МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МАРІУПОЛЬСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ЕКОНОМІКО-ПРАВОВИЙ ФАКУЛЬТЕТ
КАФЕДРА СИСТЕМНОГО АНАЛІЗУ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

До захисту допустити:
В.о. зав. кафедри

Ганна МАРТИНЮК

«04» червня 2025 р.

«ВЕБ-ЗАСТОСУНОК МАГАЗИНУ КОСМІЧНИХ ТОВАРІВ»

Кваліфікаційна робота
здобувача вищої освіти першого (бакалаврського) рівня вищої освіти освітньо-професійної програми «Комп'ютерні науки»
Д`яченка Вадима Дмитровича
Науковий керівник:
Дрейс Юрій Олександрович,
кандидат технічних наук, доцент,
доцент кафедри системного аналізу та інформаційних технологій
Рецензент:
Нечипорук Олена Петрівна,
доктор технічних наук, професор, завідувач кафедри інтелектуальних кібернетичних систем Державного університету «Київського авіаційного університету»

Кваліфікаційна робота захищена
з оцінкою відмінно 95 (А)
Секретар ЕК

«11» червня 2025 р.

Київ – 2025

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Скорочення / Позначення	Розшифрування / Пояснення
API	Application Programming Interface — інтерфейс прикладного програмування
CRUD	Create, Read, Update, Delete — базові операції з даними
CSS	Cascading Style Sheets — каскадні таблиці стилів
DOM	Document Object Model — об’єктна модель документа
HTML	HyperText Markup Language — мова гіпертекстової розмітки
JS	JavaScript — мова програмування для веб-розробки
JWT	JSON Web Token — формат передачі токенів для автентифікації

MongoDB	Документо-орієнтована база даних NoSQL типу
Next.js	React-фреймворк для створення серверних та клієнтських веб-застосунків
NextAuth	Бібліотека для реалізації автентифікації в Next.js
Node.js	Платформа для виконання JS-коду на сервері
REST API	Representational State Transfer — стиль архітектури для побудови API
SPA	Single Page Application — односторінковий веб-застосунок
SSR	Server Side Rendering — серверний рендеринг сторінок
Stripe	Платіжна система для прийому онлайн-платежів
StyledComponents	Бібліотека для CSS-in-JS стилізації компонентів
TailwindCSS	CSS-фреймворк з утилітарним підходом до стилізації

UI	User Interface — інтерфейс користувача
UX	User Experience — досвід користувача
Фреймворк	Програмна платформа, що задає структуру застосунку та включає набір готових функцій і правил для розробки
Бібліотека	Збірка готового коду, що забезпечує певний функціонал і підключається до проєкту для спрощення розробки

ВСТУП

Останні роки стали періодом стрімкого розвитку цифрових технологій, особливо в галузі веб-розробки. Сьогодні майже кожен аспект повсякденного життя так чи інакше пов’язаний із використанням онлайн-сервісів — від спілкування і навчання до роботи, дозвілля й, звісно, покупок. Інтернет-магазини вже давно стали звичним явищем, але попри це, попит на нові рішення у сфері створення таких застосунків не зменшується. Навпаки — з розвитком фреймворків, бібліотек та нових архітектур зростають і очікування користувачів, і вимоги до функціональності та зручності інтерфейсу. У таких умовах створення веб-застосунків залишається актуальним як з технічної, так і з прикладної точки зору.

Розробка повноцінного магазину у формі веб-застосунку — це завдання, яке дозволяє не лише застосувати знання з сучасної веб-розробки, але й протестувати, наскільки ефективно різні інструменти й підходи працюють разом у реальних умовах. Такий проєкт має як практичне значення, так і навчальну цінність, оскільки дає змогу закріпити теоретичні знання на прикладі цілісного функціонального продукту.

