

**Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення**

**КВАЛІФІКАЦІЙНА РОБОТА
ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»**

**РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ПРОДАЖУ КАВИ ТА ЧАЮ ІЗ
ВИКОРИСТАННЯМ NEXT.JS ТА NESTJS**

**DEVELOPMENT OF A WEB APPLICATION FOR SELLING COFFEE AND
TEA USING NEXT.JS AND NESTJS**

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконав: здобувач вищої освіти
групи ІПЗ-43
Васильєв Андрій Віталійович

Керівник:
Ліщина Наталія Миколаївна

Кваліфікаційну роботу
допущено до захисту
«__» _____ 20__ р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних та інформаційних технологій

Кафедра інженерії програмного забезпечення

Ступінь вищої освіти бакалавр

Галузь знань: 12 «Інформаційні технології»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри

«__» _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Васильєву Андрію Віталійовичу

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка вебзастосунку для продажу кави та чаю із використанням Next.js та NestJS»

Керівник роботи: к.т.н., доцент Ліщина Н. М.

затверджені наказом закладу вищої освіти від «31» грудня 2025 р. № 541/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи бакалавра «29» травня 2026 р.

3. Вихідні дані до роботи: технічне завдання на розроблення веб-застосунку для роздрібно́ї онлайн-торгівлі кавою та чаєм, функціональні вимоги до системи з підтримкою трьох ролей користувачів (анонімний відвідувач, зареєстрований покупець, адміністратор), нефункціональні вимоги до продуктивності, безпеки і доступності, технологічні обмеження: Next.js 16, NestJS 11, Prisma ORM 6, MySQL 8.4, рекомендації OWASP ASVS рівня 1, ДСТУ 3008:2015 і ДСТУ 8302:2015.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що потрібно розробити): аналіз онлайн-торгівлі кавою та чаєм, огляд рішень і постановка завдань, визначення вимог, вибір технологій і проєктування архітектури, розробка системи (Next.js 16, NestJS 11, MySQL 8.4 + Prisma), впровадження безпеки (JWT, RBAC, bcrypt), тестування та підготовка документації і розгортання.

5. Перелік графічного матеріалу: 16 рисунків, 11 лістингів, 11 таблиць, 3 додатки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	Ліщина Н. М.		
Специфікація вимог до розробленої системи	Ліщина Н. М.		
Розробка об'єкта проектування	Ліщина Н. М.		
Нормоконтроль	Суринович О. М.		
Гарант ОП	Ліщина Н. М.		
Показник запозичень тексту	_____ %		
Академічна доброчесність	Ліщина Н. М.		

7. Дата видачі завдання «3» лютого 2026 р.

№	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 14.02.2026 р.	
2	Аналіз проблеми розробки та впровадження об'єкту проектування	до 03.03.2026 р.	
3	Обґрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 14.03.2026 р.	
4	Розробка функціонально-структурної схеми роботи об'єкта проектування та проектування бази даних	до 14.04.2026 р.	
5	Практична реалізація об'єкта проектування та розробка бази даних	до 20.04.2026 р.	
6	Тестування та налагодження об'єкта проектування	до 04.05.2026 р.	
7	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедрі	до 29.05.2026 р.	

Здобувач вищої освіти _____

Андрій ВАСИЛЬЄВ

Керівник кваліфікаційної роботи _____

Наталія ЛІЩИНА

АНОТАЦІЯ

Васильєв А. В. Розробка вебзастосунку для продажу кави та чаю із використанням Next.js та NestJS. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності 121 «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків.

У першому розділі досліджено сучасний стан онлайн-торгівлі кавою та чаєм в Україні, виконано порівняльний аналіз чотирьох діючих інтернет-магазинів і виявлено обмеження готових хмарних рішень. Сформульовано перелік завдань, які повинна вирішувати власна розробка. У другому розділі визначено категорії користувачів, складено функціональні та нефункціональні вимоги, обґрунтовано вибір технологічного стека. Підготовлено діаграму варіантів використання, компонентну архітектуру системи та перелік API-методів з прив'язкою до ролей доступу. У третьому розділі описано реалізацію клієнтської частини на Next.js 16 з App Router, серверної частини на NestJS 11 з Prisma 6, схему бази даних MySQL, підсистему безпеки на основі JWT і RBAC, план тестування і документацію проєкту. Описано дефекти, що виникли під час розробки, та шляхи їх усунення.

У результаті виконання роботи створено працездатний веб-додаток «Brewly», що містить публічний каталог, особистий кабінет покупця, оформлення замовлень для гостей і зареєстрованих користувачів, а також адміністративну панель для управління товарами, замовленнями та користувачами.

Ключові слова: електронна комерція, інтернет-магазин, кава, чай, Next.js, NestJS, Prisma, MySQL, JWT-автентифікація, RBAC.

ABSTRACT

Vasyliiev A. V. Development of a Web Application for Selling Coffee and Tea Using Next.JS and NestJS. Bachelor's qualification thesis in the educational program "Software Engineering", specialty 121 "Software Engineering". Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter examines the current state of online coffee and tea commerce in Ukraine, provides a comparative analysis of four existing online stores, and identifies the limitations of ready-made cloud solutions. A list of tasks that the custom-developed system should solve is formulated. The second chapter defines user categories, specifies functional and non-functional requirements, and substantiates the choice of the technological stack. A use case diagram, the component architecture of the system, and a list of API methods linked to access roles are prepared. The third chapter describes the implementation of the client side using Next.js 16 with App Router, the server side using NestJS 11 with Prisma 6, the MySQL database schema, the security subsystem based on JWT and RBAC, the testing plan, and the project documentation. Defects encountered during development and methods of their resolution are also described.

As a result of the work, a fully functional web application called "Brewly" was developed. The application includes a public catalog, a customer account, order placement for guests and registered users, and an administrative panel for managing products, orders, and users.

Keywords: e-commerce, online store, coffee, tea, Next.js, NestJS, Prisma, MySQL, JWT authentication, RBAC.

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1 АНАЛІЗ МЕТОДІВ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ	10
1.1 Аналіз сучасного стану проблеми	10
1.2 Постановка завдання на кваліфікаційну роботу бакалавра	13
Висновки до розділу 1	14
РОЗДІЛ 2 СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ...	15
2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення	15
2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання ...	21
Висновки до розділу 2	24
РОЗДІЛ 3 РОЗРОБКА СИСТЕМИ «BREWLY».....	25
3.1 Практична реалізація об'єкта проєктування.....	25
3.2 Тестування та налагодження інформаційно-комп'ютерної системи	34
3.3 Розробка бази даних	37
3.4 Захист інформаційної системи	41
3.5 Розробка документації для програмного продукту	44
Висновки до розділу 3.....	45
ВИСНОВКИ	47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	49
ДОДАТКИ	51

ВСТУП

Актуальність теми. Український ринок кави та чаю протягом останніх років активно розвивається в мережі Інтернет. Сучасні споживачі очікують детальної інформації про продукцію, зокрема походження зерна, ступінь обсмаження, рекомендації щодо приготування та дату обсмаження, а не лише фотографію упаковки. Більшість українських виробників кави та чаю використовують сайти, створені на базі конструкторів або рішень на основі CMS (Content Management System, система керування вмістом), які мають обмежені можливості для автоматизованого управління каталогом товарів.

Найпоширенішою практикою у сегменті малих виробників є використання SaaS-платформ (Software as a Service), зокрема Shopify або «Хорошоп». Такий підхід забезпечує швидкий запуск інтернет-магазину, однак прив'язує бізнес до умов тарифного плану, обмежує вибір платіжних провайдерів і ускладнює інтеграцію з власною CRM-системою. Власна реалізація на сучасних веб-технологіях усуває частину цих обмежень, але потребує системного проєктування архітектури, безпеки та подальшого супроводу.

У межах кваліфікаційної роботи спроектовано веб-застосунок електронної комерції «Brewly», призначений для продажу кави та чаю одним продавцем. Система поєднує публічний каталог із фільтрацією товарів, кошик, оформлення замовлень для гостей і зареєстрованих покупців, особистий кабінет користувача та адміністративну частину для керування товарами, замовленнями і користувачами. Клієнтську частину реалізовано на Next.js 16, серверну частину – на NestJS 11 за архітектурним стилем REST API (Representational State Transfer Application Programming Interface), а зберігання даних організовано в MySQL 8.4 через Prisma ORM (Object-Relational Mapping).

Світові тенденції у сфері електронної комерції підтверджують актуальність обраного напрямку розробки. Згідно з дослідженнями Baymard Institute, спеціалізовані інтернет-магазини демонструють вищу ефективність завдяки детальному опису товарів, зручній навігації та продуманому

користувацькому інтерфейсу [1]. У розроблюваному веб-застосунку передбачено підтримку гостьового оформлення замовлення та окрему логіку його обробки на серверній частині. Під час проєктування системи також враховано рекомендації стандарту OWASP ASVS (Application Security Verification Standard), що визначає базові вимоги до безпеки веб-застосунків.

Метою роботи є проєктування і реалізація програмного забезпечення веб-додатка для продажу кави та чаю з власним каталогом, оформленням замовлень, адміністративною панеллю та підсистемою безпеки на основі JWT (JSON Web Token, маркер доступу) і RBAC (Role-Based Access Control, рольове керування доступом).

Об'єктом дослідження є процес автоматизації роздрібної онлайн-торгівлі товарами категорії «кава та чай».

Предметом дослідження є моделі, методи та програмні засоби побудови повнофункціонального веб-застосунку електронної комерції з розділенням клієнтської та серверної частин.

Для досягнення мети поставлено такі завдання:

- 1) проаналізувати предметну область, дослідити існуючі рішення та сформулювати функціональні обмеження;
- 2) визначити функціональні та нефункціональні вимоги до системи з урахуванням трьох ролей користувачів;
- 3) обґрунтувати вибір технологічного стека для клієнтської і серверної частин та СУБД;
- 4) спроектувати схему бази даних з підтримкою каталогу, замовлень, відгуків, списку бажань і промо-банерів;
- 5) реалізувати клієнтську частину з публічним каталогом, особистим кабінетом і адміністративною панеллю;
- 6) реалізувати серверну частину з REST API, JWT-автентифікацією, RBAC;
- 7) виконати тестування основних сценаріїв роботи системи та задокументувати усунені дефекти;

8) підготувати документацію, інструкції з розгортання та мінімальні системні вимоги.

Методами дослідження є системний аналіз технологічних альтернатив, моделювання предметної області засобами UML (Unified Modeling Language, уніфікована мова моделювання), об'єктно-орієнтоване проєктування із використанням принципів IoC (Inversion of Control, інверсія керування) та DI (Dependency Injection, впровадження залежностей), ручне тестування системи за сценаріями використання, а також статичний аналіз коду засобами TypeScript.

Практичне значення отриманих результатів полягає у створенні функціонального веб-застосунку, який може бути адаптований до потреб реального продавця кави та чаю. Для впровадження системи достатньо налаштувати початкові дані каталогу, інтегрувати платіжний сервіс та виконати базове розгортання веб-застосунку. Запропоновані архітектурні рішення також можуть бути використані для створення систем електронної комерції в суміжних товарних категоріях.

РОЗДІЛ 1

АНАЛІЗ МЕТОДІВ РОЗРОБКИ ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПОСТАНОВКА ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Сегмент онлайн-продажу кави та чаю в Україні у 2017-2019 роках почав активно розвиватися, а впродовж 2024-2026 років набув ознак окремого напрямку електронної комерції. Частина покупців уже не сприймає каву лише як повсякденний товар, а звертає увагу на виробника, країну походження зерна, тип продукту, ступінь обсмаження та рекомендації щодо способу приготування. Подібні зміни помітні й у чайному сегменті: розширення асортименту пуерів, оолонгів, матча та інших спеціалізованих сортів зумовлює потребу в зручній системі категоризації, фільтрації та представлення товарів.

З технічного погляду більшість українських продавців обирає одне з трьох рішень: конструктори за моделлю SaaS, зокрема Хорошоп, Prom.ua, Shopify, готові платформи з відкритим кодом, такі як PrestaShop, Sylius, або власну розробку веб-застосунку. Кожен із цих підходів має певні переваги та обмеження, тому для визначення доцільності створення власної системи необхідно провести порівняльний аналіз існуючих аналогів.

Огляд аналогів проведено на прикладі чотирьох представників українського ринку:

- «Kavova» (kavova.choiceqr.com) – інтернет-магазин кави з широким асортиментом зернової продукції та позиціями преміального сегмента. До переваг належить зручна структура каталогу й базова система фільтрів. Водночас платформа побудована на типовому шаблоні, що обмежує можливості налаштування інтерфейсу;

- «Львівська копальня кави» (koralnia.coffee) – відомий український бренд, для якого веб-магазин є допоміжним каналом продажу поряд з офлайн-мережею. Перевагою є впізнаваний фірмовий стиль і якісне візуальне оформлення сайту.

Натомість відсутні розширені фільтри за типом зерна, способом обробки та ступенем обсмаження;

– «Віденська Кава» (wkava.com.ua) – інтернет-магазин кави з широким асортиментом зернової продукції, зокрема кави преміального сегмента. Магазин пропонує широкий асортимент як продукції, так і професійного обладнання, проте структура каталогу перевантажена, а індивідуалізація взаємодії з користувачем недостатня;

– «Surkoff» (surkoff.com.ua) – спеціалізований магазин кави та аксесуарів для приготування. Його перевагою є детальний опис походження зерна та смакового профілю. Водночас система фільтрації за країною походження, способом обробки та ступенем обсмаження залишається обмеженою.

Проведений аналіз показав, що більшість існуючих рішень орієнтується на базовий продаж і не поєднує одночасно гнучку систему фільтрації, детальний опис характеристик товару, зручне гостьове оформлення замовлення та повноцінну адміністративну панель. Отже, на ринку наразі бракує рішення, яке б об'єднувало всі зазначені функції, що обґрунтовує доцільність розробки власного веб-застосунку, адаптованого до специфіки продажу кави та чаю.

Окрему увагу слід приділити технічним аспектам реалізації подібних систем. SaaS-конструктори часто обмежують можливості оптимізації продуктивності, SEO (Search Engine Optimization, пошукової оптимізації) та кастомізації інтерфейсу. Використання React Server Components (серверних компонентів React) дає змогу зменшити обсяг клієнтського JavaScript і оптимізувати рендеринг сторінок [2, 3]. Додатково застосування ISR (Incremental Static Regeneration, інкрементальної статичної регенерації) дозволяє підвищити швидкодію веб-застосунку та покращити показники Core Web Vitals [4]. Окрему роль у забезпеченні масштабованості та інтеграційних можливостей відіграє REST API, що створює основу для подальшого розширення системи, зокрема розробки мобільного клієнта та взаємодії із зовнішніми сервісами [5].

