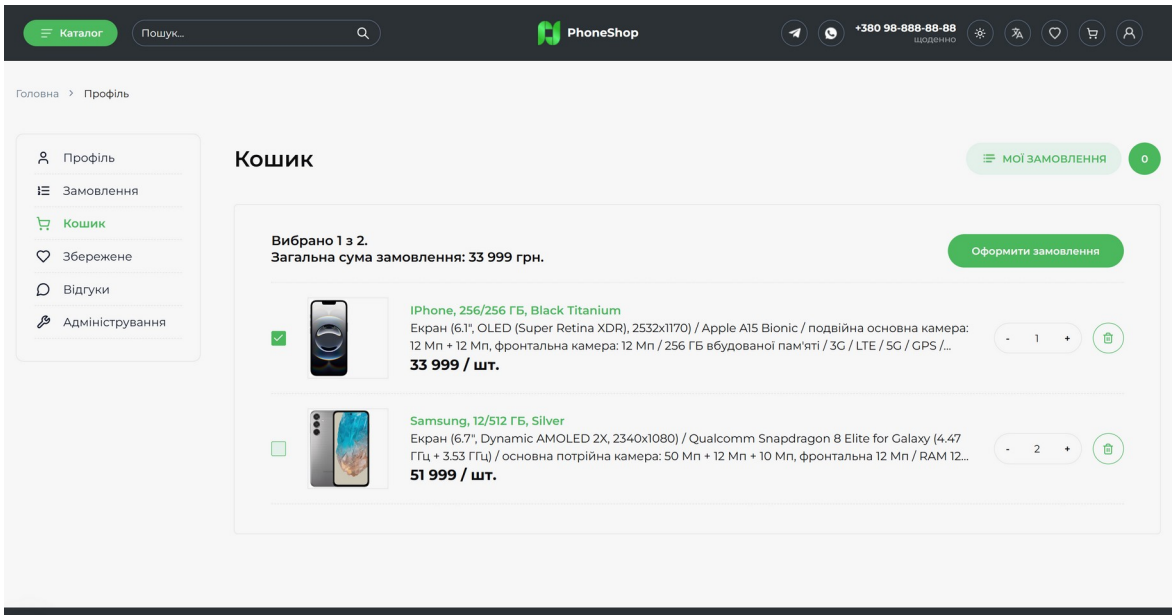
Інтерфейс сторінки "Замовлення"

