МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Навчально-науковий інститут інформаційних технологій та бізнесу
Кафедра інформаційних технологій та аналітики даних

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Проєктування та розробка інтернет-магазину косметичних засобів з фільтрами за складом продуктів та системою персоналізованих рекомендацій»**

Виконала: студентка 4 курсу, групи КН-41
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Денищук Вікторія Сергіївна

Керівник: *старший викладач кафедри інформаційних технологій та аналітики даних*
Клебан Юрій Вікторович

Рецензент: *Розробник програмного забезпечення в ТОВ ОБНОВА-ЄВРОШОП, викладач кафедри інформаційних технологій та аналітики даних НаУОА*
Шведюк Володимир Володимирович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ
Завідувач кафедри інформаційних технологій та аналітики даних _____
(проф., д.е.н. Кривицька О.Р.)
Протокол №10 від 28 травня 2025 р.

Острог, 2025

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Проєктування та розробка інтернет-магазину косметичних засобів з фільтрами за складом продуктів та системою персоналізованих рекомендацій.

Автор: Денищук Вікторія Сергіївна.

Науковий керівник: старший викладач кафедри економіко-математичного моделювання та інформаційних технологій Клебан Юрій Вікторович.

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: 69 с., 14 рис., 1 табл., 3 додатки, 40 джерел.

Ключові слова: інтернет-магазин, персоналізовані рекомендації, фільтрація за складом, штучний інтелект, голосовий помічник, ASP.NET Core, Web Speech API, PostgreSQL, Mistral, Ollama, Entity Framework Core, чат-бот.

Короткий зміст праці:
У кваліфікаційній роботі розроблено сучасну інформаційну систему для інтернет-магазину косметичних засобів з підтримкою фільтрації товарів за складом, типом шкіри та іншими характеристиками, а також з реалізацією системи персоналізованих рекомендацій. Особливу увагу приділено покращенню доступності та зручності користування завдяки інтеграції голосового помічника та чат-підтримки на основі штучного інтелекту, що особливо корисно для людей з обмеженими можливостями.
Застосунок розроблено на основі ASP.NET Core із використанням Entity Framework Core та бази даних PostgreSQL. Він забезпечує фільтрацію косметичних товарів за складом, типом шкіри та ефектами, а також перевірку безпечності інгредієнтів. У систему інтегровано голосовий помічник на базі Web Speech API, що дозволяє користувачам керувати інтерфейсом голосом, та чат-бот, побудований на локальній мовній моделі Mistral із використанням платформи Ollama, який забезпечує інтелектуальну підтримку в реальному часі. Також реалізовано персоналізовані рекомендації, функцію збереження улюблених товарів, історію чатів, автоматичне очищення застарілих даних та рольову систему доступу.
Розроблена система поєднує гнучку архітектуру, інклюзивність, зручність використання та сучасні підходи у сфері цифрових технологій і штучного інтелекту.

_____

ANNOTATION
of qualification paper
for bachelor's degree

Title: System Design and Development of an Online Store for Cosmetic Products with Ingredient-Based Filtering and a Personalized Recommendation System.

Author: Viktoriia Serhiivna Denyshchuk

Scientific advisor: Senior Lecturer of the Department of Economic-Mathematical Modeling and Information Technologies, Yurii Viktorovych Kleban.

Defended «……»…… 2025.

Explanatory note to the qualification thesis: 69 pages, 14 figures, 1 tables, 3 appendices, 40 references.

Keywords: online store, personalized recommendations, ingredient-based filtering, artificial intelligence, voice assistant, ASP.NET Core, Web Speech API, PostgreSQL, Mistral, Ollama, Entity Framework Core, chatbot.

Abstract:
This bachelor's thesis presents the development of a modern information system for an online cosmetics store that supports product filtering by ingredients, skin type, and other characteristics, as well as the implementation of a personalized recommendation system. Special attention is given to improving accessibility and user convenience through the integration of a voice assistant and AI-powered chat support, which is particularly beneficial for users with disabilities.
The application is developed using ASP.NET Core with Entity Framework Core and a PostgreSQL database. It provides filtering of cosmetic products by ingredients, skin type, and desired effects, along with safety checks for ingredients. The system integrates a voice assistant based on the Web Speech API, allowing users to control the interface via voice commands, and a chatbot built on the local Mistral language model using the Ollama platform, offering real-time intelligent assistance.
Additionally, the system includes personalized product recommendations, a favorites feature, chat history management, automatic data cleanup, and a role-based access control mechanism.
The developed system combines flexible architecture, inclusiveness, ease of use, and state-of-the-art approaches in digital technologies and artificial intelligence.

_____

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

API	Application Programming Interface — набір правил і методів, за допомогою яких окремі компоненти програм можуть взаємодіяти між собою.
ASP.NET Core	Фреймворк від Microsoft, що використовується для створення високопродуктивних вебзастосунків і сервісів.
CRUD	Create, Read, Update, Delete — чотири основні операції, які використовуються при роботі з інформацією в базах даних.
ID	Application PIdentifier — унікальне цифрове або символьне позначення, що дозволяє ідентифікувати елемент у системі.rogramming Interface — набір правил і методів, за допомогою яких окремі компоненти програм можуть взаємодіяти між собою.
JSON	JavaScript Object Notation — зручний формат обміну структурованими даними між клієнтом і сервером.
LINQ	Language Integrated Query — мова запитів, інтегрована в C#, яка дозволяє писати запити до колекцій у вигляді звичайного коду.
LLM	Large Language Model — потужна модель штучного інтелекту, здатна генерувати текст і відповідати на запити природною мовою.
Ollama	Платформа, що дає змогу запускати великі мовні моделі локально, без підключення до хмари.

ORM	Object-Relational Mapping — спосіб взаємодії з базою даних через об'єкти програмного коду, що відповідають таблицям.
PostgreSQL	Сучасна система управління базами даних з відкритим кодом, яка підтримує складні запити та об’єктно-реляційний підхід.
.NET	Універсальна платформа від Microsoft для розробки програмного забезпечення на різні операційні системи та пристрої.
DOM	Document Object Model — модель, що представляє HTML або XML-документ у вигляді дерева об'єктів, з якими можна взаємодіяти через скрипти.
AI	Artificial Intelligence — галузь комп’ютерних наук, що займається створенням систем, здатних виконувати завдання, які потребують штучного інтелекту.
UML	Unified Modeling Language — стандартизована мова для побудови діаграм, які описують структуру та поведінку програмних систем.
БД	База даних — структурований набір інформації, що зберігається та керується за допомогою СУБД.
СУБД	Система управління базами даних — програмне забезпечення для створення, обслуговування та взаємодії з базами даних.

ВСТУП

У наш час онлайн-шопінг набуває все більшої популярності і активно розвивається ринок косметичних засобів. З кожним роком зростає кількість людей, які обирають купувати засоби догляду за шкірою, волоссям та декоративну косметику через Інтернет. Це швидко, зручно і дозволяє порівнювати продукти, не виходячи з дому. Проте водночас виникає і низка труднощів — надмірна кількість товарів, відсутність детальної інформації про склад, алергенність або сумісність з конкретним типом шкіри ускладнюють вибір.

Розробка інтернет-магазину з можливістю фільтрації косметики за складом, персональними особливостями користувача, а також із голосовим помічником та інтегрованим чатом дозволяє зробити процес вибору більш зручним, безпечним та персоналізованим. Такий підхід не тільки спрощує пошук потрібного товару, а й дозволяє уникнути небажаних реакцій, алергій або неефективного використання косметики.

Косметичний ринок є одним з найбільших і найдинамічніших у світі. Для покупців важливо не просто знайти продукт за хорошою ціною, а впевнено обрати саме той, що підходить за складом і характеристиками. Особливо це актуально для людей із чутливою шкірою, індивідуальними особливостями чи конкретними вимогами до компонентів продукції. У такому середовищі важливо мати онлайн-платформу, яка враховує не тільки тип продукту, але й потреби конкретного користувача.

Мета цього рішення не просто продавати косметику, а створити зручний онлайн-простір, де кожен зможе швидко знайти те, що потрібно саме йому. Для досягнення цієї мети потрібно реалізувати такі ключові завдання:

- зібрати та проаналізувати інформацію про сучасні підходи до створення інтернет-магазинів у сфері косметики;
- спроєктувати логіку взаємодії користувача з системою: від пошуку до купівлі;

- реалізувати зручну систему фільтрації товарів за складом;
- впровадити персоналізовану систему рекомендацій;
- розробити чат на базі штучного інтелекту;
- додати голосового асистента.

Об’єктом дослідження є вебзастосунок інтернет-магазин косметичних засобів з інтелектуальними функціями.

Предметом дослідження виступають методи реалізації персоналізованих рекомендацій, алгоритми фільтрації товарів за складом, технології забезпечення цифрової доступності та інструменти для інтеграції голосового помічника.

Отже, створення подібного онлайн-магазину дозволяє не лише зробити косметичний шопінг більш зручним і ефективним, а й забезпечує сучасний рівень взаємодії з користувачем завдяки новітнім технологіям. Такий підхід розширює можливості покупців та робить вибір продукції більш свідомим, безпечним і персоналізованим.

РОЗДІЛ 1. ТЕОРЕТИЧНІ АСПЕКТИ ПРОЄКТУВАННЯ ТА РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ КОСМЕТИЧНИХ ЗАСОБІВ

1.1. Опис предметного середовища інтернет-магазину косметики

Все далі інтернет-магазини набувають популярності та стають одним із основних каналів збуту для компаній, які працюють у сфері космеичних засобів. Онлай-продажі дозволяють охопити широку аудиторію та надають клієнтам зручність у виборі товарів, особливо за допомогою персоналізованих інструментів, що взаємодіють з покупцями.

Інтернет-магазин косметичних засобів є веб-додатком, що спеціалізується на продажу продукції для догляду за шкірою, волоссям та тілом. Це сучасний інструмент електронної комерції, який дозволяє користувачам обирати, купувати й отримувати товари через інтернет.

Предметним середовищем у даному випадку слугує ринок онлайн-продажу косметичних виробів. Ключовими учасниками даного процесу виступають: користувачі (покупці), платформа (вебсайт), база даних продуктів, модулі персоналізації, голосовий асистент та чат-бот.

Автоматизації підлягає процес пошуку, підбору та купівлі косметики, враховуючи індивідуальні особливості клієнта. Планується створити вебдодаток, який дає можливість не тільки придбати товар, а й полегшує його вибір шляхом фільтрації за складом, персоналізованими порадами та інтерактивною підтримкою.

За результатами дослідження інформації зі сторінки “Економічна правда” було виявлено, що 80% населення України віком від 16 років користуються інтернетом і з них приблизно 36% здійснюють покупки через інтернет. Найчастіше купують онлайн одяг (47%), побутову техніку і електроніку (46%), а також косметику і парфумерію (37%) [1]. На основі цих даних можна зрозуміти, що тема онлайн-магазину косметичних засобів є досить актуальною в наш час.

Згідно з даними компанії Meest Shopping, косметика посідає третє місце серед найпопулярніших товарних категорій в інтернет-магазинах України (рис. 1.1). Частка покупців, які замовляють косметичну продукцію онлайн, становить 48%, причому переважають жінки віком від 25 до 34 років, які складають 65% загальної аудиторії онлайн-шопінгу [2].

Рис. 1.1. Популярність покупок в інтернет-магазинах України(2023-2024)

Джерело:[2]

Таким чином, можна зазначити, що предметне середовище інтернет-магазину косметичних засобів є комплексною системою, яка поєднує різноманітні цифрові інструменти та платформи персоналізації. Успішне функціонування застосунку залежить не лише від дизайну або асортименту, а від здатності забезпечити ефективну й зручну взаємодію з клієнтом на кожному етапі онлайн-покупки.

1.2. Огляд наявних аналогів для інтернет-магазинів косметики

Для порівняння було вибрано два великих бренди косметики, що продаються через інтернет-магазини в Україні: L'Oréal Paris (loreal-paris.ua) і Avon (avon.ua). Ці бренди мають великі асортиментні лінії косметичних засобів, але при цьому не

забезпечують глибокої фільтрації за складом продуктів або персоналізованих рекомендацій, підтримки чату і голосового помічника. Розглянемо детальніше кожен з цих сайтів, а також їхні сильні та слабкі сторони в контексті функціональності. Нижче наведена таблиця 1.1 із порівнянням двох сайтів-аналогів.

Таблиця 1.1

Порівняльна характеристика сайтів-аналогів для запланованого продукту

Назва веб-сайту	**loreal-paris.ua**	**avon.ua**
Адреса веб-сайту	https://loreal-paris.ua	https://avon.ua
Функціональність	Фільтрація за типом шкіри, типом волосся, віковими категоріями	Фільтрація за ціною, алфавітом, рейтингом
Інтерфейс користувача	Простий, але обмежений за можливостями персоналізації	Зрозумілий, простий інтерфейс, але без персоналізації
Персоналізовані рекомендації	Персоналізація за типом шкіри	Обмежена персоналізація за типом шкіри і волосся
Фільтрація за складом	Немає можливості фільтрувати за складом	Немає фільтрації за складом
Можливість виключення небажаних інгредієнтів	Немає	Немає
Чат-підтримка	Відсутня	Відсутня
Голосовий помічник	Відсутній	Відсутній
Додаткові функції	Розширена фільтрація за ціною і брендом	Наявність акцій і знижок

Рис. 1.3. Таблиця порівняльної характеристика сайтів-аналогів

Джерело: розроблено автором

На представлених сайтах реалізована базова система фільтрації за категоріями товарів, типом шкіри, віковими ознаками тощо, але L'Oréal Paris тільки частково підтримує дану функцію (рис. 1.2), а Avon не надає можливості фільтрувати продукти за складом (рис. 1.3), а це є критично важливим для людей з алергією,

чутливою шкірою або для тих, хто свідомо уникають шкідливих інгредієнтів. Запропонований продукт усуває ці недоліки та допомагає користувачеві зробити вибір швидко та безпечно.

