БАКАЛАВРСЬКА РОБОТА

БР.ІП-30.00.00.000 ПЗ

Група ІП-22-2

Абрам Олег

2026

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Абрам Олег Віталійович

(прізвище, ім'я, по батькові)

УДК 004.41

(індекс)

БАКАЛАВРСЬКА РОБОТА

Застосування принципів UI/UX-дизайну у фронт-енд веб-розробці

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 – Інженерія програмного забезпечення

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Абрам О. В.

(підпис, ініціали та прізвище здобувача)

Науковий керівник Григорчук Любомир Іванович, доцент, к.п.н.

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

Допущено до захисту

Завідувач кафедри,

доцент _____ _____ Бандура В. В.

(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення
Ступінь вищої освіти бакалавр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою ІПЗ
доцент В.В. Бандура
“ ____ ” _____ 2026 р.

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ
Абраму Олегу Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема проекту (роботи) "Застосування принципів UI/UX-дизайну у фронт-енд веб-розробці"

керівник проекту (роботи) Григорчук Любомир Іванович, доцент, к.п.н.

затверджені наказом вищого навчального закладу від “ 21 ” квітня 2026 р. № 179/7

2. Строк подання студентом проекту (роботи) 20 травня 2026 р.

3. Вихідні дані до проекту (роботи) Матеріали навчального проєкту AptekaNear.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1 Теоретичні основи UI/UX у фронт-енд веброзробці

2 Аналіз предметної області та вимог

3 Проєктування вебзастосунку

4 Програмна реалізація

5 Тестування та впровадження

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1 Головна сторінка вебзастосунку AptekaNear (рис. 4.1, ст. 52)

2 Картка препарату з пропозиціями аптек і сортуванням (рис. 4.2, ст. 53)

3 Підтвердження бронювання у вебзастосунку AptekaNear (рис. 4.3, ст. 55)

4 Форма бронювання препарату у вебзастосунку AptekaNear (рис. 4.4, ст. 56)

5 Стан «нічого не знайдено» у вебзастосунку AptekaNear (рис. 4.5, ст. 57)

6. Консультанти розділів проекту (роботи)

Розділ	***Консультант***	***Підпис, дата***	
		Завдання видав	***Завдання прийняв***
1–5	*Григорчук Л. І.*		

7. Дата видачі завдання *січня 2026 року*

Керівник _____ *(підпис)* *Григорчук Л. І.* *(розшифровка підпису)*

Завдання прийняв до виконання _____ *(підпис)* *Абрам О. В.* *(розшифровка підпису)*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	***Назва етапів дипломного проекту (роботи)***	***Строк виконання етапів проекту***	***Примітка***
1	Визначення та обґрунтування теми роботи	15.12.2025	виконано
2	Опрацювання теоретичних основ UI/UX-дизайну та фронт-енд веб-розробки	31.01.2026	виконано
3	Аналіз предметної області, формування вимог і UX-моделі AptekaNear	15.03.2026	виконано
4	Проєктування, програмна реалізація, тестування та оформлення додатків	30.04.2026	виконано
5	Оформлення пояснювальної записки бакалаврської роботи	20.05.2026	виконано

Студент–дипломник _____ *(підпис)* *Абрам О. В.* *(розшифровка підпису)*

Керівник роботи _____ *(підпис)* *Григорчук Л. І.* *(розшифровка підпису)*

АНОТАЦІЯ

Бакалаврська робота містить 72 сторінки, 5 рисунків, 12 таблиць, список використаних джерел із 32 найменувань та 6 додатків.

Метою роботи є обґрунтування системного застосування принципів UI/UX-дизайну у фронт-енд веб-розробці на прикладі AptekaNear.

Об'єкт дослідження: процес фронт-енд веб-розробки інтерактивних вебзастосунків.

Предмет дослідження: принципи, методи та засоби UI/UX-дизайну, що застосовуються під час проєктування та реалізації клієнтської частини вебсистем.

У першому розділі розглянуто теоретичні основи UI/UX-дизайну у фронт-енд веб-розробці та систематизовано принципи побудови інтерфейсів.

У другому розділі виконано аналіз предметної області AptekaNear, визначено цільові групи користувачів і сформовано вимоги до MVP.

У третьому розділі спроєктовано інтерфейсні та архітектурні рішення: структуру екранів, клієнт-серверну архітектуру та модель даних.

У четвертому розділі описано реалізацію клієнтської частини AptekaNear: сценарну структуру сторінок на React, Vite і TypeScript, стани інтерфейсу, валідацію форми бронювання, адаптивність і вебдоступність.

У п'ятому розділі розглянуто тестування, оцінювання та впровадження вебзастосунку й визначено напрями його вдосконалення.

Висновки роботи підтверджують, що інтеграція UI/UX-рішень із фронт-енд реалізацією спрощує взаємодію з вебзастосунком і підвищує підтримуваність продукту.

Отримані результати дослідження включають рекомендації щодо поєднання UI/UX-дизайну з фронт-енд інженерією для реалізації AptekaNear та подібних вебзастосунків.

КЛЮЧОВІ СЛОВА: UI, UX, ФРОНТ-ЕНД, ВЕБРОЗРОБКА, ЮЗАБІЛІТІ, АДАПТИВНИЙ ДИЗАЙН, ДОСТУПНІСТЬ, ІНФОРМАЦІЙНА АРХІТЕКТУРА, ВЕБЗАСТОСУНОК, APTEKANEAR.

ANNOTATION

The bachelor's thesis contains 72 pages, 5 figures, 12 tables, a list of used sources with 32 items, and 6 appendices.

The purpose of the thesis is to justify the systematic use of UI/UX design principles in front-end web development, applied to the AptekaNear project.

Object of the study: the process of front-end web development of interactive web applications.

The subject of the study is the principles, methods, and tools of UI/UX design used in the design and implementation of the client side of web systems.

The first section examines the theoretical foundations of UI/UX design in front-end web development and systematizes interface design principles.

The second section analyzes the domain of AptekaNear, identifies the target user groups, and formulates the requirements for the MVP.

The third section designs the interface and architectural solutions: the screen structure, client-server architecture, and data model.

The fourth section describes the implementation of the AptekaNear client side: scenario-based page structure built with React, Vite, and TypeScript, interface states, booking form validation, responsiveness, and accessibility.

The fifth section covers testing, evaluation, and deployment of the web application and outlines directions for its improvement.

The conclusions of the work confirm that the integration of UI/UX solutions with front-end implementation simplifies interaction with the web application and improves the maintainability of the product.

The results of the study include recommendations for combining UI/UX design with front-end engineering for implementing AptekaNear and similar web applications.

KEYWORDS: UI, UX, FRONT-END, WEB DEVELOPMENT, USABILITY, RESPONSIVE DESIGN, ACCESSIBILITY, INFORMATION ARCHITECTURE, WEB APPLICATION, APTEKANEAR.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

У роботі використано такі скорочення:

UI (User Interface) – інтерфейс користувача.

UX (User Experience) – користувацький досвід.

MVP (Minimum Viable Product) – мінімально життєздатний продукт.

API (Application Programming Interface) – програмний інтерфейс застосунку.

REST (Representational State Transfer) – архітектурний стиль побудови вебсервісів.

HTML (HyperText Markup Language) – мова гіпертекстової розмітки.

CSS (Cascading Style Sheets) – каскадні таблиці стилів.

DI (Dependency Injection) – впровадження залежностей.

WCAG (Web Content Accessibility Guidelines) – настанови щодо доступності вебконтенту.

WAI (Web Accessibility Initiative) – ініціатива вебдоступності.

ARIA (Accessible Rich Internet Applications) – набір атрибутів для підвищення доступності інтерактивних вебінтерфейсів.

W3C (World Wide Web Consortium) – Консорціум Всесвітньої павутини.

БД – база даних.

ВСТУП

Актуальність теми дослідження зумовлена тим, що сучасний вебзастосунок оцінюється користувачем не лише за наявністю потрібного функціоналу, а й за швидкістю сприйняття інтерфейсу, передбачуваністю навігації, простотою виконання дій та відчуттям контролю над системою. У фронт-енд веб-розробці саме рівень реалізації UI/UX-рішень визначає, чи зможе користувач без зайвих зусиль знайти потрібну інформацію, правильно інтерпретувати стани елементів та успішно завершити свій сценарій. Тому питання інтеграції принципів UI/UX-дизайну у процес розроблення клієнтської частини вебзастосунків є не другорядним етапом «після програмування», а повноцінною складовою інженерії програмного забезпечення [21, 23, 24].

У міру зростання складності вебпродуктів фронт-енд перестав бути лише засобом візуалізації даних. Він забезпечує логіку взаємодії, керує станами екранів, обробляє помилки, підказує наступні кроки, адаптується до різних пристроїв, враховує вимоги доступності та впливає на суб’єктивну оцінку якості продукту. Якщо інтерфейс не підтримує ментальну модель користувача, навіть технічно коректна система сприймається як незручна. Навпаки, продумана структура екранів, семантична розмітка, чітка візуальна ієрархія та послідовний зворотний зв’язок підвищують ефективність взаємодії та скорочують кількість помилок [4, 15, 23].

Для практичного опрацювання теми у роботі використано навчальний вебпроєкт AptekaNear - сервіс пошуку лікарських засобів, порівняння цін, перегляду наявності в аптеках та оформлення бронювання. Саме такий домен є показовим для UI/UX-дослідження, адже користувачі очікують максимально швидкого пошуку, мінімальної кількості кроків до результату, зрозумілих повідомлень про статуси товару та високої довіри до інтерфейсу. У сервісах такого типу будь-яка неузгодженість у навігації, формі бронювання чи поданні цін безпосередньо впливає на користувацький досвід і конверсію критичних сценаріїв.

У бакалаврській роботі досліджено застосування принципів UI/UX-дизайну у фронт-енд веб-розробці та показано, що якість користувацького досвіду формується не окремо від програмної реалізації, а безпосередньо в процесі побудови клієнтської частини вебзастосунку. На основі аналізу наукової і професійної літератури узагальнено роль UI як системи візуальних і поведінкових елементів та UX як цілісного досвіду взаємодії користувача із системою.

У теоретичній частині роботи розглянуто основні принципи побудови інтерфейсів: візуальну ієрархію, консистентність, зворотний зв'язок, зручність форм, орієнтацію на завдання користувача, відповідність ментальній моделі, зниження когнітивного навантаження, адаптивність і доступність. Показано, що у фронт-енд веброзробці ці принципи набувають конкретної технічної форми через структуру HTML-розмітки, CSS-токени, компонентну архітектуру, стани елементів, маршрутизацію та обробку асинхронних подій.

У практичній частині виконано аналіз предметної області та вимог до вебзастосунку AptekaNear. Сформовано ключові користувацькі сценарії, описано групи користувачів, функціональні й нефункціональні вимоги, узагальнено роль wireframe-прототипу як інструмента узгодження UX і технічної структури фронт-енду. Обґрунтовано доцільність клієнт-серверної багаторівневої архітектури, визначено основні сутності даних, екрани, компоненти та API-виклики.

Окремо показано, що підтримка якісного UX вимагає не лише продуманого інтерфейсу, а й стабільної серверної частини. У матеріалах проєкту використано модульний підхід, Dependency Injection, патерни Facade, Strategy і Factory Method, а також базове логування й аналітику помилок. Ці рішення підвищують підтримуваність системи, роблять взаємодію між шарами передбачуваною і полегшують подальшу фронт-енд реалізацію.

Об'єктом дослідження є процес фронт-енд веб-розробки інтерактивних застосунків.

Предметом дослідження є принципи, методи та засоби UI/UX-дизайну, що застосовуються під час проєктування та реалізації клієнтської частини вебсистем.

Метою роботи є обґрунтування доцільності системного використання принципів UI/UX-дизайну у фронт-енд веб-розробці та демонстрація практичного застосування цих принципів на прикладі проєкту AptekaNear.

Для досягнення поставленої мети сформовані завдання: проаналізувати теоретичні засади; узагальнити принципи побудови зручних, доступних та адаптивних інтерфейсів; розглянути особливості фронт-енд реалізації дизайнерських рішень; проаналізувати предметну область проєкту, визначити цільові групи користувачів і сформувати вимоги до системи.

Наступні методи дослідження: аналіз наукової та професійної літератури; порівняльний аналіз інтерфейсних підходів; методи структурного та об'єктно-орієнтованого проєктування; моделювання користувацьких сценаріїв; експертну оцінку юзабіліті; узагальнення результатів розроблення навчального проєкту.

Наукова новизна одержаних результатів. У роботі уточнено підхід до застосування принципів UI/UX-дизайну у фронт-енд веб-розробці на прикладі навчального вебпроєкту AptekaNear. Запропоновано розглядати UI/UX-рішення не лише як етап візуального оформлення, а як складову проєктування та реалізації клієнтської частини вебзастосунку.

Практичне значення одержаних результатів. У роботі сформовано цілісний набір рекомендацій щодо поєднання UI/UX-дизайну з фронт-енд інженерією. Запропоновано підходи до побудови користувацьких сценаріїв, структури екранів, компонентів, API-взаємодії, тестування та оцінювання UX. Практичне застосування результатів роботи охоплює створення й удосконалення фронт-енд частини вебзастосунків із урахуванням принципів UI/UX-дизайну.

Робота складається зі вступу, п'яти розділів, висновків, списку використаних джерел і додатків.

Бакалаврська робота містить 72 сторінки, 5 рисунків і 12 таблиць.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАСТОСУВАННЯ UI/UX-ДИЗАЙНУ У ФРОНТ-ЕНД ВЕБ-РОЗРОБЦІ

1.1 Теоретичні засади UI та UX у вебзастосунках

Термін UI (User Interface) у межах веброзробки охоплює засоби візуальної і структурної взаємодії користувача із системою: композицію екрана, типографіку, колір, форми, кнопки, іконографіку, стани елементів, підказки та інші об’єкти, з якими користувач безпосередньо працює. UX (User Experience) має ширший зміст і описує загальне враження від користування продуктом: чи легко зрозуміти логіку системи, чи швидко досягається мета, наскільки передбачуваними є дії застосунку та чи викликає інтерфейс довіру [9, 18, 19].

У практиці веброзробки UI та UX тісно пов’язані між собою. Естетично привабливий інтерфейс, який не підтримує природний сценарій користувача, не забезпечує якісного досвіду. Так само логічно спроєктований сценарій втрачає ефективність, якщо реалізація у фронт-енді перевантажена візуальним шумом, має нечітку ієрархію або недостатній контраст. Отже, UI є видимою формою взаємодії, а UX - результатом цієї взаємодії у поєднанні з очікуваннями користувача і контекстом використання [5, 9, 24, 31].

Людиноорієнтований підхід до проєктування, закріплений у стандарті ISO 9241-210, передбачає глибоке розуміння потреб користувачів, контексту використання, циклічне уточнення рішень та оцінювання результатів за участю користувача або експертів. Для вебзастосунків це означає, що інтерфейс повинен вибудовуватися не навколо внутрішньої логіки системи, а навколо найцінніших користувацьких сценаріїв: пошуку, вибору, оформлення дії, підтвердження результату, повернення до історії або профілю [18, 19].

В умовах конкурентного цифрового ринку якісний UX стає фактором утримання користувача. Якщо система вимагає зайвих дій, створює невизначеність або не пояснює помилки, користувач швидко залишає продукт, навіть якщо той має корисну функціональність. Тому у фронт-енд веброзробці робота з досвідом користувача має розглядатися як інженерне завдання, що охоплює не лише дизайн-

макети, а й архітектуру компонентів, обробку подій, оптимізацію продуктивності та доступність [14, 15, 21].

Узагальнені дані наведено в табл. 1.1.

Таблиця 1.1

Узагальнена характеристика UI та UX у вебзастосунках

Критерій	**UI**	**UX**
Основний фокус	Візуальна форма взаємодії	Повний досвід користувача під час виконання сценарію
Ключові елементи	Композиція, колір, типографіка, кнопки, поля, стани	Навігація, логіка кроків, зрозумілість, довіра, ефективність
Основне питання	Що користувач бачить і з чим взаємодіє?	Наскільки легко, швидко і комфортно користувач досягає мети?
Вплив на фронт-енд	Реалізація компонентів, стилів, адаптивності	Побудова потоків, обробка помилок, зворотний зв’язок, продуктивність

Базові принципи побудови інтерфейсу користувача

Одним із базових принципів UI є візуальна ієрархія. На екрані завжди мають бути очевидно визначені первинні, вторинні та допоміжні дії. Досягти цього можна за допомогою розміру шрифту, контрасту, відступів, групування елементів та візуальних акцентів. Коли ієрархія порушена, користувач витрачає додатковий час на пошук основної кнопки або важливої інформації. Для фронт-енд розробника це означає потребу в єдиній системі стилів, повторюваних правилах верстки та контролі станів компонентів [9, 14, 20, 23].

Не менш важливим є принцип консистентності. Елементи однакового призначення повинні мати однаковий вигляд і однаково поводитися на всіх екранах. Якщо на одній сторінці основна дія позначена синьою кнопкою, а на іншій - текстовим посиланням, користувач змушений заново інтерпретувати інтерфейс. Консистентність зменшує когнітивне навантаження, полегшує навчання новим сценаріям і прискорює виконання повторюваних дій [14, 23, 31, 32].

Велике значення має типографіка. Читабельний вебінтерфейс потребує контрастного шрифту, передбачуваної ієрархії заголовків, достатніх інтерліньяжів та узгоджених відступів. Недоліки типографіки рідко сприймаються користувачем

як окрема проблема, але саме вони часто створюють відчуття складності, перевантаженості або непрофесійності продукту. У фронт-енді це означає роботу з токенами типографіки, масштабом шрифтів та адаптацією текстових блоків до різних розмірів екрана [6, 9, 20].

Ще один важливий принцип - явний зворотний зв’язок. Інтерфейс має повідомляти користувача про результат його дії: натискання кнопки, успішне збереження форми, завантаження даних, попередження або помилку. Відсутність реакції системи породжує невизначеність і підвищує ризик повторних, зайвих або помилкових дій. Тому під час фронт-енд реалізації потрібно проєктувати стани hover, focus, disabled, loading, success та error як повноцінні складові компонента, а не як другорядні декоративні елементи [5, 15, 23, 24].

Особливу увагу слід приділяти формам введення. Саме форми найчастіше стають вузьким місцем користувацького потоку. Ефективна форма містить мінімально необхідну кількість полів, зрозумілі підписи, доречні значення за замовчуванням, валідацію біля поля та помилки, сформульовані мовою дії. Для фронт-енд розробника це означає необхідність коректної роботи з фокусом, масками введення, семантичними label, допоміжними повідомленнями й захистом від випадкових повторних надсилань [4, 13, 14].

1.2 UX-принципи, якість взаємодії та фронт-енд інтерпретація

UX-дизайн зосереджений не на окремому елементі, а на сценарії як цілісній послідовності кроків. Успішний досвід користувача виникає тоді, коли шлях від наміру до результату є коротким, логічним і передбачуваним. Тому одним із центральних принципів UX є орієнтація на завдання користувача.

Система повинна приховувати від користувача зайву технічну складність і виводити на перший план саме те, що допомагає завершити дію: знайти товар, відфільтрувати результати, оформити бронювання, підтвердити дію або повернутися до попереднього стану [11, 19, 24, 31].

Важливим принципом є відповідність ментальній моделі користувача. Люди очікують, що вебсервіс працюватиме у спосіб, подібний до інших знайомих

їм продуктів: логотип повертає на головну сторінку, поле пошуку розміщене у верхній частині екрана, кнопка підтвердження виділена сильніше за другорядні дії, а результати сортування можна змінити без перезавантаження всього сценарію. Порушення ментальної моделі веде до дезорієнтації та зростання кількості помилок [23, 24, 31].

Під час веброзробки UX також пов’язаний з інформаційною архітектурою. Навіть якісні окремі екрани не забезпечать доброго досвіду, якщо зв’язки між ними неузгоджені або занадто складні. Інформаційна архітектура визначає, як класифікуються дані, які блоки вважаються первинними, як побудовано навігаційні шляхи, де користувач очікує знайти історію дій, профіль, налаштування чи деталі об’єкта. Для фронт-енду це виливається у структуру маршрутів, вкладених сторінок, компонентів навігації та правил повернення назад [9, 11, 19].

Принцип зниження когнітивного навантаження є особливо важливим для сервісів, де користувач перебуває у стресовому або терміновому контексті. У таких системах не слід перевантажувати перший екран зайвою інформацією, складною термінологією чи великою кількістю конкурентних акцентів. Бажано ділити завдання на зрозумілі кроки, використовувати звичні назви дій, уникати дублювання повідомлень та підкріплювати вибір короткими підказками. Інженерно це означає оптимізацію сценаріїв, зменшення кількості модальних вікон та відмову від непотрібних переривань користувача [18, 23, 24].

Важливим орієнтиром залишаються евристики юзабіліті: видимість стану системи, відповідність реальному світу, свобода і контроль користувача, консистентність, запобігання помилкам, розпізнавання замість пригадування, гнучкість і ефективність, естетичний мінімалізм, зрозумілі помилки та підтримка відновлення після них. У фронт-енд реалізації ці вимоги конкретизуються в маршрутизації, механізмах undo/redo, станах форм, підказках, структурі компонентів та побудові API-повідомлень, які відображаються користувачеві [23, 24].