Для забезпечення безпеки користувацьких даних інтернет-магазин повинен підтримувати безпечну реєстрацію, зберігання облікових даних у

захищеному вигляді, аудит дій адміністраторів та запобігання несанкціонованому доступу до особистих кабінетів. SaaS-рішення зазвичай делегують питання безпеки провайдеру, тоді як власна розробка потребує їх явної реалізації. У проєкті «Brewly» для захисту паролів використано bcrypt-хешування [6]. Для автентифікації застосовано JWT-токени з коротким часом життя та механізмом refresh-токенів [7]. Розмежування доступу реалізовано за допомогою рольових вартових (guards) RBAC [8]. Детальний опис реалізації наведено у розділі 3.4.

Підсумовуючи огляд, доцільність власної розробки веб-застосунку для продажу кави та чаю обумовлена трьома чинниками: відсутністю готового рішення на ринку з потрібним набором функцій, технічними обмеженнями SaaS-платформ і вимогами до контролю над безпекою. Порівняльну оцінку підходів представлено у таблиці 1.1.

Таблиця 1.1 – Порівняння підходів інтернет-магазину кави та чаю

Підхід	Переваги	Недоліки	Висновок для проєкту «Brewly»
SaaS-конструктор (Хорошоп, Prom, Shopify)	Швидкий запуск, готові шаблони, інтеграція з провайдерами доставки	Обмежені поля картки товару, плата за тариф, важка міграція даних, обмежений SEO	Не підходить через брак фільтрів специфічних для кави/чаю
Готовий open-source двигун (PrestaShop, Sylius)	Безкоштовне ядро, спільнота, плагіни	Складна кастомізація, важка міграція між мажорними версіями, високі вимоги до хостингу	Не задовольняє вимоги через значну складність кастомізації, залежність від сторонніх модулів і витрати на міграцію між версіями
Замовлення у студії під ключ	Готовий продукт, авторський дизайн	Висока вартість, залежність від виконавця, мінімальний контроль після здачі	Не є оптимальним для кваліфікаційної роботи, оскільки обмежує самостійне проєктування та аналіз архітектурних рішень

Продовження таблиці 1.1

Підхід	Переваги	Недоліки	Висновок для проєкту «Brewly»
Власний CMS на CMS-фреймворку (Strapi, Sanity)	Гнучкі моделі контенту, headless-підхід	Обмежений каталог-функціонал, відсутність комерційних модулів без додаткових налаштувань	Частково підходить, але потребує суттєвої добудови комерційної логіки
Власний веб-застосунок електронної комерції на Next.js + NestJS + Prisma (Brewly)	Повний контроль над моделлю даних, JWT/RBAC, можливість мобільного клієнта, безпека за OWASP ASVS, контейнеризація	Триваліший термін розробки, вищі вимоги до кваліфікації розробника	Саме цей підхід обрано: відповідає функціональним і нефункціональним вимогам, має потенціал для масштабування

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

Виконаний у підрозділі 1.1 аналіз показав, що проєктована система повинна закривати функціональність каталогу, гостьове і автентифіковане оформлення замовлення, особистий кабінет покупця та повноцінну адміністративну панель з безпечним розмежуванням ролей. Метою кваліфікаційної роботи є розробка прототипу такої системи на сучасному стеку JavaScript-технологій.

Для досягнення поставленої мети необхідно виконати такі завдання:

- 1) проаналізувати предметну область онлайн-торгівлі кавою та чаєм, оглянути існуючі рішення, виявити їхні обмеження;
- 2) визначити функціональні та нефункціональні вимоги до інформаційної системи електронної комерції з урахуванням трьох ролей (анонімний відвідувач, зареєстрований покупець, адміністратор);
- 3) обґрунтувати вибір технологічного стека – фреймворків клієнтської та серверної частин, ORM, СУБД, бібліотек валідації та автентифікації;

4) спроектувати схему бази даних, що підтримує каталог (продукти, категорії, бренди, зображення), замовлення (з гостьовим і авторизованим режимами), відгуки, список бажань, промо-банери, ролі доступу;

5) реалізувати клієнтську частину на Next.js 16 з публічним каталогом, кошиком, оформленням замовлення, особистим кабінетом і адміністративною панеллю;

6) реалізувати серверну частину на NestJS 11 з REST API, JWT-автентифікацією, ролевим розмежуванням та модульною структурою;

7) виконати тестування основних сценаріїв роботи системи, описати ручні тест-кейси і знайдені дефекти;

8) підготувати документацію, інструкції з розгортання, мінімальні системні вимоги та опис вбудованої довідки.

У межах роботи свідомо не реалізовано інтеграцію з платіжними провайдерами (monobank, Fondy, LiqPay) та API служб доставки (Nova Poshta, Укрпошта), оскільки це потребує реальних договорів. Натомість передбачено вибір способу доставки й оплати на рівні даних замовлення з можливістю подальшого розширення через додавання нового модуля. Кінцевий продукт є демонстраційним прототипом, готовим до перетворення на повноцінний інтернет-магазин після узгодження з платіжними та логістичними сервісами.

Висновки до розділу 1

За підсумками огляду сформульовано вісім завдань кваліфікаційної роботи, які охоплюють етапи аналізу, проєктування, реалізації, тестування та документування системи. Інтеграції з платіжними й логістичними сервісами свідомо обмежено рівнем моделі даних, оскільки їх повноцінне підключення потребує окремого узгодження з відповідними провайдерами. Постановка завдання визначає чіткий обсяг робіт для подальшої реалізації системи та враховує баланс між технічною складністю й можливостями виконання проєкту в межах бакалаврської роботи.

РОЗДІЛ 2

СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Підставою для формування переліку вимог стали результати аналізу, наведені у підрозділі 1.1, рекомендації стандарту OWASP ASVS та типові сценарії використання інтернет-магазинів. Вимоги поділено на функціональні та нефункціональні. Функціональні вимоги визначають основні можливості системи, а нефункціональні характеризують її безпеку, продуктивність та якість роботи. Таке розділення дозволило чітко визначити межі функціонування системи та встановити критерії оцінювання її ефективності під час реалізації й тестування.

У системі визначено три категорії користувачів. Анонімний відвідувач має доступ до публічного каталогу, сторінки товару, локального кошика та гостьового оформлення замовлення. Зареєстрований покупець (роль CUSTOMER у моделі User) отримує особистий кабінет з історією замовлень, списком бажань, можливістю керування адресами доставки та залишення відгуків. Адміністратор (роль ADMIN) має доступ до операцій створення, читання, оновлення та видалення (CRUD, Create, Read, Update, Delete) над товарами, категоріями, брендами та банерами, а також до перегляду користувачів і замовлень та керування статусами замовлень.

Для виявлення вимог використано два підходи. Аналіз сценаріїв використання представлено у вигляді діаграми варіантів використання (рис. 2.1). Аналіз реалізованих сторінок веб-застосунку та API дозволив зіставити визначені вимоги з наявною функціональністю системи. Додатково було враховано особливості взаємодії між клієнтською та серверною частинами застосунку, що дало змогу точніше сформулювати вимоги до обробки даних, авторизації користувачів та стабільності роботи системи.

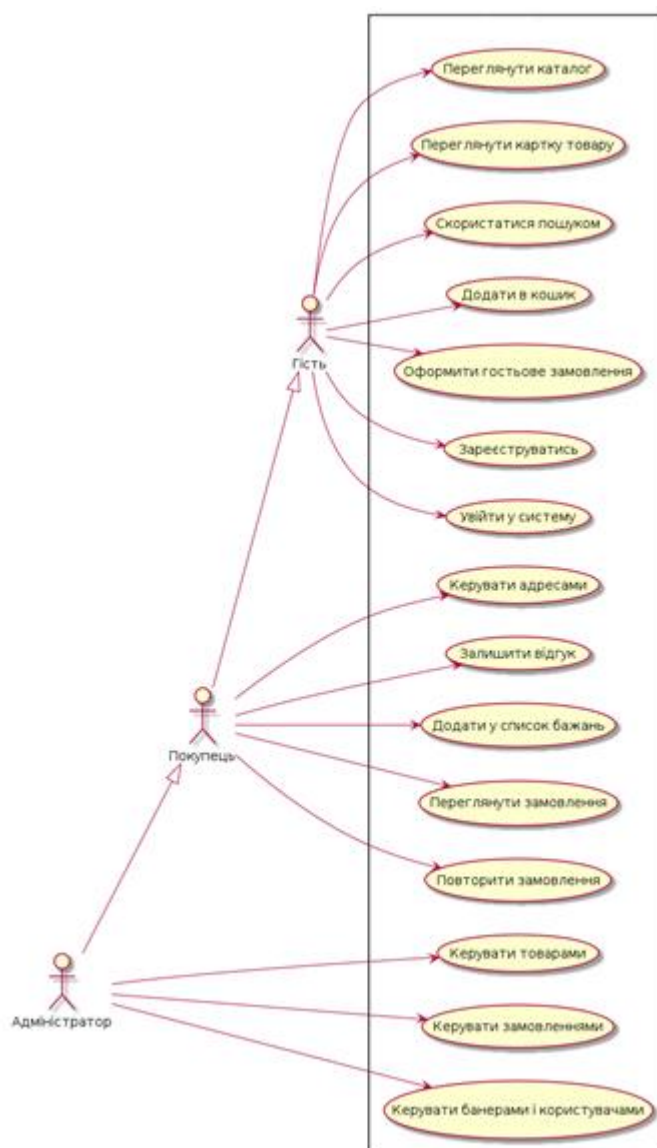


Рисунок 2.1 – Діаграма варіантів використання системи «Brewly»

Маршрутна сітка вітрини побудована за принципом App Router у Next.js 16: публічні маршрути «/», «/catalog», «/catalog/[slug]», «/cart», «/checkout», «/auth/login», «/auth/register» доступні без автентифікації. Кабінет автентифікованого користувача охоплює маршрути «/account», «/account/orders», «/account/wishlist», «/account/addresses». Адміністративна частина – це окрема група маршрутів під «/admin/*»: панель керування, продукти (з підмаршрутом створення «/admin/products/new»), категорії, бренди, банери, замовлення, користувачі. Захист маршрутів реалізовано на рівні middleware (client/proxy.ts), яка перевіряє наявність access-токена для «/account/*» та додатково роль ADMIN для «/admin/*».

Серверні кінцеві точки API побудовано як уніфіковану REST-схему з префіксом `/api/v1`. У таблиці 2.1 наведено ключові API-методи, що охоплюють основні сценарії роботи системи. Кожен метод, що потребує автентифікації, отримує JWT-токен у заголовку `Authorization: Bearer ....` Адміністративні методи додатково проходять через `RolesGuard`, який перевіряє метадані `@Roles(Role.ADMIN)`. Усі публічні методи позначені декоратором `@Public()`, який обходить глобальний `JwtAuthGuard`.

Таблиця 2.1 – Основні API-методи системи

Метод	Endpoint	Guard / роль	Опис
POST	<code>/api/v1/auth/register</code>	<code>@Public</code>	Реєстрація нового користувача
POST	<code>/api/v1/auth/login</code>	<code>@Public</code>	Вхід, повертає <code>access+refresh</code>
POST	<code>/api/v1/auth/refresh</code>	<code>RefreshJwtGuard</code>	Оновлення пари токенів
GET	<code>/api/v1/auth/me</code>	<code>JwtAuthGuard</code>	Дані поточного користувача
POST	<code>/api/v1/auth/logout</code>	<code>JwtAuthGuard</code>	Очищення <code>refreshTokenHash</code>
GET	<code>/api/v1/products</code>	<code>@Public</code>	Список товарів з фільтрами
GET	<code>/api/v1/products/:slug</code>	<code>@Public</code>	Картка товару
GET	<code>/api/v1/products/filters/meta</code>	<code>@Public</code>	Метадані фільтрів
GET	<code>/api/v1/products/search-suggestions</code>	<code>@Public</code>	Автодоповнення пошуку
GET	<code>/api/v1/categories</code>	<code>@Public</code>	Список категорій
GET	<code>/api/v1/brands</code>	<code>@Public</code>	Список брендів
GET	<code>/api/v1/promo-banners</code>	<code>@Public</code>	Активні промо-банери
POST	<code>/api/v1/cart/promo/validate</code>	<code>@Public</code>	Перевірка промо-коду
POST	<code>/api/v1/orders/guest</code>	<code>@Public</code>	Гостьове замовлення
POST	<code>/api/v1/orders</code>	<code>JwtAuthGuard</code>	Замовлення авторизованого
GET	<code>/api/v1/orders/my</code>	<code>JwtAuthGuard</code>	Історія замовлень
POST	<code>/api/v1/orders/:id/reorder</code>	<code>JwtAuthGuard</code>	Повтор замовлення
GET	<code>/api/v1/users/profile</code>	<code>JwtAuthGuard</code>	Профіль і адреси
POST	<code>/api/v1/wishlist/items</code>	<code>JwtAuthGuard</code>	Додати в список бажань

Продовження таблиці 2.1

Метод	Endpoint	Guard / роль	Опис
POST	/api/v1/reviews	JwtAuthGuard	Залишити відгук
POST	/api/v1/admin/products	@Roles(ADMIN)	Створити товар
PATCH	/api/v1/admin/products/:id	@Roles(ADMIN)	Оновити товар
DELETE	/api/v1/admin/products/:id	@Roles(ADMIN)	Видалити товар
GET	/api/v1/admin/dashboard	@Roles(ADMIN)	Метрики панелі керування
PATCH	/api/v1/admin/orders/:id/status	@Roles(ADMIN)	Зміна статусу замовлення

Підхід до користувацького досвіду (UX, User Experience) ґрунтується на принципі поступового подання інформації. На стартовій сторінці акцент зроблено на ключових товарних блоках, бестселерах і промо-банерах. Каталог дає можливість фільтрувати товари за типом продукту, категорією, брендом, наявністю на складі та діапазоном цін. Картка товару показує опис, специфікації у форматі ключ-значення, поточну ціну, попередню ціну (якщо є знижка), середній рейтинг і список відгуків. У кошику застосовується клієнтська логіка зберігання у localStorage [9].

Інформаційні потоки системи описуються трьома сценаріями. Сценарій реєстрації та входу передбачає надсилання клієнтом форми з email і паролем, після чого API повертає пару access/refresh токенів, які зберігаються у cookie з назвами `brewly_access_token` і `brewly_refresh_token`. Під час оформлення замовлення клієнт у `/checkout` заповнює форму, валідовану Zod-схемою, і викликає `POST /api/v1/orders` (для авторизованого) або `POST /api/v1/orders/guest`. Сценарій адміністрування передбачає створення товару адміністратором через `/admin/products/new` та `POST /api/v1/admin/products`; на стороні API DTO (Data Transfer Object, об'єкт передавання даних) валідується `class-validator` [10].