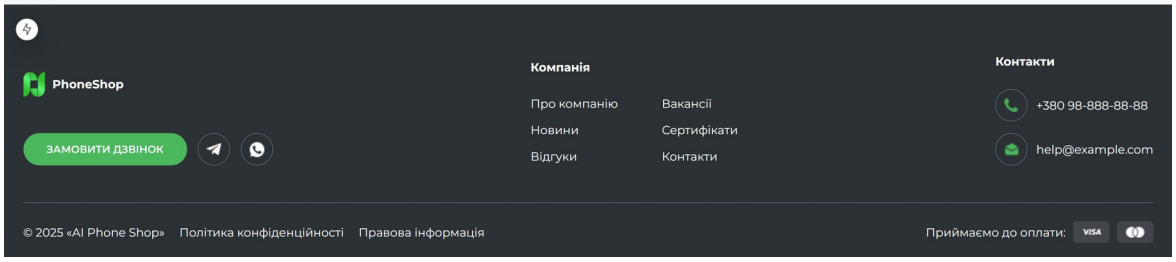


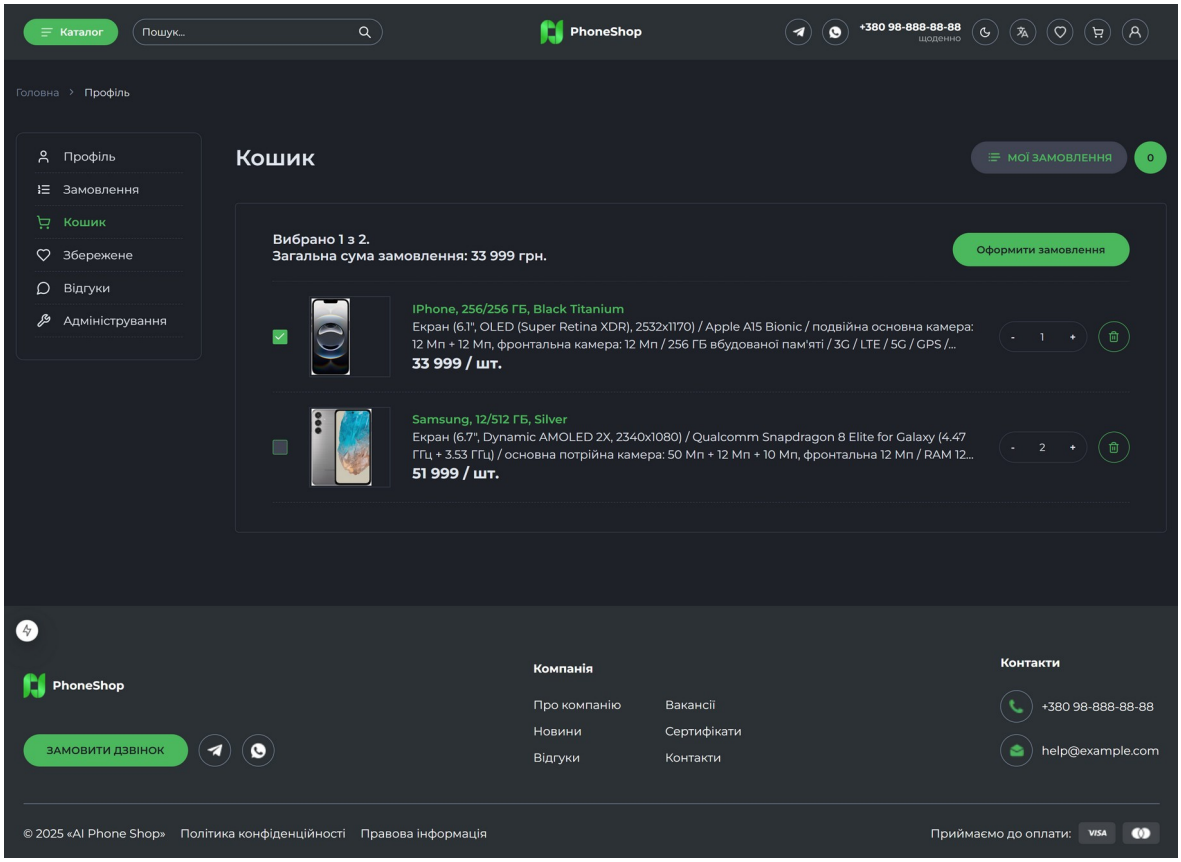


ДОДАТОК АП

Інтерфейс сторінки "Кошик"

