Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра автоматизації, комп'ютерних наук і технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти – бакалавр
за освітньо-професійною програмою
«Комп'ютерні науки»
зі спеціальності
122 – Комп'ютерні науки

тема роботи:

«Розробка інтернет-магазину косметичних засобів на базі технології ASP.NET Core»

Виконав студент гр. КН-21 _____ Ковтуненко Д. О.

Керівник _____ Тронь В. В.

Нормоконтроль _____ Маринич І. А.

Завідувач кафедри _____ Рубан С. А.

Кривий Ріг – 2025

КРИВОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет: *інформаційних технологій*

Кафедра: *автоматизації, комп'ютерних наук і технологій*

Ступінь вищої освіти: *Бакалавр*

Спеціальність: *122 – Комп'ютерні науки*

ЗАТВЕРДЖУЮ

Зав. кафедрою: *к.т.н. Рубан С.А.*

_____

« *29* » *січня* 2025 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

студенту групи КН-21 *Ковтуненку Данилу Олександровичу*

1. Тема кваліфікаційної роботи: *«Розробка інтернет-магазину косметичних засобів на базі технології ASP.NET Core»*

затверджено наказом по університету № *254с* від *13.05.2025 р.*

2. Термін здачі кваліфікаційної роботи: *05.06.2025 р.*

3. Склад кваліфікаційної роботи: *Пояснювальна записка обсягом 70 с., додатки, презентація у Microsoft PowerPoint (13 слайдів) в електронному та друкованому вигляді*

4. Консультанти кваліфікаційної роботи:

Розділ 1-2 доц. Тронь В. В.

Нормоконтроль доц. Маринич І. А.

5. Календарний план:

№	Етапи роботи	Термін виконання
1	*Вступ*	*01.04.25*
2	*Розділ 1*	*05.04.25*
3	*Розділ 2*	*01.05.25*
4	*Висновки*	*25.05.25*
5	*Оформлення кваліфікаційної роботи*	*28.05.25*
6	*Підготовка презентації та графічного матеріалу*	*20.05.25*
7	*Підготовка доповіді до захисту*	*05.06.25*

6. Дата видачі завдання: *29.01.2025р.*

Керівник _____ / Тронь В. В./

7. Запевнення: *Я, Ковтуненко Данило Олександрович, запевняю, що ця кваліфікаційна робота виконана самостійно, не містить академічного плагіату, фабрикації, фальсифікації. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.*

Із чинним Положенням про академічну доброчесність Криворізького національного університету ознайомлений.

Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі умисних порушень робота не допускається до захисту або оцінюється незадовільно.

Студент _____ / Ковтуненко Д. О./

АНОТАЦІЯ

Ковтуненко Д. О. Розробка інтернет-магазину косметичних засобів на базі технології ASP.NET Core : кваліфікаційна робота бакалавра : 122 – Комп’ютерні науки. Кривий Ріг. Криворізький національний університет, 2025. 70 с.

Кваліфікаційна робота присвячена розробці веб-застосунку – інтернет-магазину косметичних засобів, реалізованого з використанням сучасної технології ASP.NET Core MVC.

Метою розробки є створення функціонального, масштабованого та безпечного веб-застосунку з інтуїтивно зрозумілим інтерфейсом та широким спектром можливостей, таких як управління товарами, категоріями, брендами, обробка замовлень, кошик покупця та підтримка зовнішніх провайдерів аутентифікації (Google, Facebook).

У вступі обґрунтовано актуальність обраної теми, сформульовано мету та завдання дослідження. У першому розділі виконано огляд сучасних вітчизняних та зарубіжних інтернет-магазинів, проведено аналіз функціональних можливостей та технологічних рішень, а також сформульовано вимоги до майбутнього застосування. Другий розділ присвячено архітектурі веб-застосунку, описано каркас проєкту, виконано проєктування бази даних, компонентів інтерфейсу користувача, виконано реалізацію основного функціоналу інтернет-магазину, окремо детально описано реалізацію кошику товарів та функціоналу обробки замовлень.

Практична цінність роботи полягає у створенні універсального веб-застосунку, який може бути використаний як основа для комерційного інтернет-магазину або навчальний приклад для вивчення веб-розробки.

Ключові слова:

ЕЛЕКТРОННА КОМЕРЦІЯ, ІНТЕРНЕТ-МАГАЗИН, КОСМЕТИЧНІ ЗАСОБИ, БАЗА ДАНИХ, ФРЕЙМВОРК, ASP.NET CORE, MVC

ЗМІСТ

ВСТУП

У сучасних умовах швидкого розвитку цифрових технологій онлайн-торгівля стає невід’ємною складовою економіки, а електронна комерція виступає потужним інструментом взаємодії між продавцями та споживачами. Особливу популярність набули спеціалізовані інтернет-магазини, що пропонують косметичні засоби, оскільки забезпечують зручний доступ до широкого асортименту продукції, можливість ознайомлення з відгуками, гнучку систему фільтрації та доставки. У зв’язку з цим актуальною є розробка високоякісних веб-рішень, орієнтованих на покращення користувацького досвіду, розширення функціональності та підтримку сучасних стандартів безпеки.

Метою кваліфікаційної роботи є створення функціонального веб-застосунку – інтернет-магазину косметичних засобів, реалізованого за допомогою технології ASP.NET Core MVC , що забезпечує розділення логіки, представлення та обробки даних.

Для досягнення поставленої мети необхідно виконати такі завдання:

– здійснити аналіз існуючих інтернет-магазинів косметики та виділити ключові функціональні характеристики;

– сформулювати перелік вимог до майбутнього додатку;

– розробити архітектуру системи на основі MVC-патерну;

– розробити структуру БД для даного застосунку та забезпечити маніпуляції з даними у додатку з використанням ORM-фреймворку Entity Framework Core з використанням підходу Code First;

– реалізувати функціонал реєстрації, аутентифікації (включаючи OAuth2), управління товарами, категоріями, брендами;

– створити кошик, систему замовлень, ієрархічне меню та адміністративну панель.

– виконати практичну апробацію розробленого інтернет-магазину та навести опис методики його використання.

Практична цінність розробленого застосунку полягає у можливості його адаптації під реальні потреби малого та середнього бізнесу у сфері онлайн-продажу косметичних засобів, а також використання як навчального прикладу для вивчення сучасних підходів до веб-розробки.

РОЗДІЛ 1
ДОСЛІДЖЕННЯ МОЖЛИВИХ ТЕХНОЛОГІЙ ТА ІНСТРУМЕНТІВ ДЛЯ РОЗРОБКИ ІНТЕРНЕТ-МАГАЗИНУ КОСМЕТИЧНИХ ЗАСОБІВ

1.1 Аналіз існуючих інтернет-магазинів косметичних засобів, їх функціоналу та технологій розробки

У процесі розробки інтернет-магазину доцільним є вивчення найкращих прикладів функціонування аналогічних рішень у сфері електронної комерції. Аналіз зразкових реалізацій дозволяє сформувати обґрунтований перелік функціональних вимог, що забезпечують ефективну взаємодію користувача з системою та високу конкурентоспроможність веб-ресурсу.

Інтернет-магазин Sephora (www.sephora.com) є одним із найвідоміших світових ритейлерів косметики та парфумерії (рис. 1.1). Його веб-платформа демонструє високу ступінь функціонального та візуального опрацювання.

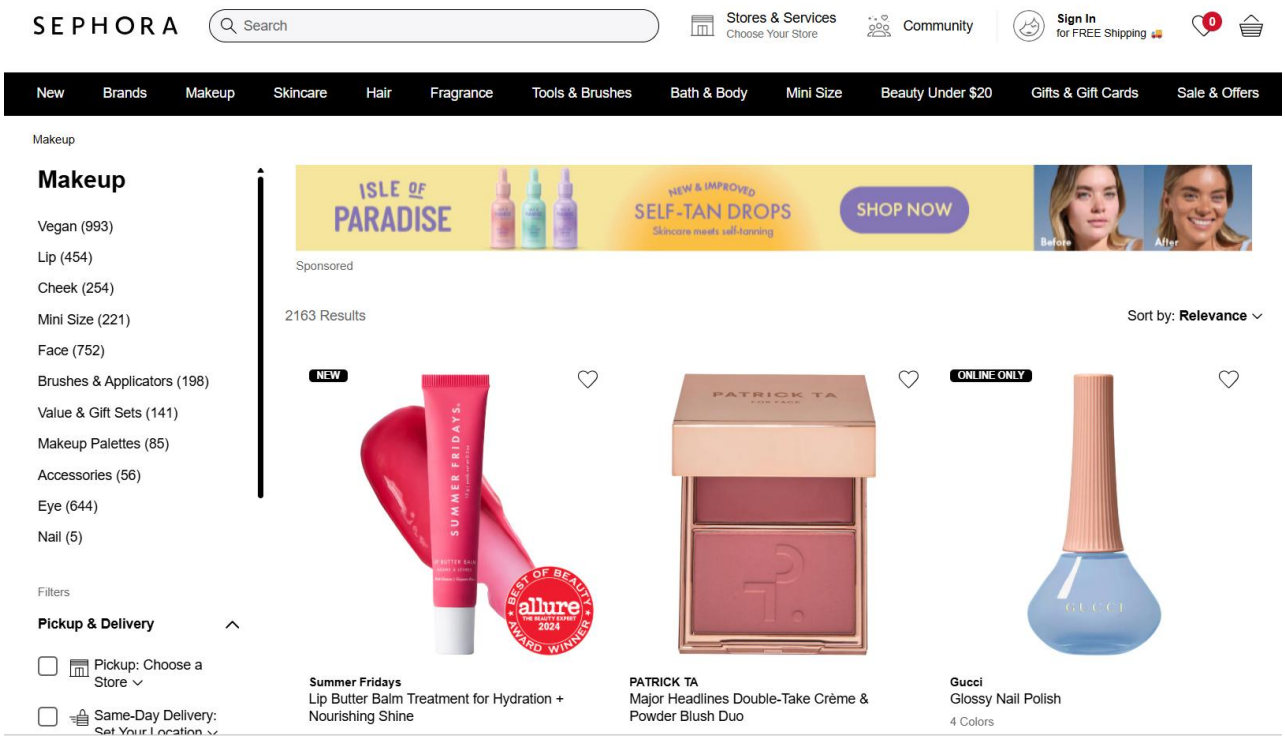


Рисунок 1.1 – Інтерфейс сторінки групи товарів інтернет-магазину Sephora

Особливу увагу приділено персоналізації користувацького досвіду. Зокрема, реалізовано розширену систему фільтрації товарів за різноманітними критеріями: бренд, тип шкіри, колір, складники, ціна, рейтинг тощо. Також передбачено функціонал авторизації через облікові записи Google, Facebook та Apple ID, що спрощує доступ до персоналізованих функцій.

Серед інноваційних можливостей слід відзначити модуль віртуальної примірки (Virtual Try-On), який дозволяє в режимі доповненої реальності оцінити вигляд декоративної косметики. Крім того, кожна сторінка товару містить детальний опис, відео-огляди, рецензії користувачів, а також рекомендації на основі попередніх переглядів. Усі ці функції сприяють підвищенню залученості користувачів і зростанню продажів.

На основі інформації з відкритих джерел та застосування інструментів для аналізу веб-технологій (BuiltWith, Wappalyzer), було визначено, що сайт Sephora використовує наступні основні технології:

– Frontend: React.js, Backbone.js (в окремих модулях), Handlebars;

– Backend: Oracle Commerce (раніше ATG Web Commerce), серверна частина – Java EE.

У якості інструментів CDN та технологій оптимізації використовуються Akamai, Cloudflare. Також варто відзначити використання таких сервісів, як Google Tag Manager, Google Analytics, Algolia – для швидкого пошуку по сайту, Adobe Experience Manager – для керування контентом.

До характерних особливостей можна віднести високий рівень персоналізації, використання Headless-комерції (окрема фронтенд частина, яка працює з API backend-системи).

Веб-сайт компанії Glossier (www.glossier.com) орієнтований на молодіжну аудиторію та відрізняється мінімалістичним дизайном, який зосереджує увагу на візуальних аспектах товару (рис. 1.2). Основними елементами інтерфейсу є великі зображення продукції, короткі описи, а також можливість ознайомитися з реальними відгуками покупців. Дизайн побудовано за принципами user-centric підходу: сайт інтуїтивно зрозумілий, а навігація логічно структурована.

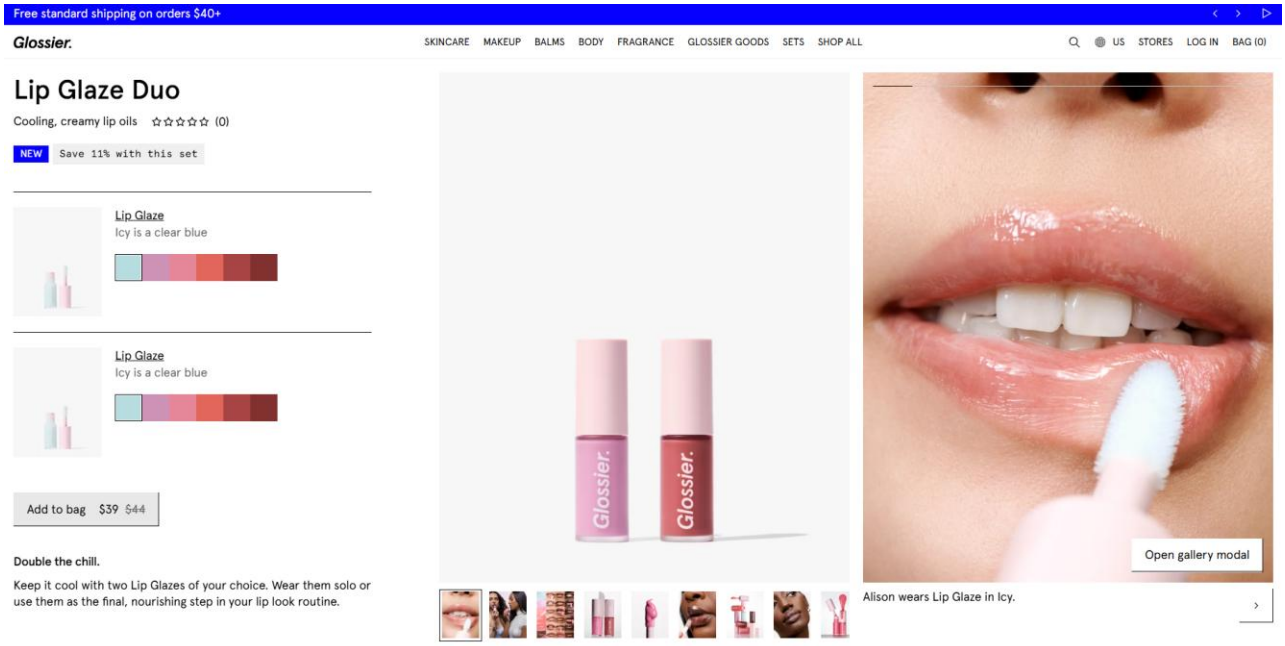


Рисунок 1.2 – Інтерфейс сторінки окремого товару інтернет-магазину Glossier

Окрему роль у комунікації з клієнтами відіграє інтеграція з Instagram, яка дозволяє переглядати дописи покупців, що використовують продукцію бренду. Такий підхід не лише слугує джерелом довіри, а й формує спільноту навколо бренду. Додатково реалізовано функціонал створення добірок за темою (наприклад, сезонні товари або рекомендації стилістів), що може слугувати прикладом ефективного маркетингового інструменту.

Сайт Glossier використовує наступні основні технології:

– Frontend: React.js, Next.js;

– Backend: Ruby on Rails (із GraphQL / REST API).

Інфраструктура інтернет-магазину розгорнута на Amazon Web Services (AWS) та Fastly.

Ресурс використовує платіжну систему Stripe, а також набір інструментів аналітики і маркетингу, зокрема Segment, Hotjar, Klaviyo, Facebook Pixel, Google Analytics. До особливостей треба віднести використання сучасного стеку технологій на базі JAMstack (JavaScript + APIs + Markup) та великий акцент на SEO та швидкість завантаження.

Онлайн-магазин компанії Lush (www.lush.com) позиціонується як етичний бренд з акцентом на екологічність, відмову від тестування на тваринах та використання натуральних інгредієнтів. Візуальне оформлення ресурсу відображає цінності бренду: природні кольори, простий інтерфейс, великий обсяг текстової інформації з описом процесу виготовлення продукції (рис. 1.3). Сайт оснащений розділом із відео-оглядами, що дозволяє клієнтам краще ознайомитися з товарами до моменту придбання.

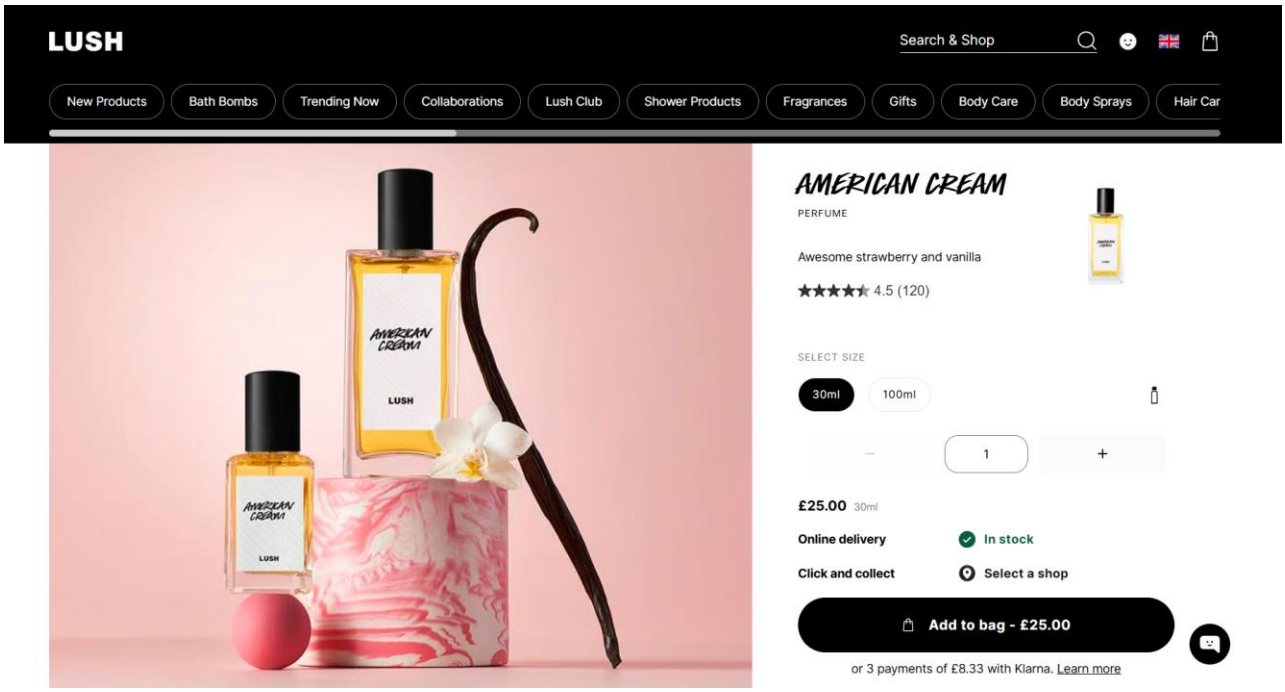


Рисунок 1.3 – Інтерфейс сторінки окремого товару інтернет-магазину Lush

Серед функціональних можливостей слід виокремити наявність інтерактивного онлайн-чату з консультантом, багатомовну підтримку, а також розширену систему фільтрів, яка враховує не лише характеристики товару, а й етичні критерії (наприклад, «веганські», «без сульфатів»). Також варто відзначити функціонал «Подарункові набори», який дозволяє користувачеві формувати власні комплекти на основі персоналізованих уподобань (рис. 1.4).

Інтернет-магазин Lush використовує наступні основні технології:

– Frontend: Vue.js (в нових версіях сайту), HTML5, CSS3;

– Backend: раніше використовувалась Magento, нині переходять до PWA-архітектури (Progressive Web App).