Функціональні вимоги до системи розподілено за ролями і наведено у таблиці 2.2. Нефункціональні вимоги, які стосуються продуктивності, безпеки,

доступності, переносності та супроводу, представлено у таблиці 2.3. Загальну матрицю доступу – у таблиці 2.4.

Таблиця 2.2 – Функціональні вимоги до системи «Brewly»

ID	Роль	Вимога	Реалізація
F01	Анонімний	Переглянути публічний каталог з фільтрацією	GET /products + /catalog
F02	Анонімний	Переглянути картку товару за slug	GET /products/:slug + /catalog/[slug]
F03	Анонімний	Скористатись автодоповненням пошуку	GET /products/search-suggestions?q=
F04	Анонімний	Додати/видалити товар у кошику	CartProvider (localStorage)
F05	Анонімний	Оформити замовлення без реєстрації	POST /orders/guest
F06	Анонімний	Зареєструватись як покупець	POST /auth/register
F07	Покупець	Увійти і отримати JWT-пару	POST /auth/login
F08	Покупець	Переглянути особистий профіль	GET /users/profile
F09	Покупець	Керувати власними адресами	POST/PATCH/DELETE /users/addresses
F10	Покупець	Сформувати список бажань	POST/DELETE /wishlist/items
F11	Покупець	Залишити відгук на товар	POST /reviews
F12	Покупець	Переглянути історію замовлень	GET /orders/my
F13	Покупець	Повторити минуле замовлення	POST /orders/:id/reorder
F14	Покупець	Застосувати промо-код у кошику	POST /cart/promo/validate
F15	Адмін	Переглянути зведену панель керування	GET /admin/dashboard
F16	Адмін	Створити/змінити/видалити товар	POST/PATCH/DELETE /admin/products/...
F17	Адмін	Керувати категоріями, брендами, банерами	/admin/categories, /brands, /banners
F18	Адмін	Змінити статус замовлення	PATCH /admin/orders/:id/status

Таблиця 2.3 – Нефункціональні вимоги до системи «Brewly»

Категорія	Вимога	Реалізація	Перевірка
Доступність	Підтримка темної/світлої теми, контрастний UI	next-themes + Tailwind tokens	Перевірка у Lighthouse

Продовження таблиці 2.3

Категорія	Вимога	Реалізація	Перевірка
Безпека (Auth)	Паролі зберігаються лише у вигляді хешу	bcrypt з кількістю раундів з ENV (Environment Variables) BCRYPT_ROUNDS	Перевірка БД, що passwordHash нечитабельний
Безпека (Tokens)	JWT-токени короткі, із оновленням	access 15 хв + refresh 7 днів, через POST /auth/refresh	Декодування через jwt.io
Безпека (Validation)	Серверна валідація усіх DTO	Глобальний ValidationPipe({whitelist, forbidNonWhitelisted})	Спроба надіслати зайве поле – 400
Продуктивність	TTFB (Time To First Byte – час до отримання першого байта відповіді) не більше 800 мс для каталогу при 1000 продуктах	Серверні компоненти, індекси Prisma на категорії і бренді	Профілювання у DevTools

Таблиця 2.4 – Ролі користувачів та права доступу

Роль	Frontend	Backend API	Обмеження
Анонімний	/, /catalog, /catalog/[slug], /cart, /checkout, /auth/*	усі endpoint-и з декоратором @Public()	заборонено доступ до /account/*, /admin/*
Покупець (CUSTOMER)	усе вище + /account/*	endpoint-и з JWT (без RBAC-обмежень)	заборонено /admin/* (redirect до головної)
Адміністратор (ADMIN)	усе вище + /admin/*	усі endpoint-и + ті, що мають @Roles(Role.ADMIN)	відсутні; повний контроль

Архітектурне рішення є класичною трирівневою моделлю: рівень презентації (Next.js + React), рівень бізнес-логіки (NestJS REST API з валідаторами і вартовими доступу), рівень даних (MySQL через Prisma). Структура усіх компонентів і зовнішніх інтеграцій (поки що відсутніх) ілюструє рисунок 2.2.

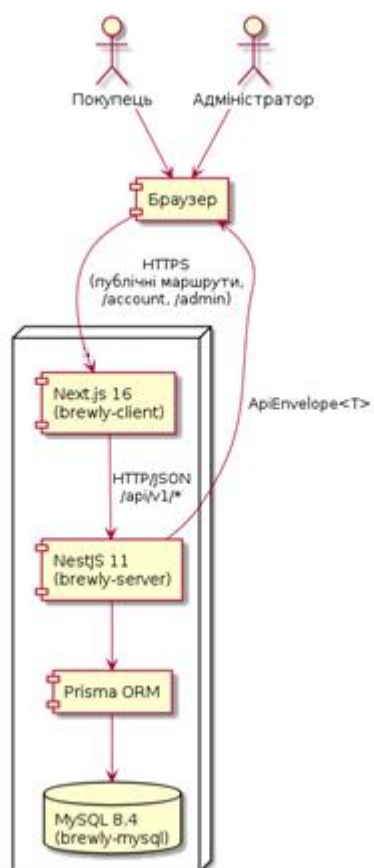


Рисунок 2.2 – Архітектура системи «Brewly»

Особливістю обраного підходу є вибір клієнтського кошика без серверного представлення. Такий підхід знижує навантаження на API під час перегляду каталогу, дозволяє підтримувати гостьовий сценарій без створення анонімного користувача в базі і спрощує синхронізацію між вкладками браузера через подію storage. У замовлення кошик потрапляє лише в момент POST /orders або POST /orders/guest як масив OrderItemInputDto.

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

Технологічний стек було обрано на основі порівняльного аналізу альтернативних рішень для кожного рівня системи. Під час вибору враховувалися функціональні можливості технологій, складність освоєння,

рівень підтримки спільнотою та перспективи подальшого розвитку. Перелік обраних технологій і їх призначення наведено у таблиці 2.5.

Таблиця 2.5 – Технологічний стек системи «Brewly»

Рівень	Технологія	Версія	Призначення
Клієнтська частина	Next.js	16.0	App Router, серверні компоненти, SSR/CSR
Клієнтська частина	React	19.0	Бібліотека UI-компонентів
Клієнтська частина	TypeScript	5.7	Статична типізація
Клієнтська частина	TanStack Query	5.62	Серверний кеш і мутації
Клієнтська частина	React Hook Form	7.54	Робота з формами
Клієнтська частина	Zod	3.24	Декларативна валідація форм
Клієнтська частина	Tailwind CSS	3.4	Утилітарний CSS і дизайн-токени
Серверна частина	NestJS	11.0	Модульний REST-фреймворк
Серверна частина	Prisma ORM	6.0	Типобезпечний доступ до БД
Серверна частина	Passport-JWT	4.0	Стратегії автентифікації
Серверна частина	bcrypt	5.1	Хешування паролів і refresh-токенів
Серверна частина	class-validator	0.14	Валідація DTO
СУБД	MySQL	8.4	Реляційна СУБД
Інфраструктура	Node.js	22	Середовище виконання

Next.js обрано як фреймворк клієнтської частини на основі React. Альтернативою розглядався Remix. Вибір зумовлено підтримкою серверного рендерингу, App Router та Server Components, що дозволяє поєднувати клієнтську та серверну логіку в межах одного застосунку і зменшує навантаження на API [11].

NestJS обрано замість мінімального Express-сервера через вбудовану модульну організацію коду та IoC-контейнер [12]. Додатковою перевагою стала готова інтеграція з Passport-стратегіями для реалізації автентифікації [13]. Також NestJS надає ValidationPipe на основі class-validator, що спрощує перевірку вхідних даних. Цикл життя контролерів і провайдерів стандартизовано, що зменшує складність підтримки системи. Альтернативою в JavaScript-екосистемі могло б бути використання Fastify з готовими плагінами, однак такий підхід потребує додаткового обсягу робіт із безпеки, валідації та документації, який виходить за межі поставлених завдань кваліфікаційної роботи.

Prisma ORM обрано замість TypeORM. Prisma дає типобезпечний клієнт, який генерується з декларативної схеми (schema.prisma), що зменшує кількість помилок під час рефакторингу [14]. TypeORM спирається на декоратори на класах, але має складніший механізм міграцій і синхронізації сутностей. Ключовим для проєкту є наявність генератора TypeScript-типів та інструментів prisma db push / prisma migrate dev для швидкого застосування змін схеми у середовищі розробки [15].

MySQL обрано замість PostgreSQL з урахуванням цільових інфраструктурних вимог проєкту та простоти розгортання в типовому середовищі хостингу [16]. Для реалізації поточного функціоналу можливостей MySQL є достатньо, а сама система добре підтримується більшістю хостинг-провайдерів. Оскільки Prisma підтримує обидві СУБД, у майбутньому зберігається можливість міграції на PostgreSQL без суттєвих змін у прикладній логіці застосунку.

Zod обрано для валідації клієнтських форм [17]. Альтернативою був Yup, але Zod краще інтегрується з TypeScript: типи виводяться через z.infer<>, відсутній окремий API для типів і валідаторів. У парі з @hookform/resolvers/zod це дає однорідний потік обробки форм без додаткових адаптерів.

TanStack Query (React Query) обрано замість SWR. Обидві бібліотеки вирішують одну задачу – серверний кеш у React, – однак TanStack Query має ширший API (мутації, нескінченні запити, інвалідація за ключами), кращу

підтримку SSR через HydrationBoundary та офіційні DevTools [18]. Це важливо для адмін-панелі, де список товарів кешується і повинен інвалідуватися після кожної CRUD-операції.

Tailwind CSS використано для системи дизайн-токенів. Альтернативою була CSS-in-JS бібліотека vanilla-extract або стилі через CSS Modules. Tailwind обрано через швидкий цикл внесення змін і їхнього перегляду, а також відсутністю додаткових runtime-обчислень стилів, оскільки вони компонуються на етапі збирання [19]. Колірна палітра спирається на CSS-змінні --background, --foreground, --primary, --secondary, що дозволяє підключити темну тему через class="dark".

Для автентифікації обрано пару JWT access + refresh. Альтернативою були серверні сесії на основі cookie з випадковим ідентифікатором. JWT-стратегія дозволяє валідувати запити без звернень до БД, а refresh-токен перевіряється через bcrypt-порівняння з полем refreshTokenHash у моделі User. Час життя access-токена – 15 хвилин, refresh-токена – 7 днів.

Висновки до розділу 2

У розділі сформовано таблиці функціональних і нефункціональних вимог, матрицю доступу, а також підготовлено діаграму варіантів використання і компонентну архітектуру. У таблиці 2.1 наведено ключові API-методи, які охоплюють основні рольові сценарії системи.

Технологічний стек обрано на основі порівняльного аналізу. Next.js, NestJS, Prisma і MySQL разом утворюють збалансовану платформу: типобезпечну, добре документовану і придатну до контейнеризації. JWT-автентифікація відповідає вимозі до середовища без збереження стану між запитами.

Сформована специфікація вимог і архітектурна модель визначають основу для подальшого опису реалізації клієнтської та серверної частин, бази даних, підсистеми безпеки та документації у розділі 3.

РОЗДІЛ 3

РОЗРОБКА СИСТЕМИ «BREWLY»

3.1 Практична реалізація об'єкта проєктування

Структура проєкту організована як monorepository з ізольованими модулями клієнтської та серверної частин. Такий підхід забезпечує логічне розмежування відповідальності, спрощує супровід програмного коду та уніфікує процес локального розгортання системи без необхідності дублювати конфігураційні файли у кількох незалежних репозиторіях. Крім того, використання єдиної структури репозиторію спрощує керування залежностями, спільними типами даних та процесом збірки застосунку.

Клієнтська частина побудована на Next.js 16 з App Router. Корінь маршрутів – каталог client/app/, з підмаршрутами auth/, catalog/, cart/, checkout/, account/ і admin/. У файлі app/layout.tsx підключаються глобальні стилі Tailwind, шрифти і провайдери з src/shared/providers/app-providers.tsx: ThemeProvider, QueryClientProvider (TanStack Query), CartProvider, Toaster від sonner. Така організація забезпечує централізоване керування станом застосунку, стилями та глобальними компонентами інтерфейсу.

Логічну організацію клієнтського коду виконано за підходом розподілу за функціональними модулями. Каталог src/shared/ містить бібліотеку багаторазових UI-компонентів (Button, Input, Textarea, Card, Badge, Skeleton), компоненти макета (Header, Footer, MobileMenu), допоміжні модулі. Каталог src/features/ об'єднує бізнес-логіку домену: auth (форми та схеми Zod), cart (контекст і збереження у localStorage), catalog (компонент-сторінка), checkout (форма оформлення замовлення, як показано на рис. 3.1). Каталог src/widgets/ зарезервовано під великі інтерфейсні блоки – поки що тут зберігається лише product-details.tsx. Каталог src/services/api/ містить типізовані сервісні функції: auth.service.ts, catalog.service.ts, orders.service.ts, users.service.ts, admin.service.ts та базовий fetch-обгортку http.ts.

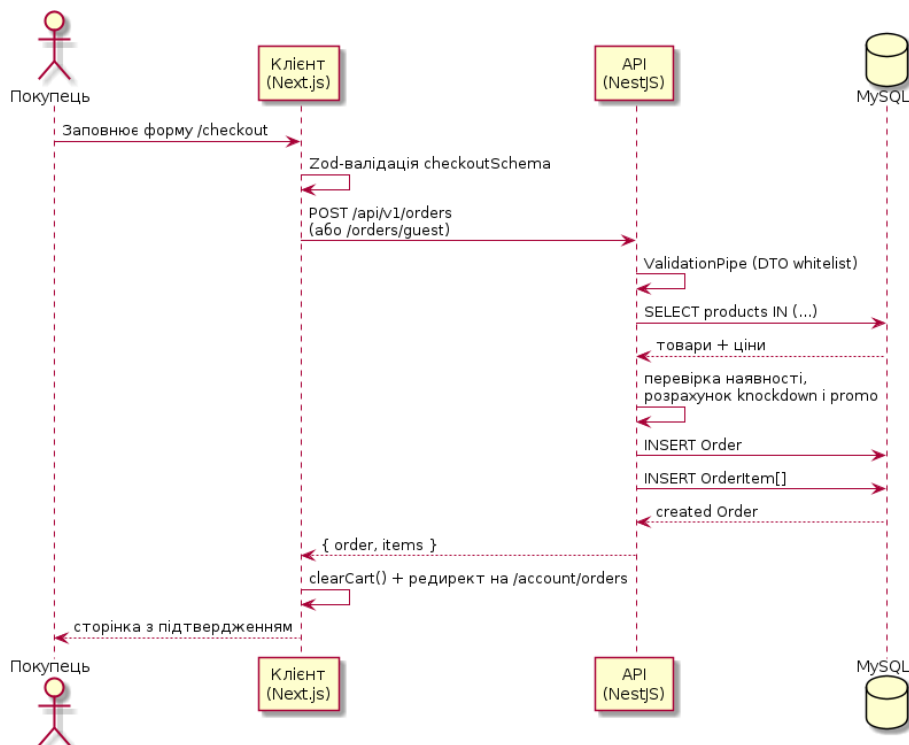


Рисунок 3.1 – Послідовність оформлення замовлення

Контекст кошика реалізує дві властивості, важливі для UX інтернет-магазину: збереження стану при перезавантаженні сторінки та миттєвий розрахунок підсумків. Початкове завантаження зчитує JSON з ключа `brewly_cart` у `localStorage`, а будь-яке наступне оновлення стейту синхронно записує його туди ж. Підсумки `subtotal` і `total` обчислюються через `useMemo`. Це показано у лістингу 3.1.