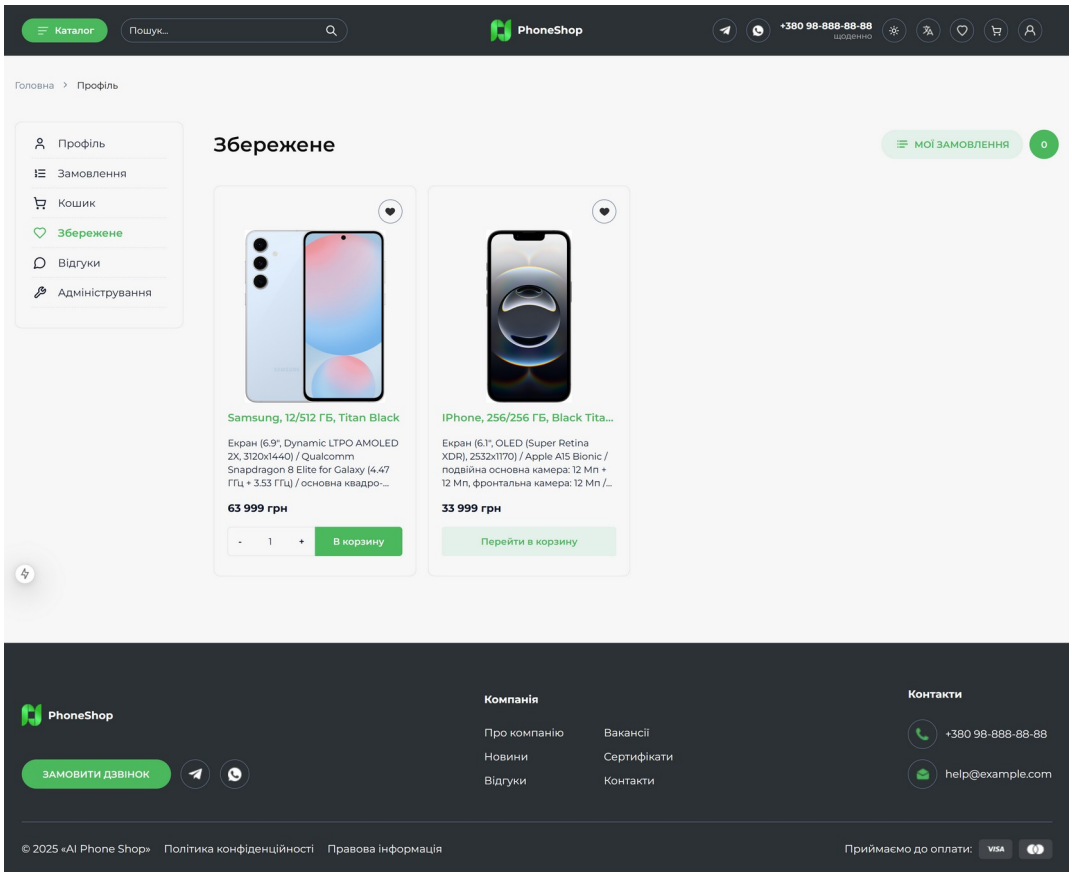


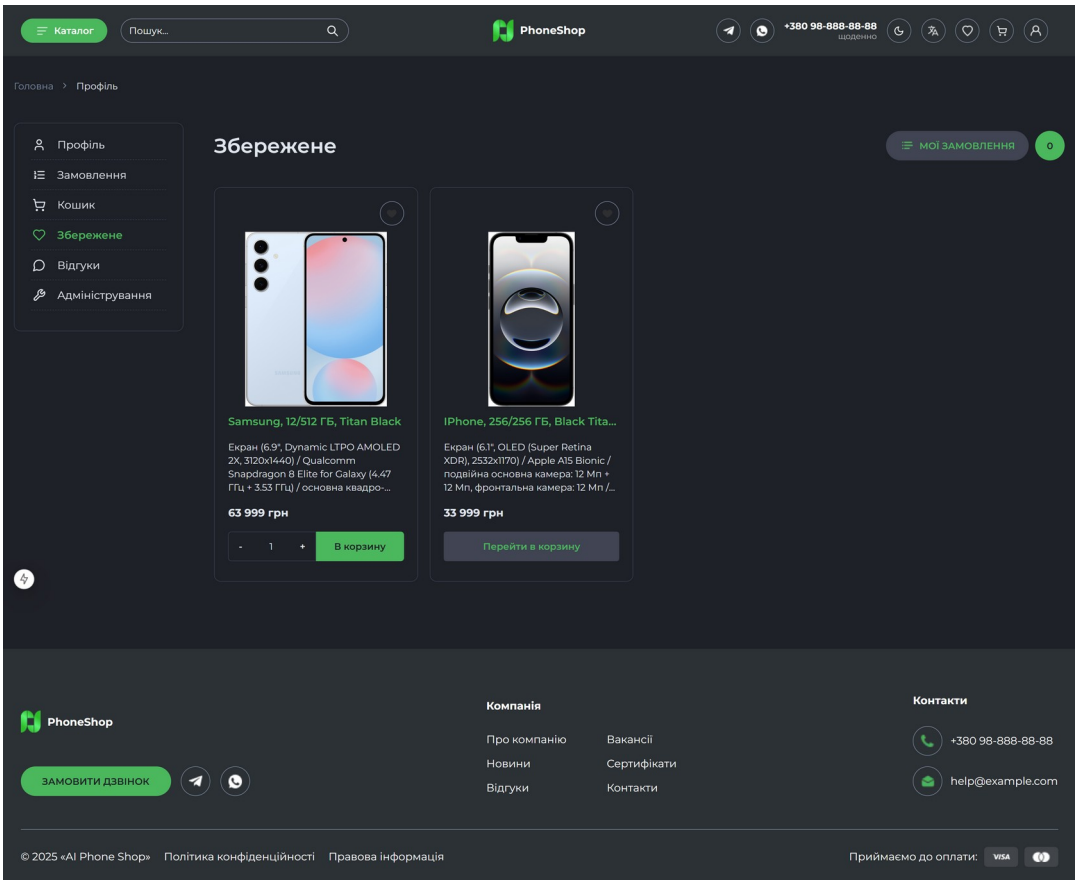




ДОДАТОК АР

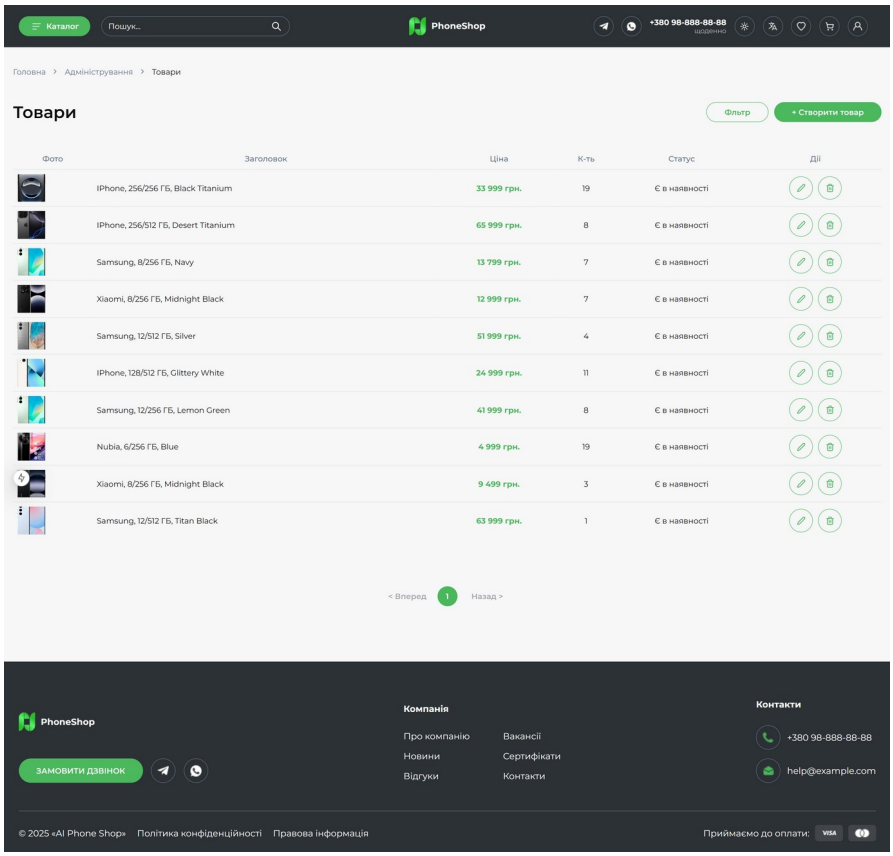