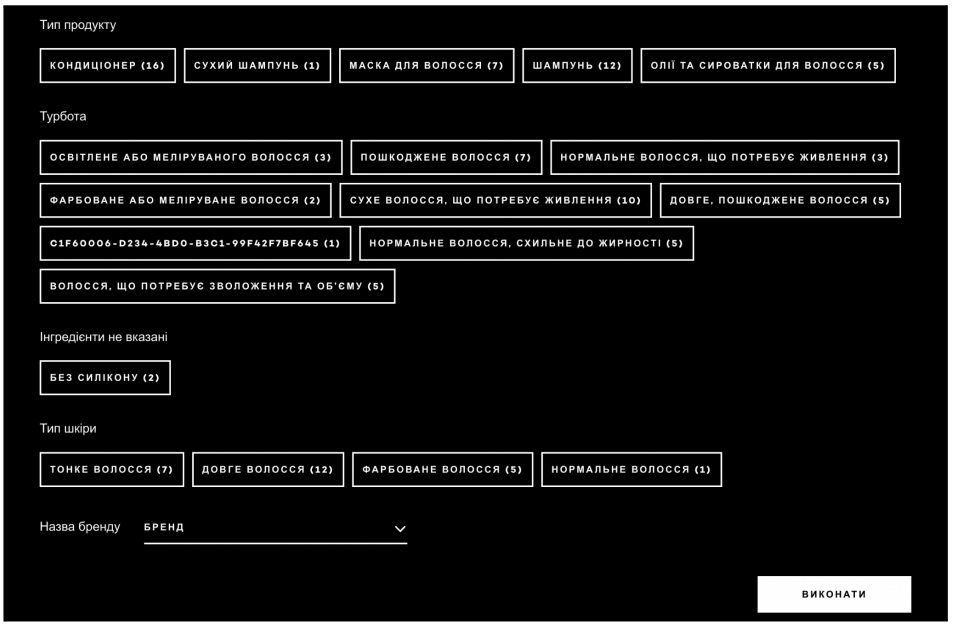

Рис. 1.2. Фільтр веб-сторінки із аналогом, “loreal-paris.ua”

Джерело: https://www.loreal-paris.ua/volossya/doglyad-za-volossyam

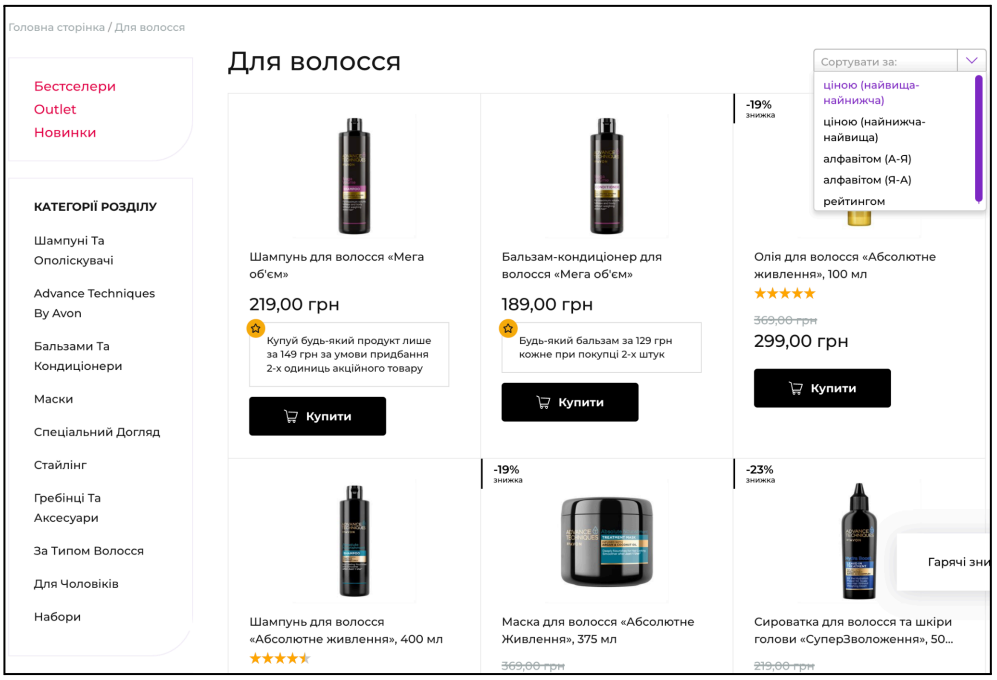

Рис. 1.3. Фільтр веб-сторінки із аналогом, “avon.ua”

Джерело: https://my.avon.ua/341/dlya-volossya

Обидва конкуренти реалізують рекомендації лише за загальними критеріями, а запропонований інтернет-магазин буде враховувати повний профіль користувача включаючи тип шкіри, волосся, бажаної дії продукту, наявність алергій тощо. Це забезпечить якісний підбір засобів, який буде точнішим та кориснішим за існуючі рішення.

Чат з підтримкою на основі штучного інтелекту не передбачено ні в L'Oréal, ні в Avon. У розроблювальному проєкті користувач отримає доступ до чату з віртуальним консультантом, який допомагає у виборі засобів, відповідає на питання про склад, спосіб застосування і також може надавати поради.

Жоден із проаналізованих сайтів не використовує голосовий інтерфейс. Запропонований проєкт містить інтегрованого голосового асистента, який автоматично активується при вході на сайт та пропонує користувачу навігацію голосом. Це забезпечує доступність для людей з вадами зору та покращує зручність користування.

Запланований інтернет-магазин значно перевищує розглянуті аналоги за рахунок поєднання класичних можливостей з інноваційними функціями: голосовим асистентом, персоналізованим підбором, чат-помічником, фільтрацією за складом та можливістю відстежування замовлення. Такий комплекс функцій дозволяє говорити не просто про інтернет-магазин, а про систему орієнтовану на індивідуальний комфорт та безпеку користувача.

1.3. Постановка задачі для розробки інтернет-магазину косметики

Для створення сучасного і функціонально інтернет-магазину косметичних засобів недостатньо реалізувати лише стандартний набір функцій, таких як каталог товарів і корзина. Сучасний користувач очікує значно більше: простоту у використанні, швидкість індивідуальний підхід, розширену інформацію про товари та зручну підтримку. Для досягнення цього, на початку розробки важливо чітко визначити, що саме має робити система, які функції вона повинна мати, яким буде

інтерфейс та як взаємодія з нею буде виглядати для різних типів користувачів. Запропонований проєкт – це не просто онлайн-магазин, а інтелектуальна платформа, яка враховує потреби покупця та пропонує максимально зручний спосіб пошуку, вибору та придбання косметичних засобів.

При створенні інтернет-магазину косметичних засобів з орієнтацією на індивідуальні потреби користувачів виникає низка складних завдань, кожне з яких має свої переваги та недоліки. Розробка такої платформи вимагає комплексного підходу, де кожна функція повинна бути належним чином інтегрована в систему для забезпечення зручності та безпеки користування. Завдання, з якими стикаються розробники, є багатокритеріальними, що передбачає необхідність оцінки та порівняння різних рішень за кількома важливими параметрами одночасно.

Метою розробки є створення веб застосунку, який дозволяє користувачу швидко та зручно підбирати косметичні засоби з урахуванням індивідуальних потреб і складу продуктів, отримувати рекомендації, взаємодіяти із системою голосом або в чаті, оформлювати замовлення та відстежувати його статус. У рамках проєкту система повинна виконувати основні фунуції:

1. Користувач повинен мати можливість створити особистий профіль.
2. Всі товари мають бути структуровані за категоріями. Кожен товар повинен мати докладний опис, склад, зображення та іншу інформацію.
3. Можливість фільтрувати товари не лише за типом шкіри та волосся, а й за інгредієнтами, алергенами, дією продукту.
4. Додавання продуктів які сподобались до списку улюблених.
5. Можливість перевірити інгредієнти та зробити детальний аналіз складу продукту і оцінити його безпечність.
6. Після покупки користувач повинен мати змогу перевірити статус доставки і перевірити на якому етапі перебуває його замовлення.
7. Перегляд списку замовлень та їхні деталі.
8. Сторінка замовення повинна бути інтуїтивно зрозумілою зі зручним інтерфейсом.

9. На сайті повинен бути чат, який працює за допомогою штучного інтелекту та рекомендує користувачеві засоби на основі бази даних представленого інтернет-магазину.
10. При вході на сайт та кліку в будь яке місце, повинен автоматично запускатися голосовий помічник, який буде вітати клієнтів та допомагати їм з будь якими діями на сайті якщо цього забажає користувач. Особливо важливо це реалізувати на сторінці з чатом для того, щоб надавати консультацію та допомагати з вибором продуктів користувачам з обмеженими можливостями.

Окрім основного функціоналу, майбутня система повинна відповідати сучасним стандартам зручності, безпеки та ефективності:

- Зручний інтерфейс. Сайт має бути інтуїтивно зрозумілим з адаптивною верстkою.
- Безпека. Всі пермогалізовані дані користувачів мають бути захищеними.
- Продуктивність. Сторінки повинні відкриватися швидко, а фільтрація працювати майже миттєво.
- Інклюзивність. Сайт повинен бути зручним для людей з обмеженими можливостями.
- Масштабованість. У майбутньому сайт має легко розширюватись: можна буде додати нові функції.

Запланований інтернет-магазин створюється для широкої аудиторії: від звичайних користувачів, які шукають якісну косметику онлайн, до людей з конкретними косметичними потребами або чутливою шкірою. Окрему увагу приділено користувачам, які хочуть або потребують використовувати голосового помічника для зручності, швидкості або через фізичні обмеження.

Реалізація зазначених бізнес-процесів потребує вирішення таких технічних завдань. Зокрема, важливо розробити алгоритми для визначення параметрів фільтрації товарів, що дозволить аналізувати їхній склад і вибирати найбільш відповідні продукти. Крім того, необхідно визначити критерії для персоналізованих рекомендацій, враховуючи тип шкіри, наявність алергій тощо. Дане технічне

завдання охоплює всі ключові вимоги до системи, яка повинна не лише виконувати функції стандартного онлайн-магазину, а й надавати розширені, інноваційні можливості.

Висновки до розділу 1

У даному розділі було виявлено, що ринок онлайн-торгівлі косметикою стрімко зростає, а користувачі все більше потребують зручних та індивідуальних інструментів для здійснення покупок. Особливий акцент зроблено на важливості персоналізації, можливості фільтрації товарів за складом, використанні голосових асистентів, що сьогодні є актуальним і практично відсутнім у більшості конкурентних рішень.

Аналіз існуючих платформ, таких як сайти брендів L'Oréal Paris та Avon, показав обмеженість їх функціональності в контексті індивідуального підбору засобів, відсутність глибокої фільтрації, чату-консультацій та голосової взаємодії. Це відкриває простір для створення сучасного, конкурентного продукту, який буде враховувати потреби широкого кола користувачів, зокрема тих, хто має специфічні запити або потребує додаткової доступності.

Також у межах розділу було сформульовано постановку задачі та технічне бачення майбутнього програмного продукту. Визначено функціональні та нефункціональні вимоги до системи, які мають на меті забезпечити інтуїтивну взаємодію, безпеку, швидкість роботи та масштабованість. Особливу увагу приділено адаптації сайту для людей з особливими потребами та реалізації голосового інтерфейсу.

Отже, на основі проведеного аналізу сформовано обґрунтовану потребу у створенні вебзастосунку, який стане не просто інтернет-магазином, а повноцінним цифровим простором для безпечного, зручного та персоналізованого вибору косметичних засобів.

РОЗДІЛ 2. ПРОЄКТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ ІНТЕРНЕТ-МАГАЗИНУ КОСМЕТИЧНИХ ЗАСОБІВ

2.1. Аналіз предметної області програми

Першим кроком у розробці подібної системи є аналіз предметної області, тобто вивчення того, як повинна працювати система, які її основні складові, які дані вона обробляє, які функції виконує і які обмеження має. Це дозволяє побудувати логічну та зрозумілу структуру, яка в майбутньому стане основою для програмної реалізації. У процесі аналізу було визначено кілька ключових об’єктів, з якими взаємодіє як користувач, так і система. Серед них:

- Користувач
- Профіль користувача
- Продукт
- Інгредієнти
- Замовлення
- Голосовий асистент
- Чат-помічник

Процес взаємодії користувача з інтернет-магазином косметичних засобів є послідовним і логічно структурованим. Його метою є забезпечити комфортний підбір товару за індивідуальними потребами користувача, а також спрощення шляху до здійснення покупки. У межах функціональної моделі системи основну роль відіграє процес автоматизації, який забезпечує швидкість та ефективність дій. Однією з ключових тенденцій є персоналізація, система автоматичного підбору косметичних засобів, враховуючи індивідуальні особливості клієнта, такі як тип шкіри, алергії, уподобання за складом тощо. У сучасному інтернет-магазині користувач не лише обирає товар, а й отримує рекомендації, підтримку консультанта через чат або голосову систему, переглядає склад продукції у зрозумілому форматі. Це дозволяє підвищити рівень задоволення клієнта та зменшити кількість повернень.

Для наочності нижче наведено UML-діаграму використання, яка відображає основні взаємодії користувача із системою (рис. 2.1).

Рис 2.1. Використання інтернет-магазину косметики

Джерело: створено автором

Інформаційна модель системи демонструє, як саме користувач може взаємодіяти з сайтом. UML-діаграма варіантів використання (рис. 2.2) дозволяє зобразити, які дії доступні користувачеві та як ці дії зв’язуються з логікою системи.

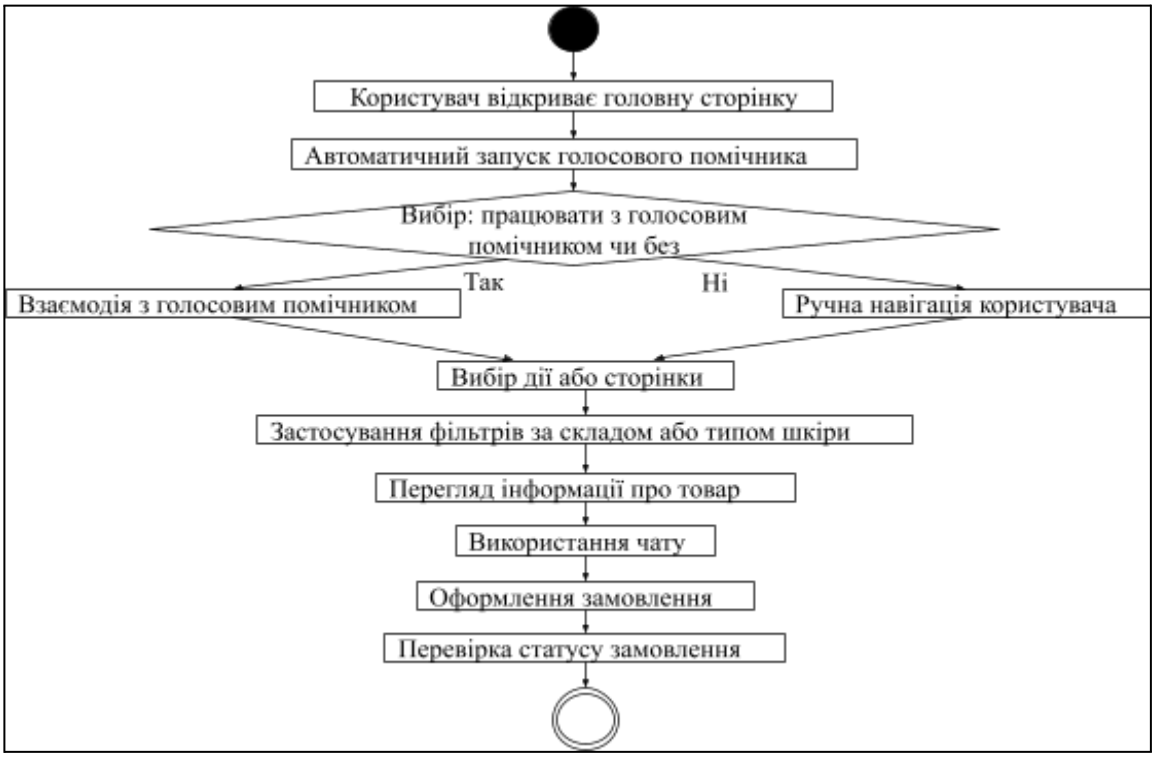

Рис. 2.2. UML-діаграма діяльності користувача в інтернет-магазині косметики

Джерело: створено автором

Даний процес починається з відкриття головної сторінки сайту та закінчується оформленням замовлення. Кожен крок має чітке призначення та виконує певну функцію в логіці системи. Визначення та опис етапів:

1. Відкриття головної сторінки. Першим етапом є вхід користувача на головну сторінку магазину. На даному етапі важливо забезпечити зрозумілий та простий інтерфейс.
2. Голосовий помічник. Одразу при запуску сайту повинен включатися голосовий помічник, який пропонує користувачеві вибіж голосовим керуванням чи ручним. Ця функція допомагає людям з обмеженими можливостями і може супроводжувати виконання всіх дій голосом.
3. Вибір дії або сторінки. Залежно від попереднього кроку користувач переходить до потрібного розділу, наприклад: “Продукти”, “Замовлення”, “Улюблені”. Це також має бути реалізовано через голосову команду.
4. Застосування фільтрів. Користувач налаштовує фільтри, щоб відібрати засоби за потрібними критеріями: складниками, типом шкіри, типом волосся, дією засобу тощо. Це допомагає зменшити кількість товарів до найбільш релевантних варіантів.
5. Перегляд інформації про товар. Обравши потрібний товар, користувач відкриває його детальну інформацію в якій представлено опис, інгредієнти, ефекти, ціна та інше. Для підвищення доступності ця інформація може озвучуватися голосом.
6. Використання чату. При виникненні запитань, користувач може звернутися до чат-бота з підтримкою штучного інтелекту. Чат підтримує типові запити (допомога з вибором, уточнення складу, косметичні поради). Ці дії також виконуються із допомогою голосового помічника.
7. Оформлення замовлення. Після вибору товару користувач переходить до оформлення покупки. Тут важливо забезпечити швидкий та безпечний процес.