Лістинг 3.1 – Контекст кошика зі збереженням стану в `localStorage`

```

const CART_STORAGE_KEY = 'brewly_cart';

export function CartProvider({ children }: { children: React.ReactNode }) {
  const [items, setItems] = useState<CartItem[]>([]);
  const [promo, setPromo] = useState<PromoValidation | null>(null);

  useEffect(() => {
    const storedCart = window.localStorage.getItem(CART_STORAGE_KEY);
    if (storedCart) {
      setItems(JSON.parse(storedCart) as CartItem[]);
    }
  }, []);
}

```

```

useEffect(() => {
  window.localStorage.setItem(CART_STORAGE_KEY, JSON.stringify(items));
}, [items]);

const value = useMemo<CartContextValue>(() => {
  const subtotal = items.reduce(
    (sum, item) => sum + Number(item.product.price) * item.quantity,
    0,
  );
  const total = Math.max(subtotal - (promo?.discountAmount ?? 0), 0);
  // ... addItem / removeItem / updateQuantity / clearCart
  return { items, promo, subtotal, total, /* ... */ } as
CartContextValue;
}, [items, promo]);

return <CartContext.Provider
value={value}>{children}</CartContext.Provider>;
}

```

кінець лістингу 3.1

Маршрути `/account/*` і `/admin/*` захищено окремим файлом `client/proxy.ts`, який працює як Next.js middleware (проміжний обробник). Перевірка виконується за cookie `brewly_access_token` і `brewly_role`. Якщо доступу до `/account/*` немає, користувача перенаправляє на `/auth/login?next=<pathname>`; якщо у разі `/admin/*` роль не дорівнює `ADMIN` – користувача повертає на головну сторінку. Повний код захисту маршрутів наведено у додатку А (ліст. А.1).

Сторонні API-виклики проходять через функцію `apiRequest` у `client/src/services/api/http.ts`. Її особливості: автоматичне додавання заголовка `Authorization`, обробка 401-помилки з одноразовою спробою оновити пару токенів через `/auth/refresh`, уніфікований формат `ApiEnvelope<T>` для розпаковки. Серверні запити у реалізації передаються через `fetch` із параметром `cache: 'no-store'` для приватних сторінок. Повний код клієнтського `fetch`-обгортувача з автоматичним `refresh` наведено у додатку А (ліст. А.2).

Серверна частина побудована як NestJS-додаток з кореневим модулем `AppModule`. Він підключає `ConfigModule` (з валідатором ENV-змінних), `PrismaModule` і дванадцять доменних модулів: `AuthModule`, `UsersModule`,

ProductsModule, CategoriesModule, BrandsModule, ReviewsModule, WishlistModule, OrdersModule, CartModule, PromoBannersModule, AdminModule. Кожен з них має контролер, сервіс і набір DTO. Глобально зареєстровано два вартові доступу (JwtAuthGuard і RolesGuard), глобальний ValidationPipe, інтерсептор відповідей і фільтр помилок.

Початкова конфігурація додатка зосереджена у server/src/main.ts: встановлення глобального префіксу /api/v1, CORS-allowlist із CLIENT_URL, активація graceful shutdown через prismaService.enableShutdownHooks. Послідовність виконання – обернена до традиційної (спочатку префікс, потім pipes, потім filter), що відповідає рекомендаціям документації Nest.

Контролер AuthController доступний за маршрутом /api/v1/auth/*. Методи register, login і refresh позначені декораторами @Public(), тоді як me і logout вимагають JWT. На реєстрації перевіряється унікальність email, обчислюється bcrypt-хеш пароля та зберігається користувач разом з парою токенів. На вході виконується bcrypt.compare для пароля; refresh-токен порівнюється з полем refreshTokenHash. Це показано у лістингу 3.2.

Лістинг 3.2 – Реєстрація користувача з bcrypt-хешем і парою токенів

```

async register(dto: RegisterDto) {
  const existingUser = await this.prisma.user.findUnique({
    where: { email: dto.email.toLowerCase() },
  });

  if (existingUser) {
    throw new ConflictException('User with this email already exists');
  }

  const passwordHash = await this.hashValue(dto.password);

  const user = await this.prisma.user.create({
    data: {
      firstName: dto.firstName,
      lastName: dto.lastName,
      email: dto.email.toLowerCase(),
      phone: dto.phone,
      passwordHash,
    },
  });
};

```

```

const tokens = await this.generateTokens(user);
await this.storeRefreshToken(user.id, tokens.refreshToken);

return {
  user: this.serializeUser(user),
  tokens,
};
}

```

кінець лістингу 3.2

Усе CRUD-розгалуження для каталогу зосереджено у `ProductsController`. Він обслуговує одночасно публічні методи (GET `/products`, `/:slug`, `/filters/meta`, `/search-suggestions`) і адміністративні (POST/PATCH/DELETE `/admin/products/...`). Логіка лістингу з пагінацією, сортуванням і фільтрами розміщена у `products.service.ts` (метод `listProducts`, ліст. 3.3). Запит формується через типобезпечний `Prisma.ProductWhereInput` і виконується паралельно з підрахунком `count` через `Promise.all`.

Лістинг 3.3 – Лістинг продуктів з фільтрацією і пагінацією

```

async listProducts(query: ProductsQueryDto) {
  const { page, pageSize, sortBy } = query;
  const where = this.buildWhere(query);
  const orderBy = this.buildSort(sortBy);

  const [items, total] = await Promise.all([
    this.prisma.product.findMany({
      where,
      orderBy,
      skip: (page - 1) * pageSize,
      take: pageSize,
      include: {
        category: true,
        brand: true,
        images: { orderBy: { sortOrder: 'asc' } },
      },
    }),
    this.prisma.product.count({ where }),
  ]);

  return {
    items,

```

```

    meta: { page, pageSize, total, pageCount: Math.ceil(total /
    pageSize) },
    };
}

```

кінець лістингу 3.3

Захист методів `@Roles(Role.ADMIN)` базується на класі `RolesGuard` (`server/src/common/roles.guard.ts`, ліст. 3.4). Вартовий доступу читає метадані `ROLES_KEY`, які проставив декоратор `@Roles(...)`, та порівнює їх з роллю користувача, видобутою з `request.user` (наповнюється `JwtAuthGuard`). Якщо метадані відсутні – доступ дозволено за замовчуванням, що зручно для публічних маршрутів.

Лістинг 3.4 – Перевірка ролі через `RolesGuard`

```

@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private readonly reflector: Reflector) {}

  canActivate(context: ExecutionContext): boolean {
    const requiredRoles =
    this.reflector.getAllAndOverride<string[]>(ROLES_KEY, [
      context.getHandler(),
      context.getClass(),
    ]);

    if (!requiredRoles || requiredRoles.length === 0) {
      return true;
    }

    const request = context.switchToHttp().getRequest();
    const user = request.user as JwtPayload | undefined;

    return Boolean(user && requiredRoles.includes(user.role));
  }
}

```

кінець лістингу 3.4

Промо-коди реалізовано як окремий модуль `CartModule` з єдиним методом `POST /cart/promo/validate`. Правила знижок створено у `cart.service.ts` (ліст. 3.5): `SPRING10` дає 10 % при підсумку від 800 грн, `TEATIME150` – фіксовану знижку

150 грн при підсумку від 800 грн. Підхід обрано свідомо: повноцінна модель промо-кампаній виходить за межі бакалаврського обсягу, а текстова конфігурація вистачає для демонстрації UX.

Лістинг 3.5 – Перевірка промо-коду на сервері

```
const PROMO_RULES = {
  SPRING10: { type: 'percentage', value: 10, minSubtotal: 800, label:
'Seasonal 10% off' },
  TEATIME150: { type: 'fixed', value: 150, minSubtotal: 900, label: 'Tea
time special discount' },
} as const;

@Injectable()
export class CartService {
  validatePromo(code: string, subtotal = 0) {
    const normalizedCode = code.trim().toUpperCase();
    const promo = PROMO_RULES[normalizedCode];

    if (!promo) {
      return { valid: false, code: normalizedCode, discountAmount: 0,
        message: 'Promo code not found' };
    }
    if (subtotal < promo.minSubtotal) {
      return { valid: false, code: normalizedCode, discountAmount: 0,
        message: `Minimum subtotal is ${promo.minSubtotal} UAH` };
    }
    const discountAmount = promo.type === 'percentage'
      ? Number(((subtotal * promo.value) / 100).toFixed(2))
      : promo.value;
    return { valid: true, code: normalizedCode, discountAmount,
      label: promo.label, type: promo.type, value: promo.value };
  }
}
```

кінець лістингу 3.5

Адміністративний модуль AdminModule не містить класичних CRUD-операцій, а об'єднує міждоменні функції: панель керування (GET /admin/dashboard), список замовлень із фільтрацією за статусом, перегляд деталей замовлення, зміну статусу та список користувачів. Це дозволяє повторно використовувати ProductService і CategoriesService без дублювання логіки.

Взаємодія клієнта і сервера реалізована через два інтерфейси: формат відповідей і модель токенів. Формат відповідей має вигляд { success: true, data: T } або { success: false, error: { statusCode, path, timestamp, message } }. Успішні відповіді формує ResponseInterceptor, а помилки – AllExceptionsFilter. На клієнті apiRequest розпаковує data і генерує Error з error.message.

Скріншоти, що супроводжують опис реалізації (рис. 3.2-3.7), показують відповідно головну сторінку, каталог з фільтрами, картку товару, кошик, форму оформлення замовлення і панель керування адміністратора.