Отже, UX-принципи у фронт-енд реалізації допомагають узгодити логіку інтерфейсу, структуру компонентів і реакцію системи на дії користувача.

Особливості фронт-енд реалізації UI/UX-рішень

Фронт-енд веброзробка є точкою, де дизайнерські принципи переходять у реальну взаємодію. Семантична HTML-розмітка впливає не лише на SEO чи формальну коректність документа, а й на доступність, роботу з клавіатурою та зрозумілість структури для допоміжних технологій. Коректне використання заголовків, списків, form, button, nav, main, section та інших елементів створює передбачувану основу для UX і полегшує подальшу підтримку інтерфейсу [4, 15, 16].

CSS у сучасному фронт-енді відповідає не просто за оформлення, а за системність дизайну. Відступи, сітки, адаптивні точки перелому, масштаби шрифтів, контрасти та стани елементів мають утворювати узгоджений набір рішень. Якщо стилізація формується хаотично, продукт швидко втрачає візуальну цілісність. Тому важливо застосовувати дизайн-систему або щонайменше набір токенів інтерфейсу, що забезпечує передбачуваність вигляду елементів на різних сторінках і пристроях [6, 20, 22, 26].

JavaScript та фронт-енд фреймворки розширюють можливості UX завдяки реактивним оновленням екранів, компонентній моделі, локальному керуванню станом і роботі з асинхронними запитами. Водночас вони можуть створювати і додаткові ризики: повільне завантаження, стрибки верстки, блокування взаємодії під час повторних запитів, втрату стану форми або неоднозначні переходи між маршрутами. Тому добрий UX у фронт-енді неможливий без уваги до оптимізації ресурсів, кешування, управління запитами і контрольованої обробки помилок [8, 20, 26, 29].

Адаптивність є ще одним ключовим чинником. Користувач працює з вебсервісом на телефоні, планшеті, ноутбуці або великому моніторі, тому сценарії мають зберігати зрозумілість незалежно від ширини екрана. Mobile-first підхід дозволяє сконцентруватися на найважливіших діях і поступово розширювати інтерфейс для більших екранів. У сервісах із пошуком і формами це особливо корисно: мобільна версія змушує розставити пріоритети і прибрати зайві елементи,

а десктопна - дає змогу комфортно показати додаткові фільтри та паралельні блоки [16, 22].

Сучасний фронт-енд повинен враховувати і вимоги доступності: достатній контраст, видимий фокус, альтернативний текст, підтримку клавіатурної навігації, коректну структуру заголовків і рольові атрибути лише там, де вони справді потрібні. Доступність не обмежується вимогами для окремих користувачів; вона підвищує загальну якість інтерфейсу для всіх, оскільки дисциплінує структуру сторінки, спрощує взаємодію і робить систему передбачуванішою [4, 5, 14, 31].

1.3 Аналоги рішень, прототипування та нормативні орієнтири в UI/UX-проєктуванні

У сфері онлайн-сервісів пошуку та бронювання товарів, зокрема в аптечному домені, можна виділити три узагальнені типи аналогів: агрегатори, що об’єднують пропозиції різних постачальників; сайти окремих мереж, орієнтовані на власний каталог; маркетплейсні рішення з розширеною системою рекомендацій і промоактивностей. З погляду UI/UX усі ці типи прагнуть розв’язати однакові базові задачі: дати швидкий пошук, показати релевантний список результатів, дозволити відсортувати пропозиції та скоротити шлях до оформлення замовлення.

Порівняльний аналіз показує, що найуспішніші рішення мають спільні риси: домінантне поле пошуку на першому екрані; явне групування фільтрів; зрозумілу картку результату з ключовими атрибутами; короткий сценарій замовлення; інформативні повідомлення про відсутність даних; послідовну мобільну версію; збереження контексту користувача під час переходу між списком, картою і сторінкою об’єкта. Натомість типовими недоліками є перевантаженість банерами, дублювання дій, надлишкові форми, розрив між мобільною і десктопною логікою та слабко пояснені статуси наявності.

Для подальшої роботи над AptekaNear доцільно взяти з узагальненого аналізу такі орієнтири: мінімізувати кількість дій на головному сценарії; відокремити обов’язкові дії від додаткових; забезпечити швидкий перехід між списком і картою; подати інформацію про препарат і аптеку в одному

інформаційному потоці; винести помилки і порожні стани в окремо спроєктовані UX-компоненти. Таким чином, аналіз аналогів підтверджує, що цінність продукту в подібному домені формується не тільки функціональністю, а й якістю організації інтерфейсу.

Узагальнені дані наведено в табл. 1.2.

Таблиця 1.2

Узагальнене порівняння типових аналогів за UI/UX-критеріями

Тип рішення	**Сильні сторони**	**Типові недоліки**	**Висновок для AptekaNear**
Агрегатор пропозицій	Швидкий пошук, широкий вибір, порівняння цін	Перевантажені списки, складна навігація в деталях	Потрібно зберегти порівняння, але уникнути візуального шуму
Сайт окремої мережі	Цілісна айдентика, простіший шлях до замовлення	Обмежений вибір, слабше порівняння альтернатив	Доцільно запозичити короткий сценарій бронювання
Маркетплейсне рішення	Багато фільтрів, персоналізація, додаткові сервіси	Надлишок акцентів, промо та другорядних дій	Слід залишити тільки функції, що підтримують основний сценарій

Метрики якості UX та способи їх оцінювання

Для інженерної оцінки користувацького досвіду важливо переходити від загальних формулювань на кшталт «інтерфейс зручний» до набору вимірюваних критеріїв. У практиці UI/UX найчастіше аналізують частку успішного виконання завдання, середній час проходження сценарію, кількість помилок, частоту повернення на попередній крок, рівень відмови на формі та суб’єктивну оцінку зрозумілості інтерфейсу. Такі метрики дають змогу поєднати дизайнерську гіпотезу з об’єктивними спостереженнями за поведінкою користувача.

У фронт-енд веброзробці метрики UX тісно пов’язані з технічними показниками. Якщо сторінка довго рендериться, макет зміщується під час завантаження або користувач не бачить стану обробки запиту, це безпосередньо погіршує сприйняття взаємодії. Тому поряд із класичними UX-методами доцільно враховувати Core Web Vitals, стабільність верстки, коректність обробки порожніх станів, а також якість повідомлень про помилки й підтвердження дій.

Для сервісу AptekaNear показовими є насамперед метрики, пов’язані з

критичним сценарієм пошуку та бронювання препарату. До них належать: час від введення запиту до появи релевантних результатів, кількість кроків до підтвердження бронювання, частота повторного пошуку після відкриття картки препарату, кількість невдалих спроб заповнення форми та частка сценаріїв, у яких користувач залишає процес після повідомлення про відсутність товару.

Отже, якісний UX не зводиться до візуальної привабливості. Він потребує системи показників, яка дозволяє виявляти слабкі місця інтерфейсу, пріоритизувати покращення і перевіряти, чи справді зміни у фронт-енд реалізації спростили взаємодію. Саме тому метрики якості UX доцільно закладати вже на етапі формування вимог і підтримувати їх упродовж усього життєвого циклу продукту.

Інструменти прототипування та роль Figma у фронт-енд процесі

У сучасній практиці веброзробки прототипування стало окремим етапом, який зменшує кількість помилок ще до початку програмної реалізації. Low-fidelity wireframe дає змогу швидко перевірити композицію екрана, послідовність блоків, наявність ключових станів і достатність навігаційних переходів. Для фронт-енд команди це означає, що структура майбутнього інтерфейсу може бути обговорена до написання компонентів і бізнес-логіки.

Серед інструментів прототипування особливо поширеним є Figma, оскільки вона поєднує каркасну модель, візуальний макет, інтерактивний сценарій переходів та спільну роботу команди. Для розробника важливо, що Figma дозволяє бачити не лише статичні екрани, а й логіку між ними: які стани мають кнопки, куди веде кожна дія, які блоки повторюються і де виникають залежності між екранами.

У межах теми бакалаврської роботи прототипування виконує роль містка між UX-моделлю та технічною постановкою задачі. Саме завдяки прототипу можна завчасно визначити межі екрану, набір компонентів, логіку маршрутизації, вимоги до адаптивності та правила відображення порожніх, помилкових і завантажувальних станів. Це знижує ризик того, що фронт-енд реалізація відхилиться від очікуваного користувацького сценарію.

Для AptekaNear прототип у Figma став базовим артефактом узгодження між

функціональними вимогами й майбутньою структурою інтерфейсу. Ілюстративний фрагмент такого wireframe-прототипу, який демонструє логіку переходів між основними екранами, наведено в додатку Д. Надалі подібний прототип може бути деталізований до дизайн-системи з токенами, компонентами та правилами адаптації під різні формати екранів.

Під час розроблення вебінтерфейсів важливо спиратися не лише на власне бачення дизайнера або розробника, а й на загальновизнані стандарти та рекомендації. Людиноорієнтований підхід розглядає інтерактивну систему через потреби користувача, контекст використання, повторне уточнення рішень і перевірку зручності взаємодії. Для бакалаврської роботи це означає, що UI/UX-рішення мають бути описані як інженерно обґрунтовані, а не суто візуальні рішення [18, 19].

Окрему роль відіграють вимоги вебдоступності. WCAG 2.2 та пов’язані рекомендації WAI дають змогу перевіряти інтерфейс за зрозумілими критеріями: сприйнятність, керованість, зрозумілість і надійність. Для фронт-енд реалізації це перетворюється на конкретні правила: достатній контраст, коректні підписи полів, видимий фокус, семантична HTML-структура, підтримка клавіатурної навігації та обережне використання ARIA-атрибутів [5, 14, 31].

Ще одним практичним орієнтиром є дизайн-системи. Вони допомагають перетворити набір окремих екранів на єдину мову продукту: з узгодженими кольорами, відступами, типографікою, компонентами, станами та правилами їх використання. Для українського контексту показовим прикладом є дизайн-система державних сайтів України та дизайн-система Дії, які демонструють значення простоти, доступності й послідовності цифрових сервісів [1, 2, 32].

Отже, нормативні орієнтири й дизайн-системи створюють основу для стабільної фронт-енд реалізації. Вони зменшують кількість випадкових рішень, підвищують передбачуваність інтерфейсу та дозволяють підтримувати якість вебзастосунку в процесі подальшого розвитку [9, 13, 32].

1.4 Висновки по розділу

У першому розділі розглянуто теоретичні основи застосування UI/UX-дизайну у фронт-енд веб-розробці. Визначено, що UI відповідає за візуальну та структурну форму взаємодії користувача із системою, а UX охоплює загальний досвід користувача під час виконання сценарію.

Узагальнено основні принципи побудови інтерфейсів: візуальну ієрархію, консистентність, зворотний зв’язок, зручність форм, адаптивність, доступність і відповідність ментальній моделі користувача. Встановлено, що ці принципи мають безпосереднє технічне втілення у фронт-енд реалізації через HTML-структуру, CSS-стилізацію, компоненти, стани інтерфейсу та обробку подій.

Зроблено висновок, що UI/UX-дизайн є важливою інженерною складовою сучасної веброзробки, яка впливає на якість, зручність і підтримуваність вебзастосунку.

РОЗДІЛ 2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ФОРМУВАННЯ ВИМОГ ДО ВЕБЗАСТОСУНКУ APTEKANEAR

2.1 Предметна область, ідея проєкту та цільові користувачі

Проєкт AptekaNear орієнтований на розв’язання практичної проблеми, з якою користувач стикається у повсякденному житті: необхідність швидко знайти потрібний лікарський засіб, перевірити його наявність у найближчих аптеках, порівняти ціну і за можливості забронювати товар для самовивозу. У такому сценарії критичними є час, зрозумілість дій і достовірність інформації. Саме тому цей домен є показовим для дослідження впливу UI/UX-принципів на фронт-енд розробку.

У попередніх матеріалах проєкту було визначено, що сервіс має поєднати декілька типів цінності: швидкий пошук препарату за назвою або діючою речовиною; перегляд доступних аптек і цін; фільтрацію за локацією; карту з прив’язкою до місця перебування користувача; картку препарату з базовою інформацією; створення бронювання; перегляд історії бронювань та списку обраного. Таким чином, основна гіпотеза продукту полягає в тому, що користувачу потрібен сервіс із мінімальною кількістю кроків від пошуку до підтвердженої дії.

Специфіка предметної області накладає і додаткові вимоги. На відміну від розважальних або контентних продуктів, сервіс пошуку ліків має працювати у контексті підвищеної відповідальності. Користувач очікує коректного подання ціни, стану наявності, адреси аптеки, часу дії бронювання та способу отримання товару. Через це інтерфейс повинен уникати неоднозначних формулювань, зайвих декоративних акцентів і неперевірених повідомлень. Формування високого рівня довіри до представлених даних стає визначальним критерієм якості користувацького досвіду у межах цього сервісу.

Ідея AptekaNear у межах бакалаврської роботи використовується як кейс, на якому можна простежити зв’язок між аналітикою вимог, побудовою wireframe-моделі, архітектурними рішеннями та практикою фронт-енд реалізації.

Це дає можливість перейти від загальних принципів UI/UX до конкретних інженерних висновків щодо структури екранів, навігації, компонентів, API-контрактів і механізмів забезпечення якості.

Для AptekaNear важливо, щоб основний шлях користувача на сайті був зручним і зрозумілим. Користувач має швидко знайти препарат, переглянути доступні аптеки, порівняти ціну та перейти до бронювання без зайвих дій. Тому інтерфейс повинен підтримувати цей сценарій і допомагати користувачу швидко досягти потрібного результату.

Цільові групи користувачів і їх потреби

У проєкті можна виділити кілька основних груп користувачів. Перша група - звичайні відвідувачі, які хочуть швидко знайти препарат без створення облікового запису. Для них критичними є просте поле пошуку, зрозумілий список результатів, базова інформація про препарат і можливість отримати відповідь без зайвих перешкод. Друга група - зареєстровані користувачі, яким потрібні історія бронювань, повторне оформлення, список обраного та персоналізовані нагадування.

Окрему групу формують представники аптек або адміністратори, які оновлюють дані про залишки, ціни та графік роботи. Хоча їхній інтерфейс не є центральним у цій роботі, він важливий з погляду цілісності досвіду: якість користувацької частини напряму залежить від актуальності довідкових даних. Саме тому адміністративний сценарій слід розглядати як допоміжний, але стратегічно важливий елемент системи.

З UX-позиції всі групи об’єднує прагнення до зниження невизначеності. Користувач хоче швидко зрозуміти, чи є препарат у наявності, де він доступний, скільки коштує і що потрібно для бронювання. Отже, навіть на етапі формування вимог інтерфейсні рішення потрібно перевіряти питанням: чи зменшує ця функція кількість сумнівів користувача і чи пришвидшує вона досягнення основної мети.

Такий поділ користувачів пов’язує функції AptekaNear з очікуваннями кожної ролі та створює основу для подальшого формування вимог до MVP.

Узагальнені дані наведено в табл. 2.1.

Таблиця 2.1

Основні групи користувачів вебзастосунку AptekaNear

Група користувачів	Потреби	Ключові UI/UX-пріоритети
Незареєстрований користувач	Швидкий пошук, перегляд цін і наявності	Домінантне поле пошуку, простий список результатів, мінімум бар'єрів
Зареєстрований користувач	Бронювання, історія, обране, нагадування	Послідовна навігація, збереження контексту, короткі форми
Адміністратор / аптека	Оновлення цін, залишків, контактів	Чіткі форми введення, валідація, зрозумілі статуси збереження

2.2 Вимоги до вебзастосунку та пріоритизація MVP

На основі проєктних матеріалів AptekaNear було сформовано набір користувацьких історій, які описують очікувану поведінку системи з точки зору ролей та цілей. Високий пріоритет для MVP мають сценарії пошуку препарату, перегляду аптек з наявністю, порівняння цін, фільтрації за локацією, відображення аптек на карті, перегляду картки препарату, бронювання, реєстрації та історії бронювань. Ці сценарії формують «ядро» продукту, навколо якого вибудовується інтерфейс.

З погляду UI/UX цінність користувацьких історій полягає в тому, що вони дозволяють переходити від абстрактного набору функцій до конкретних кроків користувача. Наприклад, історія «я хочу знайти препарат за назвою» одразу задає вимоги до розміщення поля пошуку, мінімальної довжини запиту, повідомлень про відсутність результатів, підказок та форми подання картки результату. Історія про бронювання визначає склад форми, набір обов'язкових полів, підтверджувальне повідомлення і логіку повернення до попередніх екранів.

Важливо, що функціональні вимоги не можна розглядати окремо від способу їх представлення в інтерфейсі. Та сама функція може мати різну користувацьку цінність залежно від того, чи вона легко знаходиться, чи зрозумілі її стани та чи не конфліктує з іншими діями. Саме тому формування вимог у даній роботі виконується у зв'язці з підготовкою wireframe-моделі, навігаційних переходів і переліку основних компонентів.

Для практичного застосування у фронт-енд розробці кожну ключову функцію доцільно розкласти на вхідні дані, очікуваний результат і візуальний контейнер. Такий підхід полегшує проєктування компонентів, моделювання станів loading/error/empty, а також узгодження контрактів між клієнтською та серверною частинами.

Узагальнені дані наведено в табл. 2.2.

Таблиця 2.2

Ключові функціональні вимоги MVP-застосунку

Функція	**Користувацька цінність**	**Вимоги до інтерфейсу**
Пошук препарату	Швидкий старт сценарію	Поле пошуку на першому екрані, підказка, запуск по Enter та кнопці
Перегляд результатів	Оцінка варіантів придбання	Картка результату з ціною, статусом, короткою інформацією
Фільтрація та сортування	Зменшення кількості нерелевантних варіантів	Блок фільтрів, перемикання сортування без втрати контексту
Карта аптек	Оцінка зручності розташування	Перемикач «Список/Карта», маркери, синхронізація з результатами
Бронювання	Завершення цільової дії	Коротка форма, валідація, екран підтвердження
Профіль та історія	Повернення до попередніх дій	Особистий кабінет, список бронювань, повторне використання даних

Під час пріоритизації обсягу MVP до першочергової реалізації віднесено функції, що безпосередньо забезпечують основний сценарій користувача: пошук препарату за назвою, перегляд пропозицій аптек із ціною, відстанню та залишком, сортування цих пропозицій, а також бронювання обраної позиції з підтвердженням.

Допоміжні можливості - персональний профіль, обране, нагадування, інтеграцію з картографічними сервісами та повноцінне сховище даних - свідомо віднесено до подальших етапів, щоб сфокусувати навчальний прототип на ключовій цінності для користувача.

Пріоритети встановлювалися за впливом функції на досягнення основної мети та за складністю реалізації. Сценарії, без яких неможливо завершити шлях «знайти препарат - обрати аптеку - забронювати», отримали найвищий пріоритет,

тоді як функції, що лише підвищують зручність, віднесено до нижчих рівнів. Такий підхід відповідає принципу мінімально життєздатного продукту й дозволяє перевірити життєздатність ідеї на обмеженому, але цілісному наборі функцій.

Нефункціональні вимоги та UX-критерії якості

Нефункціональні вимоги у вебзастосунках безпосередньо впливають на UX, оскільки описують не стільки «що робить система», скільки «як саме вона це робить». Для AptekaNear ключовими є зрозумілість інтерфейсу, швидкість відгуку, надійність, доступність, безпечна обробка персональних даних і підтримка мобільних пристроїв. Якщо хоча б одна з цих вимог ігнорується, користувацький досвід втрачає цілісність.

З точки зору фронт-енд реалізації важливими є час початкового завантаження, стабільність верстки, коректна робота при нестабільному з'єднанні, якісна обробка помилок API та відсутність втрати введених даних.

Наприклад, якщо після помилки користувач змушений повторно заповнювати форму бронювання, це суттєво погіршує досвід взаємодії. Тому нефункціональні вимоги мають проєктуватися на рівні компонентів і станів сторінки.

Окремий блок становлять вимоги доступності. Для AptekaNear вони особливо актуальні, оскільки частина користувачів може мати обмеження зору або користуватися сервісом у стресовому контексті.

Контраст, чіткі підписи, достатній розмір активних зон, логічний порядок фокусу та відсутність залежності від кольору як єдиного каналу передачі інформації є не лише формальними вимогами, а й передумовами зрозумілого інтерфейсу.

Таким чином, нефункціональні вимоги у цій роботі інтерпретуються як критерії якості користувацького сценарію. Вони доповнюють функціональний опис системи й задають рамки для вибору технологій, архітектурних підходів та методів тестування.

Узагальнені дані наведено в табл. 2.3.