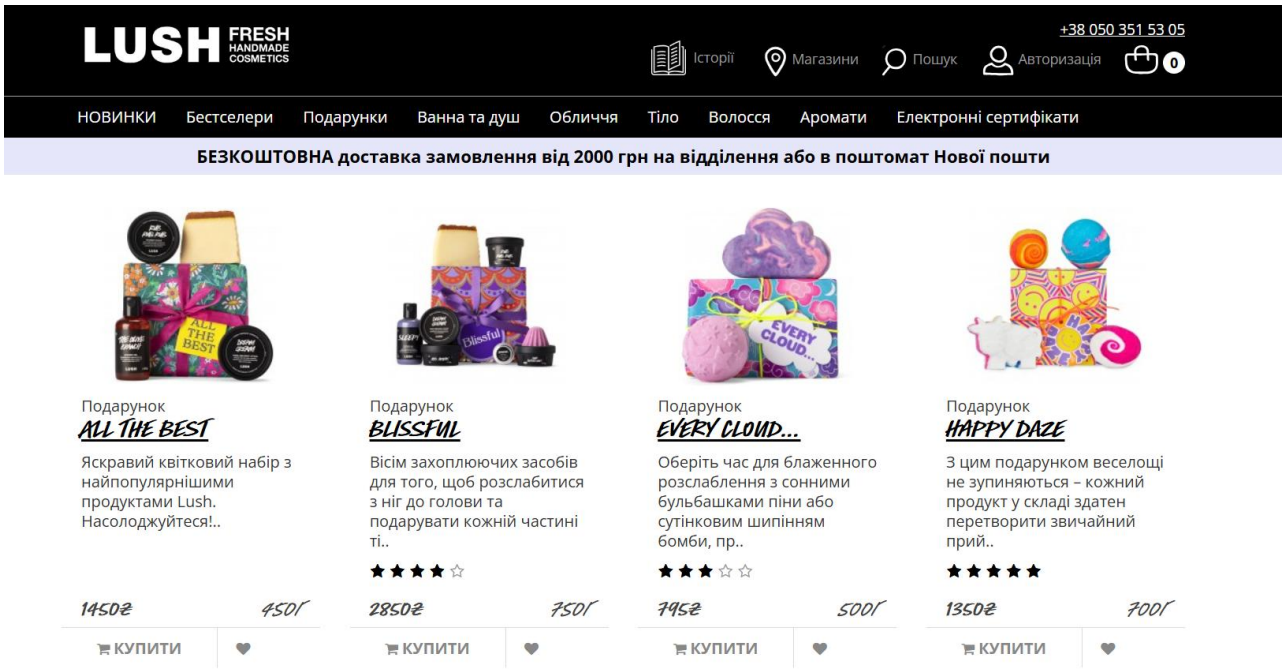


Рисунок 1.4 – Сторінка подарункових наборів на українській версії інтернет-магазину Lush

Для керування контентом адмін-частина використовує CMS Prismic, у деяких локалізаціях Contentful.

Ресурс використовує також Zendesk – для підтримки користувачів, Klarna та PayPal – для оплати, Google Optimize – для A/B тестування.

До особливостей можна віднести орієнтацію на модульну структуру та відокремлений frontend, PWA підхід.

Інтернет-магазин MakeUp (https://makeup.com.ua) є одним із найпопулярніших онлайн-ресурсів з продажу косметики в Україні. Сайт має чітко структуровану архітектуру та багатофункціональний інтерфейс користувача. Особливістю платформи є розвинена система фільтрації товарів за категоріями, брендом, типом продукту, об'ємом, країною виробництва, ціновим діапазоном та наявністю.

Сторінка кожного товару містить деталізований опис, відгуки покупців, фото у високій якості, можливість обрати об’єм/тип упаковки (рис. 1.5).

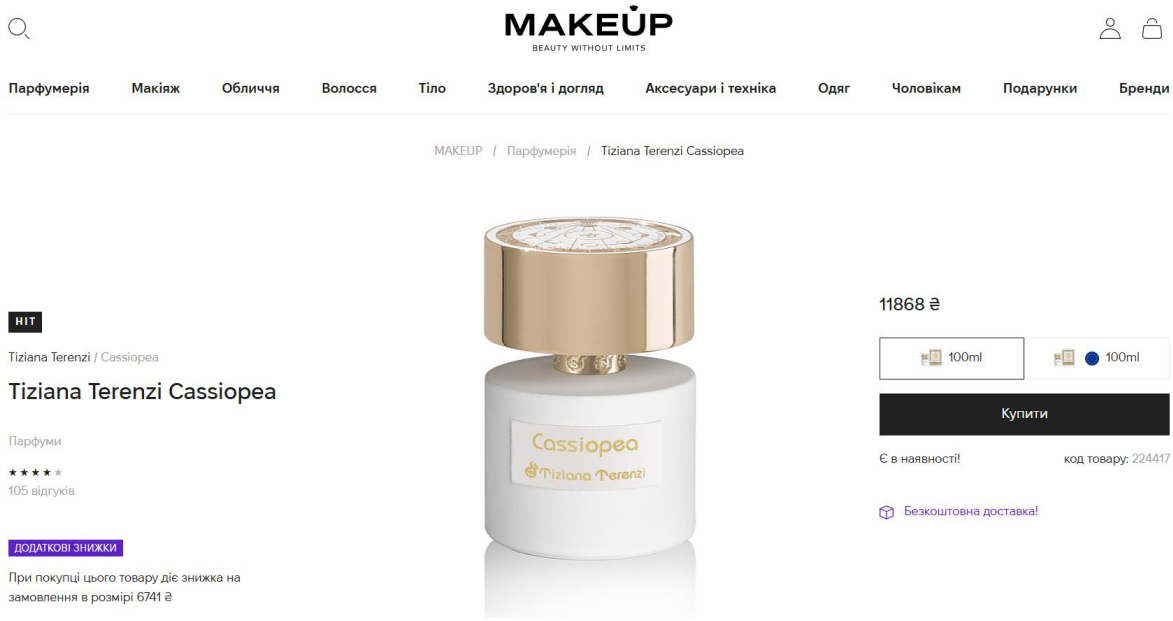


Рисунок 1.5 – Інтерфейс сторінки окремого товару інтернет-магазину MakeUp

Сайт активно використовує рекомендаційні алгоритми, які пропонують користувачу схожі або популярні товари. Також реалізовано систему персональних кабінетів, швидке оформлення замовлення без реєстрації та спрощену мобільну версію інтерфейсу.

Функціональні вимоги, які варто запозичити:

– багаторівнева система фільтрації;

– адаптивна картка товару з розширеними характеристиками;

– модуль рецензій/відгуків;

– швидке оформлення замовлення (без обов’язкової реєстрації).

Інтернет-магазин EVA.UA (eva.ua) –онлайн-платформа мережі магазинів EVA, яка поєднує традиційну роздрібну торгівлю з цифровими каналами. Сайт вирізняється високим рівнем інтеграції з бонусною програмою «EVA МОЗАЇКА» та можливістю вибору між доставкою та самовивозом із фізичних магазинів.

Інтерфейс має зручний дизайн із фокусом на категорійне меню, потужну систему фільтрації та блоки акцій/знижок. Реалізовано вхід за номером телефону, авторизацію через BankID, підтримку Apple Pay/Google Pay, а також інтерактивний чат-бот для підтримки користувачів.

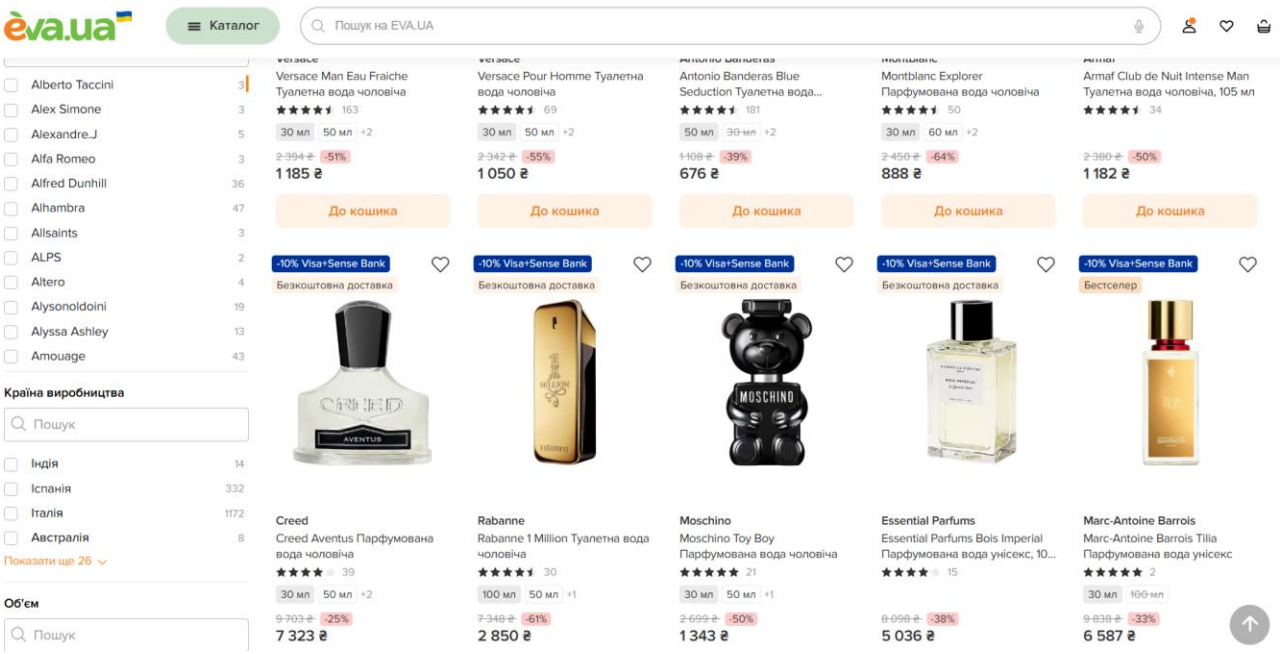


Рисунок 1.6 – Сторінка каталогу чоловічої парфумерії з набором фільтрів на сайті магазину Eva

Функціональні елементи, які заслуговують на увагу:

– інтеграція з бонусною програмою;

– мультирежим оплати (включаючи Pay-картки та післяплату);

– карта магазинів та функція самовивозу;

– мобільна адаптація та PWA-функціональність (прогресивний веб-додаток).

Міжнародна компанія Notino (notino.ua), яка успішно працює в Україні, демонструє високий рівень локалізації та клієнтоорієнтованості. Інтерфейс платформи є мінімалістичним, проте візуально привабливим, із пріоритетом на UX-дизайн та швидкий доступ до ключових розділів.

Картка товару включає докладний опис, фото, сертифікацію, рейтинг користувачів і рекомендації (рис. 1.6). Окремої уваги заслуговує система «подарункових упаковок», яка дозволяє покупцю оформити замовлення як подарунок.

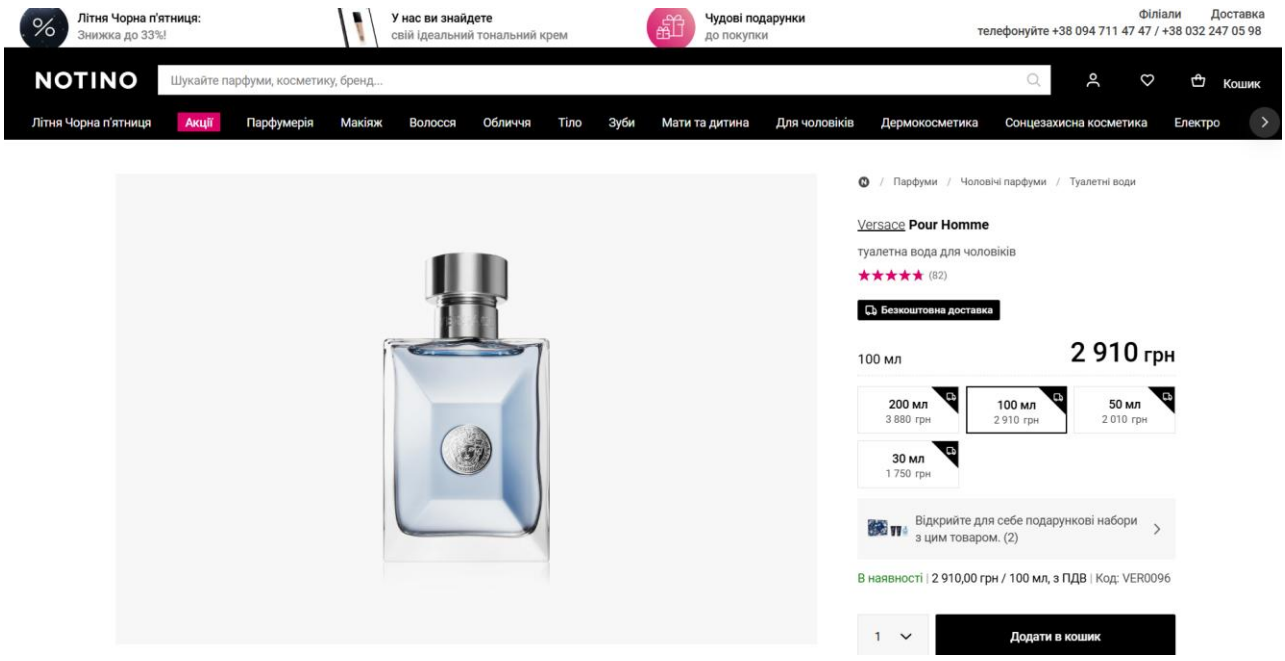


Рисунок 1.6 – Інтерфейс сторінки окремого товару інтернет-магазину Notino

Функції, які варто врахувати:

– реалізація подарункових наборів;

– підтримка багатомовності (українська, англійська);

– наявність розділу “Елітна косметика” (для сегментації товарів за преміум-ознаками);

– email-повідомлення про наявність товару, що закінчився.

На основі проведеного аналізу провідних світових (Sephora, Glossier, Lush) та українських (MakeUp, EVA.UA, Notino) інтернет-магазинів можна дійти висновку, що сучасна платформа для онлайн-торгівлі косметичними засобами має забезпечувати не лише базову функціональність перегляду й купівлі товарів, а й низку розширених можливостей, орієнтованих на зручність користувача, персоналізацію, інтеграцію з соціальними та платіжними сервісами, а також аналітику та контроль з боку адміністратора.

Спільними характеристиками аналізованих систем є:

– гнучка багаторівнева фільтрація товарів;

– мобільна адаптивність;

– підтримка відгуків і рейтингів;

– можливість замовлення як зареєстрованим користувачем, так і гостем;

– інтеграція з системами оплати, логістики та підтримки клієнтів;

– наявність інструментів для керування вмістом з боку адміністратора.

Для підвищення конкурентоспроможності доцільно також передбачити інноваційні функції, такі як онлайн-консультації, віртуальні примірки або соціальні інтеграції.

На підставі проведеного аналізу можна визначити комплекс функціональних вимог, які мають бути реалізовані у розроблюваній інформаційній системі.

1.2 Розробка функціональних вимог до розроблюваного інтернет-магазину косметичних засобів

Аналіз функціоналу провідних світових та українських інтернет-магазинів косметичних засобів дозволив виявити загальні тенденції та ключові очікування користувачів. Усі досліджені платформи надають пріоритет адаптивності, швидкості взаємодії, гнучкості оплати та функціям персоналізації (відгуки, обране, подарункові сервіси).

Результати можна узагальнити у вигляді наступних функціональних вимог. Зокрема, для користувача передбачено наступний функціонал:

1. Реєстрація та автентифікація:

– реєстрація за email або номером телефону;

– авторизація через сторонні сервіси (Google, Facebook — опційно);

– відновлення паролю;

– підтвердження email.

2. Каталог товарів:

– відображення товарів за категоріями;

– можливість сортування (за ціною, рейтингом, популярністю);

– розширена фільтрація (бренд, тип шкіри, об’єм, склад, країна виробника

тощо);

– пошук за ключовими словами.

3. Картка товару:

– зображення у високій якості;

– докладний опис та характеристики;

– відгуки та рейтинг;

– кнопка додавання до кошика / улюбленого;

– інформація про наявність.

4. Кошик і оформлення замовлення:

– огляд обраних товарів;

– модифікація кількості або видалення позицій;

– вибір способу доставки (кур’єром, самовивіз);

– вибір методу оплати (онлайн, післяплата);

– оформлення замовлення як зареєстрованим, так і гостьовим користувачем;

– отримання підтвердження про замовлення на email.

5. Особистий кабінет:

– перегляд історії замовлень;

– редагування профілю;

– перегляд збережених товарів;

– можливість залишати відгуки.

6. Додатковий функціонал:

– адаптивний дизайн для мобільних пристроїв;

– підтримка багатомовності;

– можливість створення подарункового набору;

– розділи “Акції”, “Новинки”, “Подарункові ідеї”;

– контактна форма або онлайн-чат підтримки.

Функціонал адмін-панелі недоступний для безпосереднього дослідження, але про нього можна скласти враження при користуванні клієнтською частиною. Тож, адмін-панель повинна підтримувати наступний функціонал:

1. Управління товарами:

– додавання, редагування, видалення товарів;

– зміна статусу (активний/неактивний);

– управління фото-галереєю товару;

– задання параметрів фільтрації (бренд, категорія, тип тощо).

2. Управління замовленнями:

– перегляд і редагування статусу замовлень (нове, в обробці, відправлено, завершено);

– перегляд інформації про клієнта;

– генерація списків замовлень для логістики.

3. Управління категоріями та брендами:

– додавання нових категорій/брендів;

– редагування назв, описів, порядку відображення.

4. Управління користувачами:

– перегляд зареєстрованих користувачів;

– призначення ролей (користувач, адміністратор);

– блокування/розблокування користувачів.

5. Модерація контенту:

– перевірка та видалення відгуків;

– додавання інформаційних блоків, акцій, банерів.

6. Звіти та аналітика:

– звіт про кількість замовлень за період;

– товари з найбільшою кількістю продажів;

– статистика реєстрацій та активності користувачів.

7. Налаштування сайту:

– редагування контактної інформації;

– керування мовними версіями сайту;

– налаштування способів доставки та оплати.

На основі даного аналізу було побудовано діаграму варіантів використання системи (рис. 1.7).