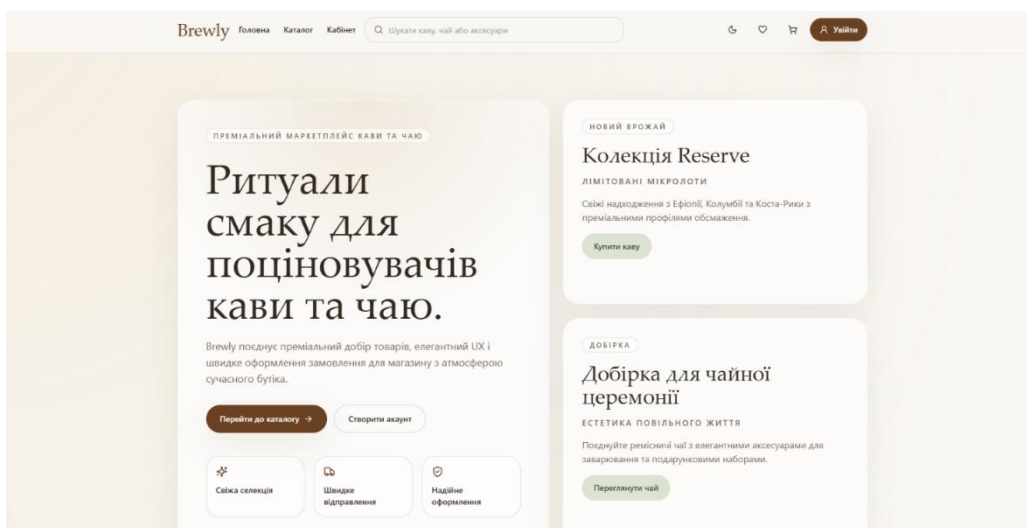


Рисунок 3.2 – Головна сторінка веб-додатку

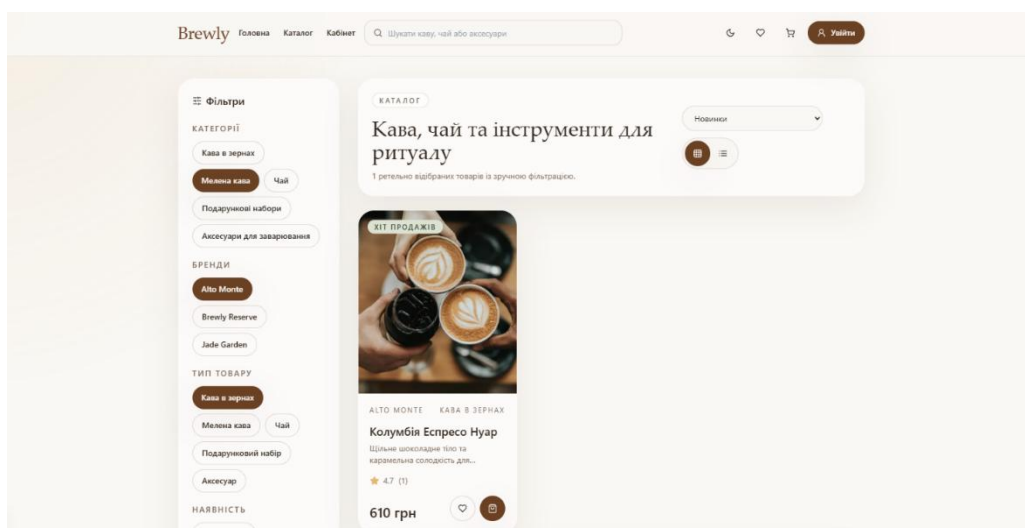


Рисунок 3.3 – Каталог товарів з фільтрами

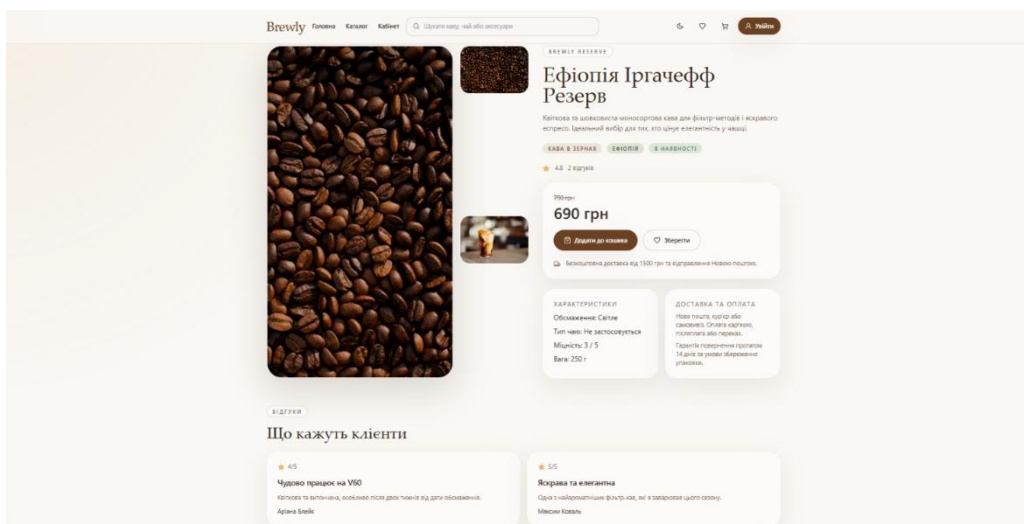


Рисунок 3.4 – Картка товару з рейтингом та відгуками

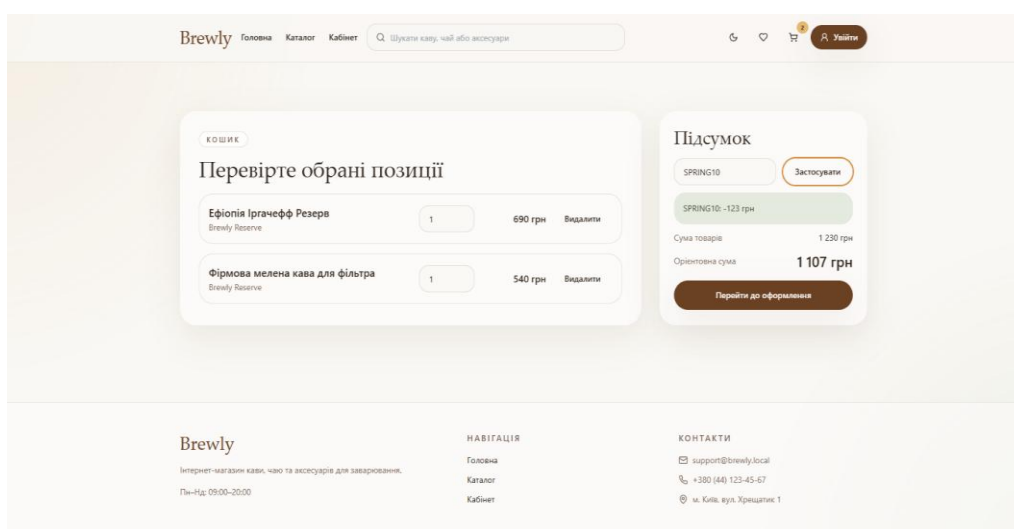


Рисунок 3.5 – Кошик з застосованим промо-кодом SPRING10

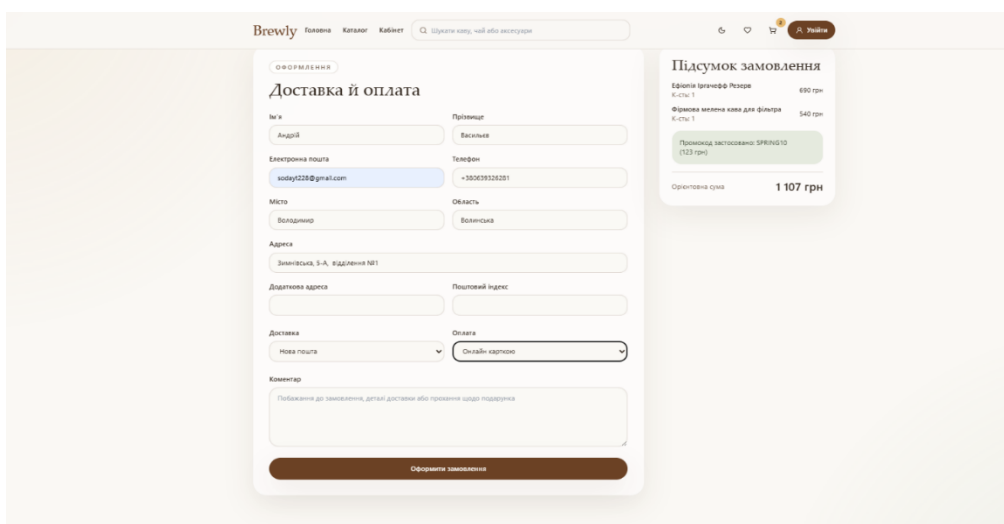


Рисунок 3.6 – Форма оформлення замовлення

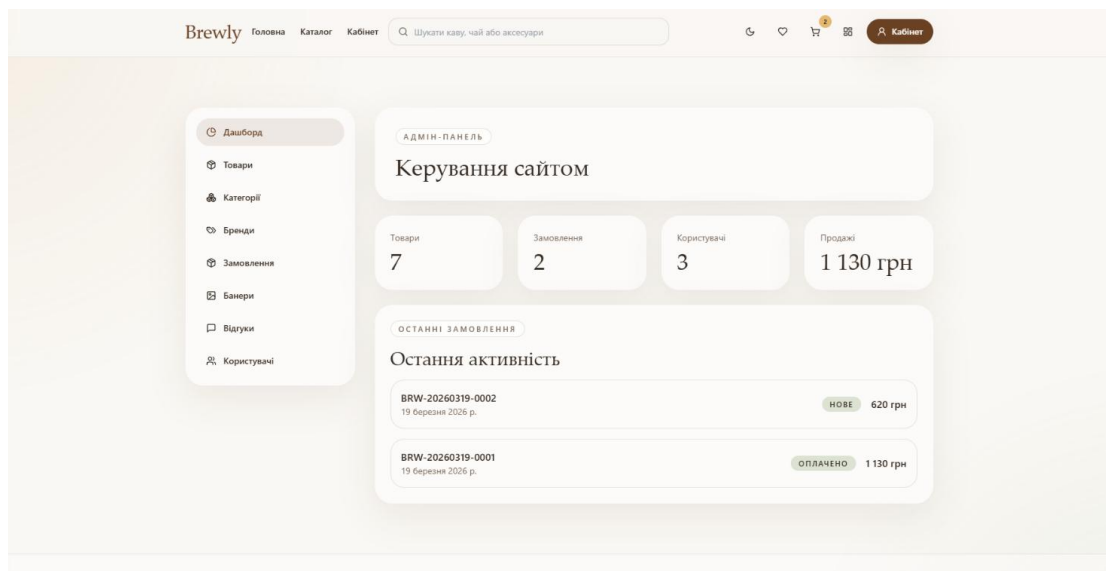


Рисунок 3.7 – Адміністративна панель керування

Решта скріншотів веб-додатку, що ілюструють окремі сторінки та функціональні можливості системи, наведені у додатку Б (рис. Б.1-Б.6).

3.2 Тестування та налагодження інформаційно-комп'ютерної системи

Тестування системи виконувалось на двох рівнях: статичний аналіз вихідного коду через TypeScript-компілятор та ручне тестування за сценаріями використання.

Статичний аналіз запускається командою `npm run typescheck` у кожному з модулів і використовує опцію `tsc --noEmit`. Цей крок виявляє розбіжності між типами клієнта і сервера, що особливо корисно під час змін у Prisma-моделях: автогенерований клієнт `@prisma/client` оновлюється командою `npm run prisma:generate --workspace server`, після чого `typescheck` показує усі місця, де змінилося визначення типу.

Ручне тестування організовано як набір сценаріїв, прив'язаних до функціональних вимог з таблиці 2.2. Кожен сценарій виконується у двох браузерях (Google Chrome і Mozilla Firefox) у режимах світлої і темної теми. Окремо перевіряються виняткові ситуації: оформлення замовлення на велику

кількість позицій, оформлення замовлення з порожнім кошиком, спроба доступу до /admin без ролі адміністратора та повторне оновлення простроченого токена.

Під час реалізації було виявлено проблему з обробкою помилок авторизації зі статусом 401 у клієнтській частині. Початкова версія функції `apiRequest` намагалася оновити пару токенів через маршрут `/auth/refresh` у будь-якому випадку, зокрема тоді, коли токен оновлення також був простроченим. У результаті виникав повторний цикл невдалого запиту, спроби оновлення токенів і нової помилки авторизації. Проблема вирішено шляхом використання прапора `retryOnUnauthorized = true`, який вимикається під час вкладеного виклику, а також явного очищення сесії у функції `clearAuthSession()`, якщо `refreshClientSession()` повертає `null`.

Друга помітна проблема стосувалася валідації номера телефону у Zod-схемі реєстрації. Початкова схема використовувала перевірку `z.string().min(10)`, однак користувач, який вводив номер телефону з пробілами або у форматі `+38 (098) 000-00-00`, міг отримати повідомлення про недостатню довжину навіть тоді, коли кількість цифр була достатньою. Для усунення цієї проблеми рядок було нормалізовано за допомогою трансформації `transform`, після чого перевірялася довжина вже нормалізованого значення. Код Zod-схеми автентифікації наведено у додатку А (ліст. А.3).

Окремо було перевірено адміністративні сценарії. Обліковий запис адміністратора має можливість створювати новий товар, змінювати статус замовлення з NEW на PAID і додавати банер.

План тестування наведено в таблиці 3.1. У ній зазначено рівень тестування, об'єкт перевірки, відповідні файли та поточний статус виконання перевірки.

Таблиця 3.1 – План тестування системи «Brewly»

Рівень	Що перевіряється	Файли / сценарії	Статус
Статичний аналіз	Узгодженість типів клієнта і сервера	<code>npm run typecheck</code> (обидва воркспейси)	Покрито
Lint	Відповідність стилю кодування	<code>npm run lint</code> (eslint у client; tsc у server)	Частково

Продовження таблиці 3.1

Рівень	Що перевіряється	Файли / сценарії	Статус
Smoke (UX)	Перегляд каталогу і фільтрації	/catalog з трьома комбінаціями фільтрів	Покрито
Smoke (Auth)	Реєстрація, вхід, оновлення токена	/auth/register, /auth/login, повторне відкриття після 16 хв	Покрито
Інтеграційний	Оформлення гостьового і авторизованого замовлення	/checkout, два рази з різними даними	Покрито
Інтеграційний	Кошик з промо-кодами SPRING10 і TEATIME150	/cart + ввід промо	Покрито
Інтеграційний	Адміністративний CRUD продуктів	/admin/products, /admin/products/new	Покрито
Безпековий	Спроба доступу до /admin/* без ролі ADMIN	редирект на /; перевірка cookie	Покрито

Мінімальні системні вимоги для тестування та експлуатації веб-додатку наведено в таблиці 3.2. Повна ручна перевірка основних сценаріїв роботи системи займає в середньому 35-45 хвилин для одного виконавця. Її доцільно проводити перед внесенням нових функцій до основної гілки репозиторію, щоб не порушити вже реалізовану функціональність.

Таблиця 3.2 – Мінімальні системні вимоги

Категорія	Параметр	Мінімум	Рекомендовано
Робоча станція розробника	OS	Windows 10 / macOS 13 / Ubuntu 22.04	Windows 11 / macOS 15 / Ubuntu 24.04
Робоча станція розробника	Node.js	22.x LTS (Long-Term Support)	22.x LTS остання
Робоча станція розробника	RAM	8 ГБ	16 ГБ
Робоча станція розробника	Вільне місце	5 ГБ	10 ГБ
Сервер	OS	Linux з kernel 5.10+	Ubuntu 24.04 LTS
Сервер	vCPU	2	4
Сервер	СУБД	MySQL 8.4 (контейнер або хост)	MySQL 8.4

Продовження таблиці 3.2

Категорія	Параметр	Мінімум	Рекомендовано
Сервер	RAM	4 ГБ	8 ГБ
Сервер	Дисковий простір	20 ГБ	50 ГБ SSD
Клієнтський пристрій	Браузер	Google Chrome 120+ / Mozilla Firefox 121+ / Safari 17+	Google Chrome 130+
Клієнтський пристрій	Роздільність	360×640 (mobile)	1920×1080
Клієнтський пристрій	З'єднання	3G+	4G+

3.3 Розробка бази даних

База даних веб-додатку побудована на MySQL 8.4 та описана у вигляді схеми Prisma у файлі `server/prisma/schema.prisma`. Загалом у схемі визначено 12 сутностей, які охоплюють користувачів, адреси, категорії, бренди, товари, зображення товарів, списки бажань, відгуки, замовлення та банери для акційних пропозицій. Крім цього, використано 9 перелічуваних типів для опису ролей користувачів, типів товарів, рівнів обсмаження, стану наявності, статусів замовлення, способів доставки, способів оплати та статусів банерів.

Ідентифікатори записів у базі даних генеруються за допомогою функції `cuid()`, яка використовується в директиві `@default(cuid())` у схемі Prisma. Це дає змогу застосовувати такі ідентифікатори в URL-адресах без розкриття послідовності створення записів. Основні сутності бази даних наведено в таблиці 3.3.

Таблиця 3.3 – Основні сутності бази даних

Сутність	Призначення	Ключові поля	Зауваження
User	Користувачі системи	id, email, passwordHash, role, refreshTokenHash	Логічне видалення за допомогою поля <code>deletedAt</code>
Address	Адреси доставки	userId, city, line1, isDefault	Каскадне видалення разом із записом користувача

Продовження таблиці 3.3

Brand	Бренди-постачальники	slug, originCountry	Логічне видалення записів
Category	Категорії каталогу	slug, parentId	Підтримка ієрархічної структури категорій
Product	Товари	sku, slug, price, productType, teaType, roastLevel	Логічне видалення та збереження узагальненої оцінки товару
ProductImage	Зображення товару	productId, url, sortOrder, isPrimary	Каскадне видалення разом із товаром
Wishlist	Список бажань	userId (унікальний)	Один список для одного користувача
WishlistItem	Елемент списку бажань	wishlistId, productId	Унікальна пара списку бажань і товару
Review	Відгуки про товари	userId, productId, rating, comment	Один відгук користувача для одного товару
Order	Замовлення	number, status, total, deliveryMethod, paymentMethod	Підтримка гостьових і авторизованих замовлень
OrderItem	Рядок замовлення	productId, productName, unitPrice, lineTotal	Збереження даних товару на момент оформлення замовлення
PromoBanner	Промо-банери на головній	imageUrl, status, startsAt, endsAt	Відображення з урахуванням дат початку та завершення

Сутність User зберігає основні дані користувача, хеш пароля, хеш токена оновлення та поле role, яке використовується для розмежування доступу. Зв'язки з моделями Address, Order, Wishlist і Review дають змогу зберігати пов'язані з користувачем адреси доставки, замовлення, список бажань і відгуки. Логічне видалення реалізоване через поле deletedAt: запис позначається як видалений, але фізично залишається в базі даних. Це потрібно для збереження зв'язків з історією замовлень.