Таблиця 2.3

Нефункціональні вимоги та їх зв’язок із UI/UX

Нефункціональна вимога	Практичне значення	Прояв у фронт-енді
Юзабіліті	Швидке досягнення мети користувача	Короткі сценарії, зрозумілі підписи, сталі патерни взаємодії
Продуктивність	Відчуття швидкої роботи системи	Оптимізація запитів, lazy loading, мінімізація перерисовок
Надійність	Передбачуваність і довіра до результату	Порожні стани, помилки, повторні спроби, підтвердження дій
Безпека	Захист персональних даних	Контроль форм, токенів, обмеження доступу, повідомлення про авторизацію
Доступність	Можливість користування для ширшої аудиторії	Контраст, фокус, семантика, клавіатурна навігація

2.3 UX-модель інтерфейсу та її узгодження з технічною реалізацією

На етапі формування вимог для AptekaNear було створено low-fidelity wireframe-прототип у Figma. Такий формат доцільний для раннього проєктування, оскільки дозволяє зосередитися на структурі екранів, розміщенні елементів і логіці переходів без відволікання на детальну візуальну стилізацію. У межах моделі було підготовлено набір екранів, що охоплюють головні сценарії MVP: пошук, результати, карта, картка препарату, бронювання, підтвердження дії, вхід і профіль.

Для UX-аналізу важливо, що wireframe не просто демонструє вигляд сторінок, а описує шлях користувача. Було налаштовано переходи між екранами типу «натиснення - перехід», що дало змогу перевірити послідовність дій: від пошуку до перегляду результатів, далі - до картки препарату, форми бронювання та екрана підтвердження. Окремо опрацьовано переходи до профілю, історії бронювань і розділу збережених позицій.

Саме на цьому рівні стають очевидними потенційні UX-ризики: занадто довгий шлях до бронювання, дублювання інформації на екранах, відсутність чіткої вторинної навігації або конфлікт між списком і картою. Тому wireframe-етап у контексті фронт-енд розробки слід розглядати як технічну передумову правильної побудови маршрутів, вкладених компонентів і станів застосунку.

Створена каркасна модель стала базою для подальших лабораторних робіт і для формування архітектурних рішень. Це підтверджує, що UI/UX-артефакти не повинні існувати відокремлено від інженерної частини: вони мають прямо впливати на структуру фронт-енд застосунку, список компонентів, API-виклики та сценарії тестування.

Критичні користувацькі сценарії та пріоритизація MVP

Під час формування вимог до AptekaNear ключову роль відіграє виділення критичних користувацьких сценаріїв, тобто таких послідовностей дій, від успішності яких залежить сприйняття цінності всього продукту.

Для досліджуваного вебзастосунку до таких сценаріїв належать: пошук препарату за назвою, перегляд аптек з наявністю і ціною, фільтрація за локацією, відкриття картки препарату, а також бронювання позиції в конкретній аптеці.

Пріоритизація сценаріїв для MVP дає змогу обмежити першу версію системи лише тими функціями, що безпосередньо перевіряють основну продуктову гіпотезу. Якщо користувач не може швидко знайти препарат і зрозуміти, де він доступний, то розширені можливості на кшталт нагадувань або додаткових персональних налаштувань не компенсують відсутність базової зручності. Саме тому окреслені вище критичні сценарії доцільно розглядати як ядро першої версії сервісу, а менш критичні функції - як наступний етап розвитку.

З позиції UI/UX пріоритизація означає також правильний розподіл уваги в інтерфейсі. На головному екрані має домінувати пошук, на екрані результатів - список пропозицій і фільтри, у формі бронювання - підтвердження ключових даних та зрозумілий шлях завершення дії. Кожен додатковий елемент, що не підтримує критичний сценарій, підвищує когнітивне навантаження і знижує швидкість взаємодії.

Практичне значення пріоритизації полягає і в тому, що вона визначає порядок фронт-енд реалізації, тестування і приймання функціоналу. Опис основних користувацьких сценаріїв у роботі винесено також у додаток А, де вони подані як зведений матеріал для перевірки юзабіліті та подальшої інтеграційної перевірки вебзастосунку.

Узгодження UX-моделі та технічної реалізації

Перехід від UX-моделі до програмної реалізації є одним із найвідповідальніших етапів у фронт-енд розробці. На цьому кроці абстрактні вимоги до зручності, зрозумілості та передбачуваності інтерфейсу потрібно перетворити на конкретні маршрути, компоненти, стани екранів, правила валідації, повідомлення про помилки й API-контракти. Якщо такий перехід не зафіксований, навіть хороший прототип може бути реалізований непослідовно.

Для AptekaNear узгодження між UX-моделлю і технічною постановкою означає, що кожний елемент прототипу повинен мати відповідність у структурі фронт-енд застосунку: окремий маршрут, компонент або модальний сценарій. Наприклад, картка препарату задає не лише візуальний склад сторінки, а й потребу у відповідному endpoint для детальної інформації; форма бронювання формує вимоги до валідації, статусів доступності й механізму підтвердження дії.

Особливо важливо на цьому етапі описати інтерфейсні стани, які користувач бачить у реальній роботі застосунку: loading, empty, success, validation error, server error, denied location тощо. Саме ці стани найчастіше пропускаються в описі функцій, хоча вони формують значну частину фактичного користувацького досвіду. У бакалаврській роботі цей аспект використовується як аргумент на користь тісного поєднання UI/UX-проєктування з фронт-енд інженерією.

Таким чином, wireframe, user stories, acceptance criteria та архітектурні рішення не є розрізненими артефактами. Вони утворюють спільну систему постановки задачі, у якій UX-модель задає очікувану взаємодію, а технічна реалізація гарантує її стабільне відтворення у вебзастосунку. Ілюстративні матеріали до цього переходу подано в додатку Д, а ключові фрагменти реалізації - у додатку Г.

Критерії оцінювання UX-рішень у межах MVP AptekaNear

Для оцінювання UX-рішень у межах MVP AptekaNear доцільно визначити набір критеріїв, пов’язаних із ключовими сценаріями користувача. Основними показниками є швидкість знаходження потрібного препарату, зрозумілість результатів пошуку, кількість кроків до бронювання, помітність основної дії, якість

повідомлень про помилки та можливість продовжити роботу після невдалого запиту [21, 24, 30].

Перший критерій - ефективність проходження сценарію. Користувач повинен мати можливість перейти від введення назви препарату до вибору аптеки без зайвих переходів і повторного введення даних. Другий критерій - зрозумілість інформації: ціна, наявність, адреса, відстань і графік роботи мають бути представлені так, щоб користувач міг швидко порівняти варіанти [11, 23, 24].

Третій критерій - стійкість інтерфейсу до помилок. Система має пояснювати, чому пошук не дав результату, як виправити некоректне введення, що робити за відсутності геолокації та чи можна продовжити сценарій без додаткових дозволів. Це безпосередньо пов’язано з евристиками юзабіліті та рекомендаціями з доступності [14, 23, 31].

Четвертий критерій - технічна якість взаємодії. Для вебзастосунку важливо контролювати швидкість завантаження, стабільність макета і реакцію на дії користувача. Тому UI/UX-оцінювання має враховувати не лише вигляд екранів, а й фронт-енд показники, зокрема адаптивність і базові метрики Core Web Vitals [21, 22, 26].

2.4 Висновки по розділу

У другому розділі виконано аналіз предметної області та сформовано вимоги до вебзастосунку AptekaNear. Визначено основну ідею проєкту - створення сервісу для швидкого пошуку лікарських засобів, перегляду їх наявності в аптеках, порівняння цін і оформлення бронювання. Для такого сервісу ключовими є швидке отримання результату, зрозуміла навігація, коректне подання інформації та достовірність даних.

Виділено основні групи користувачів вебзастосунку: незареєстровані користувачі, зареєстровані користувачі, а також адміністратори й представники аптек. Для кожної групи визначено потреби й ключові пріоритети взаємодії з системою. На підставі проведеного аналізу сформовано функціональні та

нефункціональні вимоги до MVP. До функціональних належать: пошук препарату, перегляд результатів, фільтрація, карта аптек, бронювання та профіль користувача; до нефункціональних - продуктивність, доступність, безпека й надійність.

Розглянуто UX-модель інтерфейсу та її зв'язок з технічною реалізацією. Показано, що wireframe-прототип, користувацькі сценарії та вимоги до системи узгоджені зі структурою фронт-енд застосунку. Результати розділу стали основою для проєктування архітектури, компонентів, API-взаємодії та тестування вебзастосунку AptekaNear.

РОЗДІЛ 3. ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ APTEKANEAR

3.1 Інтерфейсні рішення та компонентна організація фронт-енду

Побудова фронт-енд частини починається з інформаційної архітектури - системи правил, за якою користувач орієнтується у вмісті продукту. Для AptekaNear центральним об’єктом є препарат, а допоміжними контекстами виступають аптека, локація, бронювання та профіль користувача. Така модель дозволяє вибудувати навігацію навколо найбільш цінного для користувача запиту: знайти потрібний препарат і швидко перейти до дії.

У структурі екранів доцільно виділити головний екран пошуку, екран результатів, картковий перегляд об’єкта, форму бронювання, екран підтвердження, сторінки авторизації та профіль. На практиці це означає, що маршрути фронт-енду повинні відображати саме ці сценарії, а не внутрішню структуру серверних модулів. Такий підхід підтримує ментальну модель користувача і дає змогу скоротити кількість навігаційних помилок.

У результатах пошуку важливо поєднати щонайменше два режими: список та карту. Список потрібен для швидкого порівняння цін і статусів, а карта - для оцінки зручності розташування. З UX-погляду ці режими не повинні бути ізольованими: користувач має перемикатися між ними, не втрачаючи вибраного запиту, фільтрів та контексту. Для фронт-енд архітектури це означає спільне керування станом пошуку і відображення різних представлень одного набору даних.

Таким чином, інформаційна архітектура AptekaNear формує основу не лише для дизайну, а й для компонентної структури фронт-енду: маршрутів, шаблонів сторінок, контейнерів даних, локального та глобального стану.

Компонентна організація фронт-енду дає змогу узгодити структуру екранів AptekaNear з основними користувацькими сценаріями: пошуком, переглядом результатів, переходом до картки препарату та бронюванням. Завдяки цьому кожен екран розглядається не як окрема сторінка, а як частина єдиного маршруту

взаємодії, де компоненти, стани та переходи підтримують послідовний користувацький досвід.

Узагальнені дані наведено в табл. 3.1.

Таблиця 3.1

Основні екрани та їх призначення

Екран	Призначення	Основні елементи
Головний екран	Запуск сценарію пошуку	Поле пошуку, підказки, швидкі дії
Результати пошуку	Порівняння знайдених варіантів	Список карток, фільтри, сортування, перемикач режиму
Карта аптек	Оцінка розташування	Маркери, коротка інформація, синхронізація зі списком
Картка препарату	Перегляд деталей	Опис, форма випуску, ціна, наявність, кнопка бронювання
Форма бронювання	Завершення дії	Поля контакту, вибір кількості, валідація
Підтвердження	Пояснення результату	Номер бронювання, термін дії, подальші дії
Профіль	Повернення до історії та налаштувань	Мої бронювання, обране, нагадування

Дизайн-система як інструмент поєднання UI та коду

Для підтримання цілісності інтерфейсу доцільно використовувати дизайн-систему - набір повторно застосовуваних правил і компонентів, що синхронізує роботу дизайну та фронт-енду. Для AptekaNear дизайн-система може включати палітру кольорів, типографічну шкалу, сітку відступів, стилі кнопок, полів, карток, бейджів статусів, повідомлень і модальних вікон. Такий підхід зменшує кількість випадкових рішень і робить інтерфейс передбачуваним.

З технічної точки зору дизайн-система має бути представлена у фронт-енді через набір токенів: кольори, розміри шрифтів, радіуси, тіні, відступи, брейкпоінти, значення для фокусних рамок та станів компонента. Токени дозволяють змінювати візуальні характеристики централізовано та забезпечують консистентність між екранами. Це особливо важливо для вебсервісів, які з часом розширюються новими модулями [8, 14, 20].

У межах AptekaNear доцільно виділити декілька базових компонентів: search bar, result card, filter panel, map toggle, detail section, status badge, primary/secondary button, input field, notification block, reservation summary.

Кожен із цих компонентів повинен мати опис станів: звичайний, наведення, активний, вимкнений, завантаження, успіх, помилка. Саме повний опис станів робить дизайн-систему придатною для реальної фронт-енд реалізації.

Особливого значення набувають статусні елементи. У сервісі, де присутні стани «в наявності», «мало», «немає», «бронювання підтверджено», «запит виконується», не можна покладатися лише на колір. Потрібні також текстові маркери, іконографіка або форма бейджа. Це одночасно підвищує доступність і зменшує ризик неправильного прочитання даних користувачем.

Отже, дизайн-система у цій роботі розглядається як зв’язувальна ланка між UI/UX-аналізом і фронт-енд кодом. Вона дозволяє перетворити дизайнерські принципи на набір технічно відтворюваних рішень.

Узагальнені дані наведено в табл. 3.2.

Таблиця 3.2

Приклад базових компонентів дизайн-системи

Компонент	**Призначення**	**Ключові стани**
Primary button	Основна цільова дія	default, hover, focus, disabled, loading
Input field	Введення даних користувача	default, focus, valid, error, disabled
Result card	Подання знайденого препарату	normal, selected, compact, empty data
Status badge	Відображення наявності	available, low stock, unavailable
Notification block	Зворотний зв’язок системи	success, warning, error, info

Адаптивність і доступність інтерфейсу

Оскільки значна частина користувачів виконує пошук і бронювання з мобільного пристрою, для AptekaNear логічним є mobile-first підхід. На малому екрані на першому плані повинні бути поле пошуку, короткий список результатів, кнопки сортування та переходу до карти. Вторинні дії - деталізація, профіль, історія - мають бути доступні, але не повинні конкурувати з головним сценарієм.

Після цього інтерфейс може поступово розширюватися для планшетів і десктопів завдяки багатоколонковим макетам та розширеному блоку фільтрів [6, 16, 22, 26].

Адаптивність не зводиться до змін ширини елементів. У різних брейкпоінтах слід переглядати саму логіку компонування: які блоки залишаються поруч, що переноситься нижче, які підписи скорочуються, як подається допоміжна інформація. Наприклад, на десктопі можна одночасно показати список і карту, тоді як на мобільному доцільніше використовувати перемикач режиму. Таке рішення напряму впливає на фронт-енд архітектуру контейнерів і компонентів.

Доступність інтерфейсу передбачає, що всі основні дії повинні бути доступними з клавіатури, а форма бронювання - мати семантичні підписи, асоційовані з полями. Повідомлення про помилку мають розташовуватися поряд із відповідним полем, а фокус після надсилання невдалої форми бажано повертати до першого проблемного елемента. Для інтерактивних карт або складних компонентів необхідно передбачати альтернативний шлях отримання тієї самої інформації, наприклад через список.

Таким чином, адаптивність і доступність є не окремими додатками до дизайну, а невід'ємними вимогами до фронт-енд реалізації. Їх урахування покращує не лише інклюзивність, а й загальну якість сценаріїв взаємодії.

3.2 Архітектура, технології та інтеграційна логіка системи

З огляду на обсяг функціоналу та потребу у подальшому розвитку для AptekaNear доцільною є клієнт-серверна багаторівнева архітектура. Вона включає рівень представлення (front-end UI), рівень бізнес-логіки (backend API), рівень доступу до даних (repository/data access) та рівень зберігання даних. Така структура дає змогу розділити відповідальності між компонентами, спростити тестування і підтримати поступовий перехід від mock-даних до повноцінної бази даних.

Для теми бакалаврської роботи важливо, що архітектура безпосередньо підтримує UX-рішення. Фронт-енд отримує передбачувані REST-контракти, може чітко обробляти стани запиту, а серверна частина зосереджується на валідації,

бізнес-правилах і формуванні стабільних відповідей. Коли межі між шарами є прозорими, значно легше реалізувати сталі повідомлення про помилки, повторні спроби запиту та інформаційні стани інтерфейсу.

Модель даних AptekaNear має підтримувати сценарії пошуку, перегляду пропозицій, бронювання, авторизації та персоналізації. Базовими сутностями є препарат, аптека, наявність/ціна, користувач, бронювання, обране та нагадування. Важливо, що сутність availability з'єднує препарат і аптеку та містить інформацію про ціну, залишок і статус. Саме ці поля визначають, як дані будуть подані у фронт-енді.

У контексті UI/UX модель даних повинна бути достатньо виразною, щоб інтерфейс не був змушений «здогадуватися» про стани. Наприклад, статус наявності має бути окремим полем, а не виводитися лише з кількості; у бронюванні потрібні код і термін дії; для профілю важливі часові позначки та зв'язок із користувачем. Така деталізація покращує якість відображення і спрощує реалізацію компонентів.

Узагальнені дані наведено в табл. 3.3.

Таблиця 3.3

Основні сутності даних проєкту

Сутність	**Ключові поля**	**Призначення**
Medicine	id, name, dosage, form, manufacturer, description	Довідкова інформація про препарат
Pharmacy	id, name, address, contacts, work_time, lat, lng	Відомості про аптеку та її місцезнаходження
Availability	medicine_id, pharmacy_id, price, quantity, status, updated_at	Зв'язок препарату з аптекою, ціна й стан наявності
User	id, name, email/phone, password_hash	Обліковий запис користувача
Reservation	id, user_id, medicine_id, pharmacy_id, reserve_code, expires_at, status	Факт бронювання та його стан
Favorite / Reminder	user_id, medicine_id, created_at / datetime	Персоналізація взаємодії

Вибір технологій для реалізації

Вибір технологій у вебзастосунку визначається вимогами функціоналу,

масштабом користувацьких сценаріїв і потребою в стабільній взаємодії з API. Для AptekaNear доцільним є використання компонентного фронт-енд стеку з підтримкою маршрутизації, станів інтерфейсу, адаптивної верстки та асинхронної роботи з даними.

Серверну частину навчального проєкту реалізовано на NestJS і TypeScript, що узгоджується з фронт-енд підходом завдяки спільній типізації, модульності, Dependency Injection і зручній побудові REST API. Це полегшує інтеграцію клієнтської частини та зменшує кількість помилок у контрактах між шарами системи.

На ранньому етапі роботи з даними доцільно використовувати seed- або mock-репозиторії, щоб перевіряти сценарії пошуку й бронювання без залежності від завершеної інфраструктури. Надалі такий підхід може бути замінений повноцінним шаром доступу до БД без зміни контрактів, видимих фронт-енду.

Допоміжними технологіями є Git для керування версіями, Figma для прототипування, засоби логування, моніторингу помилок і автоматичної перевірки якості коду. Такий набір підтримує технічну стабільність системи та послідовність користувацького досвіду.

Узагальнені дані наведено в табл. 3.4.

Таблиця 3.4

Узагальнений технологічний стек проєкту

Рівень системи	**Технологічне рішення**	**Обґрунтування**
Front-end	React, Vite, TypeScript, Tailwind CSS, React Router; компонентний підхід, адаптивна верстка	Гнучка робота зі станом інтерфейсу, адаптивність, інтерактивність
Прототипування	Figma	Швидка перевірка структури екранів і навігації
Back-end	NestJS + TypeScript	Модульність, DI, зручна побудова REST API
Дані	InMemory / seed-дані з перспективою БД	Швидкий старт MVP і подальше масштабування
Якість	Git, lint, build, логування, аналітика помилок	Контроль змін, стабільність і підтримуваність

Взаємодія фронт-енду з REST API

Ефективний UX у вебзастосунку значною мірою залежить від того, наскільки стабільно і передбачувано клієнтська частина взаємодіє з API. Для AptekaNear базові операції охоплюють пошук препаратів, отримання списку аптек із цінами і статусами, перегляд картки препарату, створення бронювання, авторизацію та роботу з історією користувача. З точки зору фронт-енду критично, щоб кожен endpoint повертав узгоджену структуру даних і кодів стану.

Для UX важливо проєктувати не лише успішні відповіді, а й помилкові сценарії. Якщо API повертає повідомлення про відсутність препарату, блокування доступу або помилку валідації, фронт-енд повинен відобразити це так, щоб користувач розумів наступну дію: змінити запит, виправити поле форми, повторити спробу або увійти в систему. Отже, UX починається ще на рівні контрактів між клієнтом і сервером.

Саме тому у роботі підтримується ідея централізованого шару доступу до API у клієнтській частині. Такий шар приховує технічні деталі запитів, уніфікує обробку помилок і дає змогу повторно використовувати логіку в різних екранах. Це знижує зв’язність компонентів і підвищує підтримуваність фронт-енду.

Компонентна організація фронт-енду

У компонентно-орієнтованому фронт-енді інтерфейс розглядається як система повторно використовуваних блоків, кожен із яких має чітко визначені вхідні дані, стани та події. Для AptekaNear доцільно виділити сторінкові компоненти (SearchPage, ResultsPage, MedicineDetailsPage, ReservationPage, ProfilePage) та менші повторні елементи (SearchBar, ResultCard, FiltersPanel, StatusBadge, ReservationForm, EmptyState, ErrorBlock). Такий підхід напряму підтримує принцип консистентності UI та робить UX передбачуванішим.

Компоненти слід розділяти на презентаційні та контейнерні. Презентаційні елементи відповідають за відображення і мають бути максимально простими, тоді як контейнерні керують отриманням даних, викликами API, маршрутизацією та передаванням подій. Це дозволяє уникнути надмірної логіки в елементах інтерфейсу і спрощує тестування окремих частин фронт-енду.