8. Перевірка статусу замовлення. Після завершення покупки користувач має можливість перейти спеціального розділу, де можна відстежити статус доставки та переглянути деталі замовлення.

2.2. Створення та моделювання бази даних застосунку

База даних — це організоване електронне середовище для збереження та обробки інформації. Вона виступає як цифрове сховище, в якому зберігаються відомості про різноманітні об’єкти та явища, що можуть бути логічно пов’язані між собою. До таких об’єктів належать, зокрема, користувачі, продукти, замовлення або інші сутності, які відображають реальні чи віртуальні процеси. Основною метою використання бази даних є забезпечення швидкого доступу до інформації, її надійного зберігання, захисту та цілісності в рамках роботи інформаційної системи [3, 4].

Одним із ключових етапів розробки інформаційної системи є побудова правильної структури бази даних. У контексті розробки онлайн-платформи для продажу косметичних засобів було поставлено завдання створити ефективну, масштабовану та логічно зв’язану базу даних. Потрібно обрати базу даних, яка буде взаємодіяти з додатком через Entity Framework Core. Забезпечити зберігання основних сутностей, таких як продукти, користувачі, замовлення, інгредієнти, що входять до складу товарів, а також визначити та додати інші необхідні елементи. Моделі, які відповідають таблицям, мають бути розроблені з урахуванням правил відображення даних, що дозволяє зручно працювати з ними на рівні програмного коду [5].

Для реалізації цього завдання було обрано СУБД PostgreSQL, що забезпечує стабільну та надійну роботу з великими обсягами даних [6, 7]. Як інструмент зв’язку між додатком і базою даних було використано Entity Framework Core — сучасний ORM-фреймворк, який дозволяє працювати з базою на рівні об’єктів та значно спрощує обробку даних [8]. Створення структури бази відбувалося за допомогою

механізму міграцій, що дало змогу поступово розширювати систему без втрати даних.

Під час проєктування були визначені основні сутності, які відповідають ключовим аспектам діяльності інтернет-магазину:

1. Product зберігає інформацію про товари. Вона включає такі властивості: назва продукту (Name, string), опис (Description, string), ціна (Price, decimal), шлях до зображення (CoverPath, string) та сам файл зображення (CoverFile, IFormFile). Ця сутність є основною для зберігання даних про косметичні товари в базі.
2. User зберігає дані про користувачів і використовує ASP.NET Identity для автентифікації. Вона містить такі властивості, як email користувача (Email, string), ім’я користувача (UserName, string) та пароль (Password, string), що дозволяють ідентифікувати користувача в системі.
3. Order описує замовлення, яке було оформлене користувачем. Властивості цієї сутності включають дату створення замовлення (OrderDate, DateTime), статус замовлення (Status, string), загальну суму замовлення (TotalAmount, decimal) та адресу доставки (ShippingAddress, string). Ці дані допомагають відстежувати прогрес замовлення та обробку доставки.
4. OrderItem використовується для зберігання інформації про товари, що входять до замовлення. До неї відносяться такі властивості, як ідентифікатор продукту (ProductId, int), кількість товару (Quantity, int), ціна за одиницю (UnitPrice, decimal) та загальна ціна (TotalPrice, decimal). Ця сутність дозволяє точно обчислити вартість замовлення.
5. Ingredient описує інгредієнти, які входять до складу продуктів. Властивості цієї сутності включають назву інгредієнта (Name, string), категорію (Category, string), рівень небезпеки (LevelOfDanger, int), а також чи є інгредієнт шкідливим (IsHarmful, bool) та опис інгредієнта (Description, string). Інформація про інгредієнти дозволяє користувачам перевіряти склад продуктів на наявність небажаних компонентів.

6. Allergen зберігає дані про алергени. Вона має таку властивість, як назва алергену (Name, string), що допомагає виявляти потенційно небезпечні компоненти для людей з алергіями.
7. Category містить інформацію про категорії продуктів. Властивість назви категорії (NameCategory, string) дозволяє організовувати продукти за певними типами, що покращує пошук та фільтрацію товарів у системі.
8. ProductRecommendation зберігає рекомендації продуктів для користувачів. Вона містить ідентифікатор користувача (UserId, string) та ідентифікатор продукту (ProductId, int), що дозволяє автоматично рекомендувати товари на основі даних користувача.
9. UserPreferences описує уподобання користувачів щодо продуктів. Властивості цієї сутності включають ідентифікатор користувача (UserId, string), тип волосся (HairType, string), тип шкіри (SkinType, string), а також чи уникає користувач алергенів (AvoidAllergens, bool) і список алергенів, яких він уникає (AvoidedAllergens, string). Це дозволяє створити персоналізовані рекомендації, що відповідають потребам кожного користувача.
10. EffectType описує ефекти, які продукти можуть мати на користувачів. Властивість назви ефекту (Name, string) допомагає визначити, чи підходить продукт для певних цілей, таких як зволоження або захист.
11. SuitableFor визначає, для кого підходить конкретний продукт. Властивість назви (Name, string) вказує, для якого типу користувачів призначено використання продукту (наприклад, для людей з чутливою шкірою).
12. ChatHistory зберігає історію взаємодій користувача з чат-ботом. До властивостей належать ідентифікатор (Id, int), ідентифікатор користувача (UserId, string), повідомлення користувача (UserMessage, string), відповідь ШІ (AIResponse, string), а також мітка часу (Timestamp, DateTime). Ця таблиця дозволяє відстежувати історію запитів і відповідей, що корисно для покращення сервісу та персоналізації обслуговування.

13. Favorite реалізує функцію "обраних" товарів. Сутність містить ідентифікатор (Id, int), ідентифікатор користувача (UserId, string) та ідентифікатор продукту (ProductId, int). Це дає змогу користувачам зберігати улюблені товари для подальшого перегляду або купівлі.

Усі проміжні таблиці, такі як ProductAllergen та ProductIngredient, використовуються для збереження зв’язків між продуктами та алергенами або інгредієнтами, що дозволяє зберігати інформацію про склад кожного товару та його взаємодію з певними алергенами або інгредієнтами.

Важливим етапом є також встановлення зв’язків між таблицями (один до багатьох, багато до багатьох, багато до одного) [9]. Для створення та модифікації таблиць потрібно використовувати інструменти міграції Entity Framework. Зведений опис зв'язків між сутностями:

Product має зв'язок багато до одного з Category, оскільки кожен продукт належить до певної категорії. Також вона має зв'язок багато до багатьох з Ingredient, оскільки продукт може містити багато інгредієнтів. Крім того, існує зв'язок багато до багатьох з Allergen, що означає, що продукт може містити кілька алергенів. Окремо, зв'язок багато до багатьох з ProductRecommendation дозволяє продукту бути рекомендованим багатьом користувачам. І нарешті, зв'язок багато до одного з OrderItem означає, що продукт може бути частиною багатьох замовлень.

User має зв'язок один до багатьох з Order, що дозволяє користувачеві мати кілька замовлень. Вона також пов'язана зв'язком один до багатьох з ProductRecommendation, оскільки користувач може мати кілька рекомендацій продуктів. Крім того, кожен користувач має одне налаштування уподобань, що визначається зв'язком один до одного з UserPreferences.

Order має зв'язок один до багатьох з OrderItem, оскільки одне замовлення може містити багато товарів. Вона також має зв'язок багато до одного з User, що означає, що замовлення належить одному користувачеві.

OrderItem має зв'язок багато до одного з Order, оскільки кожен елемент замовлення належить певному замовленню. Також цей елемент пов'язаний з Product через зв'язок багато до одного.

Ingredient має зв'язок один до багатьох з ProductIngredient, оскільки один інгредієнт може бути частиною кількох продуктів.

Allergen має зв'язок багато до багатьох з Product, що вказує на можливість зв'язку одного алергену з кількома продуктами.

Category має зв'язок один до багатьох з Product, що означає, що одна категорія може містити кілька продуктів.

ProductRecommendation пов'язана з Product та User через зв'язок багато до одного, що означає, що кожна рекомендація належить певному продукту і користувачу.

ProductAllergen зв'язок багато до одного з Product та Allergen, де кожен зв'язок вказує на конкретний продукт або алерген.

ProductIngredient має зв'язок багато до одного з Product та Ingredient, оскільки кожен зв'язок вказує на конкретний продукт та інгредієнт.

EffectType має зв'язок один до багатьох з Product, оскільки один тип ефекту може бути застосований до кількох продуктів.

SuitableFor також має зв'язок один до багатьох з Product, оскільки один тип підходу може бути застосований до кількох продуктів.

ChatHistory має зв'язок багато до одного з User, оскільки кожен запис у чаті створюється конкретним користувачем, а один користувач може мати багато записів у чаті. Це дозволяє фіксувати історію спілкування кожного користувача окремо та зберігати повний журнал взаємодії з системою.

Favorite має два основні зв'язки: багато до одного з User та багато до одного з Product. Це означає, що один користувач може додавати багато товарів до обраного, а один продукт може бути обраний багатьма користувачами. Такий підхід дозволяє гнучко реалізувати функцію «Улюблених товарів», яка є важливою для покращення користувацького досвіду та швидкого доступу до вподобаних позицій.

Отже, було реалізовано всі необхідні таблиці та зв’язки для інтернет-магазину косметичних засобів (рис. 2.3.)

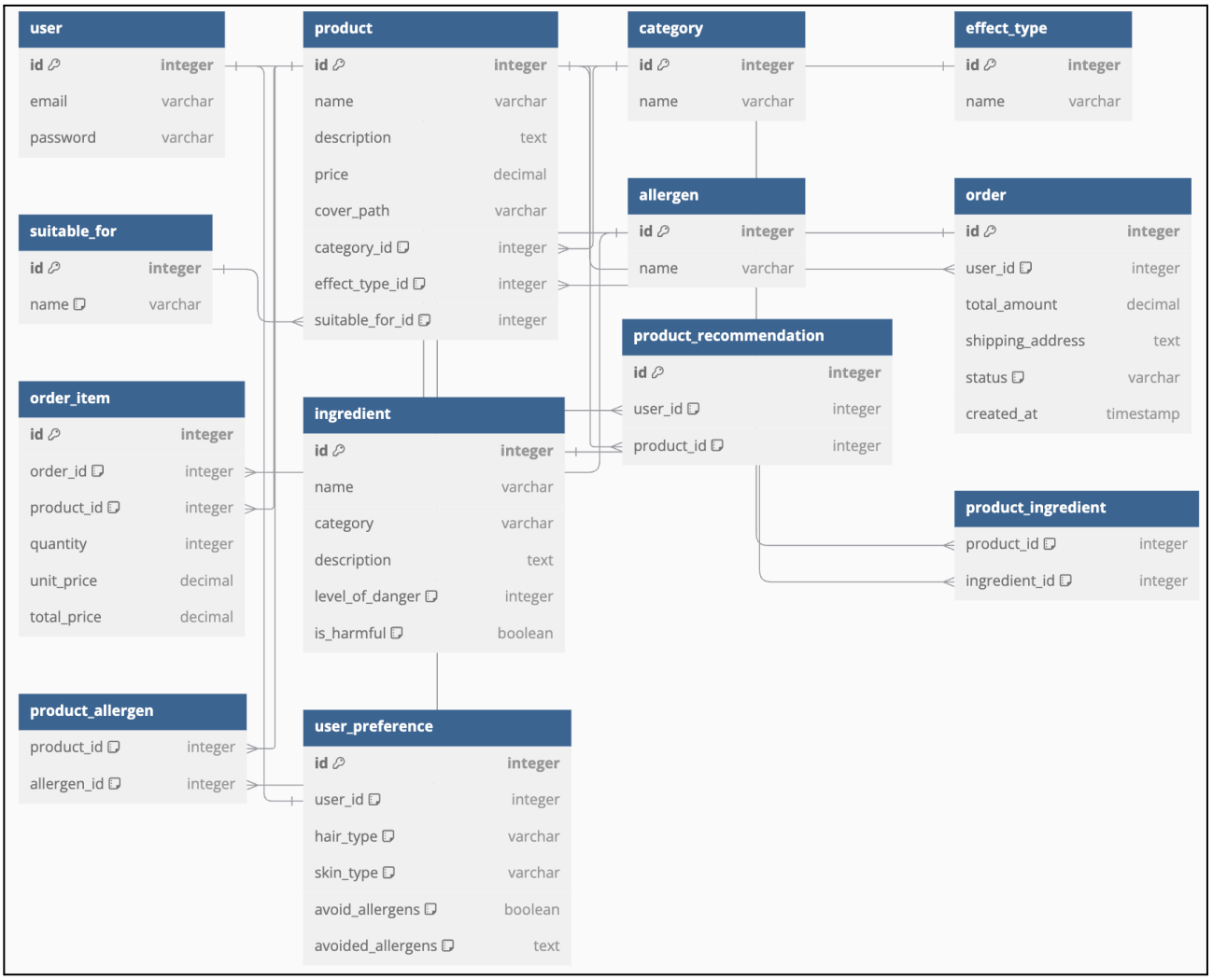

Рис. 2.3. Діаграма таблиць та зв’язків

Джерело: розроблено автором

У межах реалізації бази даних для інтернет-магазину косметичних засобів було створено низку моделей сутностей, кожна з яких відповідає певному об’єкту предметної області. Однією з ключових моделей є Product, що описує структуру зберігання даних про товари в системі. Нижче наведено приклад реалізації цієї сутності.

Лістинг 2.1. Модель сутності Product.

```
public class Product
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required]
        [MaxLength(200)]
        public string Name { get; set; }
        public string Description { get; set; }

        public decimal Price { get; set; }

        public string? CoverPath { get; set; } = "\\img\\product\\no_cover.jpg";
        [NotMapped]
        public IFormFile? CoverFile { get; set; }

        public virtual Category? Category { get; set; }
        public int? CategoryId { get; set; }

        public int? EffectTypeId { get; set; }
        public virtual EffectType? EffectType { get; set; }

        public int? SuitableForId { get; set; }
        public virtual SuitableFor? SuitableFor { get; set; }

        public virtual ICollection<ProductIngredient>? ProductIngredients { get;
set; }
        public virtual ICollection<ProductAllergen>? ProductAllergens { get; set;
}

        public virtual ICollection<OrderItem>? OrderItems { get; set; }

        public virtual ICollection<ProductRecommendation>? ProductRecommendations
{ get; set; }
    }
```

Висновки до розділу 2

У другому розділі було здійснено проєктування серверної частини інтернет-магазину косметичних засобів, що включає аналіз предметної області, моделювання взаємодії користувача із системою, а також створення ефективної бази даних для збереження та обробки даних.