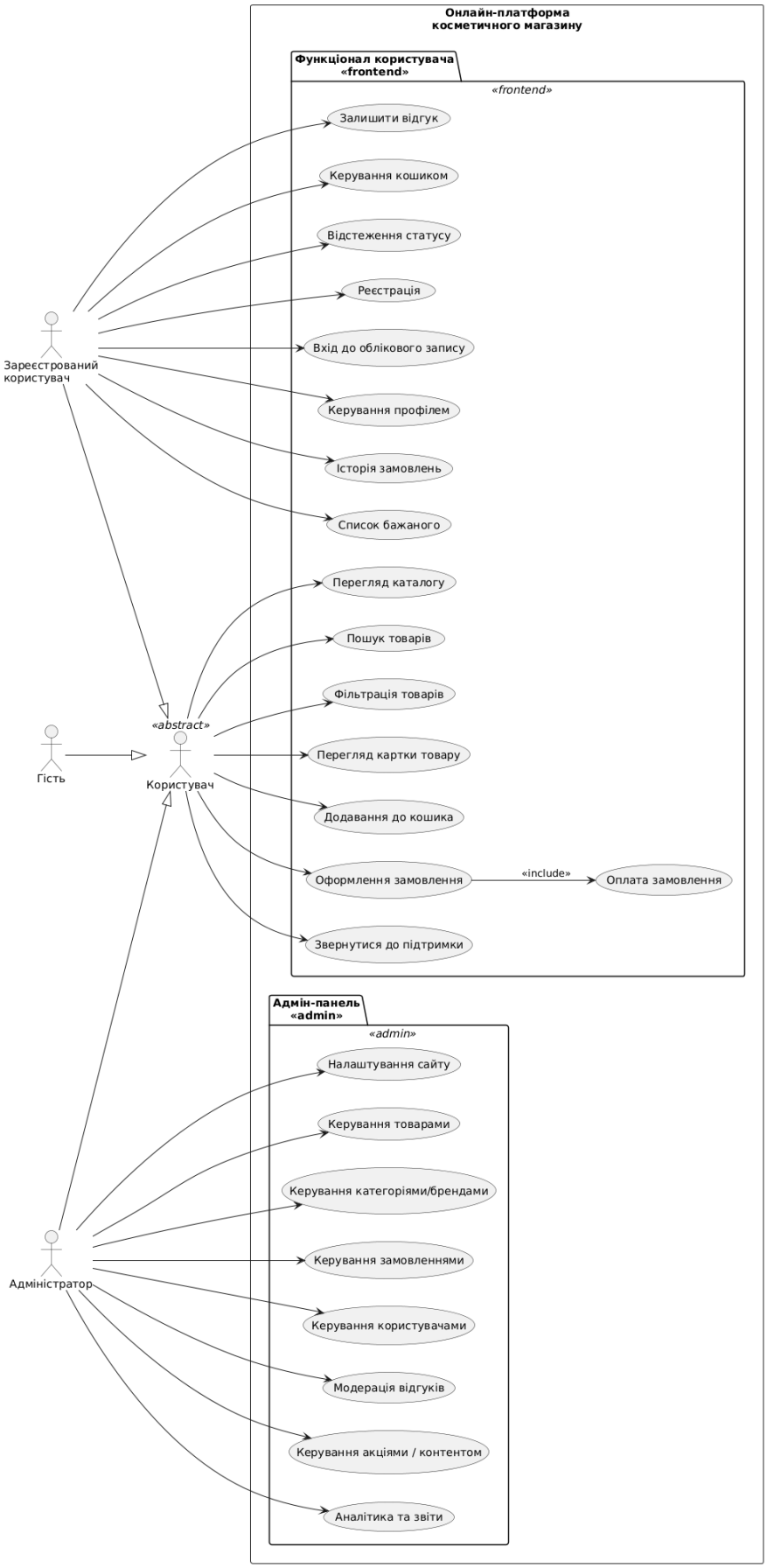


Рисунок 1.7 – Діаграма варіантів використання інтернет-магазину косметичних засобів

На основі узагальненого аналізу функціональності найуспішніших вітчизняних та зарубіжних інтернет-магазинів сформовано вичерпний перелік функціональних вимог до інформаційної системи, яка буде розроблена в межах випускної роботи бакалавру. Реалізація вищенаведених вимог дозволить створити конкурентоспроможний веб-ресурс, що забезпечить ефективну взаємодію з кінцевим користувачем та зручність керування для адміністратора системи.

1.3 Технічне обґрунтування технологій реалізації проєкту інтернет-магазину косметичних засобів

Для вибору технологічного стеку виконано аналіз сукупності функціональних (каталог, кошик, фільтри, обробка замовлень) та нефункціональних вимог (продуктивність, безпека, масштабованість, легкість підтримки, вартість експлуатації). Під час аналізу було вирішено зосередитися у першу чергу на фреймворках, які використовують архітектурний стиль MVC. Дане рішення обґрунтовується гарною індексацією контенту пошуковими системами, а також наявність експертних знань у розробника (мене). Було проаналізовано альтернативи у вигляді ASP.NET Core MVC, Node.JS/Express, Django та Laravel.

Результати виконаного аналізу наведені у таблиці 1.1. Деякі аспекти аналізу носять суб’єктивний характер, наприклад «крива навчання», і базуються на наявних знаннях мов C# та JS, які більш широко вивчалися під час навчання на освітній програмі. Якщо підвести підсумки, то завдяки показникам продуктивності, безпеки та наявності підтримки корпоративних сервісів Microsoft (зокрема аутентифікації/авторизації Identity) була обрана платформа ASP.NET Core MVC, яка, на мою думку, дозволить забезпечити найкраще співвідношення «функціонал –затрати праці» для цілей інтернет-магазину.

Таблиця 1.1 – Порівняльний аналіз можливих фреймворків для реалізації інтернет-магазину косметичних засобів

№ з/п	Критерій	ASP.NET Core MVC	Node.js/Express	Django	Laravel
1	Продуктивність	рідні async I/O, компіляція JIT	event-loop	-	-
2	Статична типізація	+	-	-	-
3	Інтеграція з MS SQL	рідна	через драйвери	ODBC/PyODBC	Doctrine
4	Безпека «з коробки»	Anti-CSRF	Мінімальна	Вбудована	Вбудована
5	Підтримка багаторівневого DI	+	сторонні	сторонні	обмежена
6	Крива навчання (при наявності знань C#)	низька	помірна	висока	висока

Вибір технології ASP.NET Core MVC для реалізації проєкту також визначає необхідність застосування пов’язаних технологій. Результати вибору та їх технічне обґрунтування наведені у таблиці 1.2.

Очікувані переваги обраного стека:

– масштабованість: горизонтальне та вертикальне масштабування через Azure App Service;

– готові шаблони MVC + Bootstrap прискорюють прототипування; EF Core автоматизує міграції;

– безпека: результати статичного аналізу (Roslyn Analyzers) та підтримка OWASP-рекомендацій у стандартних middleware (AddAntiforgery, HSTS);

– довготривала підтримка LTS-версій .NET 8, регулярні оновлення NuGet-пакетів.

Таблиця 1.2 – Вибрані технології та їх обґрунтування

Компонент	**Обрана технологія**	**Ключові аргументи**
Веб-фреймворк	ASP.NET Core MVC	Висока продуктивність (до 7 млн RPS у тестах .NET 8), кросплатформеність, відкритий код та активна підтримка Microsoft. Інтегрована DI-інфраструктура, middleware-конвеєр і власний засіб хостингу Kestrel забезпечують гнучке конфігурування та низьку латентність [1-3].
Мова	C# 12	Статична типізація, LINQ, async/await та розвинена екосистема пакетів NuGet сприяють швидкій розробці, мінімізуючи помилки на етапі компіляції.
Архітектурний стиль	MVC + Layered	Чітке розділення Presentation, Business та Data Access шарів спрощує тестування і підтримку.
ORM	Entity Framework Core	Підтримка Code-First-міграцій, change-tracking, batch-оновлень; оптимізація запитів через compiled queries й кеш перетворень [4].
СУБД	Microsoft SQL Server 2022	ACID-гарантії, вбудовані механізми реплікації та Always Encrypted, тісна інтеграція з EF Core. У 2024 р. SQL Server згадується серед «top enterprise DB» завдяки розвиненим інструментам BI та безпеки [5, 6].
UI-фреймворк	Razor Views + Bootstrap 5.3	Mobile-first сітка Flexbox, готові компоненти та висока популярність у 2024 р. (топ-3 у рейтингах front-end-фреймворків) [7-8].

Продовження табл. 1.2

Компонент	**Обрана технологія**	**Ключові аргументи**
Аутентифікація	ASP.NET Identity + OAuth 2.0	Підтримка email, соцмереж, двофакторної верифікації.
Хостинг	Azure App Service	Автоматичне горизонтальне масштабування, SLA 99,95 %, інтегрований Application Insights. На Build 2024 представлено функції autoscale на основі AI-прогнозів навантаження, що критично для пікових розпродажів [9].

Таким чином, виконаний аналіз та комплексне оцінювання показало, що фреймворк ASP.NET Core MVC з супровідними інструментами (EF Core, Bootstrap 5, Elasticsearch) та екосистемою Microsoft Azure найкраще відповідає як функціональним, так і нефункціональним вимогам проєкту інтернет-магазину косметики. Обраний стек забезпечує баланс продуктивності, безпеки та економічної доцільності, що підтверджено актуальними публікаціями й практикою впровадження e-commerce-рішень у 2024-2025 рр.

Висновки до розділу:

У межах першого розділу було здійснено систематичний аналіз сучасних веб-рішень у сфері електронної комерції, спрямований на формування функціональних і технологічних вимог до інформаційної системи інтернет-магазину косметичних засобів. Розглянуто як зарубіжні, так і українські приклади реалізації онлайн-платформ, що дозволило отримати цілісне уявлення про актуальні тенденції, інструменти та підходи у галузі.

Зокрема, проведено огляд функціональних можливостей провідних світових інтернет-магазинів, таких як Sephora, Glossier і Lush, що відзначаються

високим рівнем персоналізації користувацького досвіду, інтеграцією з системами штучного інтелекту (наприклад, віртуальна примірка), гнучкою архітектурою та мультимовною підтримкою. Усі три платформи мають розширену систему фільтрації, модуль відгуків, адаптивний інтерфейс і ефективну організацію взаємодії з користувачем. Ці приклади демонструють важливість орієнтації на користувача, швидкодію та аналітичну підтримку прийняття рішень у контексті комерційного інтерфейсу.

Окрему увагу приділено аналізу українських онлайн-магазинів косметики, зокрема MakeUp, EVA.UA та Notino.ua. Вони вирізняються локалізацією функціоналу, адаптацією до потреб національного ринку, інтеграцією з бонусними програмами, мультиканальністю (доставка/самовивіз), а також підтримкою сучасних платіжних засобів. На основі порівняння цих платформ виокремлено функції, які є стандартом для українського користувача: гостьове оформлення замовлення, розділи акцій та новинок, багаторівнева фільтрація, а також оптимізований мобільний інтерфейс.

У результаті узагальнення функціональності як глобальних, так і локальних ресурсів сформульовано комплекс функціональних вимог до проєктованої інформаційної системи. Вони охоплюють ключові підсистеми: модуль управління каталогом товарів, систему кошика й замовлень, реєстрацію та облікові записи користувачів, модуль зворотного зв’язку, адаптивність інтерфейсу, підтримку пошуку й фільтрації, багатомовність, систему рецензій, а також адміністративну панель, яка включає функції управління вмістом, користувачами, замовленнями та аналітикою.

Також у межах розділу проаналізовано технологічні основи реалізації розглянутих платформ, зокрема стек використовуваних мов програмування, архітектурні патерни (MVC, headless CMS, PWA), платформи для розгортання, системи управління базами даних і засоби забезпечення безпеки та продуктивності. Здійснено порівняння можливих технологічних варіантів реалізації з точки зору масштабованості, підтримуваності, вартості та

відповідності поставленим цілям.

На основі цього обґрунтовано вибір технологічного стеку на основі ASP.NET Core MVC, який забезпечує високу продуктивність, безпеку, підтримку сучасних архітектурних підходів і масштабованість. ASP.NET Core MVC обрано як основу для реалізації логіки бізнес-процесів, взаємодії з базами даних через ORM Entity Framework Core, організації доступу користувачів, а також розгортання платформи в хмарному середовищі (наприклад, Azure). Bootstrap використано для фронтенд-інтерфейсу, а MS SQL Server — як основну СУБД проєкту.

Таким чином, у першому розділі закладено теоретичну й аналітичну основу для подальшого проєктування системи, визначено перелік функціональних вимог до майбутнього інтернет-магазину, а також обґрунтувавши доцільність вибору відповідного стеку технологій для його реалізації.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ІНТЕРНЕТ-МАГАЗИНУ КОСМЕТИЧНИХ ЗАСОБІВ

2.1 Проєктування каркасу інтернет-магазину з використанням фреймворку ASP.NET Core

Розроблений інтернет-магазин косметичних засобів побудовано на основі сучасного веб-фреймворку ASP.NET Core MVC, що реалізує патерн Model-View-Controller. Такий підхід забезпечує чітке розмежування відповідальностей між бізнес-логікою, інтерфейсом користувача та обробкою запитів, що сприяє підвищенню гнучкості, тестованості та масштабованості проєкту [12].

Для підвищення модульності та підтримки принципів чистої архітектури, основні доменні класи, що моделюють сутності предметної області, виділені у окрему бібліотеку класів – MakeupClassLibrary (рис. 2.1). Це дозволяє відокремити бізнес-логіку від шарів презентації та доступу до даних, а також спростити повторне використання доменних моделей у різних частинах системи.

У основному проєкті дотримано наступної організації каталогів:

– Models – містить класи моделей, що передаються між контролерами та представленнями, включно з DTO (Data Transfer Objects) для передачі даних між шарами та ViewModels для формування структур, оптимізованих під потреби відображення на представленнях;

– Controllers – папка з контролерами, які реалізують бізнес-логіку та взаємодіють із сервісами, обробляють HTTP-запити і повертають відповідні результати у вигляді представлень або даних;

– Views – каталоги з Razor-представленнями, що відповідають за відображення інформації користувачу; представлення організовані відповідно до контролерів для логічного розділення інтерфейсу.

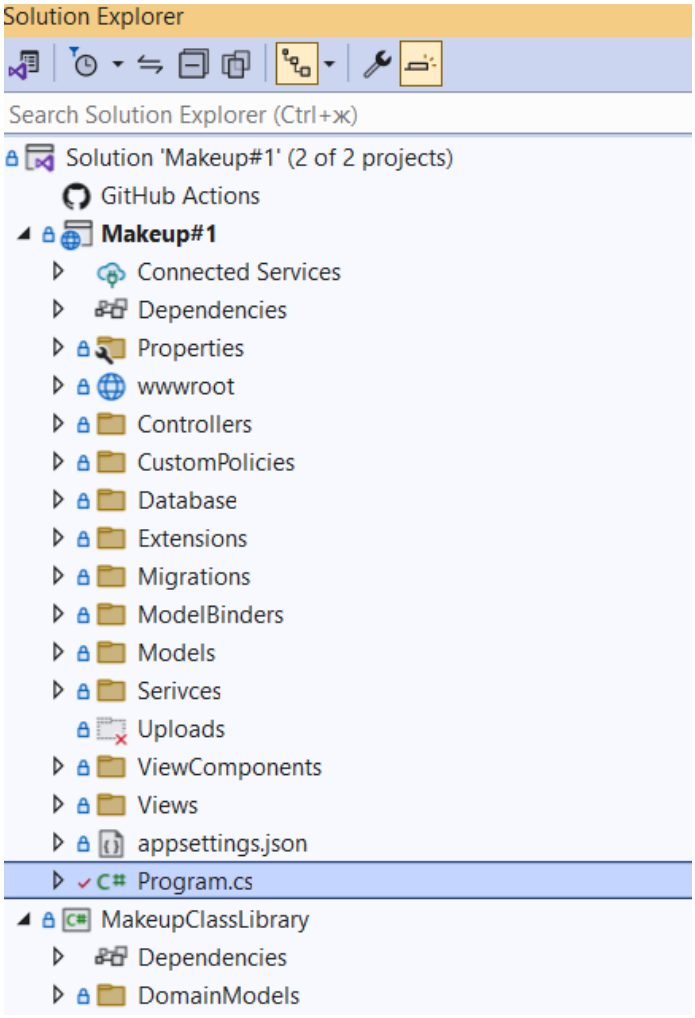


Рисунок 2.1 – Структура проєкту ASP.NET-додатку магазину косметичних засобів

– Services – сервіси, що інкапсулюють окремі функції, наприклад, сервіс відправки електронних повідомлень (EmailService); вони ізольовані від контролерів для полегшення тестування та повторного використання;

– Extensions – класи розширень, які додають додаткову функціональність до стандартних типів або фреймворку (прикладом є розширення для роботи із сесіями, що спрощує збереження та отримання даних користувача під час сеансу);

– ViewComponents – компоненти представлень, які забезпечують повторне використання частин інтерфейсу, наприклад, відображення корзини чи меню, та можуть включатися у різні сторінки;

– ModelBinders – реалізація кастомних прив’язувачів моделей, що дозволяють адаптувати стандартний механізм прив’язки даних HTTP-запиту до специфічних потреб проєкту (наприклад, складні типи або специфічна логіка десеріалізації, як от корзина);

– Database – містить клас контексту БД, що використовується ORM-фреймворком Entity Framework Core для проєкції колекцій сутностей бізнес-логіки на таблиці БД;

– Migrations – папка класів з описом міграцій БД, необхідних для підтримки життєвого циклу БД;

– CustomPolicies – реалізація власних політик авторизації, що розширюють базову функціональність системи безпеки ASP.NET Core; дозволяють гнучко налаштовувати правила доступу до ресурсів залежно від ролей, користувачів чи додаткових умов.

Розподіл функціональності по окремих папках і бібліотеках дозволяє [13]:

– підвищити модульність коду та спростити його підтримку;

– забезпечити можливість командної розробки з чітким розподілом обов’язків;

– полегшити тестування окремих компонентів;

– зменшити зв’язність між шарами і збільшити масштабованість застосунку.

Файл Program.cs у проєкті виконує роль головної точки входу для ASP.NET Core-додатка, у якій конфігуруються впровадження сервісів як залежностей у необхідні компоненти інфраструктури, бази даних, автентифікація, маршрутизація та проміжне програмне забезпечення (middleware). У межах реалізації інтернет-магазину косметичних засобів конфігурація сервісів і конвеєра HTTP-запитів була налаштована відповідно до вимог системи та принципів безпечного доступу [14].

Властивість builder.Services екземпляру класу WebApplicationBuilder, яка уявляє собою реалізацію інтерфейсу IServiceCollection, дозволяє виконати реєстрацію сервісів, необхідних для повноцінного функціонування застосунку:

1. Інфраструктура контролерів із представленнями (MVC):

```
18  builder.Services.AddControllersWithViews(options => {
19      options.ModelBinderProviders.Insert(0, new CartModelBinderProvider());
20  });
```

Забезпечує підтримку MVC-шаблону. Додатково було інтегровано спеціальний постачальник моделі CartModelBinderProvider, який відповідає за прив'язку параметрів об'єкта кошика до дій контролера [15].

2. Сервіс кешування у пам'яті:

```
14  builder.Services.AddDistributedMemoryCache();
```

Дозволяє зберігати дані сесій у оперативній пам'яті серверу, що оптимізує доступ до тимчасових даних.

3. Контекст бази даних:

```
21  builder.Services.AddDbContext<ShopContext>(options => {
22
23      options.UseSqlServer(
24          builder.Configuration
25          .GetConnectionString("MakeupDB"));
26  });
```

Підключає контекст ShopContext, який працює з СУБД SQL Server. Налаштування отримуються з конфігураційного файлу (appsettings.json) через ім'я з'єднання «MakeupDB».

4. Реєстрація кастомного обробника політик авторизації:

```
27  builder.Services.AddTransient<IAuthorizationHandler, AllowUserHandler>();
```

Додає підтримку кастомних політик доступу через реалізацію інтерфейсу IAuthorizationHandler.

5. Реєстрація сервісу електронної пошти:

```
28  builder.Services.AddTransient<IEmailService, EmailService>();
```

Сервіс EmailService використовується для надсилання повідомлень (наприклад, підтвердження замовлення, реєстрації, відновлення пароля).

6. Підтримка сесій користувача:

```
29  builder.Services.AddSession(options =>
30  {
31      options.IdleTimeout = TimeSpan.FromMinutes(10);
32      options.Cookie.IsEssential = true;
33  });
```

Забезпечує зберігання даних користувача (наприклад, кошика) між HTTP-запитами. Тайм-аут сесії становить 10 хвилин.

7. Інтеграція системи автентифікації ASP.NET Core Identity:

```
34      builder.Services
35          .AddIdentity<User, IdentityRole>((IdentityOptions options) =>
36          {
37              options.User.RequireUniqueEmail = true;
38              options.Password.RequiredLength = 1;
39              options.Password.RequireNonAlphanumeric = false;
40              options.Password.RequireLowercase = false;
41              options.Password.RequireUppercase = false;
42              options.Password.RequireDigit = false;
43          })
44          .AddEntityFrameworkStores<ShopContext>()
45          .AddDefaultTokenProviders();
```

Система Identity налаштовується з базовими параметрами безпеки. Зокрема, вимоги до паролів максимально спрощені для цілей тестування. Identity інтегрується з Entity Framework Core через AddEntityFrameworkStores<ShopContext>() [16].

8. Підтримка увтентифікації через Google та Facebook:

```
46      builder.Services.AddAuthentication()
47          .AddGoogle(googleOptions =>
48          {
49              googleOptions.ClientId = configuration["Authentication:Google:ClientId"];
50              googleOptions.ClientSecret =
51              configuration["Authentication:Google:ClientSecret"];
52              googleOptions.SignInScheme = IdentityConstants.ExternalScheme;
53          })
54          .AddFacebook(fbOptions =>
55          {
56              IConfigurationSection fbAuthSection = configuration
57              .GetSection("Authentication:Facebook");
58              fbOptions.AppId = fbAuthSection.GetSection("AppId").Value;
59              fbOptions.AppSecret = fbAuthSection.GetSection("AppSecret").Value;
60          });
```

Додається підтримка сторонніх провайдерів аутентифікації на основі протоколу OAuth 2.0. Налаштування (ID клієнта, секретні ключі) зчитуються із секції Configuration.