Каталог товарів сформовано трьома основними моделями: Category, Brand і Product. Модель Category підтримує ієрархічну структуру категорій через поле parentId, а модель Brand містить інформацію про бренд і країну походження.

Модель Product має набір атрибутів, характерних для товарів кавово-чайного сегмента: productType, teaType, roastLevel, strength, weightGrams, brewingMethod і specifications. Для збереження зображень товарів використано окрему модель ProductImage, у якій передбачено поля isPrimary і sortOrder для визначення головного зображення та порядку відображення в галереї.

Замовлення описується моделями Order та OrderItem. Модель Order зберігає загальні суми замовлення, зокрема subtotal, shippingTotal, discountTotal і total, а також дані доставки, спосіб доставки та спосіб оплати. Модель OrderItem містить зафіксовані на момент оформлення замовлення значення productName, productSlug, unitPrice і lineTotal. Завдяки цьому історія замовлень залишається коректною навіть після зміни ціни товару або його видалення. У разі видалення товару поле productId у позиції замовлення набуває значення NULL через правило onDelete: SetNull.

Список бажань реалізовано за допомогою моделей Wishlist і WishlistItem. Для кожного користувача створюється один список бажань, а унікальне обмеження для пари wishlistId і productId не дозволяє додати один і той самий товар двічі. Подібний підхід застосовано й у моделі Review, де унікальність пари userId і productId обмежує користувача одним відгуком для одного товару.

Окрему частину структури бази даних становить сутність PromoBanner, призначена для збереження банерів головної сторінки. Вона містить дані про зображення, посилання для переходу, текст кнопки, період відображення та статус банера. Завдяки цьому адміністратор може керувати інформаційними або акційними матеріалами на головній сторінці без внесення змін до програмного коду.

Індекси задано в ключових місцях схеми бази даних. У моделі User вони використовуються для пошуку за роллю та станом активності, у моделі Product – для вибірки за категорією, брендом, типом товару, наявністю та ознаками «нове», «бестселер» і «особливе». У моделі Order індекси застосовано для пошуку за статусом, датою створення та користувачем. Це прискорює типові

запити каталогу й адміністративної панелі. Повний перелік індексів наведено у фрагменті схеми в лістингу 3.6.

Лістинг 3.6 – Фрагмент схеми Prisma з моделлю Product та індексами

```

model Product {
  id          String      @id @default(cuid())
  sku         String      @unique
  slug        String      @unique
  name        String
  description  String
  price        Decimal     @db.Decimal(10, 2)
  oldPrice     Decimal?    @db.Decimal(10, 2)
  productType ProductType
  teaType      TeaType     @default(NOT_APPLICABLE)
  roastLevel   RoastLevel  @default(NOT_APPLICABLE)
  strength     Int?
  weightGrams  Int?
  stock         Int         @default(0)
  stockStatus  StockStatus @default(IN_STOCK)
  specifications Json?
  isFeatured   Boolean     @default(false)
  isBestseller Boolean     @default(false)
  isNew        Boolean     @default(false)
  isPublished  Boolean     @default(true)
  ratingAverage Decimal    @default(0) @db.Decimal(3, 2)
  reviewCount  Int         @default(0)
  categoryId   String
  brandId      String
  category     Category    @relation(fields: [categoryId],
references: [id])
  brand        Brand       @relation(fields: [brandId], references:
[id])
  images       ProductImage[]
  orderItems   OrderItem[]
  reviews      Review[]
  wishlistItems WishlistItem[]
  createdAt    DateTime    @default(now())
  updatedAt    DateTime    @updatedAt
  deletedAt    DateTime?

  @@index([categoryId])
  @@index([brandId])
  @@index([productType, stockStatus])
  @@index([isFeatured, isBestseller, isNew])
  @@index([createdAt])
}

```

кінець лістингу 3.6

Структуру взаємозв'язків між сутностями ілюструє рисунок 3.8.

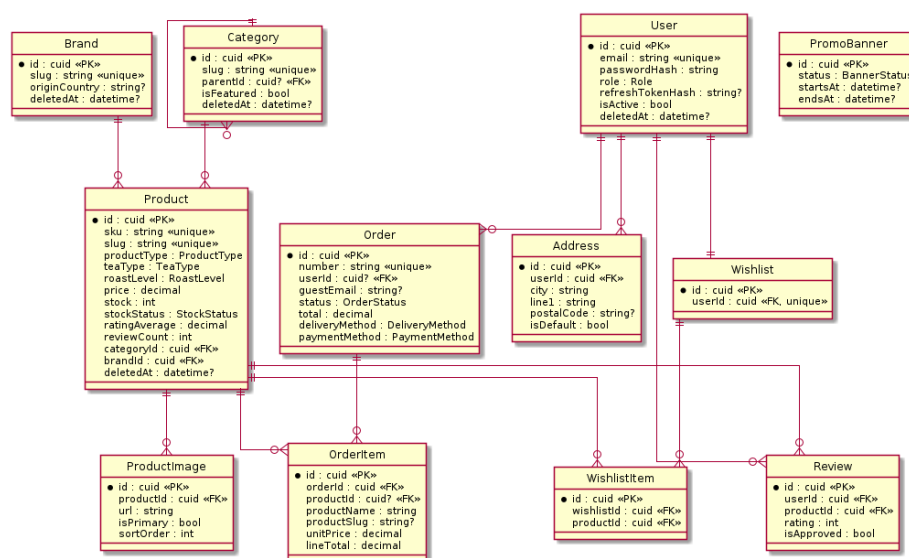


Рисунок 3.8 – Концептуальна ERD-діаграма бази даних

3.4 Захист інформаційної системи

Захист інформаційної системи «Brewly» побудовано згідно з рекомендаціями OWASP ASVS і охоплює п'ять рівнів: керування обліковими записами, керування сесіями, валідація вхідних даних, контроль доступу, обробка помилок [20].

На рівні облікових записів паролі ніколи не зберігаються у відкритому вигляді. Метод register у AuthService обчислює bcrypt-хеш з кількістю раундів, що задається змінною оточення BCRYPT_ROUNDS (за замовчуванням 10). Поле passwordHash у моделі User має тип String без додаткових індексів – це знижує ризик timing-атак при пошуку за хешем. На реєстрації email перетворюється до нижнього регістру, що запобігає створенню двох облікових записів з однаковим email у різному регістрі. Код AuthService наведений у додатку А (ліст. А.4).

Керування сесіями реалізовано через пару JWT access + refresh. Access-токен має короткий час життя (15 хвилин за замовчуванням) і передається у заголовку Authorization. Refresh-токен зберігається у вигляді bcrypt-хешу в полі

User.refreshTokenHash (його очищає метод logout) і використовується для отримання нової пари токенів через POST /auth/refresh. У cookie клієнта (brewly_access_token, brewly_refresh_token, brewly_role) встановлюються прапори SameSite=Lax (та Secure для виробничого середовища), щоб запобігти CSRF (Cross-Site Request Forgery, міжсайтовій підробці запитів).

Валідація вхідних даних виконується у двох шарах. На клієнті – Zod-схеми (loginSchema, registerSchema, checkoutSchema) у поєднанні з react-hook-form. На сервері – глобальний ValidationPipe з опціями whitelist: true, forbidNonWhitelisted: true, transform: true. whitelist відсікає від тіла запиту усі поля, не оголошені у DTO; forbidNonWhitelisted повертає 400, якщо такі поля присутні; transform перетворює рядки на enum/number відповідно до типів. Сама валідація реалізована через анотації class-validator: @IsEmail, @IsString, @MinLength, @IsEnum, @IsArray, @ValidateNested.

Контроль доступу побудовано на парі вартових JwtAuthGuard і RolesGuard. JwtAuthGuard зареєстровано глобально як APP_GUARD, тому кожен запит за замовчуванням потребує валідного access-токена. Контролер або метод може перейти у публічний режим через декоратор @Public(), який встановлює метадані IS_PUBLIC_KEY. RolesGuard додатково перевіряє роль, прочитану з request.user.role, і порівнює зі списком, заданим декоратором @Roles(Role.ADMIN).

Обробка помилок централізована у фільтрі AllExceptionsFilter. Будь-яке необроблене виключення перетворюється на стандартний об'єкт { success: false, error: { statusCode, path, timestamp, message } }, після чого повертається клієнту. Це усуває потребу дублювати catch-блоки у контролерах і знижує ризик випадкового розкриття stack trace у виробничому середовищі. Для серверних помилок 5xx виконується додаткове логування.

CORS (Cross-Origin Resource Sharing, спільне використання ресурсів між різними джерелами) налаштовується у main.ts через явний allowlist з єдиного origin – значення CLIENT_URL (наприклад, http://localhost:3000 у середовищі розробки або фактичний домен у робочому середовищі). Це блокує запити з

інших доменів без явного дозволу. Якщо у майбутньому додається мобільний застосунок або сторонній партнерський сайт, allowlist розширюється точково, а не відкривається на *.

Окремою точкою захисту є файл client/proxy.ts. Він не замінює серверну авторизацію, а лише блокує перехід на сторінки /admin/* користувачам без ролі ADMIN ще до завантаження компонентів сторінки. Це поліпшує користувацький досвід: користувач отримує перенаправлення миттєво, без короткочасного відображення адмін-панелі. Усі реальні перевірки знов виконуються на сервері при кожному API-виклику.

Зведена відповідність вимог OWASP ASVS і реалізації у проєкті подано у таблиці 3.4.

Таблиця 3.4 – Відповідність захисту системи вимогам OWASP ASVS

Категорія	Вимога ASVS (рівень 1)	Реалізація у «Brewly»	Артефакт
V2 Auth	Паролі перевіряються через сильний односторонній алгоритм	bcrypt з налаштовуваними раундами	auth.service.ts hashValue
V2 Auth	Виявлення повторного використання refresh-токена	Зберігання bcrypt(refreshToken) у User.refreshTokenHash	auth.service.ts storeRefreshToken
V3 Sessions	Час життя сесійного токена ≤ 30 хв	access-токен 15 хв	ENV JWT_ACCESS_EXPIRES=15m
V4 Access Control	Серверна перевірка ролі на кожному критичному ендпоінті	RolesGuard + @Roles(Role.ADMIN)	roles.guard.ts
V5 Validation	Суворі валідація усіх вхідних даних	ValidationPipe({ whitelist, forbidNonWhitelisted })	main.ts
V7 Error Handling	Уніфікований формат помилок без stack trace у production	AllExceptionsFilter	common/filters/
V8 Data Protection	Маскування полів паролів і refresh-хешів у відповіді	serializeUser()	auth.service.ts
V14 Config	Секрети – лише з ENV, валідація на старті	env.validation.ts	config/

3.5 Розробка документації для програмного продукту

Для веб-додатку «Brewly» підготовлено кілька видів документації: інструкцію з локального запуску, опис мінімальних системних вимог, API-довідку, користувацькі підказки в інтерфейсі та адміністративну інструкцію.

Інструкція з локального запуску призначена для розробника або технічного спеціаліста, який розгортає систему у середовищі розробки. У ній описано послідовність підготовки проєкту: отримання вихідного коду, встановлення залежностей, налаштування змінних оточення, підготовка бази даних, генерація Prisma-клієнта, застосування схеми бази даних та запуск клієнтської і серверної частин. Такий опис дозволяє відтворити робоче середовище без додаткового пояснення з боку автора проєкту.

Мінімальні системні вимоги поділено на три категорії: робоча станція розробника, серверне середовище та клієнтський пристрій. Для розробника важливими є наявність актуальної версії Node.js, доступ до MySQL, достатній обсяг оперативної пам'яті та вільного місця на диску. Серверне середовище повинно забезпечувати стабільну роботу серверної частини, бази даних і обробку запитів користувачів. Клієнтський пристрій має підтримувати сучасний браузер, адаптивну верстку, CSS Grid і CSS Custom Properties.

Користувацька довідка реалізована не як окремий документ, а як частина інтерфейсу веб-додатку. Для форм передбачено підписи полів, плейсхолдери та додаткові підказки для складніших елементів введення. Повідомлення про успішні або помилкові дії виводяться через систему сповіщень, що допомагає користувачу швидко зрозуміти результат виконаної операції.

Адміністративна документація описує основні сценарії роботи з панеллю керування. Для адміністратора передбачено послідовність дій після входу в систему: перегляд загальних показників на сторінці адмін-панелі, перевірка асортименту товарів, робота із замовленнями, керування категоріями, брендами та промо-банерами. У документації також зазначено порядок використання тестових облікових записів для демонстраційного середовища.

Окремо описано порядок роботи зі схемою бази даних. Для цього у серверній частині передбачено команди генерації Prisma-клієнта, застосування схеми до бази даних і заповнення початковими тестовими даними. Такий підхід дозволяє підтримувати узгодженість між моделями у файлі `schema.prisma` та реальною структурою бази даних. Команди, необхідні для підготовки локального середовища, наведено у лістингу 3.7.

Лістинг 3.7 – Команди розгортання локального середовища

```
# 1. Клонування репозиторію та перехід у каталог проєкту
git clone <repository-url> brewly
cd brewly

# 2. Встановлення залежностей
npm install

# 3. Підготовка змінних оточення
cp server/.env.example server/.env
cp client/.env.example client/.env.local

# 4. Генерація Prisma-клієнта
npm run prisma:generate --workspace server

# 5. Застосування схеми бази даних
npx prisma db push --schema server/prisma/schema.prisma

# 6. Заповнення бази даних початковими даними
npm run seed

# 7. Запуск клієнтської та серверної частин
npm run dev
```

кінець лістингу 3.7

Більш детальна інструкція для розгортання веб-додатку наведена у додатку В (табл. В.1).

Висновки до розділу 3

У третьому розділі описано реалізацію системи у термінах конкретних файлів і модулів. Клієнт побудовано на Next.js 16 з App Router та feature-

сегментованою структурою, сервер – на NestJS 11 з дванадцятьма доменними модулями і централізованим стеком безпеки. Контракт відповідей API уніфіковано через `ResponseInterceptor` і `AllExceptionsFilter`.

Схема бази даних охоплює одинадцять моделей з підтримкою м'якого видалення і денормалізованих знімків у `OrderItem`. Індеси проставлено в типових точках адмін-фільтра і каталогу. Підсистема безпеки відповідає вимогам OWASP ASVS рівня 1: bcrypt-паролі, JWT-сесії з коротким часом життя, валідація whitelist, серверне розмежування доступу за допомогою `RolesGuard`.