На основі аналізу функціональності інтернет-магазину були виявлені ключові об'єкти, які забезпечують індивідуальний підхід до кожного користувача: профіль, замовлення, чат-асистент, система персоналізованих рекомендацій тощо. За допомогою UML-діаграм наочно представлено логіку взаємодії користувача з функціоналом сайту, що дозволило сформувати чітке уявлення про поведінку системи.

У результаті моделювання бази даних було визначено основні сутності та реалізовано логічні зв’язки між ними (один-до-багатьох, багато-до-багатьох тощо). Використання PostgreSQL як СУБД та Entity Framework Core як ORM-фреймворку забезпечило масштабованість, зручність у розробці та підтримку міграцій, що спрощує керування схемою бази даних у процесі розвитку системи.

Даний розділ закладає надійну основу для реалізації серверної логіки, що відповідає сучасним вимогам щодо продуктивності, зручності використання та безпеки в електронній комерції.

РОЗДІЛ 3. ПРИКЛАДНА ЧАСТИНА РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ КОСМЕТИЧНИХ ЗАСОБІВ

3.1. Технічний стек та технології розробки

Основною мовою програмування обрано C#, яка працює у зв'язці з фреймворком ASP.NET Core. Це дозволяє створювати швидкі, масштабовані та захищені веб-застосунки, що є ідеальним рішенням для реалізації інтернет-магазину. ASP.NET Core забезпечує зручну архітектуру та інтеграцію з сучасними API та чудову продуктивність у хмарному середовищі. Побудова REST API реалізована з урахуванням принципів ASP.NET Core, що дозволяє забезпечити масштабованість, безпеку та простоту підтримки [10].

Для взаємодії з базою даних було використано Entity Framework Core — об'єктно-реляційний ORM-фреймворк, який підтримує підхід code-first [11]. Це дозволяє спочатку створити класи моделей у C#, а вже потім автоматично сформувати структуру бази даних. Такий підхід детально описаний у [12]. Міграції здійснюються через консольні команди, що дає змогу легко оновлювати схему БД у разі змін. Адміністрування бази здійснювалося за допомогою інструменту pgAdmin, який забезпечує графічний інтерфейс для роботи з таблицями, запитами та зв'язками.

Для розробки голосового інтерфейсу використано JavaScript у поєднанні з Web Speech API, що включає технології Speech Recognition (розпізнавання голосу) та Speech Synthesis (озвучення відповідей). Такий підхід дозволяє реалізувати голосову навігацію прямо в браузері, не потребуючи встановлення додаткових плагінів або розширень [13, 14].

Окрему увагу було приділено вибору технології для реалізації інтелектуального чат-бота та системи рекомендацій[15]. Для цього було обрано зв'язку Mistral + Ollama, що дозволяє розгорнути мовну модель локально без необхідності звернення до зовнішніх API. Mistral — це сучасна, відкрита модель

штучного інтелекту, оптимізована для швидкості роботи і помірних обчислювальних ресурсів. Завдяки підтримці локального розгортання вона забезпечує конфіденційність даних користувачів, високу швидкодію та стабільну інтеграцію у веб-додаток. Платформа Ollama спрощує процес запуску та управління LLM-моделями локально, забезпечуючи простий інтерфейс для інтеграції в архітектуру застосунку[16].

В якості середовища розробки використано Microsoft Visual Studio, яке надає всі необхідні інструменти для створення, налагодження, тестування і деплою ASP.NET Core застосунків.

3.2. Реалізація CRUD-операцій для управління даними

У межах розробки інформаційної системи для інтернет-магазину косметичних засобів ключовим функціональним блоком є реалізація CRUD-операцій: створення (Create), читання (Read), оновлення (Update) та видалення (Delete) даних. Ці операції формують основу усієї логіки взаємодії користувача з веб-застосунком, оскільки дозволяють виконувати основні дії з інформацією, яка зберігається в базі даних: додавати нові товари, редагувати їхні властивості, переглядати доступні продукти або замовлення, а також видаляти непотрібні записи.

Кожна із CRUD-операцій реалізується у вигляді окремих методів у репозиторіях (див. Додаток А), які слугують посередниками між контролерами та контекстом бази. Такий підхід сприяє дотриманню принципу розділення обов’язків, спрощує тестування та подальший супровід коду.

Формування нових записів (Create) (див. Рис. 3.1.) у базі даних було реалізовано через метод *Add*, який знаходиться у відповідних репозиторіях. Цей метод забезпечує додавання сутностей до контексту бази даних [17] і виклик збереження змін (SaveChanges) [18]. У контролерах створено відповідні POST-методи для роботи з HTTP-запитами. Вони приймають дані, надані користувачем, здійснюють початкову перевірку, наприклад, перевіряють, чи

заповнені всі обов'язкові поля. Після цього дані передаються в репозиторій для створення нового запису. Перш ніж додавати продукт у базу, проводиться перевірка, чи вказані всі важливі дані, такі як назва, ціна та категорія. Якщо є зображення продукту, воно також завантажується разом із запитом. Після перевірки збереження даних відбувається в таблиці Product.

Лістинг 3.1. Додавання нового запису до бази даних через метод Add.

```
public void Add(ProductAllergen productAllergen)
{
    _context.ProductAllergens.Add(productAllergen);
    Save();
}
```

Лістинг 3.2. Створення нового замовлення з додаванням товарів до нього.

```
[HttpPost]
public IActionResult Create(Product model)
{
    if (ModelState.IsValid)
    {
        string wwwRootPath = webHostEnvironment.WebRootPath;
        string fileName =
Path.GetFileNameWithoutExtension(model.CoverFile.FileName);
        string extension = Path.GetExtension(model.CoverFile.FileName);
        fileName = fileName + DateTime.Now.ToString("yymmssfff") + extension;
        model.CoverPath = "/img/product/" + fileName;
        string path = Path.Combine(wwwRootPath + "/img/product/", fileName);
        using (var fileStream = new FileStream(path, FileMode.Create))
        {
           model.CoverFile.CopyTo(fileStream);
        }
        productRepository.Add(model);
        return RedirectToAction(nameof(Index));
     }
     var categories = _context.Categories.ToList();
     var effectTypes = _context.EffectTypes.ToList();
     var suitableForOptions = _context.SuitableForOptions.ToList();
     ViewBag.CategoryList = new SelectList(categories, "Id", "NameCategory");
     ViewBag.EffectTypeList = new SelectList(effectTypes, "Id", "NameEffectType");
     ViewBag.SuitableForList = new SelectList(suitableForOptions, "Id",
"NameSuitableFor");
      return View(model);
}
```

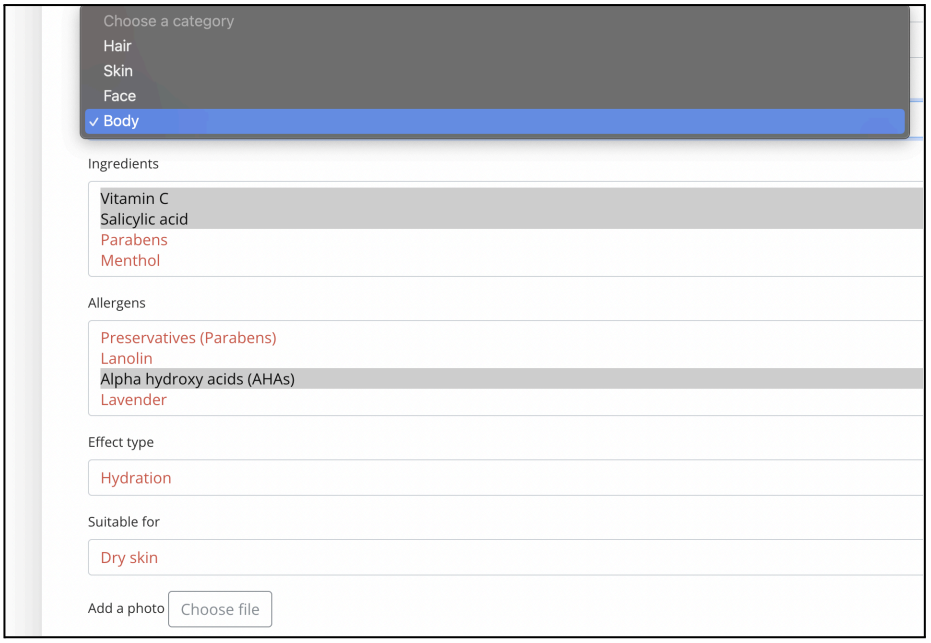

Рис. 3.1. Вигляд сторінки додавання продукту

Джерело: створено автором

Операції читання даних (Read) були реалізовані для виконання різноманітних запитів до бази даних. Існують методи для отримання всіх записів сутності, конкретного запису за його унікальним ідентифікатором, а також для отримання даних із фільтрацією за параметрами, такими як категорія продукту чи наявність певного інгредієнта. Репозиторії використовують методи GetAll [19], GetById (див. Додаток В) та спеціалізовані запити з Include [20], що дозволяють завантажувати пов’язані дані. Наприклад, при отриманні продукту (рис. 3.2.) завантажуються також його категорія, інгредієнти та алергени.

Лістинг 3.3. Отримання продукту за ідентифікатором.

```
public Product GetProductById(int productId)
{
    return _context.Products.Find(productId);
}
```

Лістинг 3.4. Отримання всіх продуктів з бази даних.

```
public IEnumerable<Product> GetAll()
{
    return _context.Products.ToList();
}
```

У контролерах GET-методи (див. Додаток В) відповідають за обробку запитів користувача, забезпечуючи повернення даних у вигляді списків або окремих записів.

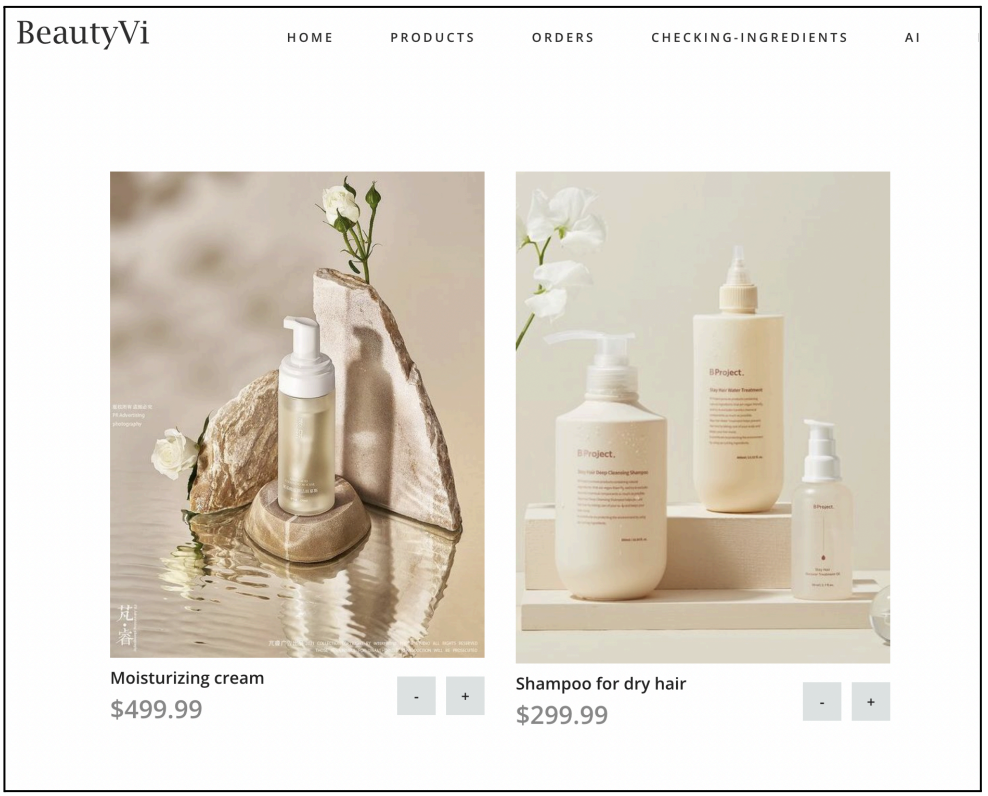

Рис. 3.2. Вигляд сторінки перегляду продуктів

Джерело: створено автором

Методи оновлення (Update) реалізовані в репозиторіях для внесення змін у наявні записи. Вони спочатку завантажують об’єкт із бази даних за його id, потім вносять зміни до його властивостей на основі наданих даних. Перед оновленням перевіряється, чи існує запис у базі. У контролерах PUT-методи (див. Додаток В) приймають дані для оновлення через API, перевіряючи актуальність даних, що надаються для оновлення, а також наявність самого запису, який має бути змінений. Після перевірки викликається метод репозиторію, і зміни зберігаються у базі даних [21]. Наприклад, оновлення продукту дозволяє змінити його ціну, опис чи зображення.

Лістинг 3.5. Оновлення продукту в базі даних.

```
public void Update(Product obj)
{
```

```
    _context.Products.Update(obj);
}
```

Видалення записів (Delete) (див. Рис. 2.6.) здійснюється через методи Delete, реалізовані в репозиторіях. Ці методи спочатку перевіряють, чи існує запис із вказаним id, а також перевіряють залежності інших таблиць, щоб не порушити цілісність даних. У контролерах DELETE-методи обробляють запити на видалення, отримують id запису та передають його в репозиторій (див. Додаток В) [22]. Видалення відбувається лише після успішної перевірки.

Ця реалізація включає механізм блокування видалення або каскадного видалення пов’язаних записів (наприклад, видалення продукту автоматично видалить усі його інгредієнти в зв’язковій таблиці).

Лістинг 3.6. Видалення продукту з бази даних.

```
public void Delete(Product obj)
{
    _context.Set<Product>().Remove(obj);
    Save();
}
```

Лістинг 3.7. Підтвердження видалення продукту.

```
public IActionResult DeleteConfirmed(int id)
{
    var product = productRepository.Get(id);
    if (product != null)
    {
        productRepository.Delete(product);
        return RedirectToAction(nameof(Index));
    }
    return NotFound();
}
```

Усі описані CRUD-операції не функціонують окремо — вони інтегровані в загальну логіку роботи сайту. Так, у панелі адміністратора реалізовано інтерфейс для керування товарами, де адміністратор може створювати, редагувати або видаляти товари. Для користувачів доступна можливість перегляду доступних продуктів, перегляду деталей і додавання товарів у замовлення.

3.3. Персоналізовані рекомендації на основі обраних характеристик користувачем

Механізм швидкого пошуку повинен бути інтегрований для того, щоб забезпечити користувачам можливість оперативно знаходити потрібні продукти, не чекаючи тривалих затримок під час пошуку. Для цього необхідно оптимізувати запити до бази даних і реалізувати механізми, які дозволяють швидко отримувати результати, не втрачаючи точності пошуку. Параметри фільтрації слід чітко розподілити на категорії (наприклад, тип шкіри, властивості, категорії товарів), щоб користувачі могли швидко орієнтуватися в налаштуваннях, для цього можна використовувати випадаючі списки, чекбокси або слайдери.