Після побудови застосунку виконується конфігурація конвеєра обробки HTTP-запитів, у якому визначено послідовність middleware-компонентів, зокрема, обробка виключень у режимі продакшн, перенаправлення з HTTP на HTTPS, обслуговування статичних файлів (CSS, JS, зображення),

маршрутизація запитів, сесії користувача, аутентифікація та авторизація, налаштування маршруту за замовчуванням та запуск застосунку:

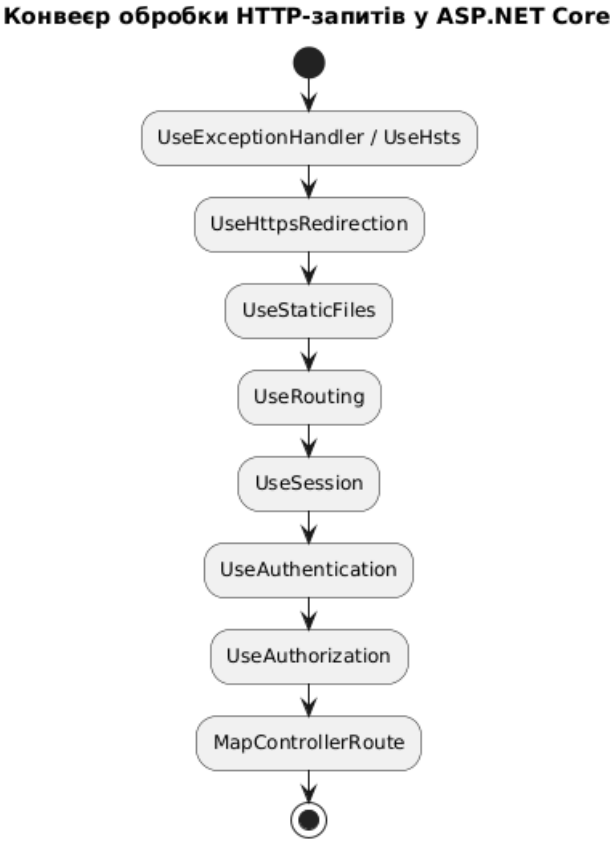
```
64      if (!app.Environment.IsDevelopment())
65      {
66          app.UseExceptionHandler("/Home/Error");
67          // The default HSTS value is 30 days. You may want to cl
68          app.UseHsts();
69      }
70
71      app.UseHttpsRedirection();
72      app.UseStaticFiles();
73
74      app.UseRouting();
75      app.UseSession();
76      app.UseAuthentication();
77      app.UseAuthorization();
78
79      app.MapControllerRoute(
80          name: "default",
81          pattern: "{controller=Home}/{action=Index}/{id?}");
82
83      app.Run();
```

Як видно з налаштувань маршруту за замовчанням, проєкт використовує стандартні трьохсегментні маршрути типу {controller}/{action}/{id?}, які дозволяють виконувати проєкцію запитів на методи дії контролерів та враховувати третій необов'язковий параметр – ідентифікатор сутності, з якою виконуються операції.

На рис. 2.2 наведено візуалізацію конвеєра обробки запитів у розробленому проєкті з використанням діаграми діяльності (activity diagram) UML.

Таким чином, виконана конфігурація додатку у Program.cs забезпечує централізоване визначення всіх служб, залежностей та конфігураційних параметрів вебзастосунку. Завдяки використанню вбудованих механізмів Dependency Injection, підтримки сесій, кастомних політик безпеки, а також інтеграції з зовнішніми OAuth-провайдерами, система отримує необхідну

гнучкість, розширюваність і безпеку, що відповідає сучасним вимогам до веб-орієнтованих інформаційних ресурсів.



2.2 Розробка структури бази для проєкту інтернет-магазину косметичних товарів

В межах проєкту реалізації інтернет-магазину косметичних засобів база даних формується з використанням підходу Code First , що передбачає створення структури бази даних на основі визначених класів-моделей у середовищі розробки [17]. Для доступу до даних застосовується фреймворк об’єктно-реляційного відображення (ORM) Entity Framework Core, що забезпечує автоматичне формування схем, зв’язків, ключів та обмежень безпосередньо на підставі коду класів основних сутностей предметної галузі.

Основними сутностями проєкту є:

1. Користувачі (ShopUser). Клас ShopUserє похідним від базового класу IdentityUser та доповнений полем YearOfBirth. Завдяки використанню ASP.NET

Core Identity забезпечується розширене управління автентифікацією та авторизацією користувачів. Один користувач може мати багато замовлень (Order) та залишати коментарі (Comment), що реалізується через зовнішні ключі до таблиці користувачів.

2. Продукти (Product). Клас Product є центральною сутністю у системі. Він містить основні характеристики товару: назву, опис, рейтинг, ціну, спосіб застосування, а також зовнішній ключ BrandId, який пов’язує товар із виробником (брендом). Продукт може мати:

– багато зображень (Image);

– багато коментарів (Comment);

– належати до декількох категорій (Category) – зв’язок багато-до-багатьох;

– входити до елементів замовлення (OrderItem).

3. Категорії (Category). Категорії реалізовано за підтримкою ієрархічної структури: кожна категорія може мати батьківську та дочірні категорії (рекурсивний зв’язок один-до-багатьох). Також категорії пов’язані з продуктами через колекцію Products, що формує зв’язок багато-до-багатьох (який EF Core реалізує через проміжну таблицю).

4. Бренди (Brand). Клас Brand описує виробника товарів: назву, країну походження. Один бренд може мати багато товарів – зв’язок один до багатьох.

5. Зображення (Image). Суть призначена для зберігання файлів Image зображень, які асоційовані з конкретним товаром (ProductId).

6. Коментарі (Comment). Коментарі містять заголовок, опис, дату створення, а також пов’язані з користувачем (зовнішній ключ UserId) та продуктом (зовнішній ключ ProductId). Таким чином, реалізуються зв’язки:

– багато коментарів до одного продукту ;

– багато коментарів від одного користувача .

7. Замовлення (Order). Сутьність Order містить дату створення та колекцію елементів замовлення (OrderItem). Також вона пов'язана з користувачем (зовнішній ключ UserId), який здійснив замовлення – зв’язок багато до одного

8. Елементи замовлення (OrderItem). Кожен елемент замовлення включає посилання на продукт (ProductId) та кількість одиниць. Елемент належить до замовлення (зовнішній ключ OrderId).

На основі даних класів побудовано клас контексту БД (рис. 2.3), який містить колекції екземплярів відповідних класів у вигляді властивостей типу DbSet<>, типізованих вказаними класами.

```
5   namespace Makeup_1.Database
6   {
        21 references
7       public class ShopContext : IdentityDbContext<User>
8       {
9
10
            0 references
11          public ShopContext(DbContextOptions<ShopContext> options)
12              : base(options) { }
13
            17 references
14          public DbSet<Brand> Brands { get; set; } = default!;
15
            24 references
16          public DbSet<Category> Categories { get; set; } = default!;
17
            20 references
18          public DbSet<Product> Products { get; set; } = default!;
19
            1 reference
20          public DbSet<Comment> Comments { get; set; } = default!;
21
            13 references
22          public DbSet<Image> Images { get; set; } = default!;
23
            0 references
24          public DbSet<OrderItem> OrderItems { get; set; } = default!;
25
            1 reference
26          public DbSet<Order> Orders { get; set; } = default!;
27      }
28  }
```

Рисунок 2.3 – Клас контексту бази даних інтернет-магазину косметичних засобів

На рис. 2.4 наведено діаграму класів, що відображає основні сутності та зв’язки між ними.

Варто відзначити наступні основні характеристики та принципи, використані при реалізації структури бази даних:

– автоматичне створення таблиць та зв’язків реалізується за допомогою механізму міграцій EF Core;

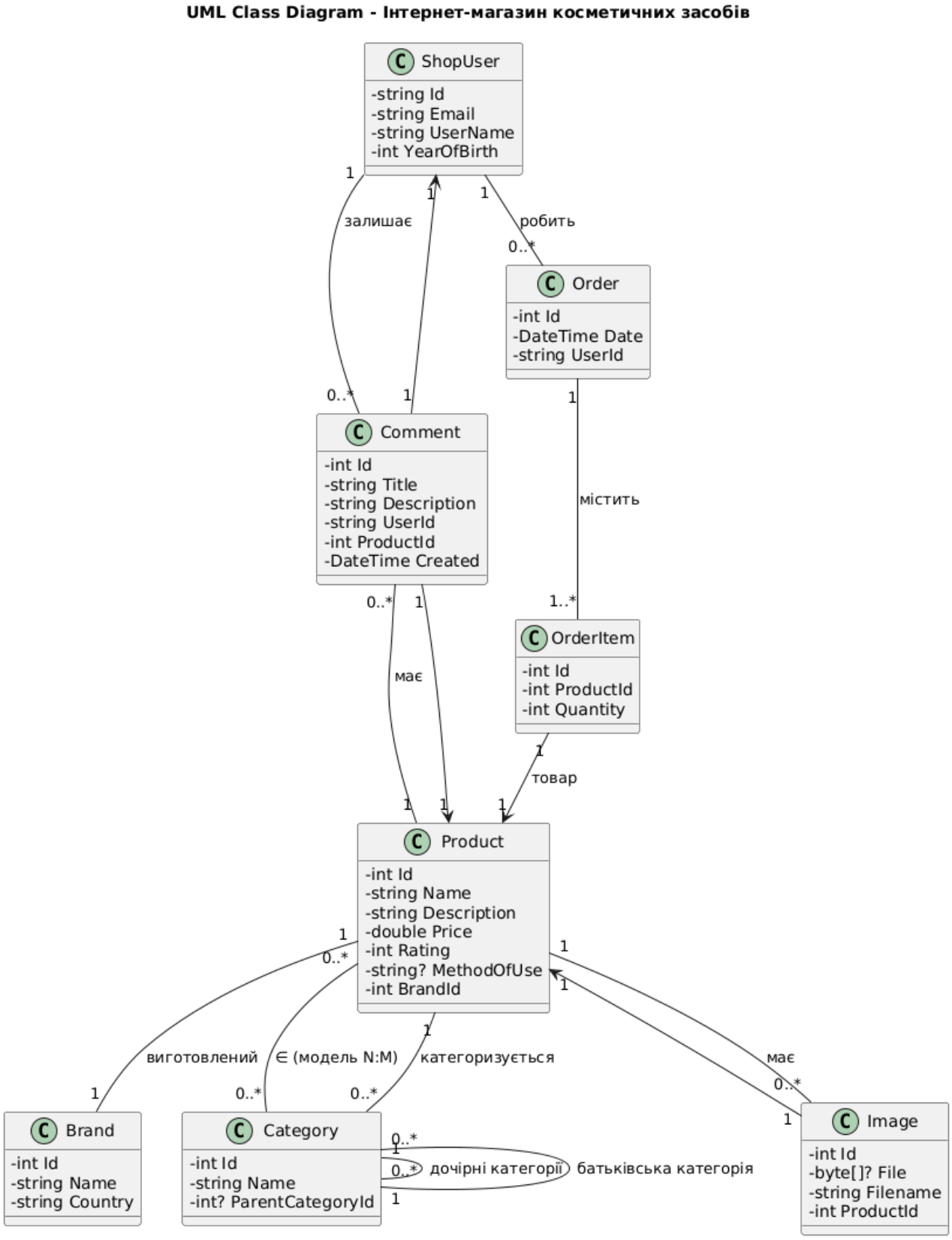


Рисунок 2.4 –UML-діаграма класів основних сутностей проєкту інтернет-магазину косметичних засобів

– первинні ключі – всі сутності містять ключове поле Id або використовують успадковані ключі (ShopUser);

– зовнішні ключі явно задаються за допомогою анотації [ForeignKey] чи неявно через навігаційні властивості;

– анотації відображення [Display(Name = "...")] використовуються для забезпечення локалізованого інтерфейсу в UI, однак не впливають на структуру БД.

– реляційна СУБД – як основа передбачається використання Microsoft SQL Server.

З метою реалізації надійної та розширюваної системи автентифікації та авторизації в межах інтернет-магазину косметичних засобів використовується інфраструктура ASP.NET Core Identity [16].

ASP.NET Core Identity надає засоби для створення та управління обліковими записами користувачів, ролями, правами доступу, а також інтеграції із зовнішніми провайдерами автентифікації на основі протоколу OAuth 2.0.

У контексті реалізації моделі доступу та управління ідентичністю користувачів у застосунку ключову роль відіграють наступні базові класи:

1. IdentityUser – є центральною сутністю, що представляє обліковий запис користувача у системі. Він забезпечує базову структуру для зберігання облікових даних, параметрів безпеки та ідентифікаційної інформації. Серед основних атрибутів класу:

– Id – унікальний ідентифікатор користувача;

– UserName, Email, PhoneNumber – атрибути ідентифікації та зв’язку з користувачем;

– PasswordHash- Збереження гешований пароль;

– SecurityStamp- службовий маркер безпеки для валідації сесій;

– TwoFactorEnabled, EmailConfirmed, LockoutEnabled – властивості, що визначають політику аутентифікації.

У межах проекту цей клас був успадкований у вигляді користувацького типу ShopUser, який додатково містить поле YearOfBirth, що дозволяє розширити профіль користувача відповідно до доменної моделі системи.

2. IdentityRole –уявляє роль у системі авторизації. Містить наступні основні властивості:

– Id – унікальний ідентифікатор ролі;

– Name, NormalizedName – текстове подання ролі, яке застосовується для порівняння;

– ConcurrencyStamp – маркер для контролю паралельного доступу.

У системі ролі дозволяють реалізувати логічну сегментацію доступу (наприклад, «Адміністратор», «Покупець», «Менеджер»), а також використовуються при конфігурації політик авторизації.

3. IdentityUserRole – є проміжною таблицею , яка реалізує зв’язок багато до багатьох між користувачами та ролями. Містить наступні основні властивості:

– UserId – зовнішній ключ до IdentityUser;

– RoleId – зовнішній ключ до IdentityRole.

Наявність такої сутності дозволяє одному користувачеві мати кілька ролей одночасно, а одній ролі охоплювати багатьох користувачів. Вище наведені властивості утворюють складений первинний ключ цієї таблиці.

4. IdentityUserClaim та IdentityRoleClaim

Класи IdentityUserClaim та IdentityRoleClaim відповідають за реалізацію моделі авторизації на основі тверджень (claims). Кожна твердження є парою тип-значення, що описує певну характеристику користувача чи ролі. Структура включає:

– ClaimType – тип твердження (наприклад, «EmailVerified», «Permission»);

– ClaimValue – Значення відповідного твердження;

– UserId або RoleId – зовнішні ключі до відповідної сутності.

Такий підхід дозволяє реалізовувати гнучку, декларативну авторизацію, яка виходитиме за межі простого поділу на ролі.

5. IdentityUserLogin – зберігає інформацію про автентифікацію через зовнішні провайдери (Google, Facebook, Microsoft тощо). Містить наступні основні властивості:

– LoginProvider – назва провайдера аутентифікації;

– ProviderKey – унікальний ідентифікатор користувача у сторонньому сервісі;

– ProviderDisplayName – текстова назва провайдера;

– UserId – зв’язок із локальним користувачем.

Таким чином, клас IdentityUserLogin забезпечує функціональність OAuth2.0-авторизації, зберігаючи прив’язку стороннього облікового запису до локального профілю.

6. IdentityUserToken – реалізує збереження токенів, які використовуються для таких функцій, як:

– підтвердження адреси електронної пошти;

– скидання пароля;

– зберігання двофакторного токена тощо.

Основними властивостями сутності є:

– UserId– зв’язок із користувачем;

– LoginProvider, Name – контекст і тип токена;

– Value – значення токена.

У відповідності до наведеної структури можна сформулювати загальну модель взаємозв’язків між класами Identity. Так, один користувач (IdentityUser) може:

– бути учасником багатьох ролей (IdentityUserRole);

– мати багато claims (IdentityUserClaim);

– бути аутентифікованим через кілька провайдерів (IdentityUserLogin);

– мати багато токенів (IdentityUserToken).

Одна роль (IdentityRole) може:

– охоплювати багато користувачів;

– містити твердження (IdentityRoleClaim).

Усі зв’язки реалізуються через зовнішні ключі та підтримуються системою міграцій Entity Framework Core.

На рис. 2.5 наведено діаграму класів, що відображає основні класи бібліотеки Microsoft Identity та зв’язки між ними.

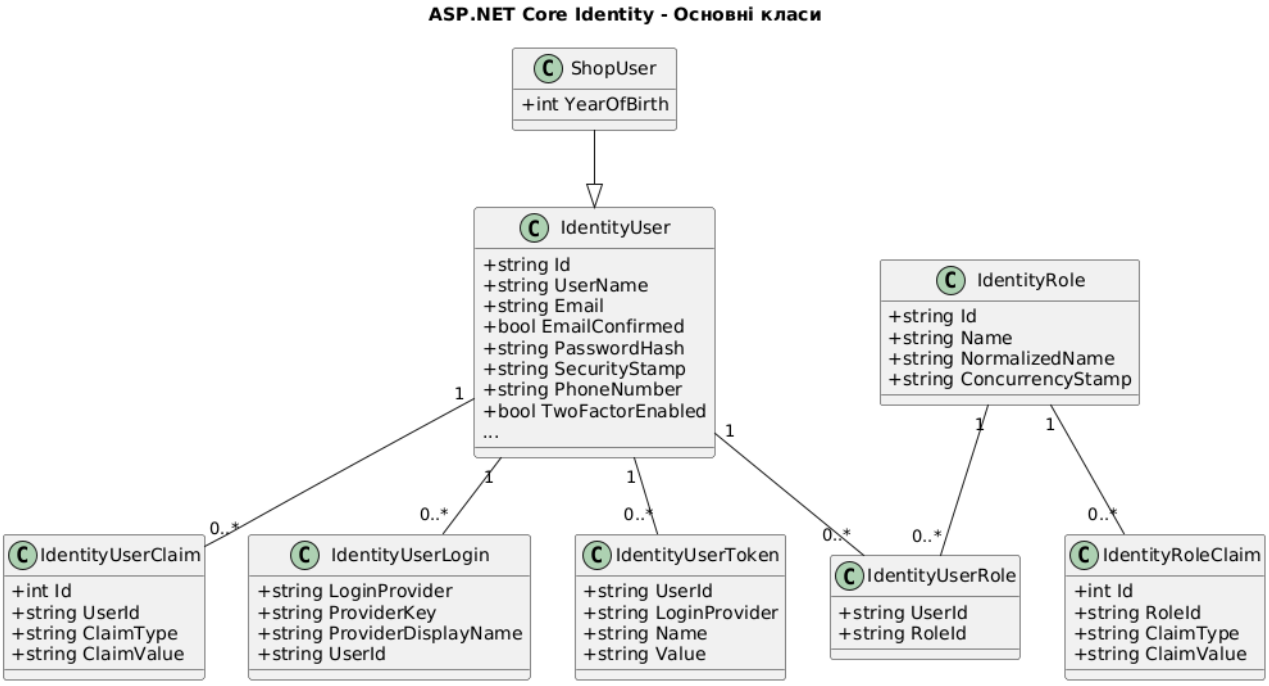


Рисунок 2.5 –UML-діаграма класів бібліотеки Identity, що використовується у проєкті

Таким чином, архітектура ASP.NET Core Identity є невід’ємною частиною інформаційної системи інтернет-магазину. Вона забезпечує повноцінну підтримку аутентифікації, авторизації, керування користувачами, а також дозволяє реалізовувати сучасні підходи до безпеки, зокрема автентифікацію через сторонні сервіси, двофакторну перевірку, керування токенами та доступ, заснований на твердженнях (claims-based). Завдяки використанню типових класів, таких як IdentityUser, IdentityRole та інших, система залишається стандартизованою, безпечною та легкою для розширення відповідно до вимог бізнес-логіки застосування.

Завдяки використанню підходу Code First база даних проекту формується таким чином, що поєднує високу гнучкість розробки з контролем над структурою даних і відповідністю бізнес-логіці системи.