Тестування на поточному етапі – переважно ручне, статичний аналіз закриває узгодженість типів. Це залишає простір для розширення проєкту автоматизованими тестами (Jest для backend, Vitest або Playwright для frontend). Документація реалізована у форматі README.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено прототип веб-додатка для продажу кави та чаю «Brewly», який поєднує публічну вітрину, особистий кабінет покупця та адміністративну панель. Система реалізована з використанням сучасного стеку вебтехнологій: Next.js 16, NestJS 11, Prisma 6 і MySQL 8.4.

Проаналізовано предметну область онлайн-торгівлі кавою та чаєм, досліджено особливості сучасних інтернет-магазинів і порівняно існуючі рішення. Виявлено типові обмеження SaaS-конструкторів та шаблонних платформ, зокрема недостатню гнучкість адміністративної частини, складність адаптації бізнес-логіки та обмежені можливості масштабування. На основі проведеного аналізу обґрунтовано доцільність створення власного програмного рішення.

Визначено функціональні та нефункціональні вимоги до інформаційної системи електронної комерції. Виконано розподіл можливостей системи за трьома ролями користувачів: анонімний відвідувач, зареєстрований покупець і адміністратор. Сформовано перелік основних функцій, що охоплюють роботу з каталогом, кошиком, замовленнями, відгуками та адміністративним керуванням. Окремо визначено вимоги до продуктивності, безпеки, масштабованості та зручності використання.

Обґрунтовано вибір технологічного стека для реалізації системи. Для клієнтської частини обрано Next.js 16, який забезпечує підтримку серверного рендерингу та сучасну маршрутизацію через App Router. Серверну частину реалізовано на NestJS 11 завдяки його модульній архітектурі та зручній інтеграції з REST API. Для взаємодії з базою даних використано Prisma ORM і СУБД MySQL 8.4. Також використано бібліотеки валідації та JWT-автентифікації для забезпечення безпечної взаємодії користувачів із системою.

Спроектровано схему бази даних, яка підтримує основні бізнес-сутності системи. Реалізовано структуру каталогу, що охоплює продукти, категорії,

бренди та зображення. Передбачено підтримку замовлень у гостьовому та авторизованому режимах, а також механізми роботи з відгуками, списком бажань, промо-банерами та ролями доступу. У структурі бази даних використано унікальні обмеження, індекси та механізм м'якого видалення для окремих сутностей.

Реалізовано клієнтську частину веб-додатка на Next.js 16. Розроблено публічний каталог товарів, сторінку товару, кошик, форму оформлення замовлення, особистий кабінет користувача та адміністративну панель. Клієнтська частина побудована на основі функціонально-сегментованої архітектури, а також містить механізми захисту маршрутів і локального збереження стану кошика.

Реалізовано серверну частину системи на NestJS 11. Архітектура побудована з використанням дванадцяти доменних модулів і REST API для взаємодії з клієнтською частиною. Реалізовано JWT-автентифікацію, рольове розмежування доступу та централізовану обробку помилок. Для перевірки вхідних даних використано глобальний ValidationPipe, що забезпечує додатковий рівень надійності та безпеки системи.

Виконано тестування основних сценаріїв роботи системи та описано ручні тест-кейси. Проведено перевірку процесів авторизації, роботи каталогу, кошика та оформлення замовлення. Також виконано статичний аналіз коду за допомогою tsc --noEmit. У процесі тестування виявлено та усунено дефекти, пов'язані з оновленням JWT-токенів і валідацією телефонного номера.

Підготовлено документацію для розгортання та використання системи. Сформовано інструкції для локального та контейнеризованого запуску застосунку, визначено мінімальні системні вимоги та описано основні сценарії використання для адміністратора й розробника. Документація README охоплює процес налаштування середовища, запуску проєкту та роботи з основними функціями системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Baymard Institute. E-Commerce UX Research Methodology. URL: <https://baymard.com/research> (дата звернення: 15.02.2026).
2. React Documentation. Meta Platforms. URL: <https://react.dev> (дата звернення: 18.02.2026).
3. React Hook Form Documentation. URL: <https://react-hook-form.com/docs> (дата звернення: 20.02.2026).
4. Core Web Vitals. web.dev. Google Developers. URL: <https://nitropack.io/blog/core-web-vitals/> (дата звернення: 23.02.2026).
5. Gravitee – REST API Tutorial. URL: <https://www.gravitee.io/blog/rest-api-tutorial?> (дата звернення: 27.02.2026).
6. GeeksforGeeks. Password Hashing with bcrypt in Node.js, 2023. URL: <https://www.geeksforgeeks.org/node-js/password-encryption-in-node-js-using-bcryptjs-module/> (дата звернення: 28.02.2026).
7. Sling Academy. Authentication and Authorization in Express.js with JWT, 2023. URL: <https://www.slingacademy.com/article/authentication-authorization-expressjs-jwt/?> (дата звернення: 08.03.2026).
8. TechTarget – What is Role-Based Access Control (RBAC)? URL: <https://www.techtarget.com/searchsecurity/definition/role-based-access-control-RBAC?> (дата звернення: 14.03.2026).
9. Web Storage API. MDN Web Docs. Mozilla Foundation. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API (дата звернення: 16.03.2026).
10. class-validator. Decorator-based property validation for classes. GitHub repository. URL: <https://github.com/typestack/class-validator> (дата звернення: 20.03.2026).
11. Next.js Documentation. URL: <https://nextjs.org/docs> (дата звернення: 25.03.2026).

12. NestJS Documentation. URL: <https://docs.nestjs.com> (дата звернення: 27.03.2026).
13. Passport.js Documentation. Jared Hanson. URL: <https://www.passportjs.org/docs/> (дата звернення: 30.03.2026).
14. Prisma ORM Documentation. Prisma Data Inc. URL: <https://www.prisma.io/docs> (дата звернення: 04.04.2026).
15. TypeScript Handbook. Microsoft. URL: <https://www.typescriptlang.org/docs/handbook/intro.html> (дата звернення: 10.04.2026).
16. MySQL 8.4 Reference Manual. Oracle Corporation. URL: <https://dev.mysql.com/doc/refman/8.4/en/> (дата звернення: 13.04.2026).
17. Zod Documentation. Colin McDonnell. URL: <https://zod.dev> (дата звернення: 17.04.2026).
18. TanStack Query v5 Documentation. TanStack. URL: <https://tanstack.com/query/v5/docs> (дата звернення: 20.04.2026).
19. Tailwind CSS Documentation. Tailwind Labs. URL: <https://tailwindcss.com/docs> (дата звернення: 21.04.2026).
20. OWASP Application Security Verification Standard 4.0.3. OWASP Foundation. URL: <https://owasp.org/www-project-application-security-verification-standard/> (дата звернення: 22.04.2026).

ДОДАТКИ

ДОДАТОК А

Лістинги основних програмних модулів

У додатку А наведено повні версії лістингів.

Лістинг А.1 – Middleware-захисту /account і /admin маршрутів

```
import type { NextRequest } from 'next/server';
import { NextResponse } from 'next/server';

const ACCESS_TOKEN_COOKIE = 'brewly_access_token';
const ROLE_COOKIE = 'brewly_role';

export function proxy(request: NextRequest) {
  const { pathname } = request.nextUrl;
  const accessToken = request.cookies.get(ACCESS_TOKEN_COOKIE)?.value;
  const role = request.cookies.get(ROLE_COOKIE)?.value;

  if (pathname.startsWith('/account') && !accessToken) {
    const loginUrl = new URL('/auth/login', request.url);
    loginUrl.searchParams.set('next', pathname);
    return NextResponse.redirect(loginUrl);
  }

  if (pathname.startsWith('/admin')) {
    if (!accessToken) {
      const loginUrl = new URL('/auth/login', request.url);
      loginUrl.searchParams.set('next', pathname);
      return NextResponse.redirect(loginUrl);
    }
    if (role !== 'ADMIN') {
      return NextResponse.redirect(new URL('/', request.url));
    }
  }
  return NextResponse.next();
}

export const config = {
  matcher: ['/account/:path*', '/admin/:path*'],
};
```

кінець лістингу А.1

Лістинг A.2 – Клієнтський fetch-обгортувач з автоматичним refresh

```

import { clearAuthSession, getAccessToken, getRefreshToken,
updateSessionTokens } from '@lib/auth';
import type { ApiEnvelope } from '@types/api';

const API_URL = process.env.NEXT_PUBLIC_API_URL ??
'http://localhost:4000/api/v1';
const INTERNAL_API_URL = process.env.INTERNAL_API_URL ?? API_URL;

type RequestOptions = RequestInit & { token?: string;
retryOnUnauthorized?: boolean };

async function refreshClientSession() {
  const refreshToken = getRefreshToken();
  if (!refreshToken) return null;

  const response = await fetch(`${API_URL}/auth/refresh`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ refreshToken }),
  });
  const payload = (await response.json()) as ApiEnvelope<{
    tokens: { accessToken: string; refreshToken: string }
  >;
  if (!response.ok || !payload.success) {
    clearAuthSession();
    return null;
  }
  updateSessionTokens(payload.data.tokens.accessToken,
payload.data.tokens.refreshToken);
  return payload.data.tokens.accessToken;
}

export async function apiRequest<T>(path: string, options: RequestOptions
= {}): Promise<T> {
  const { token, retryOnUnauthorized = true, headers, body, ...rest } =
options;
  const accessToken = token ?? getAccessToken();

  const response = await fetch(`${API_URL}${path}`, {
    ...rest,
    headers: {
      'Content-Type': 'application/json',
      ...(accessToken ? { Authorization: `Bearer ${accessToken}` } : {}),
      ...headers,
    },
    body,
    cache: 'no-store',
  });
}

```

```

    if (response.status === 401 && retryOnUnauthorized && typeof window !==
'undefined') {
        const renewed = await refreshClientSession();
        if (renewed) {
            return apiRequest<T>(path, { ...options, token: renewed,
retryOnUnauthorized: false });
        }
    }

    const payload = (await response.json()) as ApiEnvelope<T>;
    if (!response.ok || !payload.success) {
        throw new Error(payload.error?.message ?? 'Не вдалося виконати
запит');
    }
    return payload.data;
}

```

кінець лістингу A.2

Лістинг A.3 – Zod-схеми клієнтських форм автентифікації

```

import { z } from 'zod';

export const loginSchema = z.object({
    email: z.string().email('Введіть коректний email'),
    password: z.string().min(8, 'Мінімум 8 символів'),
});

export const registerSchema = loginSchema.extend({
    firstName: z.string().min(2, 'Мінімум 2 символи'),
    lastName: z.string().min(2, 'Мінімум 2 символи'),
    phone: z.string().min(10, 'Введіть коректний номер телефону'),
});

export type LoginSchema = z.infer<typeof loginSchema>;
export type RegisterSchema = z.infer<typeof registerSchema>;

```

кінець лістингу A.3

Лістинг A.4 – Повний код AuthService

```

import {
    ConflictException,
    Injectable,
    UnauthorizedException,
} from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { JwtService } from '@nestjs/jwt';
import { Role, User } from '@prisma/client';

```

```

import * as bcrypt from 'bcrypt';
import type { SignOptions } from 'jsonwebtoken';
import { PrismaService } from '../common/prisma.service';
import { RegisterDto } from '../dto/register.dto';
import { LoginDto } from '../dto/login.dto';
import { JwtPayload } from '../common/types/jwt-payload.type';

type AuthTokens = { accessToken: string; refreshToken: string };

@Injectable()
export class AuthService {
  constructor(
    private readonly prisma: PrismaService,
    private readonly jwtService: JwtService,
    private readonly configService: ConfigService,
  ) {}

  async register(dto: RegisterDto) {
    const existingUser = await this.prisma.user.findUnique({
      where: { email: dto.email.toLowerCase() },
    });
    if (existingUser) throw new ConflictException('User with this email already exists');

    const passwordHash = await this.hashValue(dto.password);
    const user = await this.prisma.user.create({
      data: {
        firstName: dto.firstName,
        lastName: dto.lastName,
        email: dto.email.toLowerCase(),
        phone: dto.phone,
        passwordHash,
      },
    });
    const tokens = await this.generateTokens(user);
    await this.storeRefreshToken(user.id, tokens.refreshToken);
    return { user: this.serializeUser(user), tokens };
  }

  async login(dto: LoginDto) {
    const user = await this.prisma.user.findUnique({
      where: { email: dto.email.toLowerCase() },
    });
    if (!user || !user.isActive) throw new UnauthorizedException('Invalid credentials');

    const passwordsMatch = await bcrypt.compare(dto.password, user.passwordHash);
    if (!passwordsMatch) throw new UnauthorizedException('Invalid credentials');
  }
}

```

```

    const tokens = await this.generateTokens(user);
    await this.storeRefreshToken(user.id, tokens.refreshToken);
    return { user: this.serializeUser(user), tokens };
}

async refresh(userId: string, refreshToken: string) {
    const user = await this.prisma.user.findUnique({ where: { id:
userId } });
    if (!user || !user.refreshTokenHash) throw new
UnauthorizedException('Invalid refresh token');

    const ok = await bcrypt.compare(refreshToken, user.refreshTokenHash);
    if (!ok) throw new UnauthorizedException('Invalid refresh token');

    const tokens = await this.generateTokens(user);
    await this.storeRefreshToken(user.id, tokens.refreshToken);
    return { user: this.serializeUser(user), tokens };
}

async logout(userId: string) {
    await this.prisma.user.update({
        where: { id: userId },
        data: { refreshTokenHash: null },
    });
    return { success: true, message: 'Logged out successfully' };
}

async me(userId: string) {
    const user = await this.prisma.user.findUniqueOrThrow({
        where: { id: userId },
        include: { addresses: true },
    });
    return this.serializeUser(user);
}

private async generateTokens(user: User): Promise<AuthTokens> {
    const payload: JwtPayload = { sub: user.id, email: user.email, role:
user.role };
    const accessExpiresIn =
this.configService.getOrThrow<string>('auth.accessExpires') as
SignOptions['expiresIn'];
    const refreshExpiresIn =
this.configService.getOrThrow<string>('auth.refreshExpires') as
SignOptions['expiresIn'];

    const [accessToken, refreshToken] = await Promise.all([
        this.jwtService.signAsync(payload, {
            secret:
this.configService.getOrThrow<string>('auth.accessSecret'),

```

```

        expiresIn: accessExpiresIn,
    )),
    this.jwtService.signAsync(payload, {
        secret:
this.configService.getOrThrow<string>('auth.refreshSecret'),
        expiresIn: refreshExpiresIn,
    )),
    ]));
    return { accessToken, refreshToken };
}

private async storeRefreshToken(userId: string, refreshToken: string) {
    const refreshTokenHash = await this.hashValue(refreshToken);
    await this.prisma.user.update({ where: { id: userId }, data:
{ refreshTokenHash } });
}

private async hashValue(value: string) {
    const rounds = this.configService.get<number>('auth.bcryptRounds') ??
10;
    return bcrypt.hash(value, rounds);
}

private serializeUser(user: User & { addresses?: unknown }) {
    const { passwordHash: _ph, refreshTokenHash: _rh, ...safe } = user;
    return safe;
}
}

```

кінець лістингу А.4

ДОДАТОК Б

Додаткові скріншоти інтерфейсу

У додатку Б наведено додаткові скріншоти інтерфейсу.