Для реалізації цього завдання у веб-додатку було розроблено спеціальний метод у репозиторії, який приймає параметри фільтрації, що надсилаються з інтерфейсу користувача, і формує динамічний запит до бази даних. Параметри можуть включати тип шкіри, тип волосся, бажані властивості продукту, категорію та інші характеристики, що дозволяють здійснювати точний пошук серед великого обсягу товарів. Гнучкість цього методу дозволяє працювати з множинними фільтрами одночасно, зберігаючи ефективність та швидкість запитів.

Основою для фільтрації є використання динамічних LINQ-запитів (див. Додаток В), які дозволяють формувати умови на основі вибраних параметрів [23]. Якщо користувач не обирає певний фільтр, то відповідна умова не додається до запиту. Це дає змогу формувати лише ті запити, які стосуються реальних потреб користувача, не навантажуючи базу даних непотрібними умовами. Наприклад, якщо користувач не вибирає тип шкіри, запит не міститиме умови для цього параметра, а якщо вибрані параметри включають тип шкіри і властивості продукту, запит буде відповідно комбінувати ці умови.

Для того щоб забезпечити точність результатів, було важливо, щоб вибрані товари відповідали усім заданим критеріям одночасно. Це дозволяє фільтрувати товари таким чином, щоб, наприклад, результати показували тільки ті продукти, які

підходять для сухої шкіри і одночасно мають властивості зволоження та очищення. Кожен параметр фільтрації перевіряється окремо, і лише ті товари, які відповідають усім вимогам, потрапляють у кінцевий список.

Оскільки фільтрація може повертати велику кількість продуктів, для оптимізації запитів до бази даних було застосовано метод .Include() [20], який дозволяє підвантажувати пов'язані дані, такі як категорії, інгредієнти чи властивості продуктів. Це зменшує кількість запитів до бази, що підвищує ефективність роботи додатку та дозволяє отримати всі необхідні дані за один раз.

Інтерфейс користувача був реалізований таким чином, щоб забезпечити легке та інтуїтивно зрозуміле використання фільтрів. У вигляді були інтегровані чекбокси для вибору кількох властивостей продуктів, випадаючі списки для вибору типу шкіри та волосся. Ці елементи інтерфейсу дозволяють користувачам швидко і зручно обирати необхідні параметри фільтрації, що полегшує процес пошуку відповідних товарів (див. Рис. 3.3.).

Користувацький інтерфейс інтегровано з серверною частиною через API-запити. Коли користувач вибирає фільтри, параметри передаються на сервер, де контролер обробляє їх і передає до репозиторію для формування запиту.

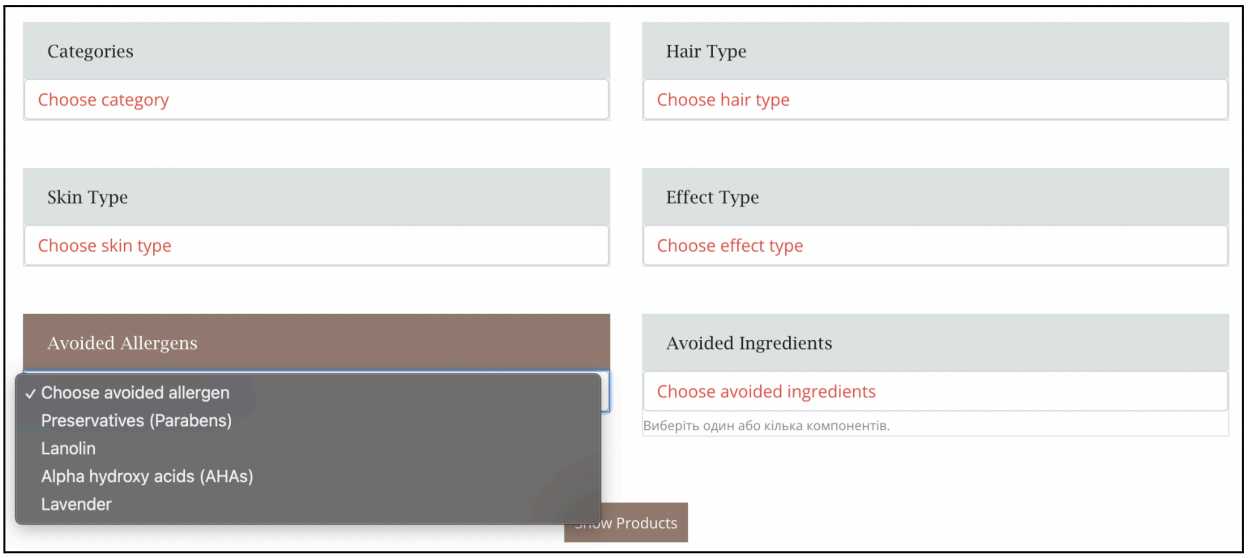

Рис. 3.3. Вигляд фільтрів для засобів

Джерело: створено автором

3.4. Фільтрація за складом продуктів

Розроблено спеціальний механізм для реалізації фільтрації продуктів за складом, зокрема за наявністю або відсутністю небажаних інгредієнтів, у веб-додатку, що дає можливість користувачам виключати небажані компоненти при пошуку косметичних засобів (рис. 3.4).

Рис. 3.4. Вигляд фільтру для складу продуктів

Джерело:створено автором

Для зберігання інформації про інгредієнти був створений окремий зв'язок між таблицями продуктів та інгредієнтів. Кожен продукт у базі даних містить посилання на таблицю інгредієнтів, що дозволяє зберігати перелік усіх складових, зокрема алергени чи інші небажані компоненти. У таблиці інгредієнтів є перелік компонентів із зазначенням їх типу, що дозволяє користувачам легко виключати певні інгредієнти з пошуку.

Створено метод, який відповідає за формування динамічного запиту до бази даних на основі вибраних користувачем параметрів, що передаються з інтерфейсу і включають список інгредієнтів, які потрібно виключити з результатів пошуку (див. Додаток В). Метод, який здійснює фільтрацію, використовує динамічне формування запиту за допомогою LINQ [23]. Зокрема, він застосовує методи .Where(), щоб додавати умови для фільтрації продуктів на основі того, які інгредієнти вибрано виключити. При цьому користувач може вибирати інгредієнти для виключення через інтерфейс, наприклад, за допомогою чекбоксів. Коли користувач обирає певні компоненти, сервер отримує ці параметри та передає їх у запит до бази даних.

Для того, щоб мінімізувати кількість запитів до бази даних, були використані методи для підвантаження додаткових даних. Наприклад, було застосовано метод .Include(), який дозволяє одночасно завантажити пов'язані дані, такі як категорії продуктів або властивості. Це дозволяє отримати всі необхідні дані за один запит, зменшуючи навантаження на базу даних і покращуючи ефективність роботи додатку.

3.5. Перевірка безпеки складу косметичного продукту

Механізм перевірки складу продукту має включати оцінку безпеки кожного інгредієнта на основі наданої інформації. Користувачі повинні мати можливість переглядати деталі інгредієнтів, включаючи їхню безпечність, можливі алергени та інші важливі характеристики. Крім того, система повинна надавати пояснення щодо кожного інгредієнта, зокрема, чому він може бути небезпечним чи безпечним для здоров'я або шкіри.

Інформація про кожен інгредієнт повинна зберігатися в базі даних і містити дані про рівень безпеки та додаткові характеристики, такі як опис інгредієнта і потенційні алергени. Першим кроком для зберігання даних про інгредієнти було створено окремі таблиці в базі даних, які містять важливу інформацію про кожен інгредієнт. Це включає рівень безпеки інгредієнта, його потенційну шкідливість, наявність алергенів та інші характеристики, які можуть впливати на здоров'я користувачів.

Крім того, для зв’язку між продуктами та інгредієнтами були налаштовані відповідні зовнішні ключі, що дозволяють визначити, які інгредієнти входять до складу кожного продукту. Це дозволяє зберігати інформацію про склади товарів та надає можливість для перевірки безпеки кожного інгредієнта в межах конкретного продукту.

Задля оцінки безпеки складу продукту було реалізовано спеціальний метод, який отримує список інгредієнтів для кожного продукту та проводить їх перевірку на основі інформації з таблиці інгредієнтів (див. Додаток С). Для цього застосовується

динамічне формування запиту, яке дозволяє перевіряти кожен інгредієнт на наявність потенційно небезпечних компонентів або алергенів. Відповідно до цих перевірок, продукт отримує оцінку безпеки.

У рамках цього процесу використовуються методи .Where() та .Any() для фільтрації інгредієнтів та перевірки на наявність певних властивостей, таких як наявність алергенів чи небажаних компонентів [24, 25]. Наприклад, якщо інгредієнт є алергеном, це враховується під час формування списку для конкретного продукту.

Окрім перевірки безпеки інгредієнтів, для кожного інгредієнта також зберігається додаткова інформація, яка пояснює, чому певний інгредієнт вважається безпечним чи небезпечним. Для цього використовуються текстові поля, які містять короткі описи, що дозволяють користувачу отримати більш детальну інформацію про кожен інгредієнт.

Для швидкої перевірки складу та отримання результатів безпеки було налаштовано ефективні запити до бази даних, які оптимізують процес фільтрації та дозволяють швидко отримувати інформацію про інгредієнти кожного продукту. Зокрема, використовуються методи .Include() для підвантаження даних про інгредієнти разом з основними даними про продукт [20].

В інтерфейсі продукту відображається список інгредієнтів, що входять до його складу, а також деталі щодо безпеки кожного компонента (рис. 3.5). Для кожного інгредієнта зазначено рівень безпеки (наприклад, безпечний, небезпечний), а також можливі алергени або шкідливі властивості. Користувач може натискати на конкретний інгредієнт, щоб отримати додаткову інформацію, таку як пояснення щодо його небезпечності чи безпеки. Це дозволяє зробити обґрунтовані висновки щодо того, чи варто використовувати цей продукт залежно від наявних компонентів.

Рис. 3.5. Вигляд перевірки складу

Джерело: створено автором

3.6. Впровадження системи контролю доступу для користувачів

Створена система дозволяє ефективно керувати доступом до функцій веб-додатку, базуючись на ролях користувачів, забезпечуючи розмежування прав доступу, захист конфіденційних даних та контроль за використанням функціоналу відповідно до обов'язків користувачів. Реалізація авторизації базується на вбудованому механізмі ASP.NET Core та враховує принципи, описані в «Securing ASP.NET Core Web Applications» [26].

Для забезпечення розмежування доступу до функціоналу веб-додатку на основі ролей було реалізовано систему контролю доступу з двома основними ролями: admin (адміністратор) і client (клієнт).

У контролері використовувався атрибут [Authorize(Roles = "...")], який дозволяє обмежувати доступ до методів залежно від ролі користувача [27].

Наприклад, методи, що відповідають за адміністративні функції, було захищено через [Authorize(Roles = "admin")], що гарантує доступ лише адміністраторам. Для методів, доступних усім зареєстрованим користувачам, використовувався атрибут [Authorize], який перевіряє лише факт авторизації без прив'язки до ролі [28].

Щоб забезпечити додаткову гнучкість, у виглядах (HTML-сторінках) були впроваджені умовні перевірки на основі ролей. За допомогою конструкцій @if (User.IsInRole("admin")) відображались елементи інтерфейсу, що доступні лише адміністраторам, наприклад, кнопки для редагування чи видалення важливих даних. Водночас функціональність, специфічна для клієнтів, приховувалася від адміністраторів завдяки аналогічним перевіркам.

3.7. Реалізація функції "Улюблені продукти"

Для збереження інформації про улюблені товари була створена окрема сутність Favorite, що містить зв'язки з користувачем та товаром. Ця таблиця містить унікальний ідентифікатор запису і ідентифікатор користувача, який додав товар у вибране. також ідентифікатор товару та навігаційні властивості для зв'язку з відповідними таблицями (див. Лістинг 3.8).

Лістинг 3.8. Оголошення класу Favorite.

```
namespace BeautyVi.Core.Entities
{
    public class Favorite
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        public virtual User? User { get; set; }
        public string? UserId { get; set; }

        public int? ProductId { get; set; }
        public virtual Product? Product { get; set; }
    }
}
```

Для взаємодії з базою даних був розроблений FavoriteRepository, який реалізує основні методи CRUD для обробки улюблених товарів. Основні функції репозиторію:

1. Додавання товару в улюблене
2. Видалення товару зі списку
3. Отримання улюблених товарів конкретного користувача
4. Повернення списку товарів, що були додані користувачем у вибране. Цей метод дозволяє отримати всі обрані товари для конкретного користувача з автоматичним завантаженням пов'язаних даних про продукт (див. Лістинг 3.9).

Лістинг 3.9. Метод GetFavoritesByUserId.

```
public IEnumerable<Favorite> GetFavoritesByUserId(string userId)
        {
            return _context.Favorites
                .Include(f => f.Product)
                .Where(f => f.UserId == userId)
                .ToList();
        }
```

Для обробки запитів користувачів було створено FavoriteController. Він містить основну логіку роботи з улюбленими товарами. Основним методом контролера є Index, тому що він відображає список улюблених товарів користувача і використовує метод GetFavoritesByUserId для отримання даних. Також важливим є AddToFavorites, який Додає товар до списку улюблених і перевіряє, чи користувач авторизований, та чи товар вже є у списку улюблених. Якщо товару немає у списку, він додається до бази даних. Є і метод Delete, що видаляє товар зі списку улюбленихі виконує видалення запису з бази даних [29].

3.8. Розробка чат-підтримки на основі штучного інтелекту

Першим етапом інтеграції AI було налаштування Ollama, який дозволяє запускати великі мовні моделі локально, без необхідності підключення до зовнішніх API [16, 30]. Це забезпечує більшу конфіденційність даних та зменшує залежність від сторонніх сервісів. Також завантажено Mistral, основну AI-модель у проєкті [31,

32]. Щоб забезпечити взаємодію між сервером і Mistral, потрібно відправляти HTTP-запити на локальний сервер Ollama, який відповідає за роботу з мовною моделлю. Для цього створимо контролер AIController і визначимо змінну, яка буде містити адресу сервера (див. Лістинг 3.10).

Лістинг 3.10. Змінна, яка містить адресу сервера.

```
private readonly string _ollamaUrl = "http://localhost:11434/api/generate";
```

Перед відправкою запиту потрібно створити об’єкт JSON, який міститиме інструкції для AI-моделі Mistral (див. Лістинг 3.11). У prompt передаємо сформований запит користувача. Stream у даному коді означає, що AI має повернути повну відповідь одразу, а не надсилати її частинами.

Лістинг 3.11. Формування JSON-запиту.

```
var requestBody = new
        {
            model = "mistral",// Використання моделі Mistral
            prompt = aiPrompt, // Текстовий запит, який отримує AI
            stream = false // Відключення потокової передачі (чекаємо на повну
відповідь)
        };
```

Для відправки запиту використовуємо HttpService, який виконує POST-запит до Ollama. Після того як AI обробить запит, отримуємо відповідь. EnsureSuccessStatusCode() перевіряє, чи сервер відповів успішно. JSON-відповідь десеріалізується у C#-об’єкт, після чого отримуємо текст, який повернула AI-модель (див. Лістинг 3.12).

Лістинг 3.12. Відправка запиту до Ollama та обробка відповіді.

```
var response = await httpService.PostDataAsync(_ollamaUrl, content);
            response.EnsureSuccessStatusCode();
var responseString = await response.Content.ReadAsStringAsync();
var jsonResponse = JsonSerializer.Deserialize<JsonElement>(responseString);
var aiResponse = jsonResponse.GetProperty("response").GetString();
```

Перед початком роботи із системою чат-бота потрібно визначити, який користувач надсилає запит. Це дозволить правильно зберігати та отримувати його історію повідомлень. Для цього у контролері AIController використовується метод

GetValidatedUserId(), який отримує userId через UserManager<IdentityUser>. Якщо userId не знайдено, викидається помилка. Цей метод гарантує, що лише авторизовані користувачі можуть взаємодіяти з AI (див. Лістинг 3.13).