Для критичних сценаріїв, таких як пошук і бронювання, важливо окремо моделювати стани інтерфейсу: початковий, завантаження, успіх, порожній результат, помилка, часткова недоступність даних. Коли такі стани визначені на рівні компонентів, користувач не залишається без пояснення того, що відбувається. Це один із ключових способів інтеграції UX-принципів безпосередньо у фронт-енд код.

Компонентна організація також сприяє масштабуванню проєкту. Додавання нових екранів або функцій - наприклад, нагадувань чи розширеного кабінету аптеки - відбувається не через дублювання верстки й логіки, а через повторне використання готових блоків і шаблонів. Це позитивно впливає як на швидкість розвитку продукту, так і на стабільність інтерфейсу.

Практичний аспект такого підходу підтверджується матеріалами проєкту AptekaNear: фрагменти реалізації Dependency Injection і базового репозиторного шару, які забезпечують передбачувану взаємодію між компонентами системи, наведено в додатку Г. Винесення цих фрагментів у додатки дозволяє не перевантажувати основний текст кодом, але водночас показує, як архітектурні рішення підтримують стабільний користувацький досвід.

Роль серверної частини у забезпеченні якісного UX

Хоча тема роботи зосереджена на фронт-енд веброзробці, якість користувацького досвіду неможливо розглядати у відриві від серверної частини. Якщо API має несталу структуру відповіді, невалідовані вхідні дані або суперечливі статуси, фронт-енд не зможе сформувати зрозумілий інтерфейс. У матеріалах проєкту AptekaNear серверну частину побудовано на модульній основі з використанням ланцюжка Controller → Service → Repository, що створює передумови для передбачуваної взаємодії з клієнтом.

Окрему роль відіграє Dependency Injection. Завдяки DI залежності не створюються жорстко всередині сервісів, а передаються через контейнер. Це дає змогу змінювати реалізації репозиторіїв, спрощує тестування і підтримує принцип слабкого зв’язку між модулями. Для користувацького досвіду це має опосередковане значення, оскільки стабільна й протестована серверна логіка

зменшує ризик непередбачуваних збоїв у ключових сценаріях взаємодії.

Використання InMemory-репозиторію на етапі MVP відповідає цілям навчального проєкту, оскільки дозволяє перевірити логіку пошуку, структуру відповідей і поведінку клієнта ще до повного розгортання інфраструктури. При цьому правильно спроєктований контракт репозиторію робить майбутню заміну джерела даних технічно безпечною для фронт-енду.

Таким чином, серверна архітектура в AptekaNear розглядається як інженерна опора для якісного UX. Вона забезпечує сталі бізнес-правила, контроль даних, повторне використання логіки та основу для тестування інтеграційних сценаріїв.

Використання патернів проєктування

У межах проєкту AptekaNear на етапі проєктування передбачено застосування патернів Facade, Strategy і Factory Method: фасад зводить багатокроковий сценарій бронювання до єдиного спрощеного інтерфейсу, стратегія забезпечує гнучке сортування пропозицій аптек за ціною, відстанню та релевантністю, а фабричний метод інкапсулює створення каналів сповіщення. Ці рішення обрано не як формальну «демонстрацію патернів», а як практичні засоби спрощення критичних користувацьких сценаріїв: для фронт-енду вони означають чистіші API-контракти та передбачувані відповіді сервера. Детальний опис програмної реалізації цих патернів із фрагментами коду наведено в підрозділі 4.2 та додатку Г.

Складання та запуск застосунку автоматизовано за допомогою стандартних інструментів екосистеми Node.js: перелік залежностей описано у файлі package.json, а параметри середовища винесено у змінні оточення. Такий підхід відокремлює конфігурацію від коду, спрощує розгортання в різних середовищах і відповідає принципам дванадцятифакторного застосунку. Запуск у режимі розробки та у промисловому середовищі відрізняється лише набором змінних оточення, що зменшує ризик помилок під час переходу між середовищами.

Поєднання статичної типізації, централізованого логування та моніторингу формує єдиний контур спостережуваності (observability) застосунку. Завдяки

цьому команда розробників отримує своєчасні сповіщення про помилки у промисловому середовищі, а також структуровані журнали для відтворення проблемних сценаріїв. Це скорочує час реагування на дефекти та підвищує довіру користувачів до стабільності роботи вебзастосунку.

Серверну частину реалізовано на платформі Node.js із застосуванням фреймворку NestJS і мови TypeScript; як HTTP-шар використано адаптер Express. NestJS обрано за вбудовану підтримку модульності, механізму впровадження залежностей і декораторів, що дозволяє чітко відокремити контролери, сервіси та доступ до даних. Така багаторівнева організація з поділом на контролер, сервіс і репозиторій спрощує тестування та подальший розвиток системи.

Взаємодія між фронт-ендом і серверною частиною побудована на стабільних REST-маршрутах, які повертають дані у форматі JSON. У межах навчального прототипу дані зберігаються у пам’яті (seed-набір препаратів і пропозицій аптек), тому окрема база даних для запуску не потрібна; натомість доступ до даних інкапсульовано за абстрактним інтерфейсом сховища, що згодом дозволить замінити реалізацію на повноцінну базу даних без зміни решти коду. Сховище й кешування залишено для подальшого розвитку.

3.3 Проєктування моделі даних та API-взаємодії вебзастосунку

Проєктування моделі даних є невід’ємною частиною загального проєктування вебзастосунку, оскільки структура збережених сутностей безпосередньо визначає, які дані фронт-енд зможе відобразити користувачеві та з якою деталізацією. Для AptekaNear модель даних має підтримувати ключові сценарії: пошук препарату, перегляд пропозицій аптек, порівняння цін і бронювання. Основними сутностями навчального прототипу є препарат (Medicine) з ідентифікатором і назвою, пропозиція аптеки (PharmacyOffer), що поєднує аптеку, ціну, відстань і залишок, та бронювання (Reservation). Користувач у поточній реалізації представлений ідентифікатором і контактом у межах бронювання, а повноцінний профіль, обране й нагадування віднесено до напрямів подальшого розвитку.

Ключовою для користувацького досвіду є пропозиція аптеки, оскільки саме вона містить ціну, відстань і залишок препарату. Ці значення дають змогу інтерфейсу відображати у списку результатів актуальні дані без додаткових обчислень на клієнті. На основі залишку інтерфейс може формувати зрозумілий статус наявності, щоб стан позиції був явним і однозначним, а не виводився користувачем лише з числа.

Проєктування API-взаємодії передбачає визначення стабільних REST-контрактів між фронт-ендом і серверною частиною.

У навчальному прототипі реалізовано базові маршрути для основного сценарію: пошук препаратів, отримання пропозицій аптек і створення бронювання. Ширший перелік спроєктованих API-операцій (зокрема перегляд картки препарату, автентифікацію та перегляд історії бронювань) наведено в додатку Б як орієнтир для подальшого розвитку. Узгодженість контрактів дозволяє фронт-енду передбачувано обробляти стани завантаження, успіху та помилки, що є передумовою якісного користувацького досвіду.

Важливим аспектом проєктування є чітке розмежування відповідальності між рівнями системи. Рівень представлення відповідає за відображення даних і реагування на дії користувача; рівень бізнес-логіки - за валідацію, бізнес-правила та формування відповідей; рівень доступу до даних - за роботу зі сховищем. Така багаторівнева організація, узгоджена з обраною клієнт-серверною архітектурою (підрозділ 3.2), спрощує тестування й дозволяє замінювати реалізацію сховища без зміни клієнтської частини.

Отже, проєктування моделі даних і API-контрактів є сполучною ланкою між UX-моделлю інтерфейсу та програмною реалізацією. Воно гарантує, що інтерфейсні рішення, описані у підрозділах 3.1-3.2, матимуть надійну технічну основу, а фронт-енд отримуватиме дані у формі, придатній для прямого відображення користувачеві.

Узгоджена модель даних і стабільні API-контракти безпосередньо впливають на побудову фронт-енд частини. Роботу з визначеними endpoint доцільно інкапсулювати в централізованому API-шарі, який уніфікує формування

запитів та обробку помилок. Завдяки цьому компоненти інтерфейсу не звертаються до мережевих викликів напряму, а оперують готовими даними та станами, що зменшує дублювання логіки й спрощує супровід клієнтської частини.

Кожному запиту до сервера у фронт-енді відповідає чітко визначений набір станів: завантаження, успішне отримання даних, порожній результат і помилка. Наприклад, під час пошуку препаратів інтерфейс спершу показує індикатор завантаження, далі - список знайдених позицій або повідомлення про відсутність результатів, а в разі збою - зрозуміле повідомлення з можливістю повторити дію.

Така узгодженість між моделлю даних і станами інтерфейсу є передумовою передбачуваного користувацького досвіду й відповідає чек-листу UI/UX, наведеному в додатку В.

Спроєктована модель даних безпосередньо відображається на основні екрани системи та сценарії взаємодії. Сутність препарату використовується на екранах пошуку та деталізації, пропозиція аптеки - у списку результатів (а в перспективі - і на карті), а бронювання - у формі бронювання. Таке відображення гарантує, що ключові користувацькі сценарії (пошук, перегляд результатів, перехід до деталей і бронювання), описані в додатку А, мають підтримку на рівні даних і не потребують додаткових перетворень на клієнті.

Окрему увагу під час проєктування приділено цілісності та актуальності даних про ціни й наявність, оскільки саме вони найбільше впливають на довіру користувача. Ціна та статус наявності зберігаються разом у межах сутності наявності й оновлюються узгоджено, що унеможливлює ситуацію, коли інтерфейс показує неактуальний статус. Базову валідацію вхідних даних (коректність ідентифікаторів, обов’язкові поля форми бронювання) виконує рівень бізнес-логіки, що убезпечує систему від некоректних запитів і покращує якість зворотного зв’язку для користувача.

Важливою частиною проєктування API-взаємодії є узгоджена обробка помилок. Для типових ситуацій (некоректний запит, відсутність ресурсу, помилка автентифікації, внутрішня помилка сервера) передбачено стандартні коди відповіді та уніфікований формат повідомлення про помилку. Завдяки цьому фронт-енд

може єдиним чином опрацьовувати помилки різних endpoint і показувати користувачеві зрозумілі підказки замість технічних деталей.

Оскільки пошук препаратів і перегляд пропозицій аптек можуть повертати велику кількість результатів, на рівні API передбачено параметри сортування та обмеження обсягу вибірки. Сортування за ціною, відстанню або наявністю задається окремим параметром запиту, що дозволяє фронт-енду змінювати порядок відображення без повторного завантаження всього сценарію. Такий підхід зменшує обсяг переданих даних і пришвидшує реакцію інтерфейсу.

Окремо спроєктовано формат запиту на бронювання: він містить ідентифікатори препарату й аптеки, кількість, дані користувача та бажаний канал сповіщення. Перед створенням бронювання серверна частина перевіряє коректність цих даних (зокрема невід’ємну кількість і достатній залишок у вибраній аптеці), що убезпечує систему від некоректних запитів і дає фронт-енду однозначний результат для відображення.

Між основними сутностями моделі даних установлено прості, але достатні для основного сценарію зв'язки: одному препарату може відповідати кілька пропозицій аптек, а кожне бронювання пов'язується з конкретним препаратом і аптекою за їхніми ідентифікаторами. Такий підхід дозволяє послідовно побудувати сценарій взаємодії користувача: від пошуку препарату до перегляду пропозицій і подальшого створення бронювання. Завдяки цьому API-взаємодія залишається зрозумілою для клієнтської частини та не потребує надмірного ускладнення моделі даних.

Особливу увагу приділено узгодженості значень між сутностями, оскільки саме вони забезпечують коректне відображення інформації на стороні інтерфейсу: ціна, відстань і залишок належать до конкретної пропозиції аптеки й передаються разом із нею. Така організація даних знижує ризик розбіжностей між результатами пошуку, списком пропозицій і станом бронювання, який бачить користувач.

3.4 Висновки по розділу

У третьому розділі спроєктовано вебзастосунок AptekaNear: визначено структуру основних екранів, компонентну організацію фронт-енду, модель даних і REST API для взаємодії із серверною частиною.

Обґрунтовано використання клієнт-серверної архітектури та дизайн-системи, що забезпечує узгоджене відображення даних і підтримку ключових користувацьких сценаріїв. У підсумку показано, що UI/UX-рішення потрібно враховувати вже на етапі проєктування, оскільки вони впливають на зручність і подальший розвиток вебзастосунку.

РОЗДІЛ 4. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ APTEKANEAR

У цьому розділі інтерфейсні рішення, спроєктовані в розділі 3, доведено до працездатного вебзастосунку AptekaNear. Спершу розглянуто серверну основу: організацію програмного коду, впровадження залежностей, застосування патернів проєктування та реалізацію логування й обробки помилок; цю частину виконано з використанням платформи Node.js та фреймворку NestJS, що підтримує модульну структуру й механізм впровадження залежностей [7, 17, 25]. Центральне місце в розділі посідає реалізація клієнтської частини (підрозділ 4.4), де принципи UI/UX-дизайну втілено в конкретних сторінках, компонентах і станах інтерфейсу. Вибрані фрагменти програмного коду наведено в додатку Г, а екранні форми реалізованого вебзастосунку - в додатку Е.

4.1 Організація програмного коду та структура проєкту

Якість і підтримуваність вебзастосунку значною мірою визначаються організацією програмного коду. Для AptekaNear застосовано модульну структуру на основі NestJS: основну предметну логіку зосереджено в модулі medicines, а спільні службові засоби - у каталозі common. У межах модуля medicines виділено окремі сервіси за відповідальністю - пошук препаратів, роботу з пропозиціями аптек, бронювання та сповіщення, - що зменшує зв’язність між частинами системи й дозволяє розвивати їх відносно незалежно.

Структура коду відображає предметні межі системи: контролер модуля medicines приймає HTTP-запити й делегує їх відповідним сервісам (пошуку, доступності, бронювання та сповіщень), а доступ до даних інкапсульовано за абстрактним інтерфейсом сховища. Спільні елементи - глобальний фільтр винятків, службу логування та інтеграцію моніторингу - винесено в каталог common. Дотримання єдиних стандартів найменування файлів, класів та інтерфейсів спрощує орієнтацію в коді, а відсутність бізнес-логіки в контролерах робить її зосередженою в сервісах.

Розмежування відповідальності між контролерами, сервісами та сховищами

безпосередньо підтримує архітектурні рішення, описані в підрозділі 3.2. Контролер приймає запит, делегує його сервісу, а сервіс взаємодіє зі сховищем через абстрактний інтерфейс. Це дозволяє замінювати реалізацію сховища (наприклад, перейти від тимчасового сховища в пам’яті до повноцінної бази даних) без зміни решти коду.

Узагальнені дані наведено в табл. 4.1.

Таблиця 4.1

Основні складові реалізації вебзастосунку AptekaNear

Складова	Призначення	Основні елементи
medicines	Основна предметна логіка	Контролер і сервіси пошуку, доступності, бронювання, сповіщень
strategies	Сортування пропозицій аптек	Стратегії за ціною, відстанню, релевантністю та контекст
facades	Спрощення сценарію бронювання	Фасад бронювання (ReservationFacade)
notifications	Імітація сповіщень	Відправники email/SMS і фабрика каналів
repositories	Доступ до даних препаратів	Інтерфейс сховища та реалізація в пам’яті
common	Спільні службові засоби	Фільтр винятків, логування, інтеграція Sentry

4.2 Впровадження залежностей та застосування патернів проєктування

Для зменшення зв’язності між компонентами системи у проєкті застосовано механізм впровадження залежностей (Dependency Injection). Замість того щоб сервіс самостійно створював конкретну реалізацію сховища, потрібну залежність передає контейнер впровадження залежностей. Це дозволяє підмінювати реалізацію (зокрема для тестування) і дотримуватися принципу інверсії залежностей.

Реєстрацію залежності в модулі medicines показано на лістингу нижче: сервіс отримує реалізацію сховища через токен MEDICINE_REPOSITORY, а конкретний клас (InMemoryMedicineRepository) задається в конфігурації модуля.

```
// medicines.module.ts
providers: [
```

```
  MedicinesService,
  { provide: MEDICINE_REPOSITORY, useClass: InMemoryMedicineRepository },
];

// medicines.service.ts
constructor(
  @Inject(MEDICINE_REPOSITORY)
  private readonly medicineRepository: MedicineRepository,
) {}
```

Для спрощення складних сценаріїв застосовано патерн Facade. Сценарій бронювання препарату передбачає послідовне звернення до кількох сервісів: отримання даних про препарат, пошук пропозиції аптеки, створення бронювання та надсилання підтвердження. Винесення цієї послідовності у фасад дозволяє контролеру працювати з одним зрозумілим методом, не знаючи про внутрішню взаємодію сервісів (повне порівняння «до/після» наведено в додатку Г).

```
// Контролер після впровадження Facade
@Post('reserve')
reserve(@Body() dto: ReserveMedicineDto) {
  return this.reservationFacade.reserveMedicine(dto);
}
```

Патерн Strategy застосовано для гнучкого сортування пропозицій аптек. Замість умовної логіки з багатьма гілками алгоритми сортування (за ціною, відстанню, наявністю) винесено в окремі стратегії, а вибір стратегії визначається параметром sortBy. Це дозволяє фронт-енду працювати з єдиним параметром сортування і спрощує додавання нових критеріїв без зміни наявного коду.

```
// Сортування пропозицій через Strategy
searchOffers(medicineId: number, sortBy: OfferSortType = 'relevance') {
  const offers = this.availabilityService.getOffersByMedicine(medicineId);
  return this.sortContext.sort(offers, sortBy);
}
```

Додатково у проєкті використано патерн Factory Method для централізованого створення об’єктів за визначеними правилами. Застосування патернів проєктування підвищує гнучкість і підтримуваність системи, зменшує дублювання коду та робить взаємодію між компонентами передбачуванішою, що позитивно впливає на стабільність роботи інтерфейсу.

Усі чотири патерни застосовано узгоджено й у межах одного модуля: впровадження залежностей задає спосіб отримання реалізацій, фабрика обирає канал сповіщення, стратегія визначає порядок сортування пропозицій, а фасад об’єднує ці кроки в один зрозумілий сценарій бронювання.

Завдяки цьому контролер залишається тонким, а зміна окремого алгоритму (наприклад, додавання нового критерію сортування) не зачіпає решту коду.

4.3 Реалізація логування, обробки помилок та контролю якості коду

Стабільність користувацького досвіду залежить не лише від коректної бізнес-логіки, а й від того, наскільки система здатна виявляти й опрацьовувати помилки. Тому у вебзастосунку реалізовано централізоване логування дій користувача та глобальну обробку винятків, що дозволяє відстежувати проблеми ще до того, як вони вплинуть на взаємодію.

Для моніторингу помилок інтегровано службу Sentry. Її ініціалізацію винесено в окремий сервіс, який активується лише за наявності змінної середовища SENTRY_DSN, тож у середовищі розробки моніторинг можна вимкнути. Глобальний фільтр винятків перехоплює помилки, фіксує їх у журналі та передає до Sentry.

```
// src/common/sentry/sentry.service.ts
@Injectable()
export class SentryService {
  private readonly logger = new Logger(SentryService.name);
  init(): void {
    const dsn = process.env.SENTRY_DSN;
    if (!dsn) {
      this.logger.log('SENTRY_DSN is not set. Sentry is disabled.');
      return;
    }
    Sentry.init({ dsn });
    this.logger.log('Sentry initialized.');
  }
}
```

Логування ключових дій (наприклад, пошуку препаратів із зазначенням запиту та кількості результатів) дає змогу аналізувати поведінку користувачів у критичних сценаріях і виявляти вузькі місця інтерфейсу. Це безпосередньо

підтримує процес оцінювання UX, оскільки об’єктивні дані про дії користувачів доповнюють експертну оцінку зручності.

```
@Get('search')
search(@Query('q') q = '') {
  const results = this.medicinesService.search(q);
  this.logger.log(`User action: search medicines (q="${q}", results=${results.length})`);
  return { q, results };
}
```

Для контролю якості коду застосовано статичну типізацію (TypeScript), що дозволяє виявляти частину помилок ще на етапі компіляції, а також єдині правила форматування й перевірки. Така дисципліна знижує ймовірність дефектів і спрощує супровід вебзастосунку [28]. Додатково на вхідні дані застосовано декларативну валідацію через глобальний ValidationPipe і декоратори бібліотеки class-validator: некоректні запити (відсутні обов’язкові поля, від’ємна кількість тощо) відхиляються з кодом 400 ще до бізнес-логіки.

Глобальний фільтр винятків перехоплює необроблені помилки, фіксує метод, маршрут і код відповіді в журналі та, за наявності активного моніторингу, передає подію до Sentry. Якщо змінну середовища SENTRY_DSN не задано, моніторинг вимкнено, і застосунок працює у звичайному режимі без зовнішніх залежностей. Email- та SMS-сповіщення в навчальному прототипі імітуються через запис у журнал, що дозволяє перевіряти сценарії сповіщення без реальної відправки.