РЕЄСТРАЦІЯ

Створіть свій преміальний акаунт

Зберігайте товари в обраному, повторюйте замовлення в один клік і тримайте дані доставки під рукою.

Ім'я: Павло

Прізвище: Коваль

Телефон: +380985675692

Електронна пошта: admin@brewly.dev

Пароль: a

Мінімум 8 символів

Створити акаунт

Уже маєте акаунт? [Увійти](#)

Рисунок Б.1 – Сторінка реєстрації зі станом валідаційних помилок

Brewly Головна Каталог Кабінет

Шукати каву, чай або аксесуари

Головна Кабінет

Профіль

Замовлення

Обране

Адреси

АКАУНТ

Огляд профілю

З нами з 19 березня 2026 р.

Вийти

Оформлено замовлень: 0

Збережені адреси: 1

Товарів в обраному: 0

РЕДАГУВАННЯ ПРОФІЛЮ

Особисті дані

Ім'я: Андрій

Прізвище: Васильєв

Телефон: +380671112233

Посилання на аватар

Зберегти зміни

Рисунок Б.2 – Особистий кабінет покупця з блоками профілю і навігацією

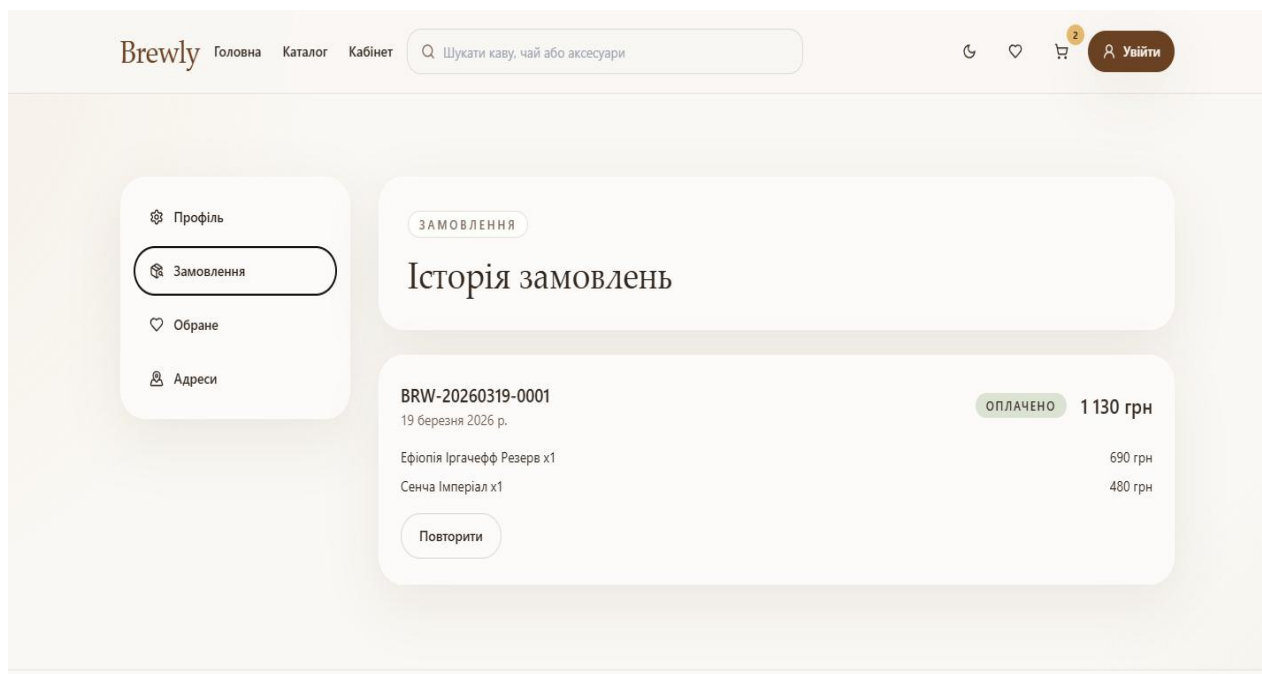


Рисунок Б.3 – Список замовлень з прикладом фактично оформленого замовлення

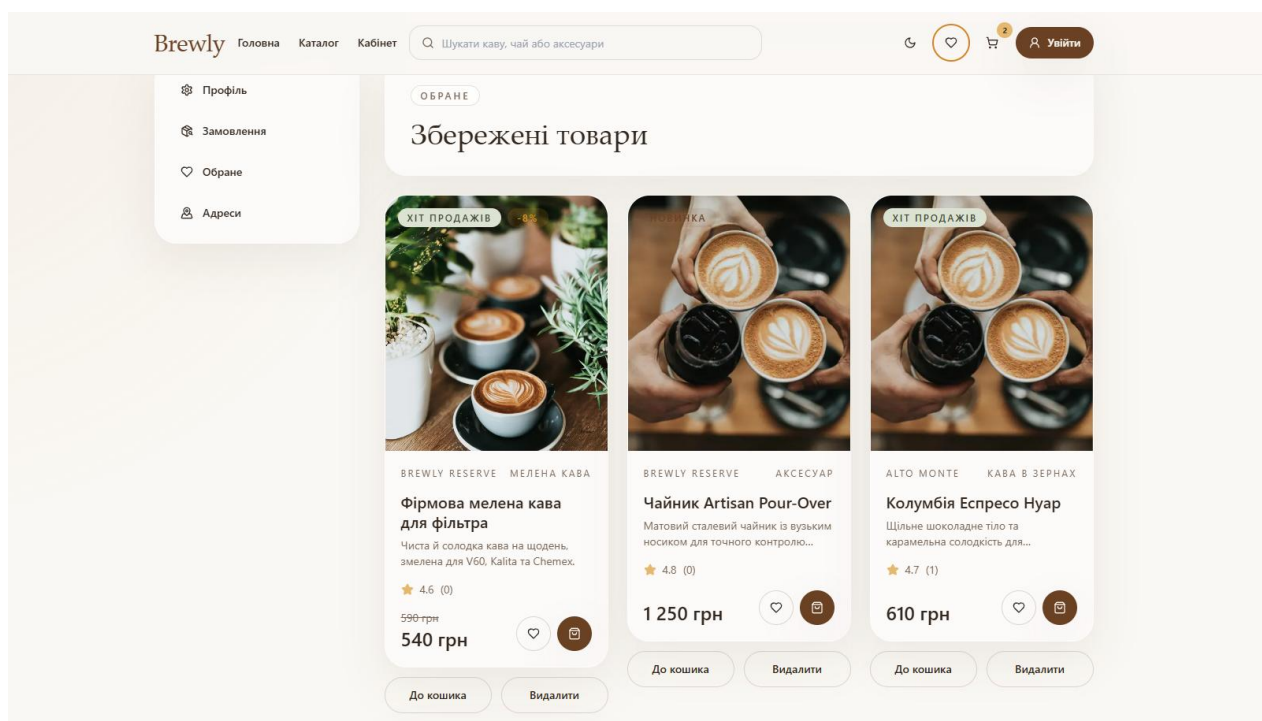


Рисунок Б.4 – Список бажань з кількома доданими товарами

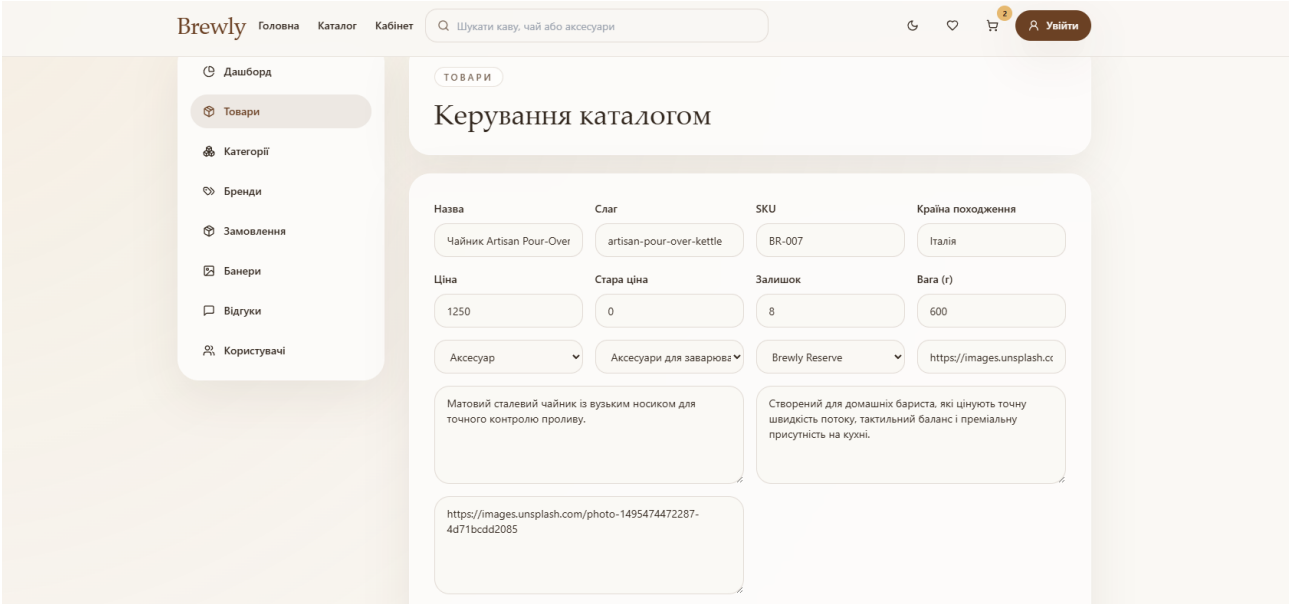


Рисунок Б.5 – Адміністративна сторінка створення товару заповненими полями

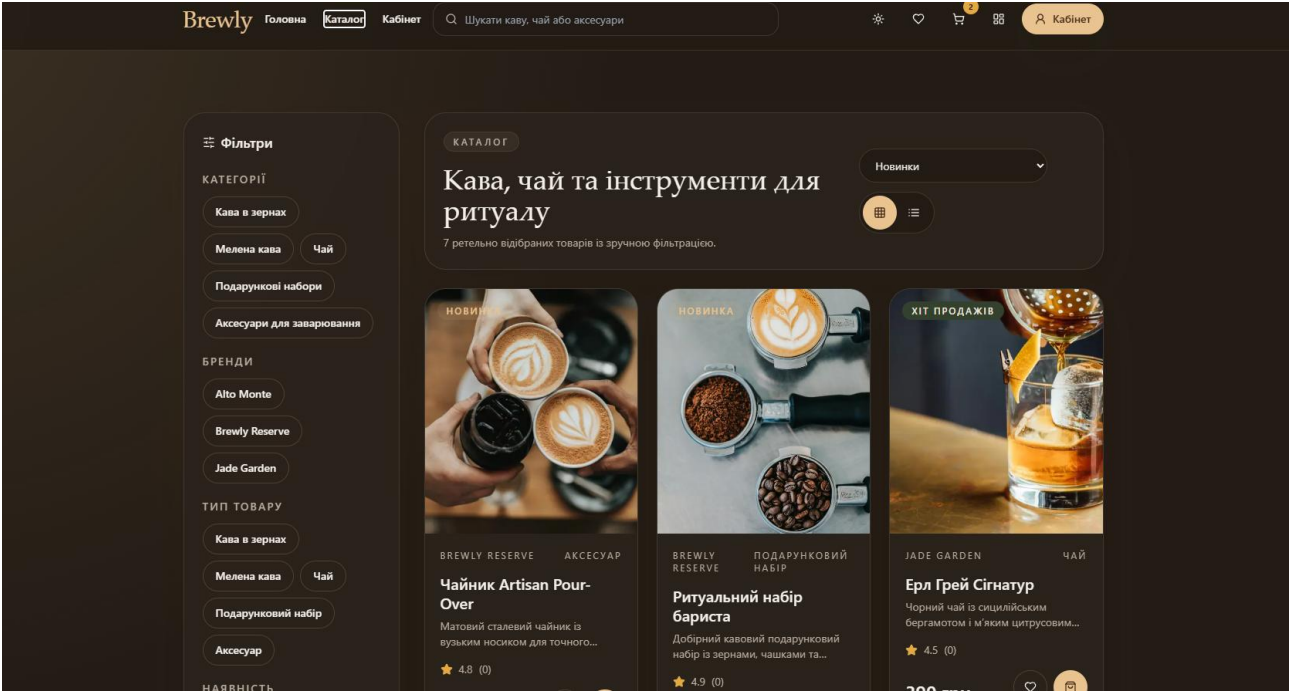


Рисунок Б.6 – Темна тема: каталог товарів після перемикання кнопки зміни кольору теми

ДОДАТОК В

Матеріали тестування та запуску

У додатку В наведено перелік команд, які використовуються для запуску проєкту.

Таблиця В.1 – Команди запуску і перевірки веб-додатку

Команда	Призначення	Результат
<code>npm install (у корені)</code>	Встановлення залежностей веб-додатку	Появи каталогів <code>node_modules</code> у корені, <code>client</code> і <code>server</code>
<code>cp server/.env.example server/.env</code>	Підготовка ENV сервера	Створено файл <code>server/.env</code>
<code>cp client/.env.example client/.env.local</code>	Підготовка ENV клієнта	Створено файл <code>client/.env.local</code>
<code>npm run prisma:generate -- workspace server</code>	Генерація типобезпечного клієнта Prisma	Згенеровано <code>node_modules/.prisma/client</code>
<code>npx prisma db push -- schema server/prisma/schema.prisma</code>	Застосування схеми до БД без міграцій	Таблиці створено у базі даних MySQL
<code>npm run seed</code>	Засів демонстраційних даних	У БД: 2 користувачі, 5 категорій, 3 бренди, 8 продуктів, 2 банери, 1 замовлення
<code>npm run dev</code>	Запуск клієнта і сервера паралельно	Клієнтська частина на <code>localhost:3000</code> , серверна частина на <code>localhost:4000</code>