Лістинг 3.13. Перевірка чи користувач авторизований.

```
private string GetValidatedUserId()
    {
        var userId = userManager.GetUserId(User);
        if (string.IsNullOrEmpty(userId))
        {
            throw new Exception("User not found.");
        }
        return userId;
    }
```

Для того щоб AI розумів контекст діалогу, перед формуванням нового запиту потрібно отримати останні повідомлення користувача, тому використовується chatHistoryRepository(у якому були створенні різні методи для роботи в контролері) для отримання останніх 5 повідомлень користувача. Повідомлення сортуються за часом (Timestamp), щоб брати найновіші. Вони форматуються у вигляді тексту для передачі в AI (див. Лістинг 3.14).

Лістинг 3.14. Отримання історії чату для контексту.

```
var userId = GetValidatedUserId();
// Отримуємо останні 5 повідомлень з історії користувача
var previousMessages = chatHistoryRepository.GetAllByUserId(userId)
            .OrderByDescending(ch => ch.Timestamp)
            .Take(5)
            .Select(ch => new { ch.UserMessage, ch.AIResponse })
            .ToList();
// Формуємо контекст діалогу
var contextMessages = string.Join("\n", previousMessages.Select(ch => $"User:
{ch.UserMessage}\nAI: {ch.AIResponse}"));
```

Для збереження історії чату, створено таблицю ChatHistory. Вона містить Id (унікальний ідентифікатор запису), UserId (зв'язок із конкретним користувачем), повідомлення користувача та відповідь AI, а також часову позначку (Timestamp). Далі зберігаємо повідомлення в базу, за допомогою методу .Add [33], щоб його можна було використати у наступних запитах.

Одним із ключових завдань чат-бота є рекомендація косметичних продуктів відповідно до запиту користувача. Для цього необхідно передати штучному інтелекту список продуктів, включаючи їх складники, алергени та категорії. Щоб AI міг аналізувати продукти, необхідно отримати їх із бази даних. Використовуємо ORM Entity Framework для отримання всіх продуктів разом із їхніми складниками та алергенами, для цього використаємо метод .Include (див. Лістинг 3.15). Після отримання списку продуктів їх потрібно перетворити у текстовий формат, що містить всю необхідну інформацію. Для цього формуємо список складників (якщо вони є, об'єднуємо їх у рядок, інакше вказуємо "No ingredients") і аналогічно робимо для списку алергенів, тоді об'єднуємо всю інформацію в один текстовий рядок.

Лістинг 3.15. Отримання списку продуктів.

```
var products = await context.Products
            .Include(p => p.ProductIngredients)
            .ThenInclude(pi => pi.Ingredient)
            .Include(p => p.ProductAllergens)
            .ThenInclude(pa => pa.Allergen)
            .AsSplitQuery()
            .ToListAsync();
```

Тепер, коли у нас є сформований список продуктів (productsList), можна передати його у запит до Mistral разом із історією чату і написати prompt в якому будть вказівки для штучного інтелекту, що він повинен відповідати вічливо, рекомендуючи наявні товари і якщо жоден продукт не підходить, він інформує про це користувача та намагається запропонувати альтернативу (див. Лістинг 3.16). Тепер можна перевірити як працює створений чат (рис. 3.6).

Лістинг 3.16. Формування prompt та передача списку продуктів.

```
var aiPrompt = $"Here is the conversation history:\n{contextMessages}\n" +
               $"New User Message: {userMessage}\n\n" +
               "Here is a list of products. Recommend ONLY those products that
meet the user's request. " +
               "Do not recommend products that do not match the user's
criteria:\n" +
               ${productsList}\n\n" +
               "Do not mention any list or database. Be friendly and helpful to
the user. " +
               "Avoid repeating greetings like 'Hello' or 'Hi' in every response.
" +
```

```
            "If no products match the user's request, politely inform them and
suggest alternatives if possible.";
```

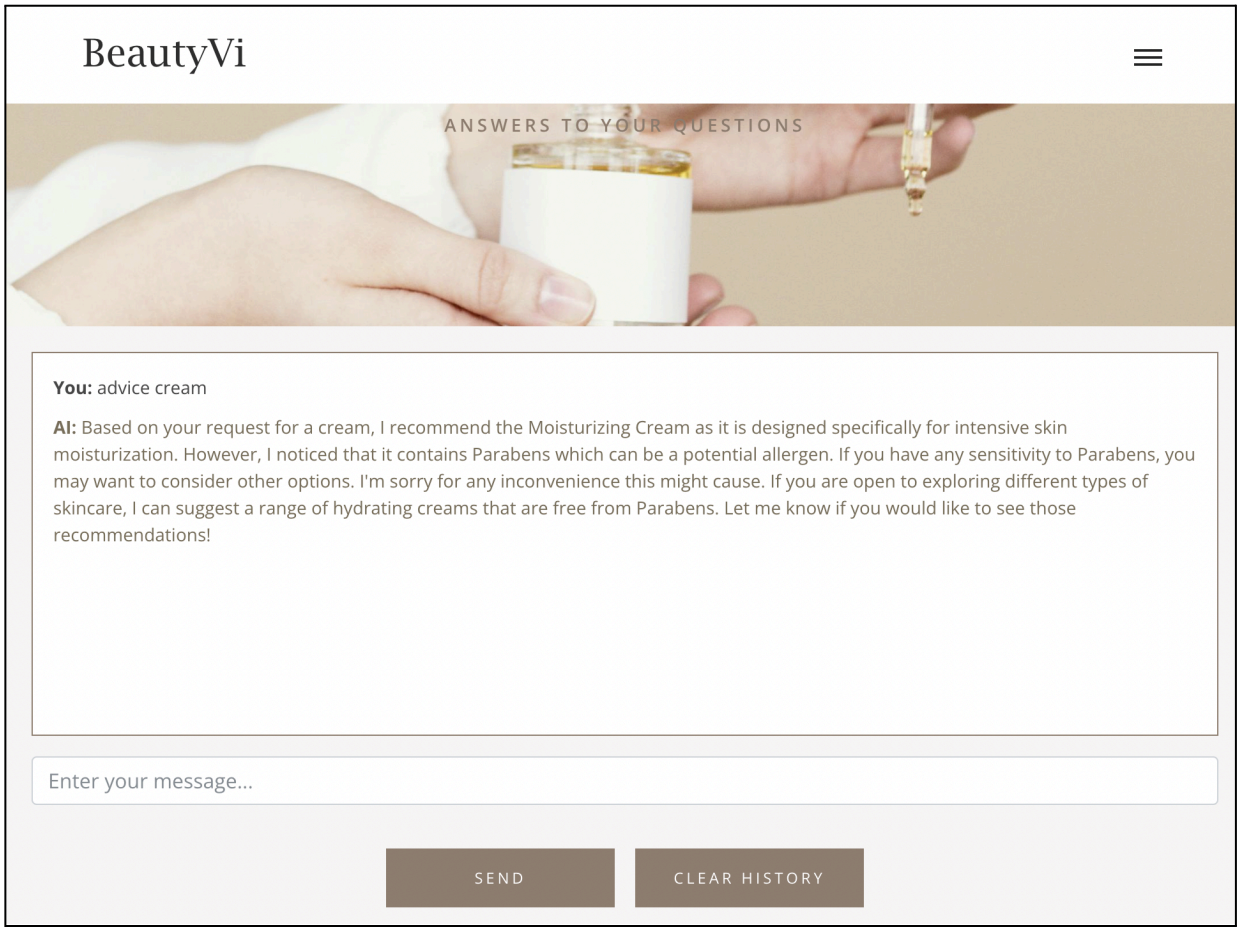

Рис. 3.6. Результат створеного чату

Джерело: створено автором

Для автоматизації процесу навчання штучного інтелекту була розроблена спеціальна фонова служба AIModelTrainingService. Вона відповідає за збір даних, їх фільтрацію та передачу на сервер навчання моделі. Ця служба реалізована в середовищі .NET Core як IHostedService, що дозволяє запускати її у фоновому режимі разом із веб-додатком [34]. Вона періодично аналізує повідомлення користувачів та оновлює модель штучного інтелекту без потреби втручання адміністратора.

При запуску застосунку у методі StartAsync створюється таймер, який кожні 24 години викликає метод TrainModelAsync() (див. Лістинг 3.18), а у методі StopAsync зупиняється таймер при завершенні роботи для економії ресурсів [35]

(див. Лістинг 3.18). Це було зроблено, щоб автоматизувати процес навчання без необхідності ручного втручання.

Лістинг 3.17. Метод StartAsync запуску таймера.

```
public Task StartAsync(CancellationToken cancellationToken)
    {
        _timer = new Timer(async _ =>
        {
            try
            {
                await TrainModelAsync();
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "An error occurred while training the
model");
            }
        }, null, TimeSpan.Zero, TimeSpan.FromHours(24)); // Run interval every 24
hours
        return Task.CompletedTask;
    }
```

Лістинг 3.18. Метод StopAsync виключення таймера.

```
public Task StopAsync(CancellationToken cancellationToken)
    {
        _timer?.Change(Timeout.Infinite, 0); // Stop the timer when the service
ends
        return Task.CompletedTask;
    }
```

Метод TrainModelAsync() створює новий контекст бази даних через IServiceScopeFactory, після чого отримує всі повідомлення, надіслані за останню добу, але якщо нових даних немає, процес навчання припиняється, і в лог записується відповідне повідомлення. Також додано перевірку, якщо за останні 24 години повідомлень немає, то навчання пропускається. Для навчання моделі лише на унікальних та корисних даних, виконується очищення, щоб видалити повторювані відповіді чи питання і також перебираються порожні повідомлення за допомогою IsNullOrWhiteSpace. Фільтрування отриманих чатів відбувається за допомогою GroupBy та Select(g => g.First()). Після фільтрації дані зберігаються у списку filteredChats. Також в логах фіксується кількість очищених записів (рис. 3.7.).

Оскільки сервер навчання не може одночасно обробити великий обсяг інформації і щоб зменшити навантаження на мережу та сервер повідомлення надсилаються пакетами розміром BatchSize по 100 записів. Для кожного пакету створюється HTTP-запит за допомогою HttpClientn [36]. Якщо запит завершується невдало, здійснюється до 3 спроб з інтервалом у 2 секунди між ними. Після кожного пакету додається невелика пауза (Task.Delay(500)) для уникнення перевантаження сервера. Забезпечується стабільність роботи служби та можливість відстеження помилок за допомогою використання ILogger [37] для логування результатів кожного кроку.

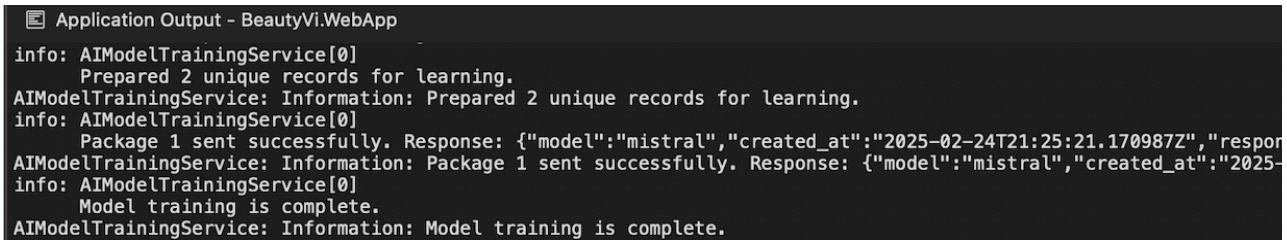

Рис. 3.7. Виведення логів

Джерело: створено автором

Для забезпечення ефективного управління даними та підтримки продуктивності системи було реалізовано механізм автоматичного та ручного видалення історії чату. Цей механізм дозволяє автоматично видаляти старі повідомлення, які вже не є актуальними, а також надає можливість користувачам вручну очищати свою історію чату.

Для автоматизації процесу очищення історії чату була розроблена фонова служба ChatCleanupService. Служба періодично аналізує історію чату та видаляє повідомлення, які старші за встановлений термін (наприклад, 30 днів). Також були додані методи StartAsync та StopAsync для запуску та вимикання таймера кожна 24 години, щоб викликати метод видалення застарілих чатів. Додано метод PerformCleanupAsync() який відповідає за автоматичне очищення історії чатів у базі даних. За допомогою ServiceProvider отримуємо екземпляр BeautyViContext, який дозволяє взаємодіяти з таблицями бази даних. Всі повідомлення, створені раніше

цієї дати, будуть видалені. Далі для видалення записів із таблиці використовуємо Where, щоб вибрати лише ті записи, які старші за 30 днів. Потім метод ExecuteDeleteAsync() видаляє всі знайдені записи за один запит, а deletedCount зберігає кількість видалених записів (див. Лістинг 3.20). ExecuteDeleteAsync() є ефективним способом видалення, оскільки він виконує один SQL-запит без завантаження всіх записів у пам’ять.

Лістинг 3.20. Видалення записів із бази даних.

```
var cutoffDate = DateTime.UtcNow.AddDays(-30);
var deletedCount = await context.ChatHistories
                                .Where(ch => ch.Timestamp < cutoffDate)
                                .ExecuteDeleteAsync();
```

Після видалення хоча б одного запису, у лог записується кількість видалених повідомлень. Якщо видалених записів немає, виводиться повідомлення про те що не знайдено старих повідомлень для очищення(рис. 3.8.).

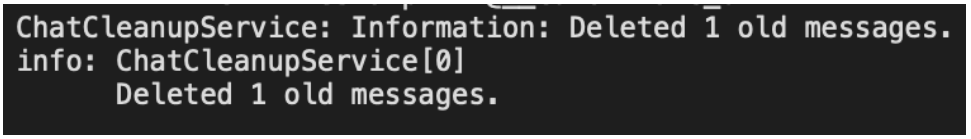

Рис. 3.8. Виведення логів

Джерело: створено автором

Для забезпечення зручності користувачів було реалізовано можливість ручного очищення історії чату. Спочатку отримуємо ідентифікатор поточного користувача та видаляємо всі записи історії чату для нього за допомогою методу .DeleteAllByUserId. Даний метод прописаний в репозиотрії ChatHistoryRepisitory і у ньому проходить фільтрація записів за переданим йому ідентифікатором користувача та відбувається видалення даних без завантаження їх у пам’ять скориставшись ExecuteDelete.

3.9. Створення голосового помічника

У рамках даного проєкту було реалізовано голосовий помічник, що дозволяє користувачу взаємодіяти з вебзастосунком через голосові команди та отримувати озвучені відповіді. Основною метою є підвищення доступності сайту та зручності

його використання, зокрема для людей з порушеннями зору або тих, хто працює в умовах, коли не можна користуватись клавіатурою (наприклад, під час водіння чи в побутових ситуаціях).

Розробка базується на вбудованих можливостях браузера — API, який не потребує зовнішніх бібліотек чи сторонніх сервісів. Зокрема, використано:

- SpeechRecognition — інтерфейс для розпізнавання голосу користувача та перетворення його на текст [38].
- SpeechSynthesis — інтерфейс для озвучення відповідей системи у відповідь на команди користувача [39].