4.4 Реалізація клієнтської частини (фронтенд)

Клієнтську частину вебзастосунку AptekaNear реалізовано як багатосторінковий вебзастосунок на основі бібліотеки React, складальника Vite та мови TypeScript; стилізацію виконано за допомогою CSS-фреймворку Tailwind, а навігацію між сторінками - засобами React Router. Структура сторінок відтворює природний шлях користувача, спроєктований у розділі 3: з головної сторінки користувач починає пошук препарату, у каталозі та результатах пошуку звужує вибір, на картці препарату порівнює пропозиції аптек, у кошику збирає потрібні

позиції, на кроці бронювання підтверджує намір і отримує екран підтвердження, а інформаційна сторінка пояснює призначення сервісу. Така організація відповідає принципу відповідності системи реальному завданню: кожна сторінка є окремим кроком сценарію, а не довільним пунктом меню.

Інтерфейс побудовано за компонентним принципом із поділом на сторінкові та повторно використовувані презентаційні компоненти: картка препарату, бейдж наявності, рядок пропозиції аптеки, кнопка додавання в кошик, індикатори станів інтерфейсу. Повторне використання тих самих компонентів на різних сторінках забезпечує візуальну й поведінкову консистентність: однакові елементи скрізь виглядають і реагують однаково, що зменшує зусилля користувача на освоєння сервісу та ризик помилкових дій. Головну сторінку реалізованого вебзастосунку наведено на рисунку 4.1.

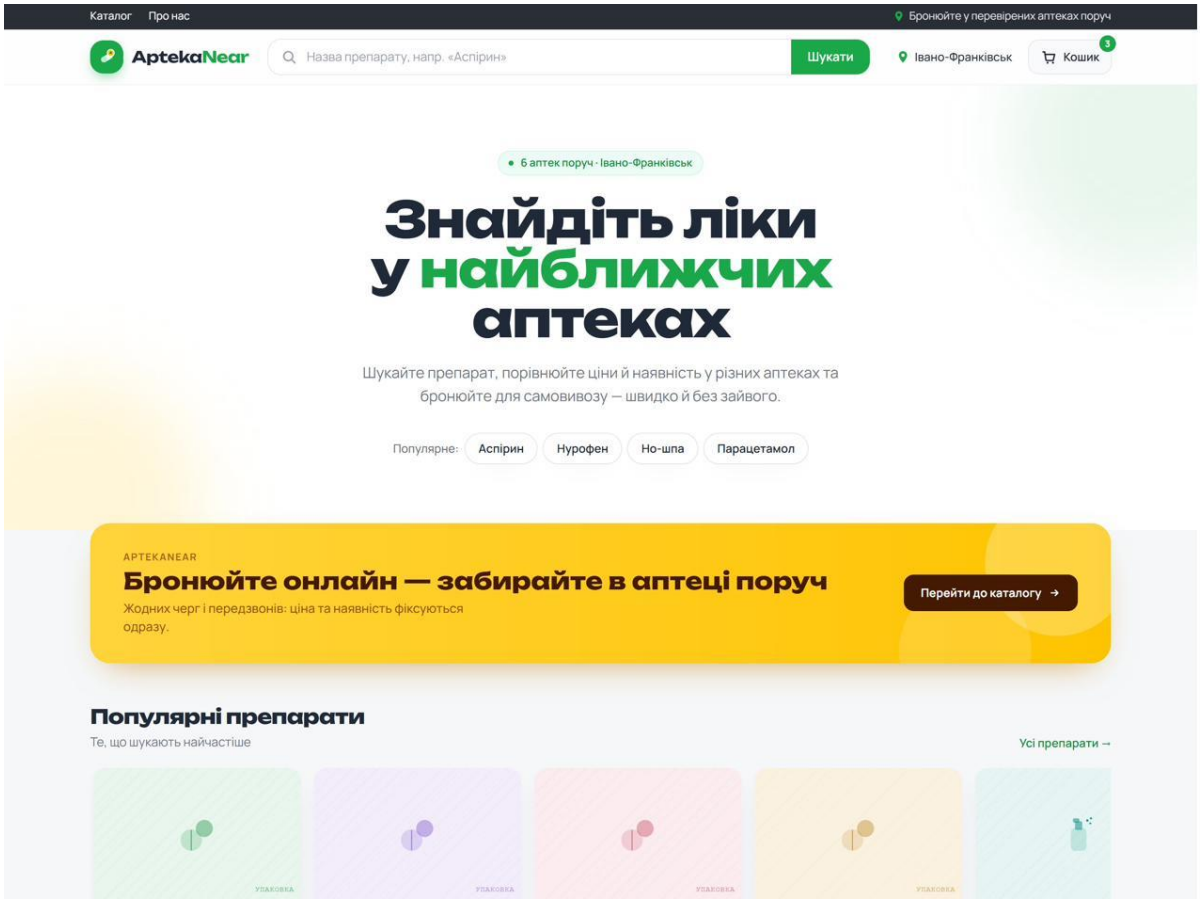

Рисунок 4.1 – Головна сторінка вебзастосунку AptekaNear

Головна сторінка виконує роль точки входу до основного сценарію:

пошуковий рядок із підказкою-прикладом розташовано на першому екрані, тож пошук доступний одразу, без додаткових переходів. Дані для відображення компоненти отримують через окремий шар клієнтської логіки, який приховує джерело даних від інтерфейсу; завдяки цьому сторінки однаково працюють на демонстраційному наборі даних і в повній конфігурації системи, а самі компоненти залишаються зосередженими виключно на відображенні та взаємодії з користувачем.

Ключовим для дослідження є екран картки препарату, що реалізує основний UX-сценарій порівняння: для обраного препарату відображається список пропозицій аптек із ціною, відстанню та статусом наявності, а також перемикач сортування (за ціною, відстанню та релевантністю), який оновлює список без перезавантаження сторінки. Зміна сортування не руйнує контекст - обраний препарат і позиція користувача на сторінці зберігаються, тому варіанти можна порівнювати послідовно, без повторних пошуків. Картку препарату з пропозиціями аптек наведено на рисунку 4.2.

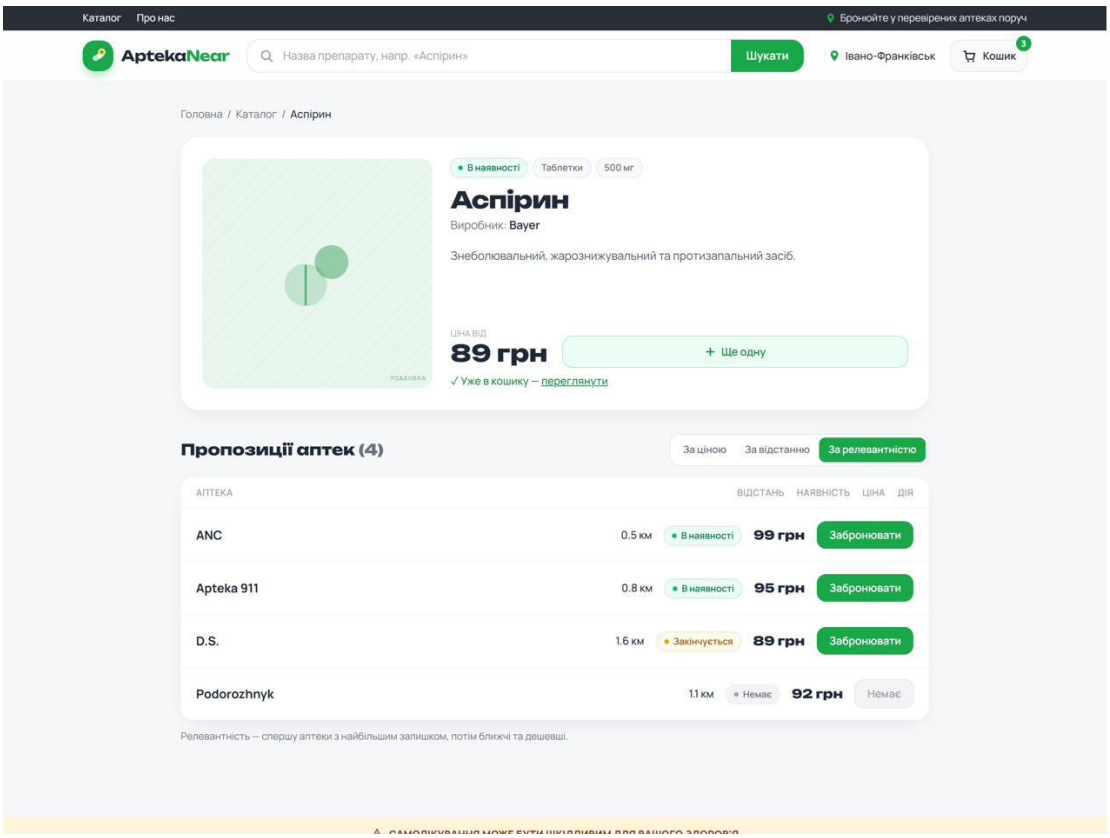

Рисунок 4.2 – Картка препарату з пропозиціями аптек і сортуванням

Перехід між кроками сценарію реалізовано декларативною маршрутизацією: кожній сторінці відповідає власна адреса, тому користувач може користуватися кнопками «назад» і «вперед» браузера, оновлювати сторінку без втрати місця в сценарії та поділитися посиланням на конкретний препарат. Навігація поводиться так, як користувач звик у звичайних вебсайтах, без «пасток», коли повернення назад руйнує введені дані або стан перегляду.

Стан кошика зберігається в локальному сховищі браузера, завдяки чому обрані позиції не втрачаються між сеансами роботи: користувач може закрити вкладку, повернутися пізніше й продовжити з того самого місця. Додавання препарату в кошик супроводжується миттєвим візуальним відгуком - лічильник позицій у шапці оновлюється одразу, підтверджуючи дію без модальних вікон, які переривали б перегляд.

Окремим UX-рішенням кошика є режим порівняння за аптеками: зібрані позиції групуються за аптеками з підрахунком доступних найменувань і загальної суми, тож користувач одразу бачить, де вигідніше забронювати всі препарати разом, а недоступні позиції явно позначаються. Це переносить на інтерфейс роботу, яку інакше довелося б виконувати вручну, переглядаючи аптеки по одній; відповідні екранні форми наведено в додатку Е.

Текстові повідомлення інтерфейсу сформульовано мовою користувача, без технічних термінів: замість кодів помилок виводяться пояснення, що сталося і що варто зробити далі, а дії, які завершують сценарій, супроводжуються явним підтвердженням. Однакові за змістом повідомлення оформлено однаково на всіх сторінках, тому користувач швидко звикає розпізнавати стани сервісу.

Для кожного кроку сценарію передбачено повний набір інтерфейсних станів - завантаження, порожній результат, помилка та успіх, - а також валідацію форми бронювання на боці клієнта. Завершення основного сценарію - екран підтвердження бронювання - наведено на рисунку 4.3.

У межах реалізації цього сценарію важливо було забезпечити безперервність переходу між етапами: від вибору препарату до підтвердження бронювання. Для цього дані про вибрану позицію передаються між сторінками

через стан застосунку та локальне сховище, що дозволяє зберігати контекст користувача після переходу до наступного екрана. Екран підтвердження виконує роль фінального зворотного зв'язку системи: після завершення бронювання користувач отримує структуровану інформацію про результат дії - номер бронювання, аптеку, препарат, кількість і спосіб сповіщення.

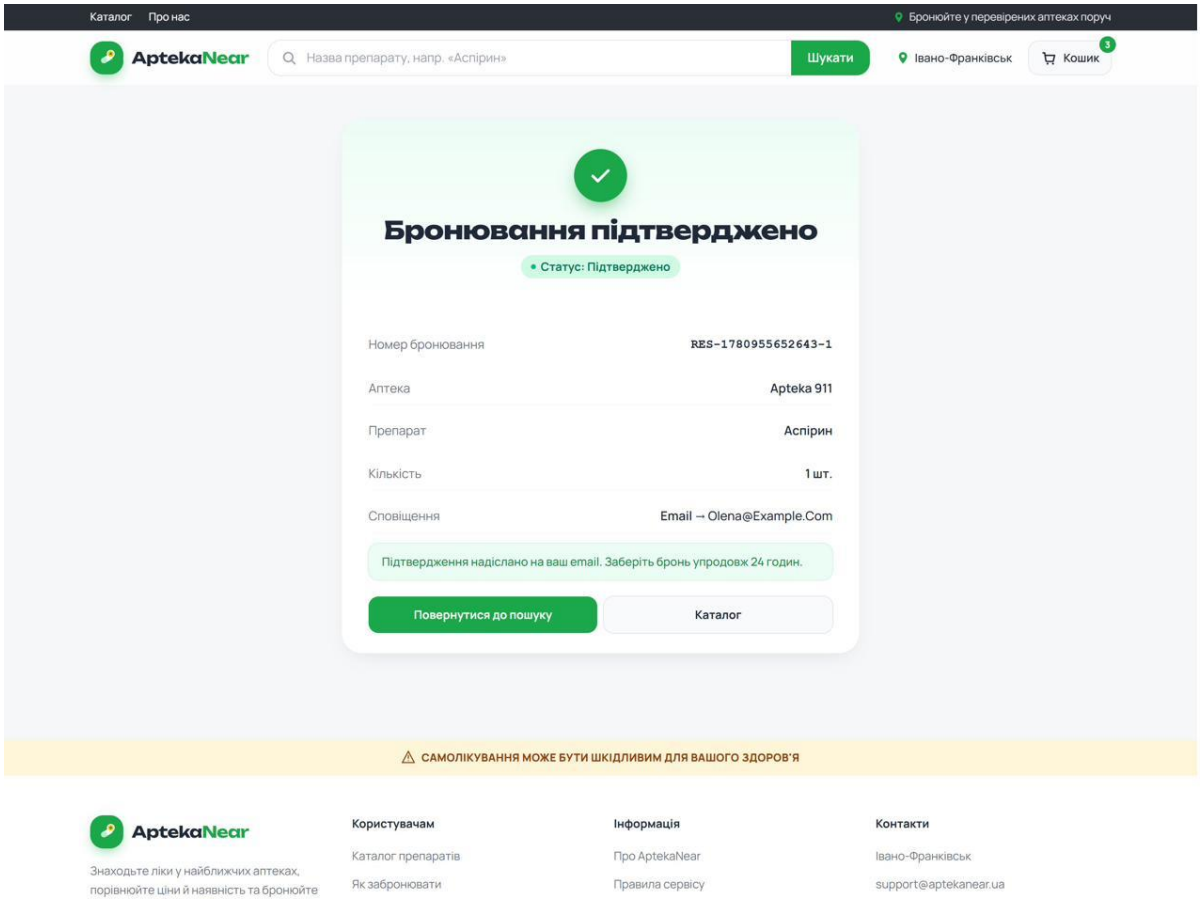

Рисунок 4.3 – Підтвердження бронювання

Екран підтвердження виконує не лише інформаційну, а й заспокійливу функцію: користувач одразу бачить статус бронювання, його номер, обрану аптеку та подальші кроки. Явне підтвердження результату дії - одна з базових евристик юзабіліті: система повідомляє про успіх у зрозумілих користувачеві термінах і не залишає сумнівів, чи завершилась операція.

На рисунку 4.4 зображено форму бронювання препарату - компактний крок, що завершує основний користувацький сценарій. Форма містить мінімально необхідний набір полів із валідацією введення та зрозумілими повідомленнями про

помилки, що зменшує ризик переривання сценарію на останньому етапі.

Під час реалізації форми бронювання основний акцент зроблено на скороченні кількості дій користувача. Поля розміщено у зрозумілій послідовності: спочатку вводяться контактні дані, після цього уточнюється кількість препарату та спосіб отримання сповіщення. Така структура відповідає логіці оформлення бронювання та допомагає завершити сценарій без зайвих повернень.

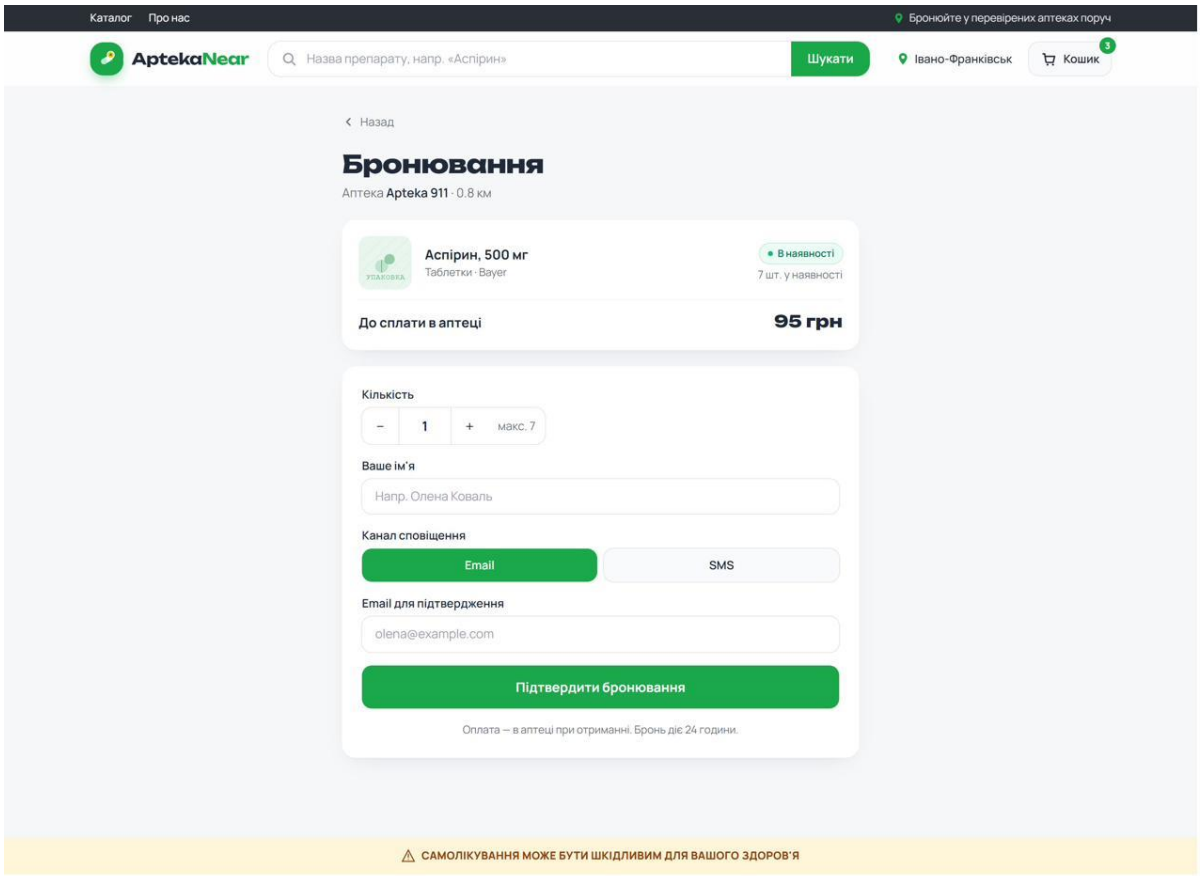

Рисунок 4.4 – Форма бронювання препарату

Валідацію форми бронювання реалізовано на боці клієнта до надсилання запиту: перевіряються заповненість обов’язкових полів, коректність контактних даних та допустима кількість одиниць препарату. Повідомлення про помилки виводяться безпосередньо біля відповідних полів у зрозумілих користувачеві формулюваннях, а кнопка підтвердження блокується на час обробки запиту, що запобігає повторному надсиланню форми.

На рисунку 4.5 зображено стан інтерфейсу, коли за пошуковим запитом

нічого не знайдено. Замість порожнього екрана користувач отримує пояснення результату та підказку щодо подальших дій, що відповідає вимозі інформативної обробки порожніх станів, сформульованій у розділі 2.

Такий стан інтерфейсу важливий для UX, оскільки відсутність результатів не повинна сприйматися як помилка системи. Інтерфейс пояснює, що запит опрацьовано коректно, але відповідних позицій у каталозі не знайдено. Користувач отримує підказку змінити запит або повернутися до каталогу.