Тематика самого магазину — а саме торгівля товарами, пов’язаними з космосом (метеорити, зразки космічного пилу, уламки астероїдів тощо) — була обрана не випадково. Вона є своєрідною стилістичною рамкою, яка додає проєкту концептуальності й дозволяє подивитися на знайому структуру інтернет-магазину під новим кутом. Такий підхід не стільки змінює суть самого магазину, скільки демонструє, як змістовне наповнення може впливати на вигляд, атмосферу і навіть функціональність веб-застосунку. Це також натяк на те, в якому напрямі може рухатися електронна комерція в майбутньому — коли кордони між фантазією та реальністю дедалі більше розмиваються завдяки технічному прогресу.

Таким чином, розробка концептуального магазину космічних товарів — це спроба не лише відтворити класичний сценарій онлайн-покупок, а й уявити, як виглядатиме подібний досвід у майбутньому. Актуальність такої теми полягає у поєднанні перевірених практик веб-розробки з оригінальною подачею ідей, яка, в свою чергу, відкриває простір для подальших експериментів і вдосконалень.

У центрі даного дослідження перебуває процес створення сучасного веб-застосунку, що відображає типову архітектуру й функціональність інтернет-магазину. Об’єктом виступає сам феномен веб-розробки як інструменту для реалізації комерційних сервісів у цифровому середовищі. Це — широка сфера, яка охоплює принципи побудови користувацьких інтерфейсів, організацію роботи з базами даних, безпекові аспекти, питання інтеграції сторонніх сервісів та взаємодію між клієнтською й серверною частинами.

Предметом дослідження є безпосередньо процес розробки концептуального веб-застосунку магазину з використанням сучасних інструментів і технологій, таких як фреймворк Next.js, база даних MongoDB, бібліотеки TailwindCSS та StyledComponents тощо. Саме вивчення взаємозв’язку між технологіями, архітектурними підходами та конкретним функціональним наповненням застосунку визначає основне спрямування цієї роботи.

На відміну від абстрактного аналізу технологій у відриві від практичного використання, у цій роботі предмет дослідження розглядається в контексті реальної реалізації: від ідеї до функціонального застосунку, який, попри концептуальну тематику, зберігає усі риси повноцінного інтернет-магазину.

Таким чином, об’єкт і предмет дослідження знаходяться у взаємозв’язку: з одного боку — загальний процес веб-розробки як технологічного явища, з іншого — конкретна реалізація цього процесу у

вигляді концептуального застосунку, що відображає певне бачення майбутнього інтернет-торгівлі.

Розробка веб-застосунків вже давно є самостійним напрямом у галузі інформаційних технологій та займає важливе місце в сучасному цифровому середовищі. Наукова та технічна література пропонує широкий спектр досліджень, присвячених архітектурним рішенням, вибору технологій, взаємодії з базами даних, реалізації безпечної авторизації, оптимізації користувацького досвіду, використанню JavaScript-фреймворків та бібліотек, організації обміну даними через API та іншим дотичним питанням.

Певна частина досліджень фокусується на великомасштабних або високонавантажених системах, а також на вузькоспеціалізованих галузях, зокрема фінансових сервісах чи платформах електронного урядування. Натомість концептуальні чи навчальні проєкти — зокрема ті, що створені з експериментальною або освітньою метою — зазвичай залишаються поза полем академічної уваги, хоча й мають значну цінність для розуміння прикладного використання сучасних технологій у межах повноцінного функціонального застосунку.

У межах цієї кваліфікаційної роботи реалізовано типовий приклад інтернет-магазину з усіма базовими функціями — перегляд товарів, оформлення замовлень, адміністрування контенту тощо. Обрана тематика товарів має радше умовний характер і не впливає на технічну реалізацію. Вона лише задає візуальну стилістику та концепцію, яка дозволяє поглянути на звичну модель продажів крізь призму гіпотетичного майбутнього. Це, у свою чергу, відкриває простір для творчого переосмислення стандартних архітектурних рішень та створення нового користувацького досвіду в межах уже знайомих механізмів.

Метою цієї кваліфікаційної роботи є створення концептуального веб-застосунку інтернет-магазину, що демонструє практичне застосування сучасних інструментів веб-розробки в умовах гіпотетичного майбутнього ринку. Застосунок реалізує стандартну модель електронної комерції, водночас

експериментує з нетиповою товарною пропозицією — як засобом візуального та тематичного переосмислення користувацького досвіду.