2.3 Реалізація аутентифікації та авторизації користувачів у проєкті інтернет-магазину косметичних товарів

Для забезпечення конфіденційності, цілісності даних та контрольованого доступу до функціональних можливостей інформаційної системи інтернет-магазину реалізовано модуль автентифікації та авторизації користувачів. В основу реалізації покладено стандартну інфраструктуру ASP.NET Core Identity, яка забезпечує гнучке керування обліковими записами, ролями, зовнішньою аувтентифікацією та механізмами безпеки [16].

Аутентифікація реалізована шляхом реєстрації та входу користувачів за допомогою логіну (email) та паролю. У системі використовується клас ShopUser, який успадковує базову функціональність класу IdentityUser і розширюється додатковими полями, зокрема YearOfBirth, що дозволяє адаптувати профіль користувача до потреб доменної області.

Крім класичної форми входу, реалізовано зовнішню аутентифікацію за допомогою OAuth 2.0 через сторонніх провайдерів – Google та Facebook (у перспективі можлива інтеграція з GitHub, Microsoft та іншими службами) [16]. Це дозволяє спростити процес доступу для користувачів та забезпечити підтримку сучасних стандартів безпеки.

Взаємодія з провайдерами аетентифікації реалізується за допомогою класу SignInManager, який викликає метод Challenge, генерує відповідні метадані OAuth2-запиту, а також обробляє callback-запити від провайдера через метод ExternalLoginSignInAsync. У разі відсутності користувача у базі, створюється новий обліковий запис на основі отриманих з Google атрибутів (email, прізвище тощо).

Авторизація в системі реалізована на основі ролей користувачів. Основними типами користувачів є:

– адміністратори мають доступ до адміністративної панелі, управління товарами, категоріями, брендами, замовленнями;

– зареєстровані користувачі – можуть переглядати продукцію, додавати товари до кошика, створювати замовлення, залишати відгуки;

– гості – можуть лише переглядати продукцію, не маючи доступу до персоналізованого функціоналу.

Для реалізації ролей використовується клас IdentityRole і відповідна проміжна сутність IdentityUserRole, що дозволяє здійснювати зв’язок «багато до багатьох» між користувачами та ролями. Додатково у проєкті підтримуються механізми claims-based авторизації через класи IdentityUserClaim та IdentityRoleClaim, що в перспективі дозволяє реалізовувати гнучку політику доступу на основі тверджень.

Усі дані, пов’язані з автентифікацією та авторизацією, зберігаються у базі даних за допомогою контексту ShopContext , який є частиною Entity Framework Core. Міграції автоматично створюють необхідні таблиці.

З точки зору архітектури система автентифікації поділяється на такі компоненти [16]:

– інтерфейс користувача (UI): форми входу, реєстрації, підтвердження пошти;

– контролери (AccountController, AdminController), які обробляють запити від UI та взаємодіють із сервісами;

– сервіси UserManager, SignInManager,RoleManager, які інкапсулюють логіку управління обліковими записами, входом та ролями;

– база даних , що зберігає всі дані автентифікації;

– зовнішні провайдери (Google, Facebook).

Відповідна діаграма компонентів (рис. 2.6) відображає залежність між складовими системами.

У розробленій інформаційній системі інтернет-магазину реалізовано механізм реєстрації нових користувачів за допомогою засобів ASP.NET Core Identity. Відповідальний за цей процес метод Register розміщено у класі AccountController, який обробляє запити, пов’язані з аутентифікацією та обліковими записами користувачів.

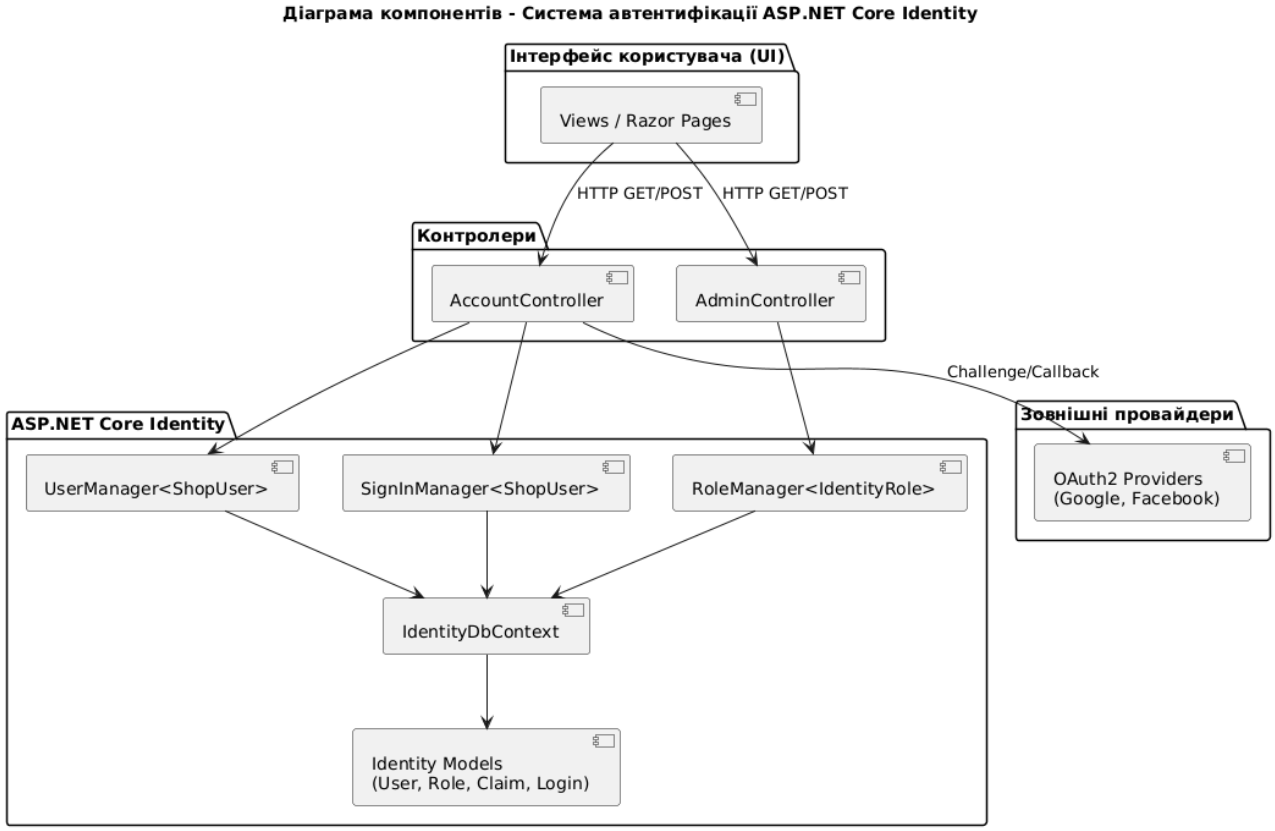


Рисунок 2.6 – UML- діаграма компонентів системи аутентифікації у проєкті інтернет-магазину

Код реалізації методів GET та POST наведено на рис. 2.7. HTTP метод GET призначений для повернення користувачу магазина форми реєстрації, а метод POST призначений для обробки переданих через форму даних.

Метод приймає модель типу RegisterUserDTO, яка містить персональні дані користувача (логін, електронну пошту, дату народження, пароль). На початковому етапі здійснюється перевірка коректності введених даних за допомогою механізму валідації моделі (ModelState.IsValid). У разі виявлення помилок, вони додаються до моделі стану, і користувачеві повторно повертається форма з відповідними повідомленнями.

У випадку, якщо вхідні дані є коректними, здійснюється створення нового об’єкта типу ShopUser, який є розширенням базового класу IdentityUser.

```
26      public IActionResult Register() => View();
27
28      [HttpPost]
        0 references
29      public async Task<IActionResult> Register(RegisterUserDTO dTO)
30      {
31          if (!ModelState.IsValid)
32              return View(dTO);
33          ShopUser shopUser = new ShopUser
34          {
35              UserName = dTO.Username,
36              Email = dTO.Email,
37              DateOfBirth = dTO.DateOfBirth,
38          };
39          var result = await userManager.CreateAsync(shopUser, dTO.Password);
40          if (result.Succeeded)
41          {
42              await signInManager.SignInAsync(shopUser, false);
43              return RedirectToAction("Index", "Home");
44          }
45          else
46          {
47              foreach (var error in result.Errors)
48              {
49                  ModelState.AddModelError(string.Empty, error.Description);
50              }
51          }
52          return View(dTO);
53      }
```

Рисунок 2.7 – Лістинг коду методів для обробки реєстрації користувачів на сайті магазину

Далі об’єкт передається до асинхронного методу CreateAsync об’єкта UserManager, що відповідає за реєстрацію облікових записів у базі даних. У разі успішної реєстрації користувача одразу авторизують за допомогою методу SignInAsync класу SignInManager, після чого здійснюється редирект на головну сторінку сайту (/Home/Index). У разі помилок при створенні облікового запису, система виводить відповідні повідомлення користувачу.

На основі вказаного методу розроблено діаграму послідовності (рис. 2.8), яка відображає основні етапи взаємодії між учасниками процесу під час реєстрації користувача.

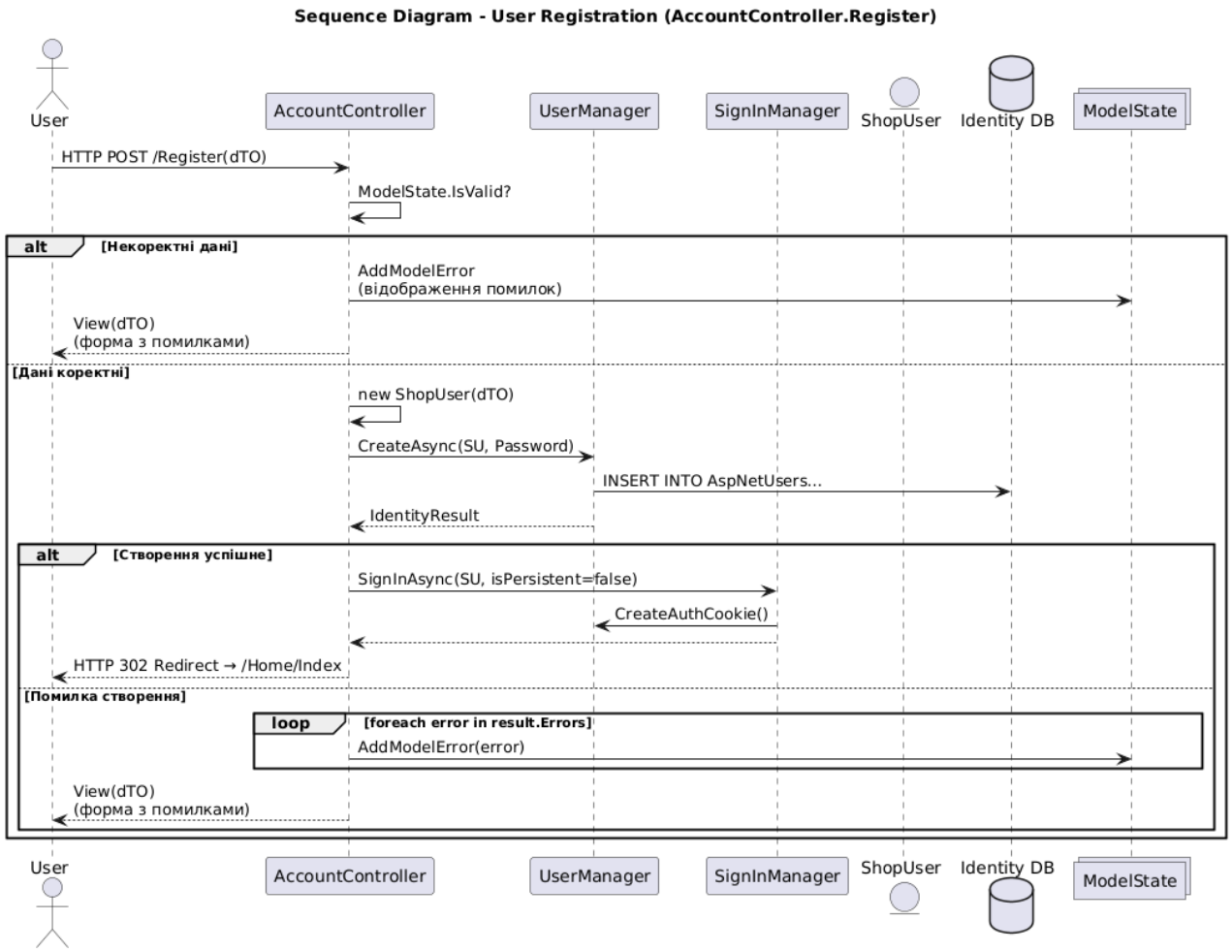


Рисунок 2.8 – UML- діаграма послідовності для реєстрації користувачів на сайті магазину

У діаграмі беруть участь такі ключові компоненти:

1. Актор User – ініціює запит на реєстрацію через форму.

2. Контролер AccountController – обробляє вхідний HTTP POST-запит, перевіряє модель, створює об’єкт користувача та координує подальші дії.

3. UserManager – компонент ASP.NET Core Identity, що реалізує логіку створення облікового запису та запису до бази даних.

4. SignInManager – забезпечує авторизацію користувача після успішної реєстрації.

5. База даних (Identity DB) – місце збереження облікових даних користувачів.

6. ModelState – система валідації моделі, що забезпечує інформування про помилки вводу.

Діаграма послідовності демонструє повну логіку процесу: від надсилання даних форми користувачем, перевірки на коректність, створення об’єкта користувача та його збереження у базі, до авторизації й перенаправлення на головну сторінку. Окремо розглянуто альтернативні гілки виконання, які реалізуються у випадку виявлення помилок валідації або невдалого створення облікового запису.

Таким чином, реалізований метод забезпечує повноцінну взаємодію між користувачем і системою керування ідентифікацією ASP.NET Core Identity, з дотриманням принципів безпеки, асинхронності та зручності користувача.

У межах реалізації функціоналу аутентифікації в інформаційній системі інтернет-магазину косметичних засобів передбачено можливість входу користувачів за допомогою облікового запису Google. Така інтеграція реалізується на основі протоколу OAuth 2.0 та стандартного інтерфейсу зовнішньої автентифікації, який надає фреймворк ASP.NET Core Identity.

Механізм реалізовано у вигляді двох методів у контролері AccountController: GoogleLogin та GoogleResponse (рис. 2.9).

Метод GoogleLogin ініціює процес аутентифікації, формуючи запит на авторизацію за допомогою зовнішнього провайдера. Для цього за допомогою SignInManager викликається метод ConfigureExternalAuthenticationProperties, який генерує параметри виклику стороннього сервісу. Сформований об’єкт передається до методу Challenge, який здійснює HTTP-перенаправлення користувача на сторінку входу в обліковий запис Google [16].

Після успішної авторизації на стороні Google користувач перенаправляється назад на визначену у returnUrl сторінку зворотного виклику — GoogleResponse. У цьому методі за допомогою GetExternalLoginInfoAsync отримується інформація про користувача, надана Google. Якщо з якихось причин відповідь відсутня, користувач повертається на сторінку звичайного входу.

```
[AllowAnonymous]
0 references
public IActionResult GoogleLogin()
{
    string? returnUrl = Url.Action("GoogleResponse", "Account");
    var properties = signInManager
        .ConfigureExternalAuthenticationProperties("Google", returnUrl);
    return Challenge(properties, "Google");
}

public async Task<IActionResult> GoogleResponse()
{
    ExternalLoginInfo? loginInfo = await signInManager
        .GetExternalLoginInfoAsync();
    if (loginInfo == null)
        return RedirectToAction("Login");
    var signInResult = await signInManager
        .ExternalLoginSignInAsync(
        loginInfo.LoginProvider,
        loginInfo.ProviderKey,
        isPersistent: false);
    if (!signInResult.Succeeded)
    {
        string[] userInfo =
        {
        loginInfo.Principal.FindFirst(ClaimTypes.Surname)!.Value!,
        loginInfo.Principal.FindFirst(ClaimTypes.Email)!.Value!
    };
        ShopUser? shopUser = await userManager.FindByEmailAsync(userInfo[1]);
        if (shopUser == null)
        {
            shopUser = new ShopUser
            {
                Email = userInfo[1],
                UserName = userInfo[1],
            };
            var createResult = await userManager.CreateAsync(shopUser);
        }
        var result = await userManager.AddLoginAsync(shopUser, loginInfo);
        await signInManager.SignInAsync(shopUser, isPersistent: false);
    }
    return RedirectToAction("Index", "Home");
}
```

Рисунок 2.9 – Лістинг коду методів для обробки аутентифікації за допомогою облікового запису Google

У разі успішного отримання зовнішньої інформації виконується спроба входу в систему через метод ExternalLoginSignInAsync. Якщо обліковий запис уже прив’язаний до відповідного зовнішнього ідентифікатора, система одразу

авторизує користувача та перенаправляє його на головну сторінку. Якщо ж обліковий запис не знайдено, зчитуються значення електронної пошти та прізвища з набору claims, які повернув провайдер.

На підставі отриманої електронної пошти виконується спроба знайти відповідного користувача у локальній базі даних через метод FindByEmailAsync. Якщо такий користувач відсутній, створюється новий об’єкт ShopUser, який реєструється в системі за допомогою методу CreateAsync. Після цього відбувається прив’язка облікового запису Google до локального профілю за допомогою AddLoginAsync, і користувач автоматично входить до системи.

Діаграма послідовності UML виконання аутентифікації в інформаційній системі інтернет-магазину косметичних засобів за допомогою облікового запису Google наведено на рис. 2.10.

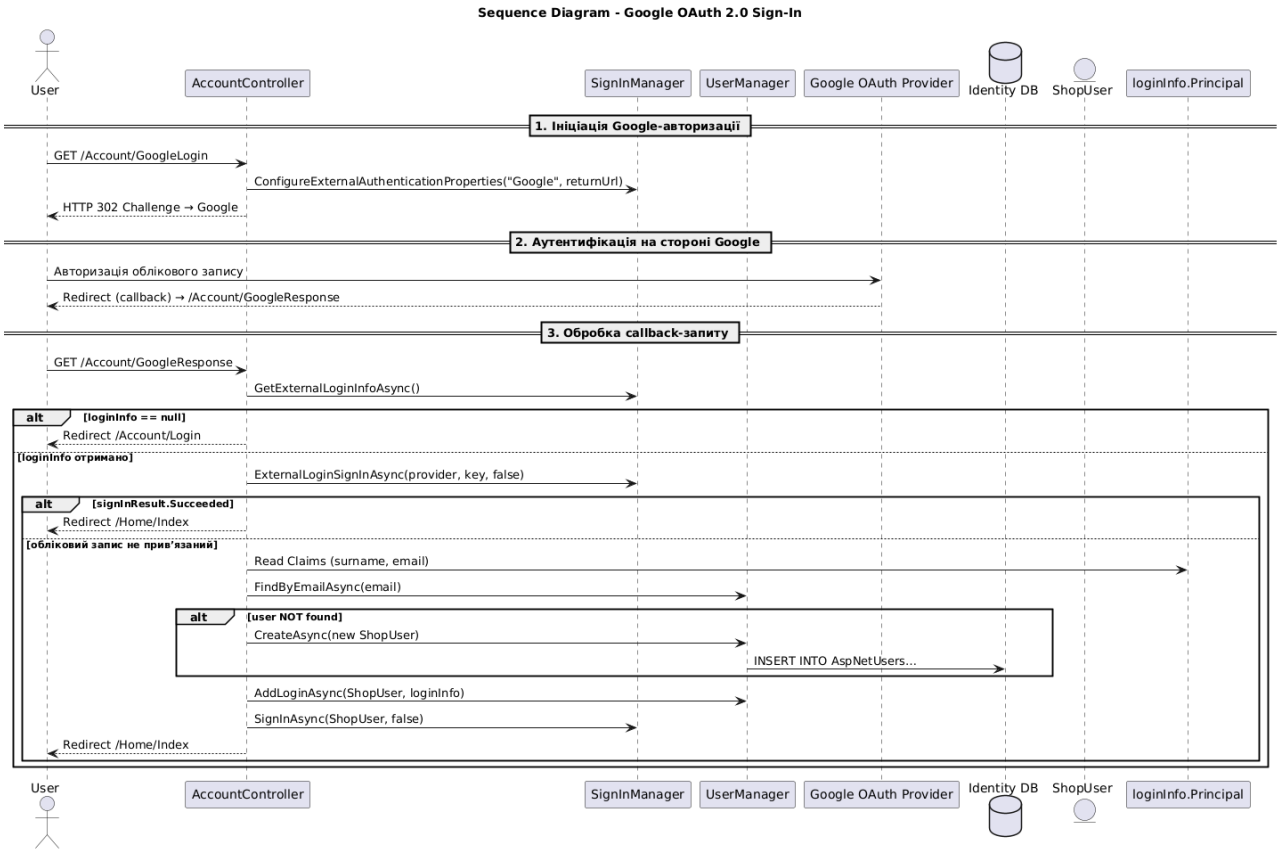


Рисунок 2.10 – UML- діаграма послідовності для аутентифікації за допомогою облікового запису Google

Реалізований механізм повністю відповідає сучасним вимогам до безпечної інтеграції з OAuth 2.0-провайдерами, дозволяє зменшити поріг входу для нових користувачів та підвищує зручність експлуатації системи. Крім того, забезпечується збереження локального облікового запису, що дозволяє у майбутньому доповнити функціонал прив’язкою кількох сторонніх акаунтів або переходом на класичний вхід через email та пароль.