Інтерфейс сторінки "Збережені товари"

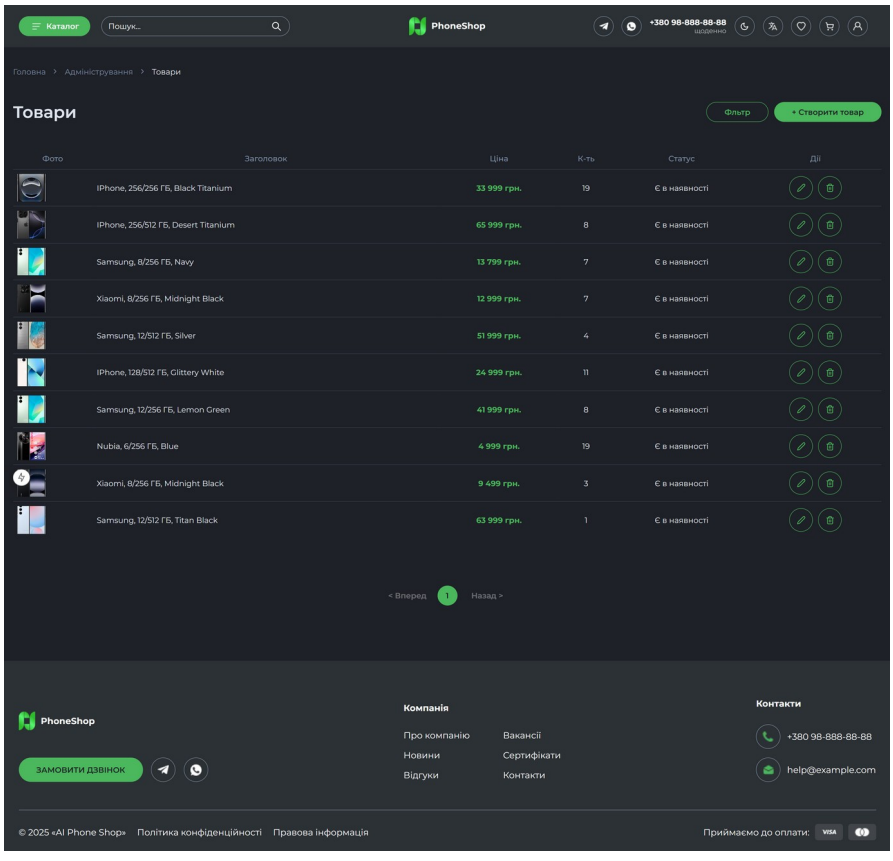




ДОДАТОК АС

Інтерфейс сторінки "Адміністрування"





ДОДАТОК АТ

Інтерфейс сторінки "Аналітика"