У межах проєкту передбачено досягнення таких завдань:

1. Провести огляд сучасних архітектурних підходів до створення веб-застосунків, зокрема з урахуванням клієнт-серверної моделі, SSR/CSR-рішень і підходів до організації API.

2. Визначити доцільність використання окремих технологій — фреймворку Next.js, бази даних MongoDB, бібліотек для авторизації, стилізації та обробки платежів — з урахуванням поставлених цілей.

3. Спроєктувати архітектуру застосунку, що складається з двох логічно відокремлених частин — клієнтської (вітрина магазину) та адміністративної (панель керування товарами й замовленнями).

4. Реалізувати усі основні функціональні модулі: перегляд товарів, додавання їх до кошика, оформлення замовлень, додавання товарів і категорій у панелі керування, перегляд замовлень, адміністрування контенту.

5. Забезпечити зв’язок між частинами застосунку через єдину базу даних, забезпечивши актуальність і цілісність інформації.

6. Інтегрувати платіжну систему для моделювання реального процесу оплати.

7. Провести локальне тестування працездатності застосунку та зробити висновки щодо його можливого розгортання в майбутньому.

Ці завдання сформовані відповідно до обраної теми, об'єкта та предмета дослідження, і разом утворюють структуру логічного та поетапного розгортання всієї кваліфікаційної роботи.

При підготовці дослідження було опрацьовано низку джерел, що відображають поточний стан розвитку веб-технологій, сучасних фреймворків, підходів до побудови архітектури застосунків, а також інструментів для організації процесів розробки, авторизації та інтеграції з платіжними сервісами. Особливу увагу було приділено офіційній документації технологій, таких як Next.js, MongoDB, TailwindCSS, StyledComponents, NextAuth та

Stripe, які безпосередньо використовуються в реалізації проєкту. Поряд із технічною базою враховано аналітичні публікації й приклади з відкритих джерел, що ілюструють кращі практики у сфері створення адаптивних та масштабованих веб-застосунків.

Методологічна основа дослідження спирається на аналітичний підхід, поєднаний із практико-орієнтованим моделюванням. Вивчення наявних рішень дозволило визначити доцільні технології для реалізації концепції, а подальше етапне проєктування і тестування компонентів дало змогу практично підтвердити їхню ефективність. Робота також включала елементи порівняльного аналізу (при виборі фреймворків і бібліотек), а також метод спостереження за поведінкою системи в умовах локального запуску.

Незважаючи на концептуальний характер проєкту, його реалізація має прикладне значення. По-перше, застосунок демонструє ефективне поєднання сучасних технологій у межах одного рішення, що може слугувати прикладом для майбутніх проєктів, незалежно від обраної тематики. По-друге, структура та логіка побудови застосунку можуть бути адаптовані для створення реальних електронних торговельних платформ. А обраний вектор футуристичного магазину є передусім творчою спробою зазирнути в потенційні напрями цифрової комерції, що невід’ємно пов’язана з розвитком інтерфейсів та доступом до нових видів продуктів.

РОЗДІЛ 1. АНАЛІЗ ВІДОМИХ АРХІТЕКТУР, ТЕХНОЛОГІЙ ТА РИНКУ

1.1. Клієнт-серверна модель. SPA та SSR

Розвиток веб-технологій від початку був тісно пов'язаний із клієнт-серверною архітектурою, яка донині залишається базовою моделлю взаємодії у мережевих застосунках. За цією моделлю, клієнт (браузер користувача) ініціює запит до сервера, а сервер, у свою чергу, обробляє цей запит і повертає відповідь. Незважаючи на її відносну простоту, саме ця модель дала поштовх до створення різноманітних підходів до реалізації інтерфейсів користувача, серед яких особливе місце посідають SPA (Single Page Application) та SSR (Server Side Rendering) [1].