Отже, реалізована система автентифікації та авторизації базується на перевірених та безпечних механізмах ASP.NET Core Identity. Вона забезпечує можливість реєстрації та входу користувачів, підтримку зовнішньої автентифікації, диференціацію прав доступу, а також масштабованість, необхідну для розширення системи у майбутньому.

2.4 Проєктування інтерфейсу користувача інтернет-магазину косметичних товарів

Інтерфейс користувача відіграє ключову роль у забезпеченні зручної та ефективної взаємодії відвідувачів із функціональністю веб-застосунку. Під час розробки інтерфейсу інтернет-магазину було дотримано принципів адаптивності, уніфікованості та модульності, що відповідає сучасним вимогам до веб-інтерфейсів.

Для побудови сторінок застосунку використовується синтаксис Razor, який є частиною MVC-платформи ASP.NET Core та забезпечує ефективне поєднання HTML-розмітки з динамічно згенерованим серверним контентом. Razor-сторінки дозволяють передавати моделі даних безпосередньо до представлення та відображати їх у контексті інтерфейсу з використанням зручного шаблонізованого синтаксису.

Сітка елементів інтерфейсу реалізована на основі фреймворку Bootstrap 5.3, який забезпечує адаптивність сторінок до різних розмірів екранів та пристроїв. Компоненти Bootstrap використовуються для реалізації меню,

форм, кнопок, карток товарів, повідомлень, а також інших елементів взаємодії з користувачем.

Центральне місце в структурі інтерфейсу займає майстер-сторінка _Layout.cshtml, яка виконує функцію спільного шаблону для всіх сторінок застосунку [14]. У межах цього шаблону розміщено головне навігаційне меню, футер (підвал сайту), а також секцію @RenderBody(), призначену для відображення вмісту дочірніх представлень. Такий підхід дозволяє забезпечити єдиний стиль оформлення та уніфіковану навігацію по всьому застосунку.

У випадку, коли користувач авторизований як адміністратор, інтерфейс доповнюється додатковим рядком меню, який містить посилання на панель адміністрування, інтерфейси керування товарами, категоріями, брендами, а також обробку замовлень. Умова відображення цієї панелі реалізується на основі перевірки ролей користувача, що забезпечує контроль доступу до адміністративного функціоналу.

У процесі розробки інтерфейсу користувача для інтернет-магазину особливу увагу було приділено реалізації механізмів, що забезпечують інтуїтивну та ефективну навігацію між категоріями товарів. З цією метою було створено ієрархічне меню, яке дозволяє користувачам швидко переміщуватись між основними категоріями та підкатегоріями.

Ієрархічне меню є важливим елементом інтерфейсу головної сторінки та шаблону макету сайту. Його метою є відображення структурованої системи категорій, що відповідає товарам, представленим на платформі. Це підвищує зручність пошуку та навігації, що є ключовим чинником у забезпеченні позитивного досвіду користувача.

Меню реалізовано з використанням компонента представлення CategoriesMenuViewComponent, що є частиною шаблону архітектури ASP.NET Core MVC. Використання компонента представлення замість традиційних часткових представлень (partial views) дає змогу ізолювати логіку вибірки та обробки даних, що робить код більш підтримуваним та тестованим.

У межах компонента реалізовано отримання даних з бази за допомогою Entity Framework Core з використанням механізму жадібного завантаження (eager loading) вкладених сутностей (рис. 2.11).

```
6   namespace Makeup_1.ViewComponents
7   {
        1 reference
8       public class CategoriesMenuViewComponent : ViewComponent
9       {
10          private readonly ShopContext context;
11
            0 references
12          public CategoriesMenuViewComponent(ShopContext context)
13          {
14              this.context = context;
15          }
16
            0 references
17          public async Task<IViewComponentResult> InvokeAsync()
18          {
19              IEnumerable<Category> categories = await context
20                  .Categories.Where(c=>c.ParentCategory == null)
21                  . Include(t => t.ChildCategories)!
22                  .ThenInclude(a=>a.ChildCategories)
23                  .ToListAsync();
24              return View(categories);
25          }
26      }
27  }
```

Рисунок 2.11 – Лістинг компоненту представлення для відображення ієрархічного меню категорій товарів

Цей підхід дозволяє за один запит отримати ієрархію категорій до третього рівня (батьківська – дочірня – «внучата»), що мінімізує кількість звернень до БД та покращує продуктивність.

Компонент представлення передає модель категорій у представлення Default.cshtml, де відбувається побудова HTML-структури меню. Для створення адаптивного, інтерактивного меню використано класи Bootstrap 5, зокрема класи dropdown, dropdown-menu та nav-item, що забезпечують випадаючі списки на різних рівнях вкладеності.

Рекурсивне відображення підкатегорій реалізовано через вкладені цикли foreach (рис. 2.12).

```
@model IEnumerable<Category>
<div class="col downBlockHeader">
    <ul class="bottomBtn bottomHeaderBtn mb-0 ps-0 me-3">
        @foreach (Category category in Model)
        {
            <li class="nav-item dropdown">

                <a class="nav-link dropdown-toggle" href="#" role="button" data-bs-toggle="dropdown"
                   aria-expanded="false">
                    @category.Name
                </a>
                @if (category.ChildCategories != null && category.ChildCategories.Count > 0)
                {
                    <ul class="dropdown-menu">
                        @foreach (Category child in category.ChildCategories)
                        {
                            <li>
                                <a class="dropdown-item"
                                   asp-action="Index"
                                   asp-controller="Home"
                                   asp-route-categoryId="@child.Id">
                                    @child.Name
                                    @if (child.ChildCategories != null
                                    && child.ChildCategories.Count > 0)
                                    {
                                        <img src="images/Vector 9.svg" alt="arrow up" />
                                    }
                                </a>
                                @if (child.ChildCategories != null
                                && child.ChildCategories.Count > 0)
                                {
                                    <ul>
                                        @foreach (Category grandChild in child.ChildCategories)
                                        {
                                            <li> <a asp-action="Index"
                                                    asp-controller="Home"
                                                    asp-route-categoryId="@grandChild.Id">
                                                    @grandChild.Name
                                                </a></li>
                                        }
                                    </ul>
                                }
                            </li>
                        }
                    </ul>
                }
            </li>
        }
    </ul>
</div>
```

Рисунок 2.12 – Лістинг коду представлення Razor компоненту представлення для відображення ієрархічного меню категорій товарів

Меню інтегрується в інтерфейс шляхом розміщення виклику компонента у головному макеті застосунку (_Layout.cshtml). Це забезпечує його автоматичне

включення на всіх сторінках, що використовують цей шаблон, та уніфікований доступ користувачів до навігації за категоріями.

2.5 Реалізація функціоналу кошика товарів

Кошик є одним із ключових елементів будь-якого інтернет-магазину, адже саме через нього реалізується взаємодія користувача із системою замовлення товарів. В рамках даного проекту реалізовано функціональний, динамічний та зручний у використанні інтерфейс кошика, який дозволяє здійснювати перегляд, зміну кількості товарів, видалення позицій та підтвердження замовлення.

Інтерфейс кошика відповідає вимогам зручності та мінімалізму. Виділення ключових елементів (вартість, кількість, загальна сума) виконано за допомогою стилів, визначених у секції @section PageStyles (рис. 2.13).

```
@section PageStyles {
    <style>
        .product-price {
            font-size: 30px;
            font-weight: 500;
            text-align: end;
        }

        .product-name {
            font-size: 16px;
            font-weight: 500;
        }

        .product-count-btn {
            cursor: pointer;
        }

        #totalSum {
            font-size: 24px;
            font-weight: 500;
        }

        #totalSumText {
            font-size: 36px;
            text-align: end;
        }

        #cartButtons a {
            text-decoration: none;
            border-width: 1px;
            border-radius: 5px;
            border-style: solid;
            border-color: rgba(141, 152, 144, 1);
            padding: 10px 20px;
            color: rgba(141, 152, 144, 1);
            font-size: 16px;
            font-weight: 500;
        }

        #cartButtons button {
            border-radius: 5px;
            border-style: solid;
            border-color: rgba(141, 152, 144, 1);
            padding: 10px 20px;
            background-color: rgba(141, 152, 144, 1);
            color: white;
            font-size: 16px;
            font-weight: 500;
        }

        .removeFromCartBtn {
            border: none;
        }
    </style>
}
```

Рисунок 2.13 – Лістинг стилістичних правил CSS для елементів сторінки відображення кошику

Сторінка адаптована для відображення на різних пристроях, а також не перевантажена лишніми візуальними ефектами, що відповідає сучасним тенденціям у вебдизайні.

Представлення кошика побудовано на базі представлень MVC з використанням технології Razor (рис. 2.14), що дозволяє поєднати серверну логіку з HTML-розміткою в рамках єдиного шаблону. До представлення передається ViewModel типу CartIndexViewModel, яка містить колекцію об’єктів типу CartItem (товари у кошику) та додаткову інформацію (наприклад, URL повернення).

```
@using Makeup_1.Models.ViewModels.CartViewModels
@model CartIndexViewModel
@{
    int i = 0;
}
@section PageStyles {
    <style>…</style>
}
<div class="row border-2 border-bottom py-3" style="align-items: center;">
    <div class="col"></div>
    <div class="col-1">кількість</div>
    <div class="col-2 text-end">вартість</div>
    <div class="col-1"></div>
</div>
@foreach (CartItem item in Model.Cart.CartItems)
{
    <div class="row border-2 border-bottom py-3" style="align-items: center;">
        <div class="col-1">
            @if (@item.Product.Images!.Count >= 1)
            {
                <img class="w-100" src="@(Convert
                        .ToBase64String(item.Product.Images?.FirstOrDefault()!.File!))"
                     alt="@item.Product.Name" style="border-radius: 10px;">
            }
        </div>
        <div class="col-1"></div>
        <div class="col">@item.Product.Name</div>
        <div class="col-1 d-flex border" style="text-align: center;">
            <span class="flex-fill p-2 product-count-btn" onclick="decCount(event, @item.Product.Id)">-</span>
            <span class="flex-fill p-2">@item.Quantity</span>
            <span class="flex-fill p-2 product-count-btn" onclick="incCount(event, @item.Product.Id)">+</span>
        </div>
        <div class="col-2 product-price">@(item.Product.Price * item.Quantity) ₴</div>
        <div class="col-1">
            <form asp-action="RemoveFromCart" method="post" asp-controller="cart">
                <input type="hidden" name="id" value="@item.Product.Id" />
                <input type="hidden" name="returnUrl" value="@Model.ReturnUrl" />
                <button type="submit" style="color: black;" class="removeFromCartBtn">
                    <svg class="bi bi-x-lg">…</svg>
                </button>
            </form>
        </div>
    </div>
}
<div id="totalSum" class="row align-items-center">
    <div class="col"></div>
    <div class="col-2">загальна сума:</div>
    <div id="totalSumText" class="col-2">@Model.Cart.GetTotalSum() ₴</div>
</div>
<div class="d-flex justify-content-between mt-5" id="cartButtons">
    <a href="@Model.ReturnUrl">Повернутися до покупок</a>
    <form asp-action="ConfirmOrder" method="post">
        <input type="hidden" name="returnUrl" value="@Model.ReturnUrl" />
        <button type="submit">Оформити замовлення</button>
    </form>
</div>
@section Scripts {
```

Рисунок 2.14 – Лістинг представлення для відображення кошику

Інтерфейс побудований з використанням Bootstrap 5.3, що забезпечує адаптивність та сучасне візуальне оформлення. Для відображення товарів використано сіткову структуру (div.row), де кожний рядок відповідає окремому товару в кошику.

Для кожного товару, що містить зображення, воно кодується у формат Base64 і відображається через тег <img>, що виключає потребу в окремих статичних файлах:

```
70      <div class="col-1">
71          @if (@item.Product.Images!.Count >= 1)
72          {
73              <img class="w-100" src="@(Convert
74                      .ToBase64String(item.Product.Images?.FirstOrDefault()!.File!))"
75                   alt="@item.Product.Name" style="border-radius: 10px;">
76          }
77      </div>
```

Зміна кількості реалізована за допомогою JavaScript-функцій incCount та decCount, які надсилають асинхронні POST-запити на відповідні маршрути /cart/incCount/{id}та /cart/decCount/{id}. Діаграму послідовності UML при збільшенні кількості товарів у кошику зображено на рис. 2.15, а код клієнтських асинхронних сценаріїв оновлення кількості мовою JavaScript – на рис. 2.16.

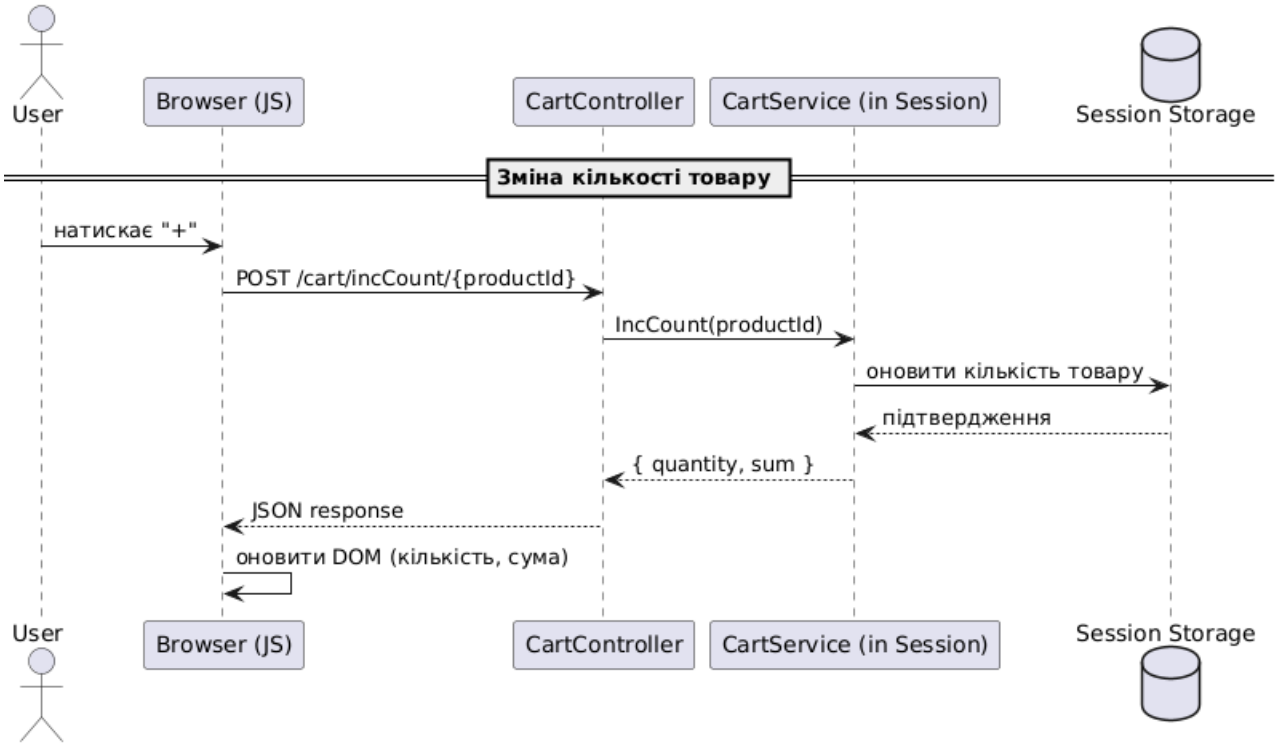


Рисунок 2.15 – UML- діаграма послідовності для операції збільшення кількості певного товару у кошику

```
113     async function incCount(e, productId) {
114         let resp = await fetch(`/cart/incCount/${productId}`,
115         { method: "post" });
116         if (resp.ok === true) {
117             let data = await resp.json();
118             let countElem = e.target.previousElementSibling;
119             // let count = parseInt(countElem.innerText);
120             countElem.innerText = `${data.quantity}`;
121             // let p = document.createElement("a");
122             let parentDiv = countElem.parentElement;
123             let newSumElem = parentDiv.nextElementSibling;
124             newSumElem.innerText = `${data.sum} ₴`;
125             await getTotalSum();
126         }
127     }
128     async function decCount(e, productId) {
129         let countElem = e.target.nextElementSibling;
130         let count = parseInt(countElem.innerText);
131         console.log(count);
132         if (count <= 1)
133             return;
134         let resp = await fetch(`/cart/decCount/${productId}`, { method: "post" });
135         if (resp.ok === true) {
136             let data = await resp.json();
137             countElem.innerText = `${data.quantity}`;
138             // let p = document.createElement("a");
139             let parentDiv = countElem.parentElement;
140             let newSumElem = parentDiv.nextElementSibling;
141             newSumElem.innerText = `${data.sum} ₴`;
142             await getTotalSum();
143         }
144     }
```

Рисунок 2.16 – Лістинг функцій збільшення та зменшення кількості товарів у кошику

Процес збільшення кількості складається з таких кроків:

1. Користувач натискає кнопку +для збільшення кількості товару.
2. JavaScript-функція incCount() надсилає POST-запит до маршруту /cart/incCount/{productId}.
3. Контролер CartController обробляє запит, змінює кількість товару у кошику (у сесії).
4. Сервер повертає нову кількість та суму по товару у форматі JSON.
5. JavaScript оновлює DOM (кількість товару та суму).

Для обробки запиту на збільшення кількості товарів у контролері CartController реалізовано метод IncCount(int id, Cart cart), який приймає ідентифікатор товару та об’єкт корзини (з використанням прив’язки і кастомного прив’язчика моделі CartModelBinder):

```
57          [HttpPost]
            0 references
58          public async Task<IActionResult> IncCount(int id, Cart cart)
59          {
60              Product? product = await context.Products.FindAsync(id);
61              if (product != null)
62              {
63                  await context.Entry(product).Collection(t => t.Images).LoadAsync();
64                  cart.AddItem(product, 1);
65                  UpdateCart(cart);
66                  int quantity = cart.CartItems.First(t => t.Product.Id == id).Quantity;
67                  return Ok(new
68                  {
69                      quantity,
70                      Sum = cart.GetSumByProduct(product)
71                  });
72              }
73              return NotFound();
74          }
```

Для зменшення кількості товарів реалізовано аналогічний метод DecCount(). Він має одну значну відмінність – при зменшенні кількості товарів до 0 товар повністю видаляється з кошику на стороні екземпляру класу Cart, який інкапсулює логіку сховища екземплярів класу CartItem.