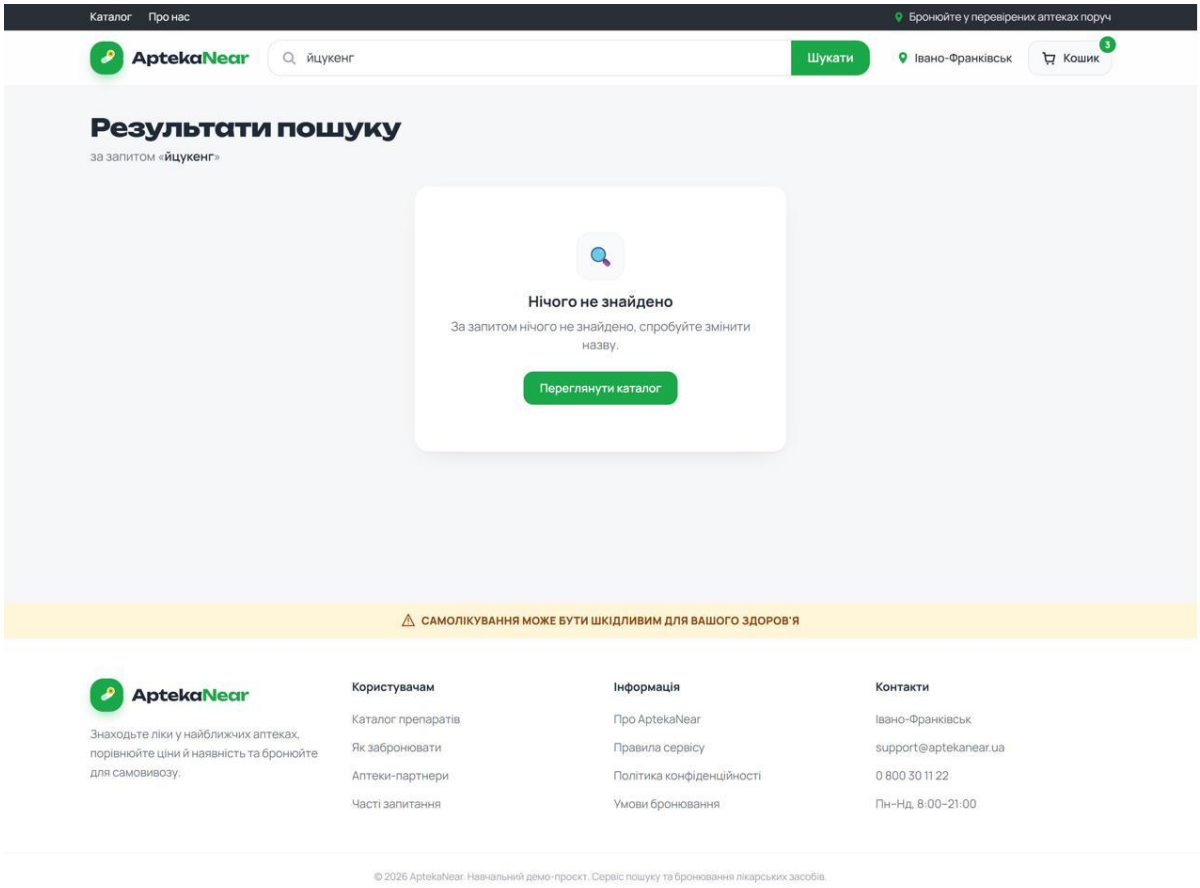

Рисунок 4.5 – Стан «нічого не знайдено» для пошуку

Окрему увагу під час реалізації приділено інтерфейсним станам. Для кожного асинхронного запиту компоненти відображають стан завантаження, а у випадку помилки - повідомлення з можливістю повторити дію без втрати введених користувачем даних. Порожні результати супроводжуються пояснювальним текстом і підказкою щодо зміни запиту (рис. 4.5), що відповідає вимогам до обробки станів, сформульованим у розділі 2. Такий підхід усуває «німі» паузи

інтерфейсу, коли користувач не розуміє, чи опрацьовує система його дію, і зменшує кількість повторних або помилкових натискань.

Решту екранних форм реалізованого вебзастосунку (каталог, результати пошуку, кошик, порівняння цін за аптеками та інформаційна сторінка) наведено в додатку Е (рисунки Е.1–Е.5).

Адаптивність інтерфейсу забезпечено утилітарними класами Tailwind зі стандартними контрольними точками: на вузьких екранах сітки карток перебудовуються в одну колонку, навігація ущільнюється, а інтерактивні елементи зберігають достатній розмір для впевненого дотику. Зображення та контейнери мають визначені розміри, що запобігає зсувам верстки під час завантаження сторінки та відповідає вимозі стабільності інтерфейсу, сформульованій серед нефункціональних вимог у розділі 2.

Під час верстки враховано базові вимоги вебдоступності, окреслені в підрозділі 1.3: сторінки використовують семантичні HTML-елементи (заголовки, списки, кнопки замість клікабельних блоків), поля форм мають текстові підписи, інтерактивні елементи доступні з клавіатури та мають видимий стан фокуса, а кольорові акценти статусу наявності дублюються текстовими позначками, щоб інформація не передавалася лише кольором.

Збирання клієнтської частини виконується засобами Vite: у режимі розробки зміни відображаються миттєво, а продукційна збірка мінімізує обсяг статичних ресурсів, що скорочує час першого завантаження сторінок. У підсумку реалізована клієнтська частина не лише відтворює спроєктовані в розділі 3 екрани, а й доводить UI/UX-принципи роботи до рівня конкретних компонентів, станів і перевірок: користувач на кожному кроці бачить зрозумілий наступний крок, отримує негайний відгук на свої дії та захищений від типових помилок введення. Саме ця наскрізна реалізація принципів UI/UX у фронт-енд частині і становить практичний результат дослідження.

4.5 Висновки по розділу

У четвертому розділі описано програмну реалізацію вебзастосунку AptekaNear. Обґрунтовано модульну організацію коду з чітким розмежуванням відповідальності між контролерами, сервісами та сховищами, що відповідає спроєктованій багаторівневій архітектурі.

Показано застосування механізму впровадження залежностей та патернів проєктування Facade, Strategy і Factory Method, які зменшують зв’язність, усувають дублювання логіки та роблять взаємодію компонентів передбачуванішою.

Описано реалізацію клієнтської частини з використанням React, Vite і TypeScript: структуру сторінок, побудовану за користувацьким сценарієм, повторно використовувані компоненти, повний набір станів інтерфейсу, клієнтську валідацію форми бронювання, локальне збереження стану кошика, а також адаптивність і базову вебдоступність, що безпосередньо розкриває тему UI/UX у фронт-енд веб-розробці.

Реалізовано централізоване логування й обробку помилок з інтеграцією моніторингу Sentry, а також застосовано статичну типізацію для контролю якості коду.

Отже, програмна реалізація AptekaNear охоплює технічну основу вебзастосунку та клієнтську фронт-енд частину, у якій практично відображено основні UI/UX-рішення: структуру екранів, компоненти, стани інтерфейсу, валідацію форми, адаптивність, доступність та взаємодію користувача із сервісом.

РОЗДІЛ 5. ТЕСТУВАННЯ ТА ВПРОВАДЖЕННЯ ВЕБЗАСТОСУНКУ

У цьому розділі розглянуто перевірку якості вебзастосунку AptekaNear, оцінювання реалізованих рішень та питання впровадження системи в експлуатацію. Тестування охоплює не лише функціональну коректність, а й перевірку ключових користувацьких сценаріїв, станів інтерфейсу, адаптивності та доступності.

5.1 Стратегії та методи тестування вебзастосунку

Тестування вебзастосунку доцільно будувати як багаторівневий процес, що поєднує різні види перевірок. Функціональне тестування підтверджує, що окремі функції (пошук, фільтрація, бронювання) працюють відповідно до вимог. Інтеграційне тестування перевіряє коректність взаємодії фронт-енду з серверною частиною через API. Тестування інтерфейсу охоплює перевірку станів завантаження, порожніх результатів, помилок валідації та підтверджень [19, 27].

Окрему увагу приділено перевірці нефункціональних характеристик. Тестування продуктивності оцінює час відгуку та стабільність верстки під час завантаження даних. Перевірка доступності контролює достатній контраст, видимий фокус, семантичну структуру та можливість клавіатурної навігації. Тестування адаптивності підтверджує збереження зручності сценаріїв на мобільних і десктопних екранах [22, 26].

Для модульного тестування серверної частини застосовують юніт-тести з підміною залежностей через мок-об’єкти, що стало можливим завдяки впровадженню залежностей (підрозділ 4.2). Автоматизацію перевірок доцільно інтегрувати у процес складання (CI), щоб виявляти регресії ще до випуску нових версій [3, 12].

У межах навчального проєкту AptekaNear тестування розглядається як перевірка не лише технічної працездатності функцій, а й цілісності основного користувацького сценарію. Особлива увага приділяється пошуку препарату, переходу до бронювання, обробці помилок і станам інтерфейсу.

Узагальнені дані наведено в табл. 5.1.

Таблиця 5.1

Види тестування вебзастосунку AptekaNear

Вид тестування	**Мета**	**Що перевіряється**
Функціональне	Коректність функцій	Пошук, фільтрація, бронювання
Інтеграційне	Взаємодія через API	Узгодженість фронт-енду та сервера
Тестування інтерфейсу	Стани взаємодії	loading, empty, error, success
Продуктивності	Швидкодія та стабільність	Час відгуку, стабільність верстки
Доступності	Інклюзивність	Контраст, фокус, семантика, клавіатура
Адаптивності	Універсальність	Мобільні та десктопні екрани

Для систематизації перевірок доцільно вести набір тестових сценаріїв, що охоплюють основні та межові випадки використання. Кожен сценарій описує початкові умови, послідовність дій користувача та очікуваний результат, завдяки чому тестування стає відтворюваним і незалежним від конкретного виконавця. Накопичення таких сценаріїв формує основу для регресійного тестування під час подальшого розвитку системи.

У навчальному прототипі реалізовано набір автоматичних перевірок. Модульні тести (Jest) покривають стратегії сортування пропозицій, фасад бронювання, репозиторій пошуку препаратів, сервіс бронювання та фабрику каналів сповіщення; наскрізні тести (Supertest) перевіряють основні маршрути - пошук препаратів, картку препарату та створення бронювання, зокрема відхилення некоректного запиту з кодом 400. Загалом це 20 модульних і 6 наскрізних перевірок, що формують основу для подальшого розширення тестового покриття. Реалізовану клієнтську частину (фронтенд) перевірено вручну за ключовим user-flow (пошук → картка препарату → кошик → бронювання → підтвердження) та за інтерфейсними станами (завантаження, порожній результат, помилка, успіх).

Поряд з автоматичними перевірками застосовано ручне тестування API за допомогою HTTP-запитів до основних маршрутів (пошук, пропозиції, бронювання) з перевіркою очікуваних відповідей і станів помилок. Для перевірки роботи

механізму обробки винятків передбачено окремий демонстраційний маршрут, який навмисно повертає помилку, що дозволяє переконатися в коректному логуванні та формуванні відповіді. Налагодження спирається на структуровані журнали дій користувача.

Результати тестування доцільно фіксувати у зведеній формі із зазначенням частки успішно пройдених перевірок за кожним видом тестування. Це дозволяє об’єктивно оцінити готовність вебзастосунку до впровадження та виявити напрями, що потребують доопрацювання. У межах навчального прототипу основну увагу приділено функціональним перевіркам ключових сценаріїв і контролю станів інтерфейсу, які найбільше впливають на користувацький досвід.

5.2 Забезпечення якості, тестування та налагодження вебзастосунку

Надійність вебзастосунку значною мірою впливає на сприйняття його зручності. Якщо система поводиться нестабільно або помилки виникають без пояснення, користувач оцінює продукт як неякісний незалежно від візуальної привабливості. Тому у проєкті було передбачено базове логування запуску застосунку, логування ключових дій користувача та централізовану обробку винятків.

Інтеграція системи збору аналітики помилок дозволяє виявляти збої ще на етапі тестування і супроводу. З інженерної точки зору це спрощує діагностику проблем, а з UX-погляду дає змогу швидше усувати дефекти у критичних сценаріях. Важливо, що така інфраструктура не повинна впливати на користувача безпосередньо; її завдання - підвищити стабільність сервісу і зменшити кількість повторюваних збоїв.

У фронт-енд частині обробка помилок має будуватися за тим самим принципом централізації: мережеві помилки, невалідні відповіді, відсутність доступу чи тайм-аути слід інтерпретувати уніфіковано і відображати користувачеві у зрозумілому вигляді. Такий підхід формує відчуття керованості системою навіть у нештатних ситуаціях.

Окремі фрагменти програмної реалізації патернів Facade, Strategy та інтеграції Sentry подано в додатку Г. Ілюстративні матеріали із запуском застосунку, логуванням дій користувача й прикладом wireframe-прототипу наведено в додатку Д. Такий розподіл матеріалу дозволяє зберегти логіку основної частини роботи й одночасно надати достатню доказову базу практичної реалізації.

Керування версіями, релізний процес і підтримуваність

Якість вебпродукту залежить не лише від коду окремого розробника, а й від організації командної роботи. У матеріалах проєкту були опрацьовані базові сценарії використання Git: робота з окремими гілками, створення Pull Request, вирішення конфліктів, релізні теги й перевірка змін. Для фронт-енд проєкту це має важливе значення, оскільки інтерфейс часто змінюється і потребує контрольованого внесення правок без деградації UX.

Наявність зрозумілого процесу рев'ю, релізів і фіксації змін підвищує підтримуваність системи. Це особливо важливо у випадку дизайн-системи та повторно використовуваних компонентів: помилка в одному спільному елементі може вплинути на багато сценаріїв одночасно. Отже, дисципліна керування версіями є організаційною передумовою стабільного користувацького досвіду.

Експертна оцінка юзабіліті для AptekaNear

Експертна оцінка юзабіліті в межах цієї роботи ґрунтується на перевірці відповідності інтерфейсу базовим евристикам і сформованим вимогам. Було проаналізовано, чи видимий стан системи під час пошуку; чи відповідають назви дій зрозумілій користувачеві мові; чи зберігається контроль над переходами між екранами; чи є інтерфейс консистентним; чи запобігає система помилкам у формах; чи достатньо інформативні повідомлення про відсутність результатів і підтвердження бронювання.

Результати такої оцінки свідчать, що найбільшу UX-цінність створюють: домінантне поле пошуку на стартовому екрані; відображення мінімальної ціни та статусу наявності без переходу до деталей; можливість перемикання між списком і картою; коротка форма бронювання; наявність екрана підтвердження. До потенційних зон покращення належать: додаткові підказки у випадку порожнього

результату, розширена робота з повторними діями та поглиблення доступності складних інтерактивних елементів.

Отже, експертна оцінка підтверджує тезу роботи: застосування принципів UI/UX-дизайну є результативним лише тоді, коли воно доведене до рівня конкретних сценаріїв, станів і компонентів, а не обмежується загальними деклараціями про «зручний інтерфейс».

Узагальнені дані наведено в табл. 5.2.

Таблиця 5.2

Приклади тестових сценаріїв для оцінювання UI/UX

Сценарій	**Очікуваний результат**	**Критерій оцінки**
Пошук препарату за назвою	Система відображає релевантний список	Час до результату, зрозумілість картки
Зміна сортування	Результати оновлюються без втрати контексту	Передбачуваність інтерфейсу
Перехід до карти	Карта відображає ті самі варіанти, що й список	Узгодженість режимів подання
Невдала форма бронювання	Показано локальні помилки біля полів	Запобігання помилкам і відновлення
Успішне бронювання	Є екран підтвердження з кодом і терміном дії	Ясність завершення сценарію
Відсутність результатів	Система пропонує змінити фільтри або запит	Підтримка користувача у порожньому стані

5.3 Оцінювання реалізації та напрями вдосконалення фронт-енд частини

Після реалізації основних сценаріїв AptekaNear важливо оцінити не тільки працездатність окремих функцій, а й цілісність користувацького досвіду. Для цього варто поєднувати функціональне тестування, перевірку доступності, аналіз адаптивності, тестування API-взаємодії та моніторинг помилок у середовищі виконання [19, 27, 31].

Оцінювання фронт-енд частини доцільно проводити за кількома напрямами. Перший напрям - перевірка ключового user flow: пошук препарату, перегляд пропозицій, вибір аптеки та бронювання.

Другий напрям - перевірка станів інтерфейсу: завантаження, порожній результат, помилка сервера, успішне підтвердження, помилка валідації. Третій напрям - контроль адаптивності на різних ширинах екрана [11, 22, 26].

З технічної точки зору подальше вдосконалення може передбачати деталізацію компонентної бібліотеки, впровадження повноцінної дизайн-системи, підключення реальної бази даних, розширення API-контрактів, покращення типізації, оптимізацію збірки та автоматизовані перевірки перед релізом [10].

Для підтримуваності проєкту важливо зберігати контроль версій, вести роботу через окремі гілки, перевіряти зміни через pull request і позначати стабільні версії тегами. Такий процес забезпечує відтворюваність релізів і спрощує супровід вебзастосунку після завершення навчального етапу [3, 12].

Оцінювання реалізованого прототипу показує, що основний користувацький сценарій - від пошуку препарату до бронювання з підтвердженням - відпрацьовує коректно на наборі демонстраційних даних. Застосування патернів проєктування та чіткий поділ на сервіси підвищили зрозумілість і підтримуваність коду, а централізоване логування дало змогу простежувати дії користувача в ключових сценаріях.

Водночас слід враховувати обмеження навчальної реалізації: дані зберігаються у пам’яті, а тестове покриття є базовим. Клієнтську частину (фронтенд) реалізовано і перевірено за спроєктованими сценаріями та інтерфейсними станами, тому оцінювання UX на цьому етапі спирається на відповідність цим сценаріям і чек-листу, а не на повноцінне дослідження за участю користувачів.

Серед основних напрямів подальшого вдосконалення - підключення реальної бази даних замість сховища в пам’яті, додавання персонального профілю, реалізація обраного й нагадувань, інтеграція з картографічними сервісами, а також розширення автоматизованого тестового покриття й перехід від імітації сповіщень до реальної відправки. Поточний навчальний прототип закладає для цього стійку архітектурну основу.

5.4 Висновки по розділу

У п’ятому розділі розглянуто тестування, оцінювання та впровадження вебзастосунку AptekaNear. Визначено стратегію тестування, що охоплює функціональні, інтеграційні та нефункціональні перевірки, зокрема контроль станів інтерфейсу, продуктивності, доступності й адаптивності.

Проаналізовано забезпечення якості, налагодження, логування, моніторинг помилок і розгортання системи. Оцінено реалізацію за основним користувацьким сценарієм, станами інтерфейсу та адаптивністю, а також визначено напрями вдосконалення фронт-енд частини.

Зроблено висновок, що якісний UI/UX у фронт-енд веб-розробці залежить від системного тестування й оцінювання, які охоплюють не лише функціональну коректність, а й зручність, доступність та стабільність взаємодії користувача з вебзастосунком.

ВИСНОВКИ

У результаті виконання бакалаврської роботи досліджено застосування принципів UI/UX-дизайну у фронт-енд веб-розробці та обґрунтовано їх значення для створення зрозумілих, адаптивних і підтримуваних вебзастосунків. На прикладі навчального проєкту AptekaNear показано, що якість користувацького досвіду формується не лише під час проєктування інтерфейсу, а й у процесі реалізації клієнтської частини, побудови компонентів, налаштування станів інтерфейсу та організації користувацьких сценаріїв.

У першому розділі розглянуто теоретичні основи UI та UX у вебзастосунках. Визначено, що UI відповідає за візуальну та структурну форму взаємодії користувача із системою, а UX охоплює загальне враження від проходження сценарію. Узагальнено принципи візуальної ієрархії, консистентності, зворотного зв'язку, зручності форм, адаптивності, доступності та відповідності ментальній моделі користувача. Встановлено, що у фронт-енд веброзробці ці принципи реалізуються через структуру екранів, компоненти, стилі, стани інтерфейсу, маршрутизацію та поведінку інтерактивних елементів.

У другому розділі виконано аналіз предметної області вебзастосунку AptekaNear. Визначено основні групи користувачів, їхні потреби та ключові сценарії взаємодії: пошук препарату, перегляд пропозицій аптек, порівняння цін, оформлення бронювання та отримання підтвердження. Сформовано функціональні й нефункціональні вимоги до MVP, а також визначено UX-критерії якості, пов'язані зі зрозумілістю інтерфейсу, швидкістю взаємодії, доступністю, стабільністю відображення та мінімізацією кількості зайвих дій.

У третьому розділі спроєктовано структуру вебзастосунку AptekaNear: основні екрани, компоненти, навігаційні переходи, модель даних і логіку взаємодії. UX-модель узгоджено з фронт-енд реалізацією: головна сторінка підтримує швидкий пошук, картка препарату допомагає порівняти пропозиції, форма бронювання забезпечує короткий сценарій, а порожні та помилкові стани дають користувачу зрозумілий зворотний зв'язок.

У четвертому розділі описано практичну реалізацію вебзастосунку AptekaNear. Показано, як інтерфейсні й архітектурні рішення, спроєктовані в попередньому розділі, доведено до працездатного вебзастосунку. Окремо розглянуто реалізацію клієнтської частини на React, Vite, TypeScript і Tailwind CSS. Описано сценарну структуру сторінок, компоненти інтерфейсу, збереження стану кошика, форму бронювання, підтвердження дії, стани loading, error, empty і success, а також адаптивність і базові вимоги вебдоступності.

У п’ятому розділі розглянуто тестування, оцінювання та впровадження вебзастосунку. Визначено підходи до перевірки функціональної коректності, стабільності інтерфейсу, адаптивності, доступності та зручності проходження основного користувацького сценарію. Зроблено висновок, що тестування фронт-енд частини має охоплювати не лише перевірку працездатності окремих функцій, а й оцінювання того, наскільки зрозуміло користувач проходить шлях від пошуку препарату до підтвердження бронювання.

Практичне значення роботи полягає в узагальненні підходів до поєднання UI/UX-дизайну з фронт-енд реалізацією вебзастосунку. У роботі показано, що якісний користувацький досвід залежить від узгодженості вимог, структури екранів, компонентної організації, поведінки інтерактивних елементів, обробки станів інтерфейсу, адаптивності та доступності.

Наукова новизна роботи полягає в уточненні підходу до розгляду UI/UX-рішень як складової фронт-енд веб-розробки. У роботі показано, що дизайн інтерфейсу не повинен сприйматися лише як візуальне оформлення, оскільки він безпосередньо пов’язаний із логікою компонентів, станами сторінок, формами введення, повідомленнями для користувача та загальною підтримуваністю клієнтської частини вебзастосунку.