Рішення реалізовано на стороні клієнта (frontend), з використанням JavaScript. Уся логіка поділена на окремі частини:

- homePage.js — відповідає за голосову взаємодію на головній сторінці сайту.
- aiPage.js — обробляє голосові команди безпосередньо в чаті з AI.
- speechHelper.js — додатковий допоміжний модуль, який забезпечує універсальні функції (ініціалізація, озвучення, обробка команд).

Для того щоб запустити голосовий помічник на сайті потрібно натиснути клавішою в будь якому місці, це зроблено для простішого ввімкнення для людей з обмеженими можливостями, щоб вони могли самостійно впоратися (див. Лістинг 3.21).

Лістинг 3.21. Ввімкнення голосового помічника.

```
document.addEventListener("click", () => {
        if (synth) {
            const intro = new SpeechSynthesisUtterance("Say a command to
continue.");
            intro.lang = "en-US";
            synth.speak(intro);
            intro.onend = () => recognition.start();
        } else {
            recognition.start();
        }
    });
```

Спочатку озвучується привітання та після цього асистент питає чи бажає користувач продовжити роботу з ним чи самостійно (див. Лістинг 3.22).

Лістинг 3.22. Привітання голосового помічника.

```
speak("Hello! I am ready to help. Please tell me what I can assist you with.", ()
=> {
        speak("Would you like me to continue?", () => {
            startListening();
        });
    });
```

При відповіді “no” помічник побажає гарного дня та вимкнеться, а якщо відповідь буде не зрозумілою прозвучить ще раз питання про подальші дії. Коли користувач хоче продовжити, то йому будуть представлені можливі опції під номерами (див. Лістинг 3.23), це зроблено для простішого розпізнання голосу.

Лістинг 3.23. Обробка вибору користувача.

```
function processInitialResponse(userInput) {
    userInput = userInput.toLowerCase().trim();
    if (userInput.includes("yes") || userInput.includes("continue")) {
        isAnswered = true;
        recognition.stop();
        currentMode = "options";
        speak("Here are your options: Say 1 to read the homepage. Say 2 to go to
the AI page.", () => {
            startListening();
        });
    } else if (userInput.includes("no") || userInput.includes("stop")) {
        isAnswered = true;
        recognition.stop();
        speak("Alright, have a great day!", () => {
            response.textContent = "Voice assistant turned off";
        });
    } else {
        attemptCount++;
        if (attemptCount >= 2) {
            isAnswered = true;
            recognition.stop();
            speak("Sorry, I still didn't understand. Let's try again another
time.");
        } else {
            speak("Sorry, I didn't catch that. Please say 'yes' to continue or
'no' to stop.", () => {
                startListening();
            });
        }
    }
}
```

Озвучування вмісту сторінки відбувається за допомогою інтерфейсу SpeechSynthesis, який є частиною Web Speech API (див. Лістинг 3.24). Його завдання

перетворити текст на мову та проговорити його через динаміки пристрою. У контексті голосового помічника, озвучення активується тоді, коли користувач вибирає відповідну команду. Після цього текстовий вміст певного елемента сторінки отримується з DOM і передається на озвучення. Відбувається вибір усіх HTML-елементів з класами, які містять важливу текстову інформацію на головній сторінці. Це заголовки, підзаголовки, рекламні блоки тощо. Текст із кожного елемента додається до однієї змінної pageText, формуючи суцільну фразу для подальшого озвучення і після цього використовується функція speak(text), яка створює об'єкт SpeechSynthesisUtterance і відтворює його через браузерний голосовий синтез.

Лістинг 3.24. Озвучення вмісту сторінки.

```
function readPageText() {
    let pageText = "";
    const elementsToRead = document.querySelectorAll(
        ".home_slider_subtitle, .home_slider_title, .promo_title, .promo_subtitle,
.section_subtitle, .section_title, .extra_1_text, .extra_2_text, .gallery_text"
    );
    elementsToRead.forEach(element => {
        pageText += element.innerText + " ";
    });
    speak(pageText);
}
```

Коли користувач обирає перейти до певної сторінки, наприклад AI, то зберігається спеціальний прапорець у localStorage [40]. Це дозволяє після завантаження нової сторінки автоматично активувати голосового помічника. Відповідна перевірка виконується одразу при ініціалізації (див. Лістинг 3.25).

Лістинг 3.25. Автоматичний запуск голосового помічника.

```
window.addEventListener("DOMContentLoaded", () => {
    const startVoiceFlag = localStorage.getItem("startAIWithVoice");
    if (startVoiceFlag === "true") {
        localStorage.removeItem("startAIWithVoice");
        startVoiceAssistant(); // функція запуску голосового режиму
    }
});
```

Голосове введення запитань у чаті реалізовано через API розпізнавання мови браузера. Після активації мікрофона користувач вимовляє питання, яке розпізнається

як текст та автоматично надсилається в чат для обробки штучним інтелектом. Надсилання реалізовано шляхом автоматичного заповнення текстового поля вводу та виклику функції надсилання повідомлення (див. Лістинг 3.26). Після отримання відповіді від AI, її текст озвучується користувачу. Потім система знову пропонує можливі подальші дії голосом.

Лістинг 3.26. Відправлення повідомлення до AI та озвучення відповіді.

```
async function sendMessage() {
    const userInput = document.getElementById("userInput").value;
    if (!userInput) return;

    const chatBox = document.getElementById("chatBox");
    chatBox.innerHTML += `
    <div class="message user" style="color: #4a4a4a; margin-bottom:
10px;"><strong>You:</strong> ${userInput}</div>`;

    const response = await fetch("/api/ai/chat", {
        method: "POST",
        headers: { "Content-Type": "application/json" },
        body: JSON.stringify(userInput)
    });
    if (!response.ok) {
        chatBox.innerHTML += `<div class="error">Server error: ${response.status}
- ${response.statusText}</div>`;
        return;
    }

    const result = await response.text();
    chatBox.innerHTML += `
    <div class="message ai" style="color: #7a6f5b; margin-bottom:
10px;"><strong>AI:</strong> ${result}</div>`;

    const synth = window.speechSynthesis;
    if (synth) {
        const utterance = new SpeechSynthesisUtterance(result);
        utterance.lang = "en-US";
        synth.speak(utterance);

        utterance.onend = () => {
            const followUp = new SpeechSynthesisUtterance(
                "What would you like to do next? Say one to ask a new question,
two to return to the home page, or three to clear the chat history."
            );
            followUp.lang = "en-US";
            synth.speak(followUp);
            followUp.onend = () => {
                if (recognition) recognition.start();
            };
        };
```

```
    }

    document.getElementById("userInput").value = "";
    chatBox.scrollTop = chatBox.scrollHeight;
}
```

Реалізація голосового помічника в цьому проєкті доводить, що голосовий інтерфейс може значно покращити зручність взаємодії з сайтом. Всі дії, включно з навігацією, запитами до AI та відповідями, можуть відбуватися без жодного кліку, що робить систему більш доступною, сучасною і корисною.

Висновки до розділу 3

У третьому розділі було детально описано практичну реалізацію інтернет-магазину косметичних засобів, що включає вибір технологічного стеку, розробку функціонального ядра системи, реалізацію голосових інтерфейсів, інтелектуальних рекомендацій та механізмів персоналізації.

Ретельно обґрунтовано використання сучасних технологій, що забезпечило високу продуктивність, безпеку та масштабованість розробленого застосунку. Для роботи з базою даних використано Entity Framework Core, що дозволив ефективно реалізувати CRUD-операції, які є основою для адміністрування інформації в системі.

Інтерфейсна частина додатку збагачена голосовим керуванням, реалізованим на основі Web Speech API, що значно покращує зручність взаємодії користувачів з системою. Водночас було реалізовано чат-бота та систему рекомендацій за допомогою мовної моделі Mistral у поєднанні з платформою Ollama, що дозволяє зберігати конфіденційність даних і інтегрувати локальні інтелектуальні функції.

Особливу увагу приділено персоналізованому пошуку та фільтрації продуктів. Створено механізм динамічної побудови запитів із використанням LINQ, який враховує індивідуальні параметри користувача, включаючи тип шкіри, властивості косметичних засобів та склад продуктів. Це дозволяє забезпечити точні та швидкі результати пошуку. Запропонована система також враховує безпеку складу

косметичних засобів, дозволяючи виключити небажані інгредієнти з результатів та переглянути детальну інформацію про кожен компонент.

Отже, реалізоване рішення поєднує сучасні підходи до розробки веб-застосунків, зручність для користувача, інтелектуальну обробку даних і персоналізацію, що в сукупності формує конкурентоспроможний інтернет-магазин нового покоління.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було реалізовано повнофункціональний інтернет-магазин косметичних засобів, орієнтований на сучасні потреби користувачів, включно з підтримкою персоналізованого підбору продукції, голосової навігації та інтелектуальної чат-підтримки. Створений застосунок поєднує в собі широкий набір функцій, інтуїтивно зрозумілий інтерфейс та потужну серверну архітектуру, що дозволяє ефективно автоматизувати процес підбору та замовлення товарів.

На етапі дослідження проведено аналіз предметного середовища та існуючих інтернет-магазинів у сфері косметики. Вивчено функціональні можливості провідних платформ та виявлено їхні основні недоліки, такі як обмежений пошуковий функціонал, відсутність персоналізації та голосової взаємодії. Отримані результати обґрунтовують необхідність розробки нової онлайн-платформи, що враховує сучасні потреби користувачів і забезпечує розширені фільтри, індивідуальні рекомендації та інтеграцію голосового асистента.

В основі системи лежить сучасний стек технологій, який охоплює ASP.NET Core для побудови серверної логіки, Entity Framework Core для роботи з базою даних, PostgreSQL як надійну СУБД, JavaScript для реалізації клієнтської логіки. Завдяки цьому підходу вдалося досягти високого рівня функціональності, продуктивності та масштабованості, що є важливими критеріями для сучасних вебсистем.

Проведено моделювання бази даних з урахуванням усіх ключових сутностей та визначено основні зв’язки між таблицями. Розроблено функціональний вебінтерфейс, в якому користувач може здійснювати реєстрацію, переглядати каталог, фільтрувати продукти за численними критеріями, додавати товари в обране, формувати замовлення. Особливу увагу приділено фільтрації за складом, що дозволяє користувачеві виключати небажані інгредієнти. Такий функціонал є особливо важливим для осіб із алергіями або чутливою шкірою.

Інноваційною частиною проєкту є впровадження голосового асистента та інтелектуального чат-бота. Голосовий помічник реалізовано на базі Web Speech API, що дає змогу керувати інтерфейсом сайту, здійснювати пошук та отримувати поради без необхідності використання миші або клавіатури, що значно підвищує інклюзивність. Для реалізації AI-чат-підтримки застосовано локальну мовну модель Mistral у поєднанні з платформою Ollama, що забезпечує приватність даних і швидку обробку запитів. Також впроваджено фоновий сервіс для періодичного очищення історії чатів і автоматичного навчання AI-моделі на основі запитів користувачів. Це дозволяє підтримувати актуальність рекомендацій та забезпечити високий рівень адаптивності системи.

Розроблена система також включає механізм персоналізованих рекомендацій, які формуються на основі індивідуальних параметрів користувача. Це дозволяє запропонувати користувачу саме ті товари, які найбільше відповідають його потребам, підвищуючи задоволеність сервісом і ймовірність покупки. Важливим є те, що користувачі мають можливість виключати небажані інгредієнти з підбору товарів, що є ключовою функцією для сучасного косметичного маркетплейсу.

Загалом, виконана кваліфікаційна робота демонструє не лише практичне втілення теоретичних знань з програмування, баз даних, веброзробки та штучного інтелекту, але й уміння створити цілісну, сучасну, масштабовану та доступну інформаційну систему. Отриманий результат може слугувати основою для реального комерційного продукту, а також бути розширеним у напрямку мобільної версії, інтеграції з платіжними системами, багатомовної підтримки, машинного зору для аналізу шкіри тощо.

Отже, поставлені завдання було повністю реалізовано, а результати свідчать про доцільність та ефективність застосованих рішень. Запропонована система не лише вирішує базові завдання онлайн-магазину, але й виводить користувацький досвід на новий рівень, поєднуючи комфорт, безпеку та інтелектуальну допомогу у виборі косметичних засобів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Економічна правда. Онлайн-продажі в Україні за період пандемії. URL: https://epravda.com.ua/news/2020/09/19/665297/ (дата звернення: 01.10.2024).
2. Meest Shopping. Покупки в інтернет-магазинах України. URL:https://biz.nv.ua/ukr/markets/yaki-onlayn-pokupki-roblyat-ukrajinci-ta-na-yaki-sumi-meest-shopping-50493284.html (дата звернення: 01.10.2024).
3. Sigma Software University. Що таке база даних? URL: https://university.sigma.software/what-is-database/ (дата звернення: 08.10.2024).
4. Telerik. ASP.NET Core Basics: Working with a Database. Telerik Blogs, 2023. URL: https://www.telerik.com/blogs/aspnet-core-basics-working-database (дата звернення: 15.10.2024).
5. Microsoft. Creating and Configuring a Model. URL: https://learn.microsoft.com/en-us/ef/core/modeling/. (дата звернення: 17.10.2024).
6. Foxminded. Що таке PostgreSQL і для чого використовується? URL: https://foxminded.ua/postgresql-shcho-tse/ (дата звернення: 18.10.2024).
7. PostgreSQL Global Development Group. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 18.10.2024).
8. Microsoft. Загальні відомості про EF Core. URL: https://learn.microsoft.com/uk-ua/training/modules/persist-data-ef-core/2-understanding-ef-core (дата звернення: 21.10.2024).
9. Microsoft. Посібник зі зв’язків між таблицями. URL: https://support.microsoft.com/uk-ua/topic/посібник-зі-зв-язків-між-таблицями-30446197-4fbe-457b-b992-2f6fb812b58f (дата звернення: 22.10.2024).
10. Shah R. Building APIs with ASP.NET Core. Packt Publishing, 2023.
11. Microsoft. (2023). Entity Framework Core Documentation. https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 24.10.2024).
12. Somers P. Practical Entity Framework Core. Apress, 2022.

13. Microsoft. Web Speech API. Розпізнавання голосу. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API (дата звернення: 13.03.2025).
14. Using the Web Speech API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Speech_API/Using_the_Web_Speech_API (дата звернення: 14.03.2025).
15. Google Cloud. Dialogflow: Створення чат-ботів з AI. URL: https://cloud.google.com/dialogflow/docs (дата звернення: 15.02.2025).
16. Ollama. Документація по локальному запуску LLM. URL: https://ollama.com (дата звернення: 12.02.2025).
17. Microsoft. System.Collections.Generic.List<T>.Add Method. URL: https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.add?view=net-6.0 (дата звернення: 16.02.2025).
18. Microsoft. EF Core – Basic Saving. URL: https://learn.microsoft.com/uk-ua/ef/core/saving/basic (дата звернення: 24.11.2024).
19. Limilabs. Метод ".GetAll". URL: https://www.limilabs.com/static/mail/documentation/html/M_Limilabs_Client_POP3_Pop3_GetAll.htm (дата звернення: 14.12.2024).
20. Microsoft. EF Core – Eager Loading with Include. URL: https://learn.microsoft.com/en-us/ef/core/querying/related-data/eager (дата звернення: 30.11.2024).
21. Microsoft Learn. Tutorial: Implement CRUD Functionality - ASP.NET Core MVC with EF Core. URL: https://learn.microsoft.com/en-us/aspnet/core/data/ef-mvc/crud?view=aspnetcore-9.0 (дата звернення: 10.11.2024).
22. Microsoft. Delete methods of an ASP.NET Core app URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/details?view=aspnetcore-9.0 (дата звернення: 13.11.2024).