Для кожної позиції передбачено кнопку видалення, реалізовану через HTML-форму з методом POST:

```
86          <div class="col-1">
87              <form asp-action="RemoveFromCart" method="post" asp-controller="cart">
88                  <input type="hidden" name="id" value="@item.Product.Id" />
89                  <input type="hidden" name="returnUrl" value="@Model.ReturnUrl" />
90                  <button type="submit" style="color: black;" class="removeFromCartBtn">
91                      <svg class="bi bi-x-lg">…</svg>
94                  </button>
95              </form>
96          </div>
```

На стороні сервера дана форма обробляється POST-методом RemoveFromCart() контролера CartController.

```
45          [HttpPost]
            0 references
46          public async Task<IActionResult> RemoveFromCart(int id, Cart cart, string returnUrl)
47          {
48              Product? product = await context.Products.FindAsync(id);
49              if (product != null)
50              {
51                  cart.RemoveItem(product);
52                  UpdateCart(cart);
53              }
54              return RedirectToAction("Index", new { returnUrl });
55          }
```

Для динамічного оновлення загальної суми товарів у кошику передбачено асинхронну (AJAX) функцію getTotalSum(), що звертається за адресою /cart/getTotalSum:

```
145     async function getTotalSum() {
146         let resp = await fetch(`/cart/getTotalSum`, { method: "post" });
147         if (resp.ok === true) {
148             let data = await resp.json();
149             let totalSumElem = document.querySelector("#totalSumText");
150             // let totalSumElem = document.querySelector("#totalSum>span");
151             totalSumElem.innerText = `${data.totalSum} ₴`;
152         }
153     }
```

На стороні контролера даний асинхронний виклик обробляє метод GetTotalSum(), який повертає загальну суму всіх товарів у кошику у форматі JSON:

```
95          [HttpPost]
            0 references
96          public IActionResult GetTotalSum(Cart cart)
97          {
98              double totalSum = cart.GetTotalSum();
99              return Ok(new {totalSum });
100         }
```

Після отримання результату запиту у форматі JSON на сторінці представлення метод getTotalSum() оновлює DOM і змінює значення загальної суми.

Ключовою особливістю реалізації є використання сесійного зберігання для фіксації стану корзини між HTTP-запитами. Метод UpdateCart() серіалізує колекцію CartItems та зберігає її у сесію:

```
126     private void UpdateCart(Cart cart)
127     {
128         HttpContext.Session.Set<IEnumerable<CartItem>>("cart", cart.CartItems);
129     }
```

Цей підхід дозволяє зберігати дані кошика незалежно від входу користувача в систему, а також дозволяє працювати з кошиком до моменту реєстрації чи авторизації.

Метод ConfirmOrder() реалізує логіку формування замовлення з позицій кошика, збереження його в базі даних та надсилання повідомлення користувача на електронну пошту із підтвердженням (рис. 2.17).

```
134     public async Task<IActionResult> ConfirmOrder( string returnUrl, Cart cart)
135     {
136
137         User user = await userManager.FindByNameAsync(User.Identity!.Name);
138         string emailTo = user.Email;
139
140         int i = 0;
141         Order order = new Order();
142         order.UserId = user.Id;
143         order.Date = DateTime.Now;
144         foreach (CartItem item in cart.CartItems)
145         {
146             OrderItem orderItem = new OrderItem();
147             orderItem.ProductId = item.Product.Id;
148             orderItem.quantity = item.Quantity;
149             order.Items.Add(orderItem);
150         }
151         context.Orders.Add(order);
152         await context.SaveChangesAsync();
153         StringBuilder builder = new StringBuilder();
154         builder.Append($"<h3>Ваше замовлення №{order.Id}: </h3><ul>");
155         builder.Append("<ul>");
156         foreach (CartItem item in cart.CartItems)
157         {
158             builder.Append($"<li>{++i}. {item.Product.Name} - {item.Quantity} шт:" +
159                 $" {item.Product.Price} грн. </li>");
160         }
161             builder.Append("</ul>");
162         builder.Append($"<h4>Итого:{cart.GetTotalSum()} грн. </h4>");
163         await emailService.SendAsync(shopEmail, emailTo,
164             $"Ваше замовлення №{order.Id} підтверждено!",
165             builder.ToString());
166         cart.Clear();
167         return Redirect(returnUrl);
168     }
```

Рисунок 2.17 – Лістинг методу підтвердження замовлення

На рис. 2.18 зображено діаграму послідовності, що реалізований у методі ConfirmOrder контролера CartController. процесу обробки замовлення в інтернет-магазині косметичних засобів.

Процес ініціюється користувачем, який надсилає запит на підтвердження замовлення. Після цього система здійснює послідовність взаємодій між основними компонентами:

1. Отримання даних користувача. Контролер звертається до менеджера користувачів (UserManager) для отримання об’єкта поточного аутентифікованого користувача.

2. Формування замовлення. На основі вмісту кошика (Cart) створюється новий об’єкт замовлення (Order), до якого додаються відповідні позиції замовлення (OrderItem) для кожного товару.

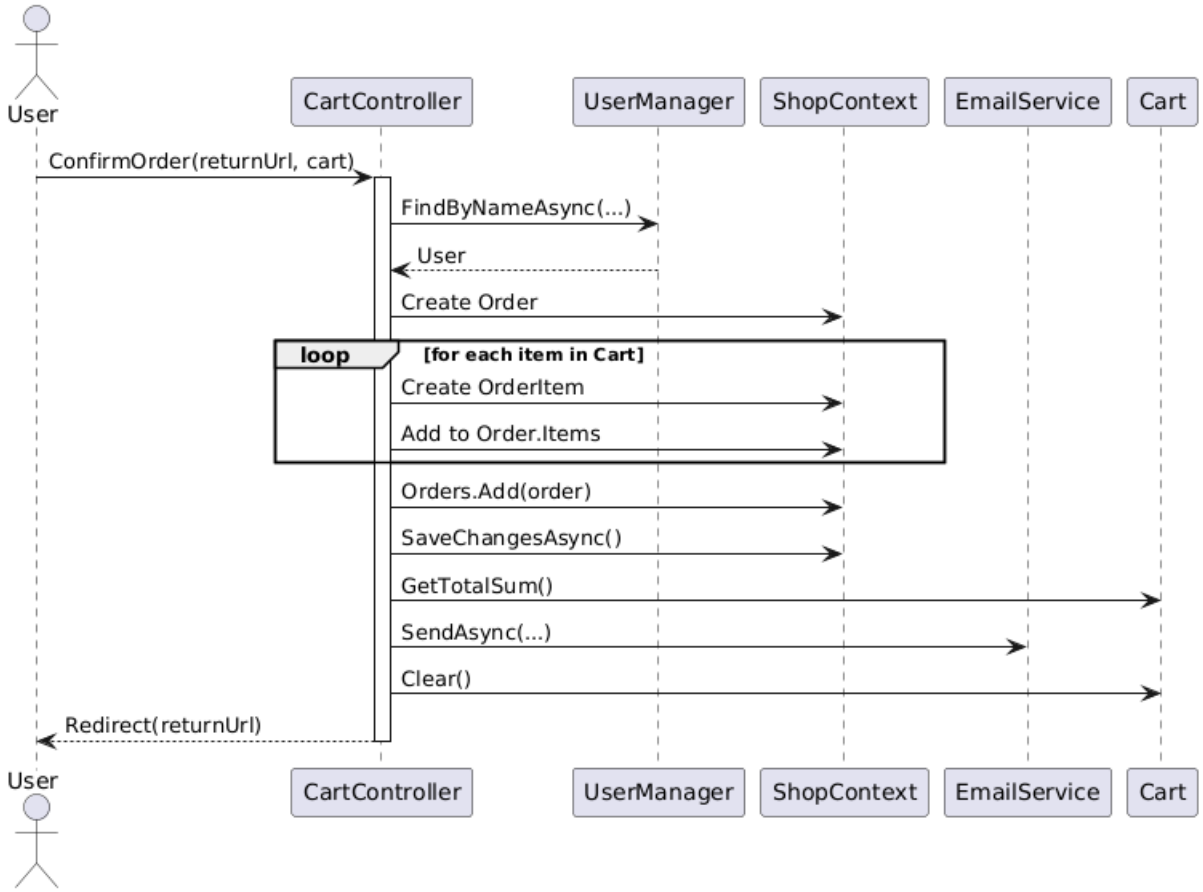


Рисунок 2.18 – UML- діаграма послідовності процесу обробки замовлення в інтернет-магазині косметичних засобів

3. Збереження в базу даних. Готове замовлення додається до контексту бази даних (DbContext) і зберігається за допомогою асинхронного методу SaveChangesAsync().

4. Обчислення загальної суми замовлення. Контролер звертається до кошика для отримання фінальної суми.

5. Надсилання підтвердження на електронну пошту. За допомогою сервісу електронної пошти (EmailService) здійснюється надсилання повідомлення про успішне оформлення замовлення.

6. Очищення кошика. Після завершення процесу, вміст кошика очищується.

7. Переадресація користувача. Користувача перенаправляють на вихідну сторінку.

Дана діаграма ілюструє взаємодію між контролером, сервісами додатку, кошиком, контекстом бази даних та об’єктом користувача, що дозволяє наочно представити логіку обробки замовлення і підтверджує дотримання принципів чіткої структурованості архітектури застосунку.

Таким чином, у роботі було реалізовано логіку формування кошику замовлень та їх оформлення. Розроблений контролер CartController з відповідними представленнями демонструє класичну реалізацію бізнес-логіки корзини в інтернет-магазині з використанням архітектури ASP.NET Core MVC. Він ефективно інтегрує роботу з моделлю, представленням, сесіями та зовнішніми сервісами (зокрема, електронною поштою), забезпечуючи цілісний та функціональний механізм обробки замовлень

2.6 Практична апробація та опис методики використання розробленого інтернет-магазину

Розроблений інтернет-магазин косметичних засобів забезпечує повний цикл управління даними та підтримку зручної взаємодії з користувачем. Нижче наведено приклади використання основних функціональних можливостей як адміністратора, так кінцевого користувача.

1. Відображення ієрархічного меню категорій

Однією з важливих складових інтерфейсу є навігаційне меню (рис. 2.19), реалізоване за допомогою компонента представлення (ViewComponent). Меню автоматично формує ієрархію категорій на основі відповідних зв'язків у базі даних (ParentCategoryId/ChildCategories) та підтримує вкладені підменю до третього рівня. Цей компонент динамічно оновлюється при додаванні або зміні категорій і виводиться у шапці (_Layout) кожної сторінки для полегшення навігації.

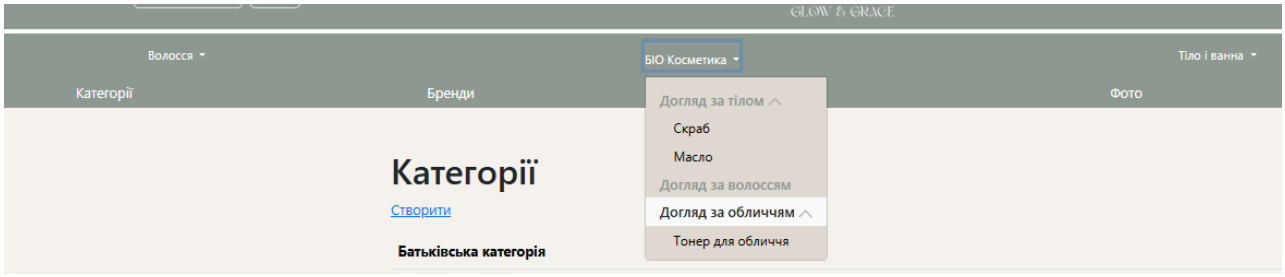


Рисунок 2.19 – Ієрархічне меню вибору категорій товарів

2. Додавання брендів, категорій та товарів

Інтерфейс адміністратора передбачає можливість додавання нових брендів косметичних засобів через окрему форму введення, що включає назву бренду та країну походження (рис. 2.20).

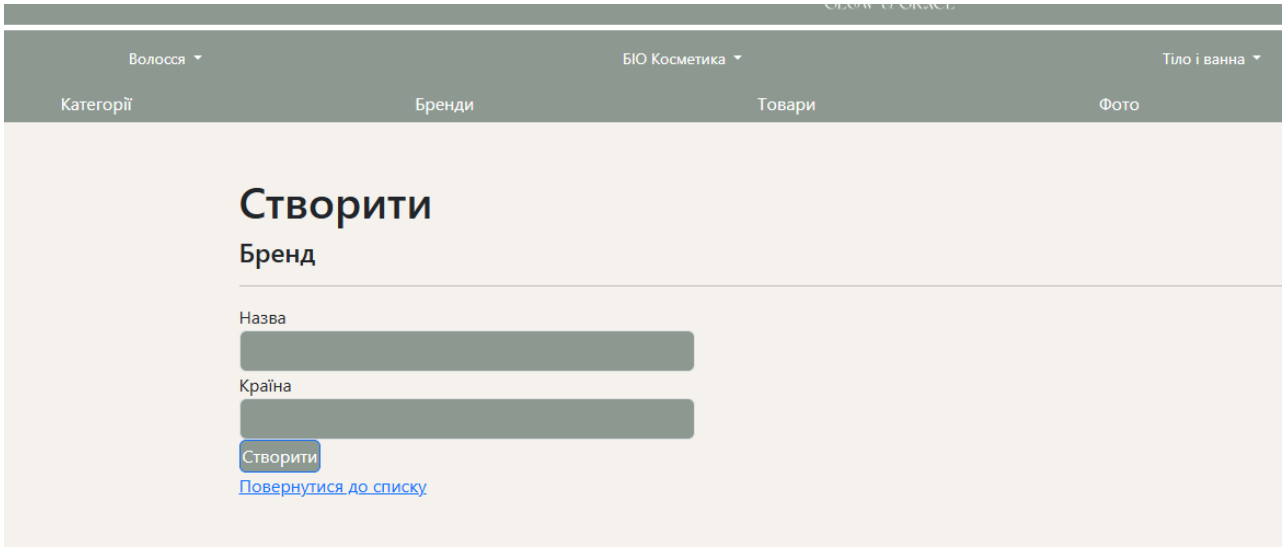


Рисунок 2.20 – Сторінка додавання бренду інтернет-магазину косметичних засобів

На наступному кроці користувач, який є членом ролі «адміністратор», має змогу створити ієрархічну структуру категорій товарів наприклад, «Уход за шкірою» → «Креми для обличчя» → «Нічні креми». Це реалізовано завдяки рекурсивній моделі категорій з полем ParentCategoryId.

На рис. 2.21 наведено список доданих адміністратором категорій товарів.

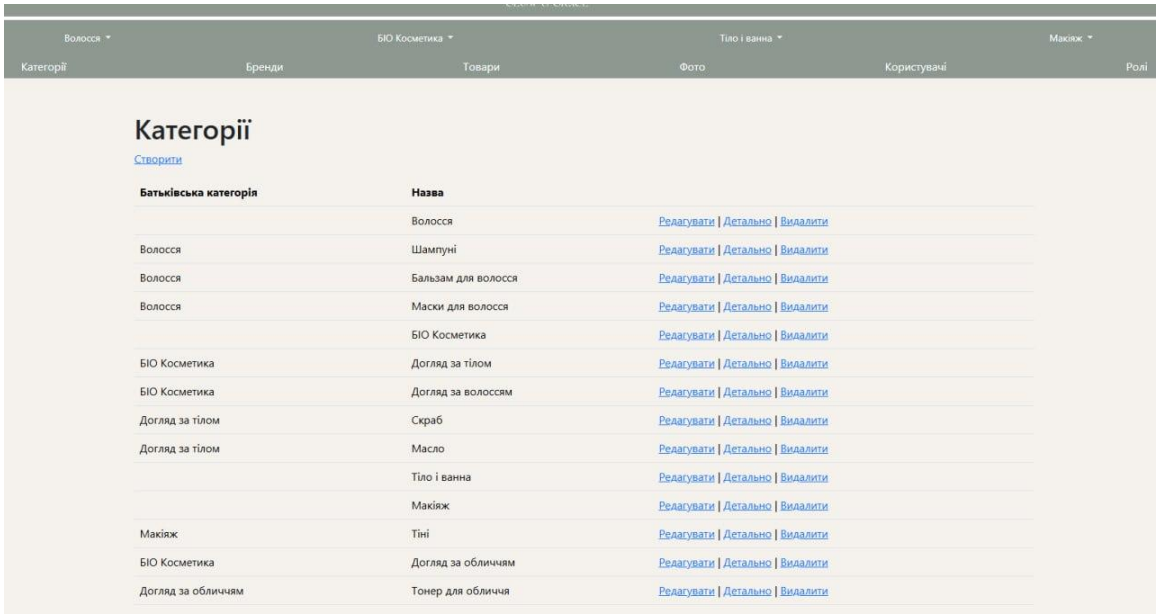


Рисунок 2.21 – Сторінка відображення списку доданих категорій товарів

Після налаштування довідників адміністратор може додати новий товар (рис. 2.22), вказавши його назву, опис, бренд, ціну, рейтинг, спосіб застосування.

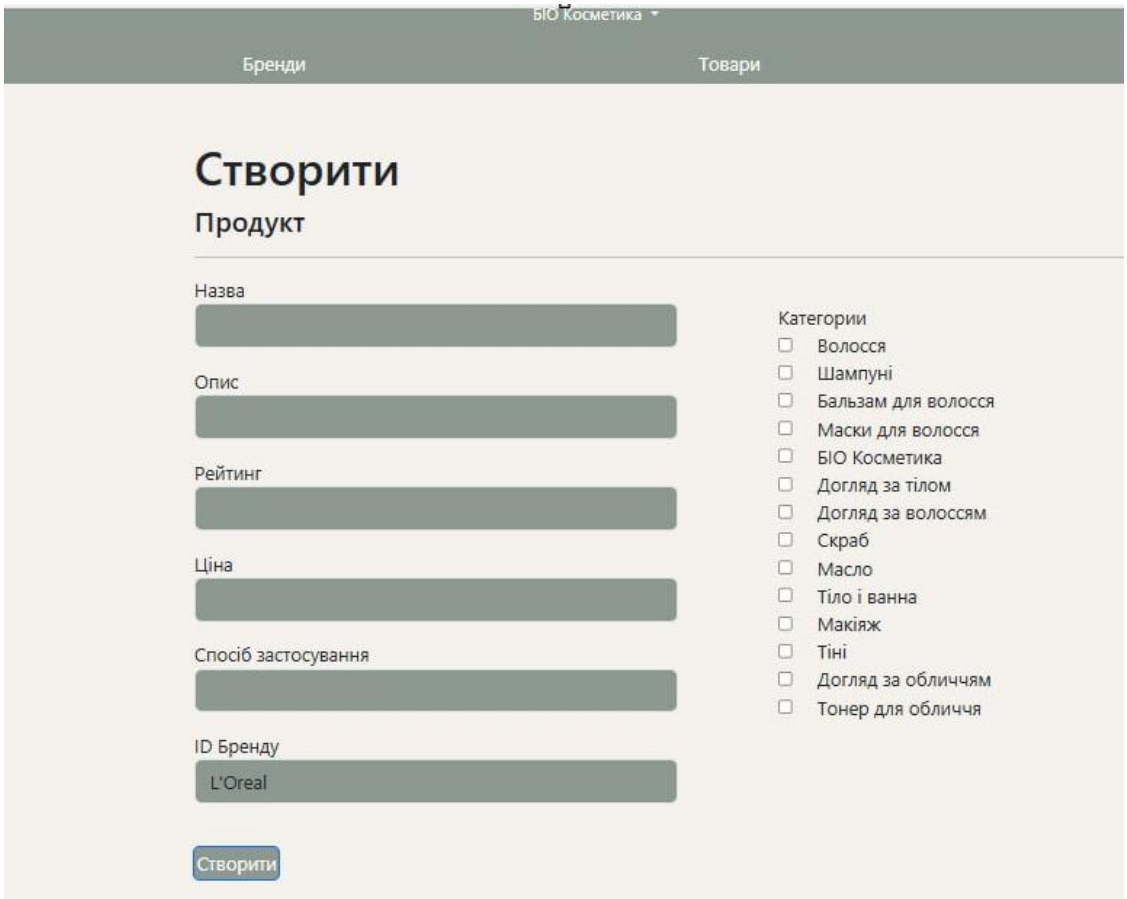


Рисунок 2.22 – Сторінка додавання нового товару

У процесі створення також можливо прив’язати товар до однієї чи кількох категорій. Збережені дані автоматично відображаються на клієнтській частині інтерфейсу.