Отже, мету бакалаврської роботи досягнуто. На прикладі AptekaNear обґрунтовано доцільність використання принципів UI/UX-дизайну у фронт-енд веб-розробці для створення зрозумілого, адаптивного й зручного вебзастосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дизайн система Дії у відкритому доступі. Міністерство цифрової трансформації України. [Електронний ресурс]. URL: https://thedigital.gov.ua/news/technologies/diia_design (дата звернення: 01.12.2025).
2. Дизайн-система державних сайтів України. [Електронний ресурс]. URL: https://design.gov.ua/ (дата звернення: 03.12.2025).
3. About pull requests. GitHub Docs. [Електронний ресурс]. URL: https://docs.github.com/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests (дата звернення: 10.12.2025).
4. Accessibility. MDN Web Docs. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/Accessibility (дата звернення: 12.12.2025).
5. ARIA Authoring Practices Guide (APG). W3C Web Accessibility Initiative. [Електронний ресурс]. URL: https://www.w3.org/WAI/ARIA/apg/ (дата звернення: 15.12.2025).
6. CSS: Cascading Style Sheets. MDN Web Docs. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/CSS (дата звернення: 18.12.2025).
7. Custom providers. NestJS Documentation. [Електронний ресурс]. URL: https://docs.nestjs.com/fundamentals/custom-providers (дата звернення: 22.12.2025).
8. Describing the UI. React Documentation. [Електронний ресурс]. URL: https://react.dev/learn/describing-the-ui (дата звернення: 05.01.2026).
9. Design Basics: UI/UX, Prototyping & Core Principles. Figma. [Електронний ресурс]. URL: https://www.figma.com/resource-library/design-basics/ (дата звернення: 09.01.2026).
10. Getting Started. Vite Documentation. [Електронний ресурс]. URL: https://vite.dev/guide/ (дата звернення: 14.01.2026).

11. Gibbons S. Journey Mapping 101. Nielsen Norman Group. [Електронний ресурс]. URL: https://www.nngroup.com/articles/journey-mapping-101/ (дата звернення: 20.01.2026).

12. Git Basics - Tagging. Pro Git Book. [Електронний ресурс]. URL: https://git-scm.com/book/en/v2/Git-Basics-Tagging (дата звернення: 25.01.2026).

13. Guide to prototyping in Figma. Figma Help Center. [Електронний ресурс]. URL: https://help.figma.com/hc/en-us/articles/360040314193-Guide-to-prototyping-in-Figma (дата звернення: 02.02.2026).

14. How to Meet WCAG (Quick Reference). W3C Web Accessibility Initiative. [Електронний ресурс]. URL: https://www.w3.org/WAI/WCAG22/quickref/ (дата звернення: 06.02.2026).

15. HTML Living Standard. WHATWG. [Електронний ресурс]. URL: https://html.spec.whatwg.org/multipage/ (дата звернення: 12.02.2026).

16. HTML: HyperText Markup Language. MDN Web Docs. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/HTML (дата звернення: 18.02.2026).

17. Introduction to Node.js. Node.js Learn. [Електронний ресурс]. URL: https://nodejs.org/learn/getting-started/introduction-to-nodejs (дата звернення: 24.02.2026).

18. ISO 9241-210:2019. Ergonomics of human-system interaction - Part 210: Human-centred design for interactive systems. [Електронний ресурс]. URL: https://www.iso.org/standard/77520.html (дата звернення: 03.03.2026).

19. ISO/IEC 25010:2023. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - Product quality model. [Електронний ресурс]. URL: https://www.iso.org/standard/78176.html (дата звернення: 07.03.2026).

20. JavaScript. MDN Web Docs. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 12.03.2026).

21. Learn Core Web Vitals. web.dev. [Електронний ресурс]. URL: https://web.dev/explore/learn-core-web-vitals (дата звернення: 18.03.2026).

22. Learn Responsive Design. web.dev. [Електронний ресурс]. URL: https://web.dev/learn/design (дата звернення: 24.03.2026).

23. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. [Електронний ресурс]. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (дата звернення: 01.04.2026).

24. Nielsen J. Usability 101: Introduction to Usability. Nielsen Norman Group. [Електронний ресурс]. URL: https://www.nngroup.com/articles/usability-101-introduction-to-usability/ (дата звернення: 05.04.2026).

25. Providers. NestJS Documentation. [Електронний ресурс]. URL: https://docs.nestjs.com/providers (дата звернення: 10.04.2026).

26. Responsive web design. MDN Web Docs. [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Core/CSS_layout/Responsive_Design (дата звернення: 16.04.2026).

27. Sentry Documentation. Nest.js. [Електронний ресурс]. URL: https://docs.sentry.io/platforms/javascript/guides/nestjs/ (дата звернення: 22.04.2026).

28. The TypeScript Handbook. TypeScript Documentation. [Електронний ресурс]. URL: https://www.typescriptlang.org/docs/handbook/intro.html (дата звернення: 28.04.2026).

29. Thinking in React. React Documentation. [Електронний ресурс]. URL: https://react.dev/learn/thinking-in-react (дата звернення: 04.05.2026).

30. Understanding Core Web Vitals and Google search results. Google Search Central. [Електронний ресурс]. URL: https://developers.google.com/search/docs/appearance/core-web-vitals (дата звернення: 08.05.2026).

31. Web Content Accessibility Guidelines (WCAG) 2.2. W3C Recommendation. [Електронний ресурс]. URL: https://www.w3.org/TR/WCAG22/ (дата звернення: 12.05.2026).

32. What Is a Design System. Figma. [Електронний ресурс]. URL: https://www.figma.com/blog/design-systems-101-what-is-a-design-system/ (дата звернення: 18.05.2026).

ДОДАТКИ

ДОДАТОК А. КЛЮЧОВІ КОРИСТУВАЦЬКІ СЦЕНАРІЇ ARTEKANEAR

Узагальнені дані наведено в табл. А.1.

Таблиця А.1

Основні сценарії взаємодії користувача із системою

Сценарій	**Початкова дія**	**Очікуваний результат**
Пошук препарату	Введення запиту на головному екрані	Список релевантних результатів
Перегляд результатів	Вибір сортування або фільтра	Оновлений список / карта без втрати контексту
Перехід до картки	Натискання на результат	Детальна інформація про препарат
Бронювання	Заповнення форми та підтвердження	Створене бронювання з кодом
Перегляд історії	Перехід у профіль	Список активних та завершених бронювань

Ключові сценарії були сформовані на основі користувацьких історій та слугують зв’язком між UX-вимогами і структурою фронт-енд застосунку.

Саме ці сценарії повинні перевірятися в першу чергу під час експертної оцінки юзабіліті й інтеграційного тестування.

ДОДАТОК Б. БАЗОВІ API-ОПЕРАЦІЇ СИСТЕМИ

Узагальнені дані наведено в табл. Б.1.

Таблиця Б.1

Приклад базових API-операцій для MVP

Метод / endpoint	**Призначення**	**Результат**
GET /api/medicines/search?q=...	Пошук препаратів за назвою	Список знайдених препаратів
GET /api/availability?medicineId=...	Отримання аптек з цінами та статусами	Результати пошуку для списку або карти
GET /api/medicines/{id}	Отримання картки препарату	Детальна інформація про препарат
POST /api/reservations	Створення бронювання	Код бронювання, статус, термін дії
POST /api/auth/login	Вхід користувача	Токен / сесія
GET /api/profile/reservations	Історія бронювань	Список бронювань користувача

У фронт-енд частині доцільно інкапсулювати роботу з цими endpoint у централізованому API-шарі, щоб уникнути дублювання логіки та уніфікувати обробку помилок.

ДОДАТОК В. ЧЕК-ЛИСТ UI/UX ДЛЯ ФРОНТ-ЕНД РЕАЛІЗАЦІЇ

Узагальнені дані наведено в табл. В.1.

Таблиця В.1

Узагальнений чек-лист UI/UX для фронт-енд реалізації

Критерій	**Перевірка**
Візуальна ієрархія	Основна дія помітніша за вторинні; текст легко сканується
Консистентність	Однакові елементи мають однаковий вигляд і поведінку
Форми	Є підписи полів, локальна валідація, зрозумілі помилки
Стани	Передбачено loading, empty, success та error state
Адаптивність	Сценарій зберігається на мобільному і десктопному екранах
Доступність	Контраст, фокус, семантика, клавіатурна навігація
Продуктивність	Сторінка швидко реагує на дії та не втрачає контекст

Чек-лист може застосовуватися як практичний інструмент під час подальшої реалізації клієнтської частини AptekaNear або аналогічних вебпроєктів.

ДОДАТОК Г. ФРАГМЕНТИ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ APTEKANEAR

У додатку наведено вибрані фрагменти програмної реалізації, які ілюструють ключові технічні рішення навчального проєкту AptekaNear. Матеріал подано в скороченому вигляді, достатньому для демонстрації архітектурного підходу, механізмів розширення та підтримки якості користувацького досвіду.

Г.1 Реєстрація репозиторію через DI-контейнер

Наведений нижче фрагмент показує, як у модулі medicines реалізовано передачу залежності через контейнер впровадження залежностей. Такий підхід зменшує зв’язність сервісу з конкретною реалізацією сховища даних.

```
// medicines.module.ts
providers: [
  MedicinesService,
  {
    provide: MEDICINE_REPOSITORY,
    useClass: InMemoryMedicineRepository,
  },
];
```

```
// medicines.service.ts
constructor(
  @Inject(MEDICINE_REPOSITORY)
  private readonly medicineRepository: MedicineRepository,
) {}
```

Г.2 Спрощення сценарію бронювання через патерн Facade

Фасад дає змогу приховати послідовність звернень до кількох сервісів і надати контролеру один зрозумілий інтерфейс виконання бронювання.

```
// До впровадження Facade
@Post('reserve')
reserve(@Body() dto: ReserveMedicineDto) {
  const medicine = this.medicinesService.getById(dto.medicineId);
  const offer =
    this.availabilityService.getOfferByMedicineAndPharmacy(
      dto.medicineId,
      dto.pharmacyId,
    );
  const reservation = this.reservationService.createReservation({
```

```
    offer,
    quantity: dto.quantity ?? 1,
    userId: dto.userId,
    contact: dto.contact,
  });
  this.notificationService.sendReservationConfirmation({
    channel: dto.channel ?? 'email',
    contact: dto.contact,
    medicineName: medicine.name,
    pharmacyName: offer.pharmacyName,
    reservationId: reservation.id,
  });
  return reservation;
}
```

```
// Після впровадження Facade
@Post('reserve')
reserve(@Body() dto: ReserveMedicineDto) {
  return this.reservationFacade.reserveMedicine(dto);
}
```

Г.3 Стратегія сортування пропозицій аптек

Винесення алгоритмів сортування в окремі стратегії дозволяє фронт-енд частині працювати з одним параметром sortBy і не дублювати умовну логіку.

```
// До впровадження Strategy
searchOffers(medicineId: number, sort: string) {
  const offers =
    this.availabilityService.getOffersByMedicine(medicineId);
  if (sort === 'price') {
    return offers.sort((a, b) => a.price - b.price);
  } else if (sort === 'distance') {
    return offers.sort((a, b) => a.distanceKm - b.distanceKm);
  } else {
    return offers.sort((a, b) => b.stock - a.stock);
  }
}
```

```
// Після впровадження Strategy
searchOffers(
  medicineId: number,
  sortBy: OfferSortType = 'relevance',
) {
  const offers =
    this.availabilityService.getOffersByMedicine(medicineId);
  return this.sortContext.sort(offers, sortBy);
}
```

Г.4 Логування та інтеграція Sentry

Фрагмент демонструє базову ініціалізацію Sentry, підключення глобального фільтра винятків і логування дій користувача в контролері medicines.

```
// src/common/sentry/sentry.service.ts
@Injectable()
export class SentryService {
  private readonly logger = new Logger(SentryService.name);
  private initialized = false;

  init(): void {
    const dsn = process.env.SENTRY_DSN;
    if (!dsn) {
      this.logger.log('SENTRY_DSN is not set. Sentry is
disabled.');
      return;
    }
    Sentry.init({ dsn });
    this.initialized = true;
    this.logger.log('Sentry initialized.');
  }
}
```

```
// src/main.ts (фрагмент)
async function bootstrap() {
  const logger = new Logger('Bootstrap');
  logger.log('Starting AptekaNear application...');
  const app = await NestFactory.create(AppModule);
  const sentryService = app.get(SentryService);
  sentryService.init();
  const httpAdapterHost = app.get(HttpAdapterHost);
  app.useGlobalFilters(
    new AppExceptionsFilter(httpAdapterHost, sentryService),
  );
  const port = Number(process.env.PORT ?? 3000);
  await app.listen(port);
  logger.log(`AptekaNear is running on http://localhost:${port}`);
}
```

```
// src/medicines/medicines.controller.ts (фрагмент)
@Get('search')
search(@Query('q') q = '') {
  const results = this.medicinesService.search(q);
  this.logger.log(
    `User action: search medicines (q="${q}",
results=${results.length})`,
  );
  return { q, results };
}
```

ДОДАТОК Д. ІЛЮСТРАТИВНІ МАТЕРІАЛИ ПРОЄКТУ

У додатку подано вибрані ілюстративні матеріали з виконаних лабораторних робіт, які підтверджують етапи прототипування, архітектурного впорядкування та контролю якості в проєкті AptekaNear. Наведені матеріали доповнюють основну частину бакалаврської роботи та демонструють практичне застосування описаних UI/UX і програмних рішень.

На рисунку Д.1 зображено фрагмент wireframe-прототипу вебзастосунку AptekaNear, створеного у Figma. Він демонструє логіку переходів між основними екранами застосунку та використовується для перевірки шляху користувача від стартового пошуку до перегляду детальніших даних.

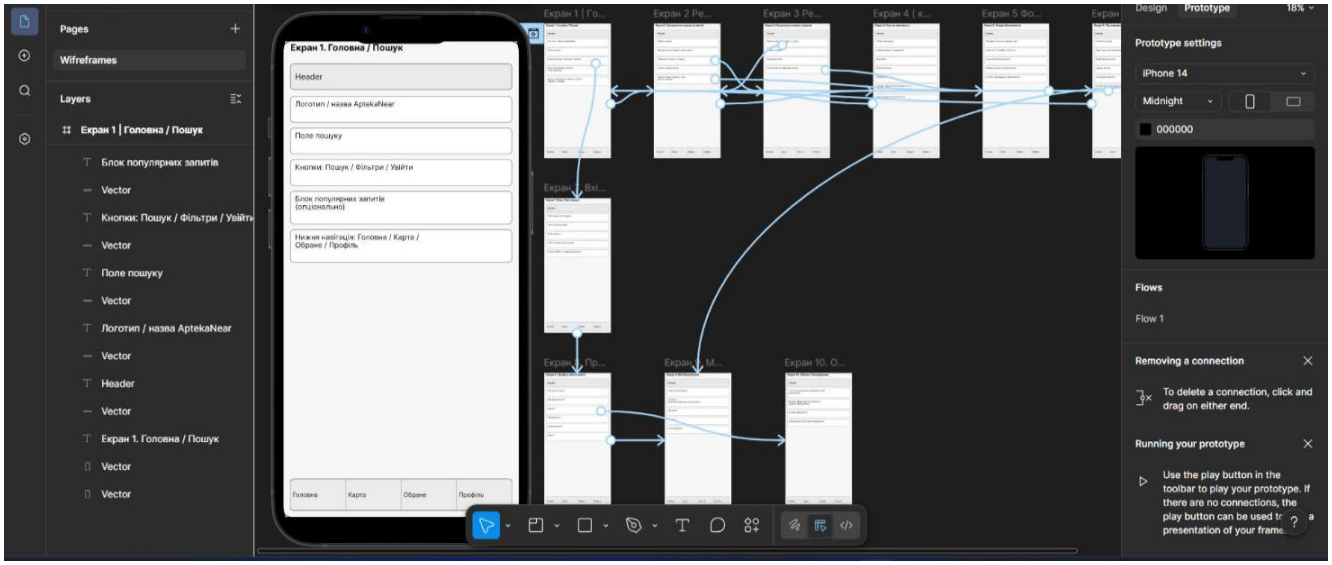

Рисунок Д.1 – Фрагмент wireframe-прототипу AptekaNear у Figma

На рисунку Д.2 зображено структуру проєкту після впровадження патернів Facade, Strategy та Factory Method. Цей матеріал демонструє, як виділення окремих фасадів, стратегій і службових модулів впорядковує програмний код, зменшує зв’язність між частинами системи та полегшує подальший розвиток вебзастосунку.

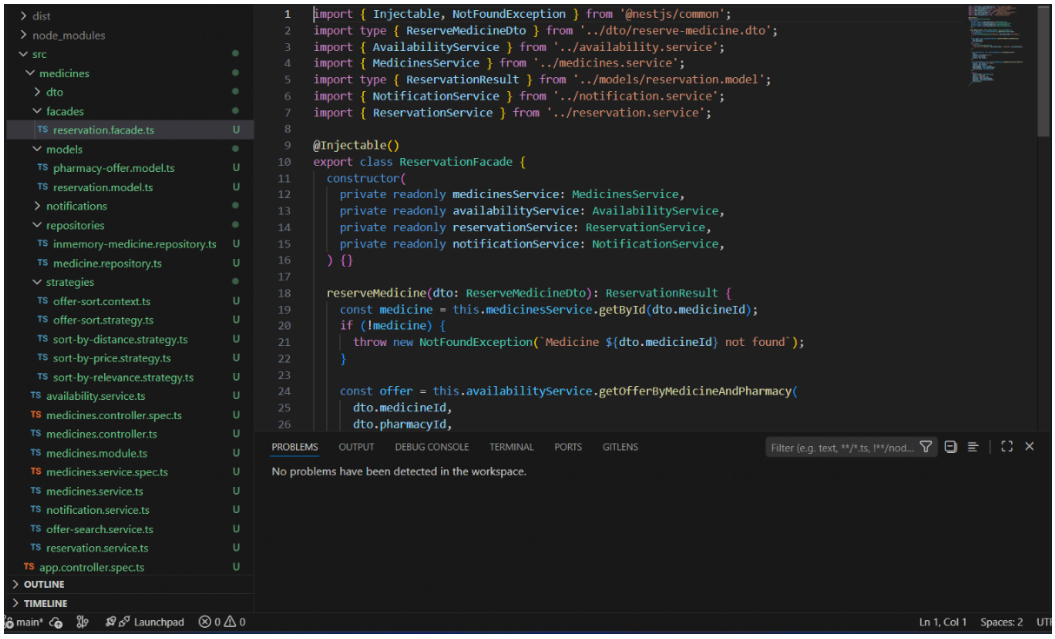

Рисунок Д.2 – Структура проєкту після впровадження Facade, Strategy та Factory Method

На рисунку Д.3 зображено запуск сервера та ініціалізацію маршрутів AptekaNear. Наведений фрагмент підтверджує коректне підключення основних модулів застосунку, реєстрацію endpoint і готовність серверної частини до подальшої інтеграції з фронт-ендом.

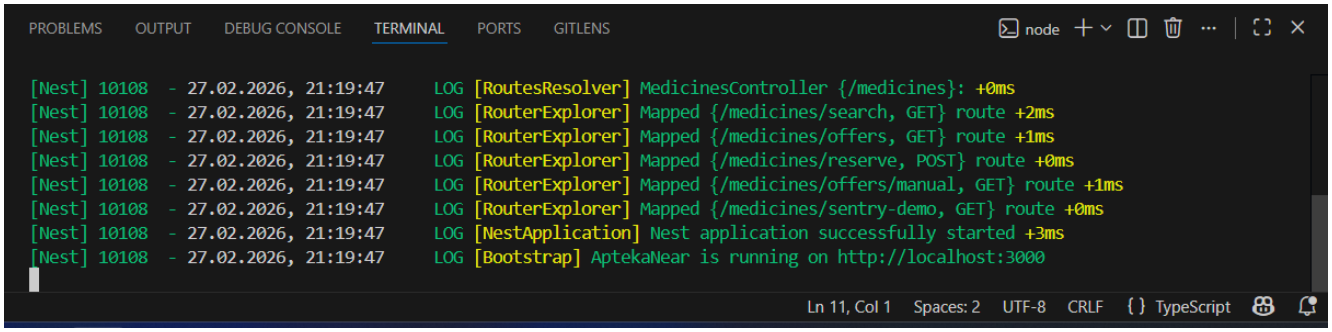

Рисунок Д.3 – Запуск сервера та ініціалізація маршрутів AptekaNear

На рисунку Д.4 зображено приклад відповіді endpoint пошуку препаратів. Цей результат ілюструє перевірку базового сценарію пошуку та підтверджує працездатність одного з ключових MVP-функціоналів вебзастосунку AptekaNear.