23. Microsoft. LINQ Tutorial: Language Integrated Query. URL: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq (дата звернення: 18.12.2024).
24. Microsoft. Where Method (System.Linq.Enumerable). Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/api/system.linq.enumerable.where?view=net-9.0 (дата звернення: 20.12.2024).
25. Microsoft. Any Method (System.Linq.Enumerable). Microsoft Learn. URL: https://learn.microsoft.com/uk-ua/dotnet/api/system.linq.enumerable.any?view=net-8.0 (дата звернення: 20.12.2024).
26. Thomas J. Securing ASP.NET Core Web Applications. Apress, 2021.
27. Microsoft. ASP.NET Core Authorization. URL: https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction (дата звернення: 14.12.2024).
28. Microsoft. Authorization: Simple authorization in ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/simple?view=aspnetcore-8.0 (дата звернення: 14.12.2024).
29. Microsoft. Робота з даними у Razor Pages та MVC. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/controller-methods-views#edit-methods-and-edit-view (дата звернення: 15.12.2024).
30. OpenAI. Використання мовних моделей у програмних продуктах. URL: https://platform.openai.com/docs (дата звернення: 19.02.2025).
31. Mistral AI. Mistral inference examples using Ollama. URL: https://ollama.com/library/mistral (дата звернення: 20.02.2025).
32. Mistral AI. Офіційна документація моделей Mistral. URL: https://mistral.ai (дата звернення: 20.02.2025).
33. Microsoft. Метод ".Add" у мові програмування C#. URL: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.add?view=net-6.0 (дата звернення: 21.02.2025).

34. Microsoft. IHostedService у .NET Core. URL: https://learn.microsoft.com/en-us/dotnet/api/microsoft.extensions.hosting.ihostedservice (дата звернення: 14.03.2025).
35. Microsoft. Реалізація таймерів у .NET. URL: https://learn.microsoft.com/en-us/dotnet/api/system.threading.timer?view=net-9.0 (дата звернення: 15.03.2025).
36. Microsoft. HttpClient Class. URL: https://learn.microsoft.com/en-us/dotnet/api/system.net.http.httpclient?view=net-9.0 (дата звернення: 18.03.2025).
37. Microsoft. Logging in C# and .Net. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/logging?tabs=command-line (дата звернення: 19.03.2025).
38. Avislab. Розпізнавання мови (Speech recognition). URL: https://blog.avislab.com/speech-recognition/ (дата звернення: 20.03.2025).
39. Microsoft. System.Speech.Synthesis Namespace. URL: https://learn.microsoft.com/uk-ua/dotnet/api/system.speech.synthesis?view=net-9.0-pp (дата звернення: 20.03.2025).
40. Mozilla Developer Network. Window.localStorage. URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (дата звернення: 21.03.2025).

ДОДАТКИ

ДОДАТОК А. Код Product Repository

Лістинг 3.27. Код Product Repository.

```
namespace BeautyVi.Repositories.Repos
{
    public class ProductRepository : IProductRepository
    {
        private BeautyViContext _context;
        public ProductRepository(BeautyViContext context)
        {
            _context = context;
        }
        public void Add(Product obj)
        {
            _context.Products.Add(obj);
            Save();
        }

        public void Delete(Product obj)
        {
            _context.Set<Product>().Remove(obj);
            Save();
        }

        public Product Get(int id)
        {
            return _context.Products.Find(id);
        }

       public IEnumerable<Product> GetAll()
        {
            return _context.Products.ToList();
        }

        public void Save()
        {
            _context.SaveChanges();
        }

        public void Update(Product obj)
        {
            _context.Products.Update(obj);
        }
        public Product Find(int id)
        {
```

```
            return _context.Products.FirstOrDefault(product => product.Id == id);
        }

        public Product GetProductById(int productId)
        {
            return _context.Products.Find(productId);
        }
    }
}
```

ДОДАТОК В. Код Product Controller

Лістинг 3.28. Код Product Controller.

```
namespace BeautyVi.WebApp.Controllers
{
    public class ProductController : Controller
    {
        private readonly IProductRepository productRepository;
        private readonly IWebHostEnvironment webHostEnvironment;
        private readonly BeautyViContext _context;

        public ProductController(IProductRepository productRepository,
            IWebHostEnvironment webHostEnviroment, [FromServices] BeautyViContext
context)
        {
            this.productRepository = productRepository;
            this.webHostEnvironment = webHostEnviroment;
            this._context = context;
        }

        public IActionResult Index()
        {
            ViewBag.HairTypes = _context.SuitableForOptions
                .Where(s => s.NameSuitableFor.Contains("волосся"))
                .ToList();

            ViewBag.SkinTypes = _context.SuitableForOptions
                .Where(s => s.NameSuitableFor.Contains("шкіра"))
                .ToList();

            ViewBag.AvoidedAllergens = _context.Allergens.ToList();
            ViewBag.EffectTypes = _context.EffectTypes.ToList();
            ViewBag.Categories = _context.Categories.ToList();
            ViewBag.Ingredients = _context.Ingredients.ToList();

            var allProducts = productRepository.GetAll();

            return View(allProducts);
        }

        [HttpGet]
        public IActionResult Search(string searchTerm)
        {
            var searchResults = _context.Products
                .Where(product => product.Name.Contains(searchTerm)
                || product.Description.Contains(searchTerm)
                || product.Category.NameCategory.Contains(searchTerm)
                || product.EffectType.NameEffectType.Contains(searchTerm)
                || product.SuitableFor.NameSuitableFor.Contains(searchTerm)
```

```
                || product.Price.ToString().Contains(searchTerm))
                .ToList();

            return View("Index", searchResults);
        }

        public IActionResult Details(int id)
        {
            var product = _context.Products
                .Include(p => p.Category)
                .Include(p => p.EffectType)
                .Include(p => p.SuitableFor)
                .Include(p => p.ProductIngredients)
                    .ThenInclude(pi => pi.Ingredient)
                .Include(p => p.ProductAllergens)
                    .ThenInclude(pi => pi.Allergen)
                .FirstOrDefault(p => p.Id == id);

            if (product == null)
            {
                return NotFound();
            }

            ViewBag.OwnerCategory = product.Category?.NameCategory;

            return View(product);
        }

        [Authorize(Roles = "Admin")]
        [HttpGet]
        public IActionResult Create()
        {
            var categories = _context.Categories.ToList();
            var effectTypes = _context.EffectTypes.ToList();
            var suitableForOptions = _context.SuitableForOptions.ToList();

            ViewBag.CategoryList = new SelectList(categories, "Id",
"NameCategory");
            ViewBag.EffectTypeList = new SelectList(effectTypes, "Id",
"NameEffectType");
            ViewBag.SuitableForList = new SelectList(suitableForOptions, "Id",
"NameSuitableFor");

            return View(new Product());
        }

        [Authorize(Roles = "Admin")]
        [HttpPost]
        public IActionResult Create(Product model)
        {
            if (ModelState.IsValid)
            {
                string wwwRootPath = webHostEnvironment.WebRootPath;
```

```
                string fileName =
Path.GetFileNameWithoutExtension(model.CoverFile.FileName);

                string extension = Path.GetExtension(model.CoverFile.FileName);
                fileName = fileName + DateTime.Now.ToString("yymmssfff") +
extension;
                model.CoverPath = "/img/product/" + fileName;
                string path = Path.Combine(wwwRootPath + "/img/product/",
fileName);

                using (var fileStream = new FileStream(path, FileMode.Create))
                {
                    model.CoverFile.CopyTo(fileStream);
                }

                productRepository.Add(model);
                return RedirectToAction(nameof(Index));
            }
            var categories = _context.Categories.ToList();
            var effectTypes = _context.EffectTypes.ToList();
            var suitableForOptions = _context.SuitableForOptions.ToList();
            ViewBag.CategoryList = new SelectList(categories, "Id",
"NameCategory");
            ViewBag.EffectTypeList = new SelectList(effectTypes, "Id",
"NameEffectType");
            ViewBag.SuitableForList = new SelectList(suitableForOptions, "Id",
"NameSuitableFor");
            return View(model);
        }

        [Authorize(Roles = "Admin")]
        public IActionResult Delete(int id)
        {
            var product = productRepository.Get(id);
            if (product == null) return NotFound();

            return View(product);
        }

        [Authorize(Roles = "Admin")]
        [HttpPost, ActionName("Delete")]
        public IActionResult DeleteConfirmed(int id)
        {
            var product = productRepository.Get(id);
            if (product != null)
            {
                productRepository.Delete(product);
                return RedirectToAction(nameof(Index));
            }
            return NotFound();
        }
```

```
        [Authorize(Roles = "Admin")]
        [HttpGet]
        public IActionResult Edit(int id)
        {
            var item = productRepository.Get(id);

            if (item == null)
            {
                return NotFound();
            }
            var categories = _context.Categories.ToList();
            var effectTypes = _context.EffectTypes.ToList();
            var suitableForOptions = _context.SuitableForOptions.ToList();
            ViewBag.CategoryList = new SelectList(categories, "Id",
"NameCategory");
            ViewBag.EffectTypeList = new SelectList(effectTypes, "Id",
"NameEffectType");
            ViewBag.SuitableForList = new SelectList(suitableForOptions, "Id",
"NameSuitableFor");

            return View(item);
        }

        [Authorize(Roles = "Admin")]
        [HttpPost]
        public IActionResult Edit(Product item)
        {
            if (ModelState.IsValid)
            {

                var existingItem = productRepository.Get(item.Id);

                if (existingItem != null)
                {
                    existingItem.Name = item.Name;
                    existingItem.Description = item.Description;
                    existingItem.Price = item.Price;
                    existingItem.Category = item.Category;
                    existingItem.EffectType = item.EffectType;
                    existingItem.SuitableFor = item.SuitableFor;

                    if (item.CoverFile != null)
                    {
                        string wwwRootPath = webHostEnvironment.WebRootPath;
                        string fileName =
Path.GetFileNameWithoutExtension(item.CoverFile.FileName);
                        string extension =
Path.GetExtension(item.CoverFile.FileName);
                        fileName = fileName + DateTime.Now.ToString("yymmssfff") +
extension;
                        existingItem.CoverPath = "/img/product/" + fileName;
                        string path = Path.Combine(wwwRootPath, "img/product",
fileName);
```

```
                        using (var fileStream = new FileStream(path,
FileMode.Create))
                        {
                            item.CoverFile.CopyTo(fileStream);
                        }
                    }
                    existingItem.CategoryId = item.CategoryId;
                    existingItem.EffectTypeId = item.EffectTypeId;
                    existingItem.SuitableForId = item.SuitableForId;

                    productRepository.Update(existingItem);
                    productRepository.Save();

                    return RedirectToAction(nameof(Index));
                }
            else
            {
                return NotFound();
            }
        }
        var categories = _context.Categories.ToList();
        var effectTypes = _context.EffectTypes.ToList();
        var suitableForOptions = _context.SuitableForOptions.ToList();
        ViewBag.CategoryList = new SelectList(categories, "Id", "NameCategory");
        ViewBag.EffectTypeList = new SelectList(effectTypes, "Id",
"NameEffectType");
        ViewBag.SuitableForList = new SelectList(suitableForOptions, "Id",
"NameSuitableFor");
        return View(item);
    }

    public IActionResult FilterByPreferences(
    string hairType,
    string skinType,
    string avoidedAllergens,
    string effectType,
    string category,
    string[] avoidedIngredients,
    bool avoidAllergens = false)
    {
        ViewBag.HairTypes = _context.SuitableForOptions
            .Where(s => s.NameSuitableFor.Contains("волосся"))
            .ToList();
        ViewBag.SkinTypes = _context.SuitableForOptions
            .Where(s => s.NameSuitableFor.Contains("шкіра"))
            .ToList();
        ViewBag.AvoidedAllergens = _context.Allergens.ToList();
        ViewBag.EffectTypes = _context.EffectTypes.ToList();
        ViewBag.Categories = _context.Categories.ToList();
        ViewBag.Ingredients = _context.Ingredients.ToList();

        var query = _context.Products
```

```
            .Include(p => p.Category)
            .Include(p => p.SuitableFor)
            .Include(p => p.EffectType)
            .Include(p => p.ProductIngredients)
                .ThenInclude(pi => pi.Ingredient)
            .Include(p => p.ProductAllergens)
                .ThenInclude(pa => pa.Allergen)
            .AsQueryable();

        if (!string.IsNullOrEmpty(category))
        {
            query = query.Where(p => p.Category.NameCategory.Contains(category));
        }

        if (!string.IsNullOrEmpty(hairType))
        {
            query = query.Where(p =>
p.SuitableFor.NameSuitableFor.Contains(hairType));
        }

        if (!string.IsNullOrEmpty(skinType))
        {
            query = query.Where(p =>
p.SuitableFor.NameSuitableFor.Contains(skinType));
        }
        if (!string.IsNullOrEmpty(avoidedAllergens))
        {
            if (avoidAllergens)
            {
                query = query.Where(p => p.ProductAllergens.Any(pa =>
pa.Allergen.Name == avoidedAllergens));
            }
            else
            {
                query = query.Where(p => !p.ProductAllergens.Any(pa =>
pa.Allergen.Name == avoidedAllergens));
            }
        }
        if (!string.IsNullOrEmpty(effectType))
        {
            query = query.Where(p =>
p.EffectType.NameEffectType.Contains(effectType));
        }
        if (avoidedIngredients != null && avoidedIngredients.Length > 0)
        {
            query = query.Where(p => !p.ProductIngredients
                .Any(pi => avoidedIngredients.Contains(pi.Ingredient.Name)));
        }
        var products = query.ToList();
        return View("FilteredProducts", products);
        }
    }
}
```

ДОДАТОК С. Код Ingredient Controller

Лістинг 3.29. Код Ingredient Controller

```
namespace BeautyVi.Controllers
{
    public class IngredientController : Controller
    {
        private readonly IIngredientRepository ingredientRepository;
        private readonly BeautyViContext _context;
        private readonly IWebHostEnvironment webHostEnvironment;

        public IngredientController(IIngredientRepository ingredientRepository,
            IWebHostEnvironment webHostEnviroment, [FromServices] BeautyViContext
context)
        {
            this.ingredientRepository = ingredientRepository;
            this.webHostEnvironment = webHostEnviroment;
            this._context = context;
        }

        public IActionResult Index()
        {
            var ingredients = ingredientRepository.GetAll();

            return View(ingredients);
        }

        [HttpGet]
        public IActionResult CheckIngredients()
        {
            return View();
        }

        [HttpPost]
        public IActionResult CheckIngredients(string ingredientsInput)
        {
            var inputIngredients = ingredientsInput?
                .Split(new[] { '\r', '\n' },
StringSplitOptions.RemoveEmptyEntries)
                .Select(i => i.Trim())
                .ToList();

            if (inputIngredients == null || inputIngredients.Count == 0)
            {
                ViewBag.Message = "Будь ласка, введіть список інгредієнтів для
перевірки.";
                return View();
            }
```

```
            var matchedIngredients = _context.Ingredients
                .Where(i => inputIngredients.Contains(i.Name))
                .ToList();

            if (matchedIngredients.Any())
            {
                ViewBag.AverageDangerLevel = matchedIngredients.Average(i =>
i.LevelOfDanger);
            }

            return View(matchedIngredients);
        }
    }
}
```