3. Відображення головної сторінки та сторінки окремого товару

На головній сторінці магазину відображається меню з доступними категоріями продукції та список популярних товарів, які динамічно формуються з бази даних. Користувач може обрати категорію, щоб переглянути відповідні товари, або скористатися пошуком (рис. 2.23).

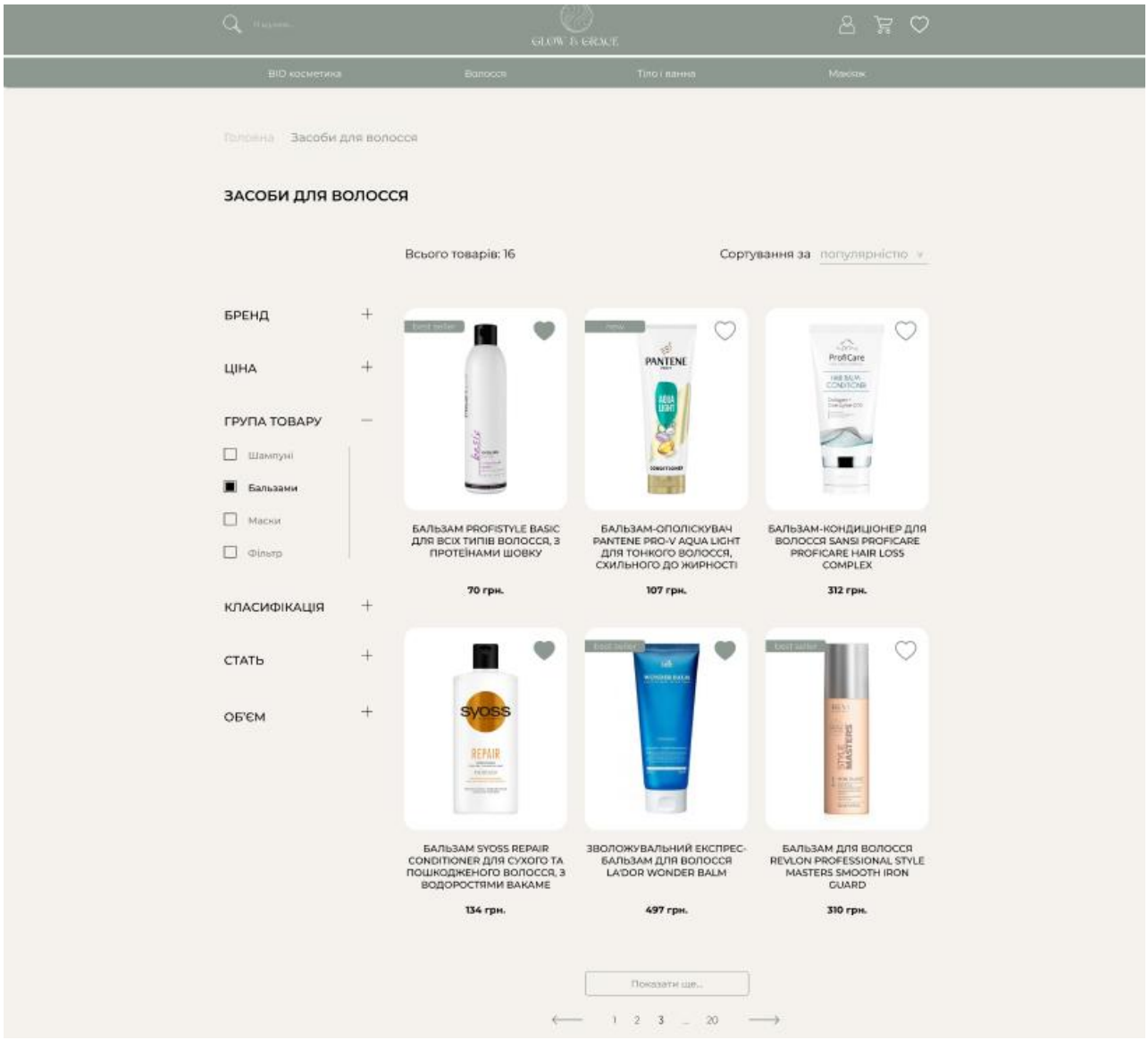


Рисунок 2.23 – Головна сторінка інтернет-магазину косметичних засобів при застосуванні фільтрів по категорії товарів

При виборі конкретного товару користувач перенаправляється на детальну сторінку продукту, де представлена інформація про товар (назва, опис, ціна, рейтинг, спосіб застосування), галерея зображень, а також коментарі інших користувачів (рис. 2.24). Відповідно до рівня авторизації, доступна можливість залишити свій коментар про товар.

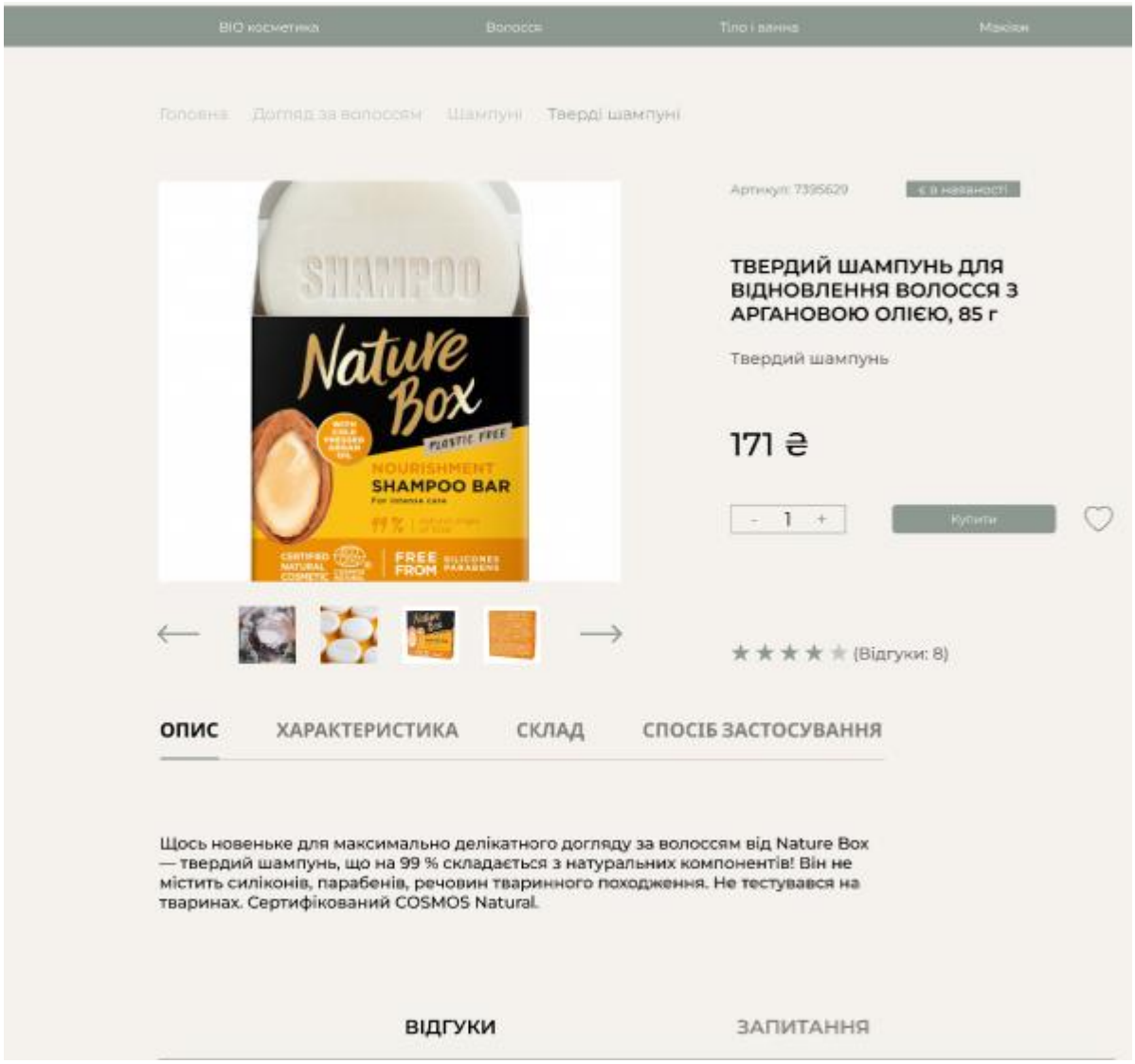


Рисунок 2.24 – Сторінка перегляду окремого товару

3. Робота з кошиком

Користувач може додати обраний товар у кошик, після чого можна перейти до кошика для перегляду повного списку вибраної продукції.

На сторінці «Кошик» відображається перелік доданих товарів із кількістю та ціною. Існує можливість змінювати кількість одиниць кожного

товару (кнопки +та -); видаляти товар із кошика. Змінення кількості товару відбувається без перезавантаження сторінки – завдяки реалізації відповідної JavaScript-логіки з використанням AJAX-запитів. Для кожної позиції у кошику обчислюється сума, а в нижній частині відображається загальна вартість замовлення (рис. 2.25).

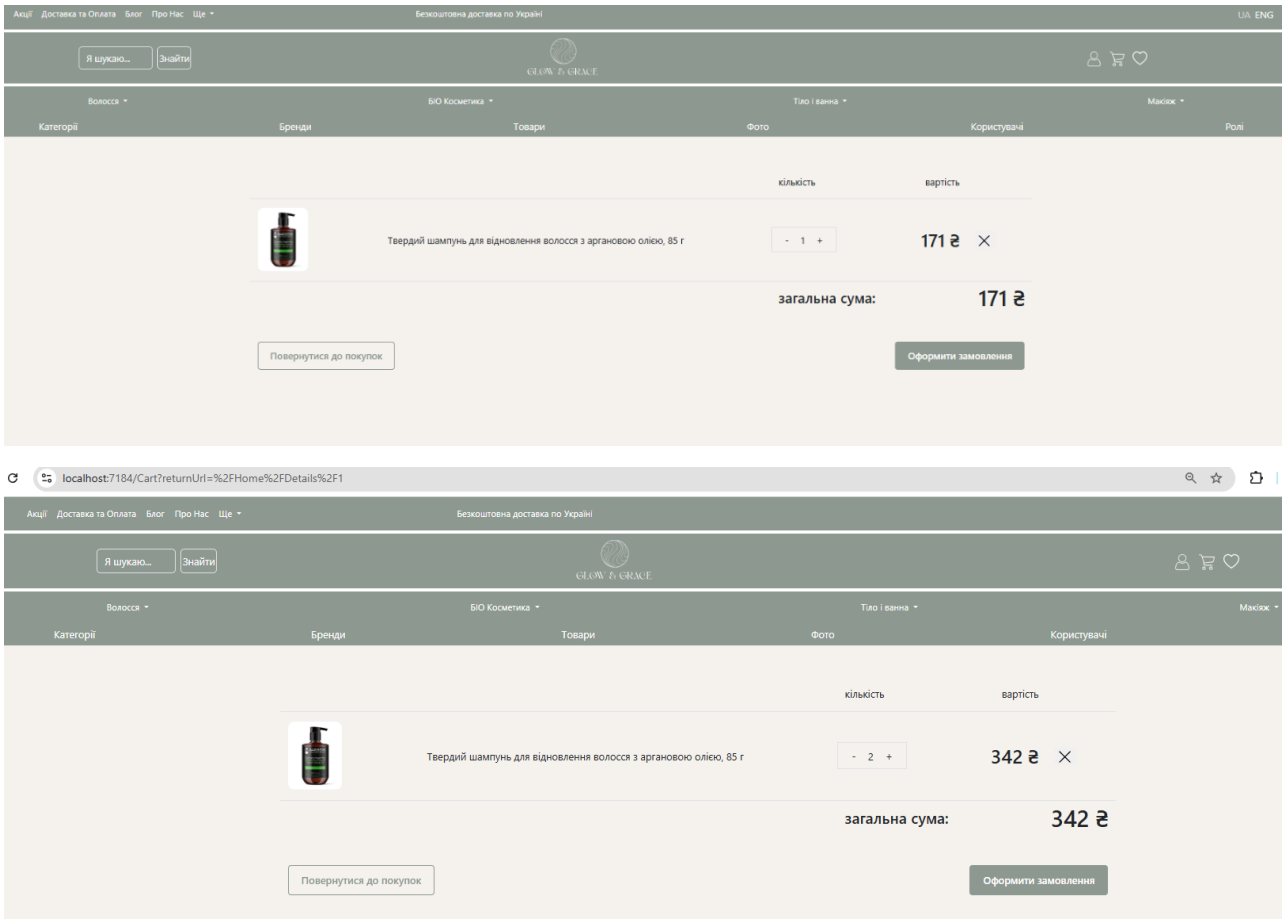


Рисунок 2.25 – Ілюстрація динамічного змінення кількості товарів на сторінці кошику

Щоб підтвердити замовлення, необхідно натиснути кнопку «Оформити замовлення». На Вашу електронну пошту буде надіслано повідомлення з деталями покупки.

Таким чином, практична апробація розробленого програмного забезпечення інтернет-магазину косметичних засобів продемонструвала, що увесь запланований до реалізації функціонал виконує всі передбачені завдання.

Розроблений інтернет-магазин має зручний та інтуїтивно зрозумілий інтерфейс та може бути рекомендований до впровадження та практичного використання.

Висновки до розділу:

У другому розділі здійснено проєктування архітектури веб-застосунку інтернет-магазину косметичних засобів на базі технології ASP.NET Core MVC. Було обґрунтовано вибір стеку технологій та моделі розробки на основі підходу Code First, що забезпечує гнучкість у побудові структури бази даних. Описано доменні класи предметної області, реалізовано їх зв’язки та взаємодії у вигляді UML-діаграм. Здійснено інтеграцію з бібліотекою Microsoft Identity, яка забезпечує аутентифікацію, авторизацію та підтримку зовнішніх провайдерів (Google, Facebook).

Окрему увагу приділено реалізації ключових компонентів: інтерфейсу користувача, адаптивного ієрархічного меню, кошика покупця та обробки замовлень. Застосовано принципи MVC-архітектури, створено компоненти представлення, контролери, сервіси та модулі розширень. Описано логіку обробки запитів, зберігання даних у сесії та механізм надсилання електронних листів з підтвердженням замовлення. У результаті сформовано функціонально повний та архітектурно обґрунтований каркас застосунку, що є основою для реалізації прикладної частини системи.

ВИСНОВКИ

У межах виконання кваліфікаційної роботи було реалізовано програмне застосування – інтернет-магазин косметичних засобів, створених з використанням технології ASP.NET Core MVC. Розробка здійснювалася відповідно до сучасних вимог у сфері веб-програмування, із дотриманням принципів модульності, масштабованості та розширюваності програмної архітектури.

На основі аналітичного огляду провідних світових та українських платформ електронної комерції (зокрема Sephora , Ulta , Makeup , Eva , Notino) було визначено набір функціональних характеристик, що є стандартом де-факто у відповідному сегменті ринку. Результати аналізу дозволили сформулювати обґрунтовані функціональні вимоги до застосування, включаючи реалізацію системи автентифікації та авторизації, кошика, адміністративної панелі, ієрархічного меню категорій, управління товарами, брендами та обробкою замовлень.

Здійснено проєктування бази даних на основі підходу Code First , що дало змогу гнучко визначити доменну модель із відповідними зв’язками. В межах реалізації використано такі ключові компоненти: контролери , моделі подань , сервіси, View-компоненти, а також механізми роботи з сесією та надсилання електронної пошти .

Особливу увагу приділено проектуванню користувацького інтерфейсу з використанням Razor -синтаксису та Bootstrap 5.3, що забезпечило адаптивність та зручність навігації.

Передбачено авторизацію та автентифікацію з підтримкою сторонніх провайдерів (Google, Facebook), адміністрування товарів, брендів і категорій, а також механізм замовлення з надсиланням електронних повідомлень.

Отриманий результат підтверджує практичну доцільність застосування обраного стеку технологій і обґрунтованість архітектурних рішень. Створений програмний продукт може бути використаний як основа для подальшого

розвитку повноцінної системи електронної комерції з розширеним функціоналом.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1 Overview of ASP.NET Core. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-9.0 (дата звернення: 26.04.2025).

2 IRKI Y. E. Is ASP.NET Core Still A Top Backend Framework In 2025?. Medium. URL: https://medium.com/@yusufeminirki/is-asp-net-core-still-a-top-backend-framework-in-2025-7b96b839dc83 (дата звернення: 26.04.2025).

3 ASP.NET Core Vs. ASP.NET MVC5: Which Is Better And Why? Zenesys. URL: https://www.zenesys.com/asp-net-core-vs-asp-net-mvc-5 (дата звернення: 26.04.2025).

4 Manpreetkaur. Mastering Entity Framework Core: A Deep Dive. Medium. URL: https://medium.com/@manpreetkaur6311062/mastering-entity-framework-core-a-deep-dive-d5b1f8c98817 (дата звернення: 26.04.2025).

5 Best SQL databases: 2024 Edition. Best SQL databases: 2024 Edition. URL: https://risingwave.com/blog/best-sql-databases-2024-edition (дата звернення: 29.04.2025).

6 Relational Database: Structure, Benefits, and Key Use Cases. Acceldata | Agentic Data Management Platform | Enterprise Data Observability. URL: https://www.acceldata.io/blog/what-is-a-relational-database-architecture-features-and-real-world-applications (дата звернення: 29.04.2025).

7 Top 10 Front-End Frameworks for Responsive Design 2024. daily.dev | Where developers suffer together. URL: https://daily.dev/blog/top-10-front-end-frameworks-for-responsive-design-2024 (дата звернення: 29.04.2025).

8 Abdulsamad. Responsive Web Design: Best Practices for 2024. Medium. URL: https://medium.com/@abdulsamad18090/responsive-web-design-best-practices-for-2024-492a42635a4c (дата звернення: 29.04.2025).

9 Popat M. Effortless Web App Deployment with Azure App Service: Your Complete Guide to Scalable Cloud. Medium. URL:

https://mihirpopat.medium.com/effortless-web-app-deployment-with-azure-app-service-your-complete-guide-to-scalable-cloud-e9f1d30961ac (дата звернення: 29.04.2025).

10 Palanikalyan. Transforming E-commerce: A Complete Azure Migration Journey. Medium. URL: https://medium.com/@palanikalyan27/transforming-e-commerce-a-complete-azure-migration-journey-45051a0f323c (дата звернення: 29.04.2025).

11 Scalable Web Application for Business | Benefits. Digital Transformation & Managed Services | Star Knowledge. URL: https://star-knowledge.com/blog/build-scalable-web-application-azure (дата звернення: 29.04.2025).

12 Lock A. ASP. NET Core in Action, Second Edition. Manning Publications Co. LLC, 2021. 832 с.

13 Murach's ASP. NET Core MVC. Murach & Associates, Incorporated, Mike, 2020. 780 p.

14 Pro ASP. NET Core 7. Manning Publications Co. LLC, 2023.

15 Spasojevic M. Ultimate ASP.NET Core Web API - From Zero To Six-Figure Backend Developer. Syncfusion Inc., 2023.

16 Wenz C. ASP. NET Core Security. Manning Publications Co. LLC, 2022.

17 Booth J. Database Design Succinctly. Morrisville, NC : Syncfusion Inc, 2022. URL: https://www.syncfusion.com/succinctly-free-ebooks/downloads/database-design-succinctly.pdf (дата звернення: 01.06.2025).

18 Introduction to Identity on ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity (дата звернення 10.05.2023).

19 Моркун Н. В., Завсегдашня І. В. Методичні вказівки до виконання кваліфікаційної роботи бакалавру для студентів спеціальності 122 “Комп’ютерні науки”. Кривий Ріг : Видавничий центр КНУ, 2021. 50 с.

20 ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлення. Київ, ДП «УкрННЦ», 2015. 26с. (Інформація та документація).

21 ДСТУ 8302:2015. Бібліографічне посилання. Загальні вимоги та правила складання Київ, ДП «УкрННЦ», 2016. 16 с. (Інформація та документація).

22 ДСТУ 3582:2013. Бібліографічний опис. Скорочення слів і словосполучень в українській мові. Загальні вимоги та правила. Київ, ДП «УкрННЦ», 2013. 23 с. (Інформація та документація).