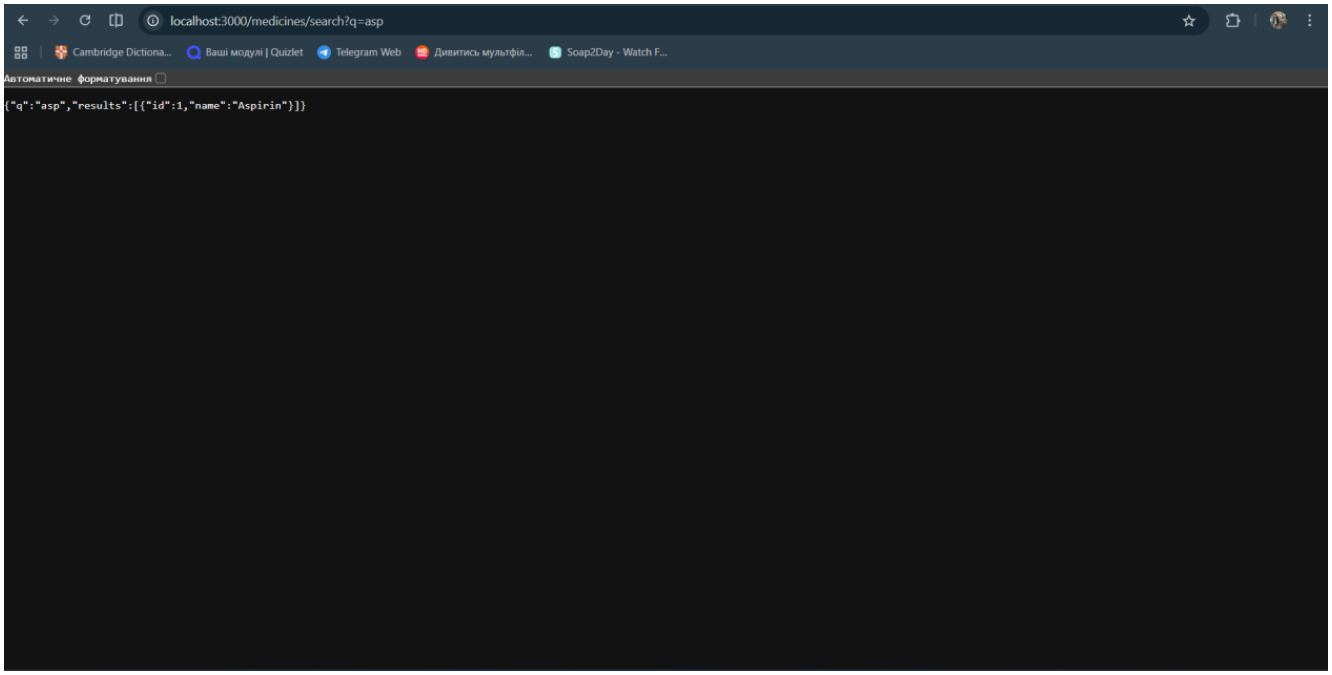

Рисунок Д.4 – Приклад відповіді endpoint пошуку препаратів

ДОДАТОК Е. ЕКРАННІ ФОРМИ РЕАЛІЗОВАНОГО ВЕБЗАСТОСУНКУ APTEKANEAR

У додатку наведено екранні форми реалізованої клієнтської частини вебзастосунку AptekaNear, що доповнюють опис у підрозділі 4.4. Зображення демонструють основні сторінки та стани інтерфейсу на демонстраційному наборі даних.

На рисунку Е.1 зображено каталог лікарських засобів.

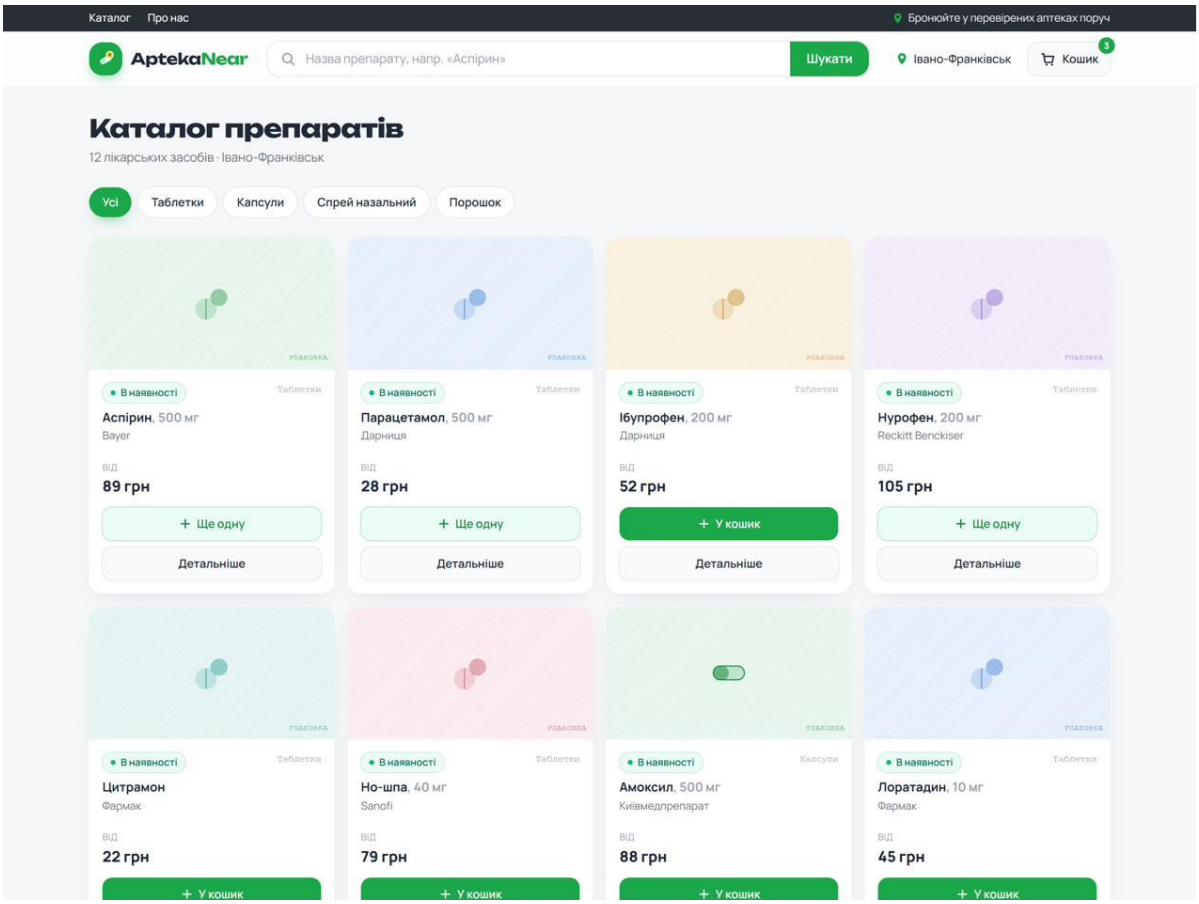

Рисунок Е.1 – Каталог лікарських засобів

На рисунку Е.2 зображено результати пошуку препарату за назвою.

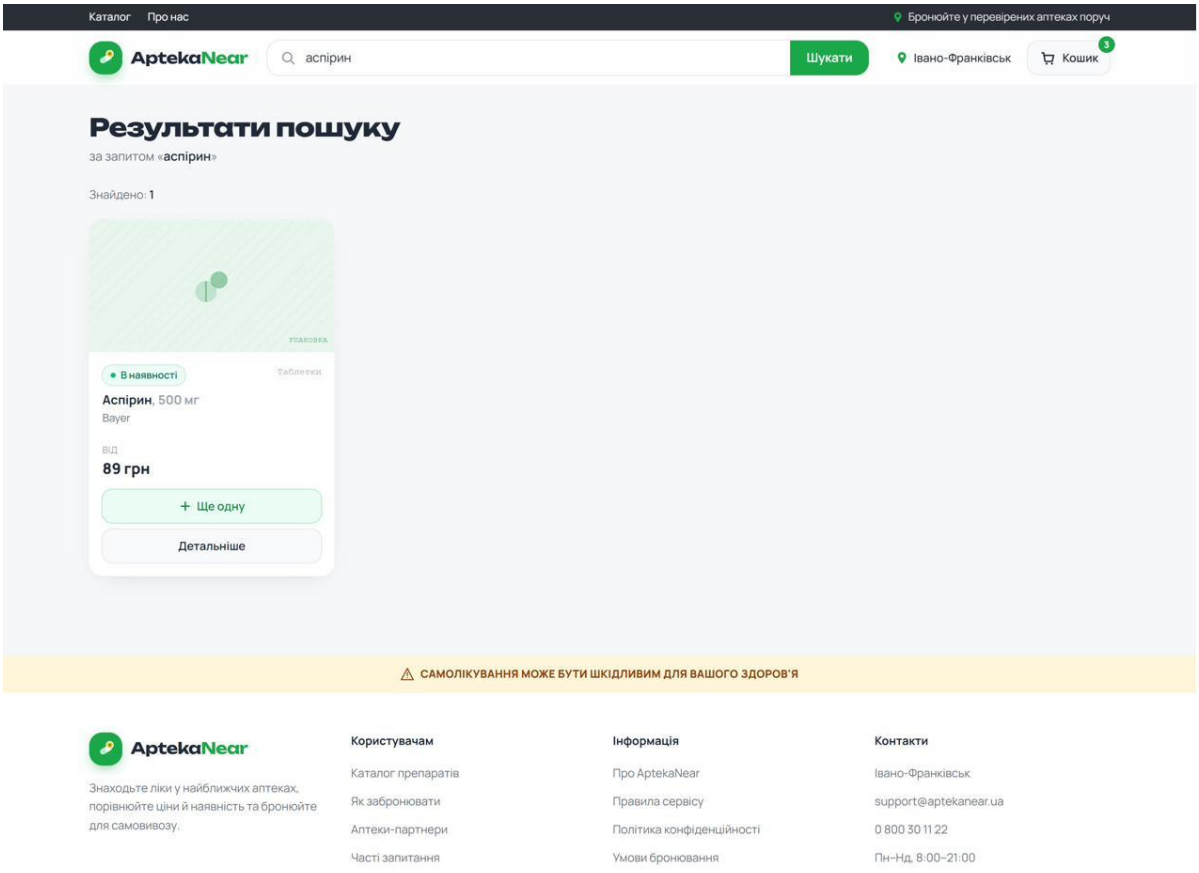

Рисунок Е.2 – Результати пошуку препарату

На рисунку Е.3 зображено кошик користувача.

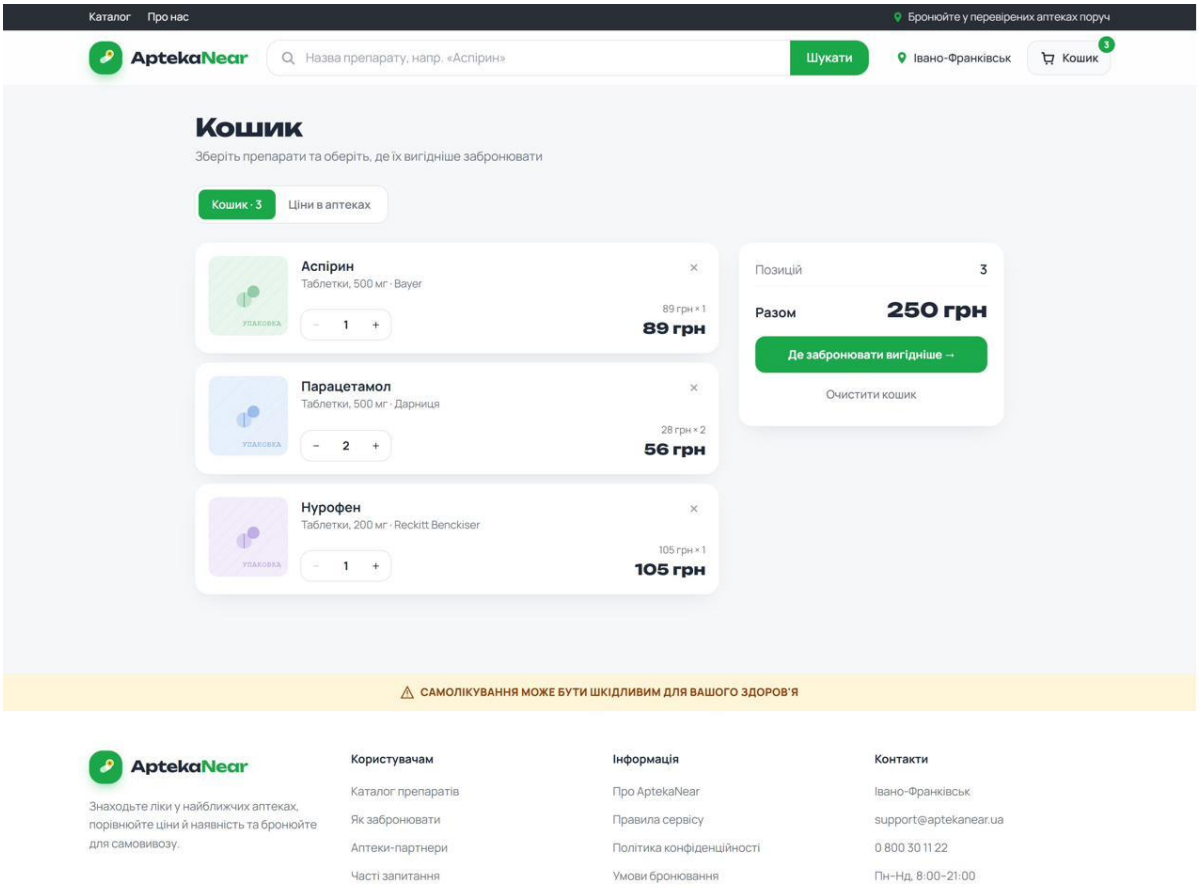

Рисунок Е.3 – Кошик користувача

На рисунку Е.4 зображено порівняння цін за аптеками (вкладка «Ціни в аптеках»).

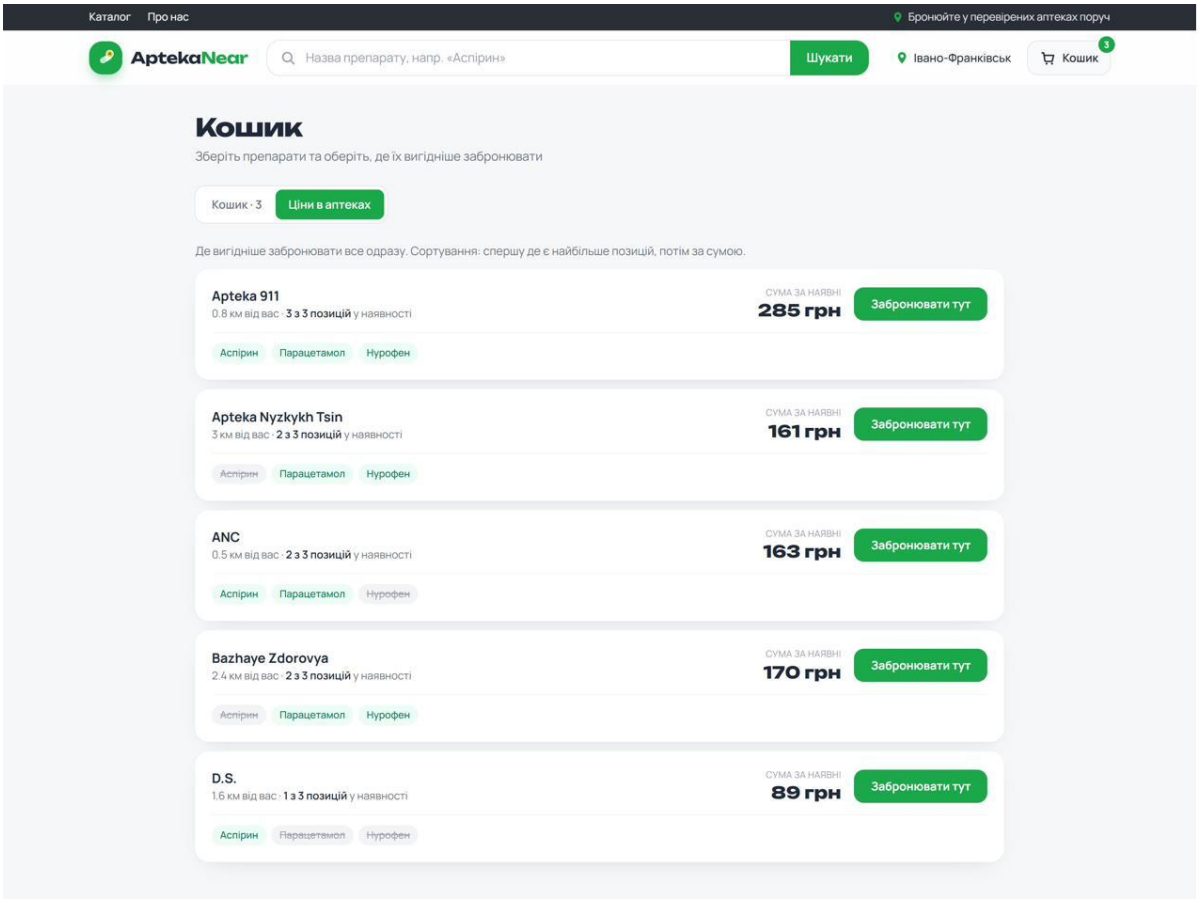

Рисунок Е.4 – Порівняння цін за аптеками

На рисунку Е.5 зображено інформаційну сторінку «Про нас».

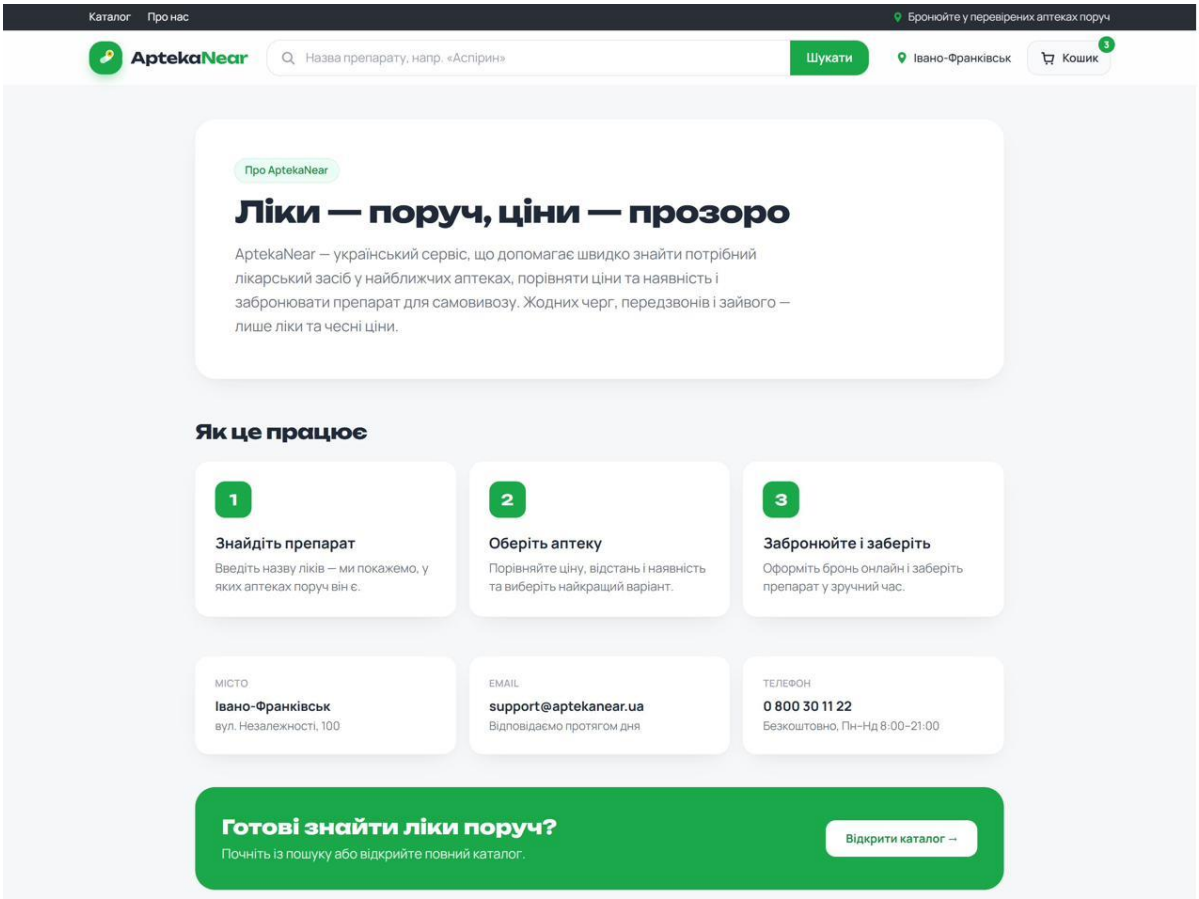

Рисунок Е.5 – Інформаційна сторінка «Про нас»

БІБЛІОГРАФІЧНА ДОВІДКА

Тема бакалаврської роботи: «Застосування принципів UI/UX-дизайну у фронт-енд веб-розробці»

Обсяг пояснювальної записки: 72 аркушів

Дата закінчення бакалаврської роботи «____» _____ 2026 р.

Підпис студента _____