Односторінкові застосунки (SPA) стали популярними завдяки своїй здатності створювати динамічні, швидкі й інтерактивні інтерфейси. У таких проєктах основне завантаження сторінки відбувається один раз, а подальші зміни інтерфейсу реалізуються на клієнтському боці за допомогою JavaScript. Користувач взаємодіє з додатком без потреби в повному оновленні сторінки, що створює враження нативного застосунку. Цей підхід став можливим завдяки появі потужних JavaScript-бібліотек і фреймворків, серед яких одним із найпоширеніших є React [10].

На відміну від SPA, серверний рендеринг (SSR) передбачає формування вмісту сторінки безпосередньо на сервері перед її передачею клієнту. Таким чином, користувач отримує готовий HTML-код, який може бути відображений у браузері ще до повного завантаження JavaScript. Це дає перевагу в контексті пошукової оптимізації (SEO, англ. Search Engine Optimization), тобто здатності сайту бути коректно індексованим пошуковими системами (наприклад, Google) та з’являтися у видачі за релевантними запитами користувачів. Важливо зазначити, що сучасні фреймворки, зокрема Next.js, пропонують можливість поєднання обох підходів — тобто створення гібридних

застосунків, у яких окремі сторінки можуть рендеритись на сервері, а інші — функціонувати за логікою SPA [2].

Така гнучкість дозволяє розробникам адаптувати архітектуру застосунку до конкретних потреб: швидкість, динаміка взаємодії, вимоги до SEO, безпека чи навантаження на сервер. З огляду на це, аналіз SPA та SSR є важливим етапом при виборі технологічної основи для будь-якого сучасного веб-застосунку, зокрема й того, що розробляється в межах цієї кваліфікаційної роботи.

1.2. Серверні та клієнтські технології

У сучасній веб-розробці будь-який застосунок умовно поділяється на дві ключові складові — клієнтську (frontend) та серверну (backend). Перша відповідає за те, як виглядає й функціонує застосунок в браузері користувача, друга — за обробку даних, взаємодію з базою даних, авторизацію, захист та бізнес-логіку.

Значну частину клієнтської частини в сучасних умовах розробляють із використанням бібліотек або фреймворків. Хоча ці поняття часто плутають, між ними існує суттєва різниця: бібліотека надає розробнику набір функцій, якими він керує сам, тоді як фреймворк диктує певну структуру та організацію коду, в яку розробник має вписатися.

Найпопулярнішими клієнтськими технологіями сьогодні є: React — JavaScript-бібліотека для побудови інтерфейсів. Вона працює за компонентним принципом, дозволяє ефективно оновлювати DOM завдяки використанню Virtual DOM, і має потужну екосистему [10].

Vue.js — прогресивний фреймворк, який поєднує простоту синтаксису з гнучкістю масштабування [9]. Angular — потужний фреймворк від Google з жорсткою архітектурою, що підходить для великих корпоративних проєктів [9].

Ці технології часто поєднують із серверними інструментами, створюючи так звані технологічні зв’язки. Наприклад: React + Node.js —

поєднання бібліотеки для створення інтерфейсу з JavaScript-середовищем для серверної логіки. Такий зв’язок дозволяє писати як frontend, так і backend на одній мові — JavaScript — що спрощує командну роботу, зменшує залежність від сторонніх технологій і прискорює розробку [10][23]. Vue + Express — схожий зв’язок, де Express використовується як легкий backend-фреймворк для Node.js.

У таких зв’язках фронтенд та бекенд можуть існувати як окремі сервіси, що взаємодіють через API, або бути об’єднані в одному проєкті за допомогою фреймворків на зразок Next.js (у випадку з React), який дозволяє реалізовувати SSR, покращувати SEO та організовувати маршрутизацію [2].

На серверній частині найбільш поширеними є:
Node.js — середовище виконання JavaScript поза браузером. Його перевага — асинхронна модель обробки запитів, що дає змогу ефективно працювати з великою кількістю одночасних користувачів [23].
Python (Django, Flask) — мова, що дозволяє швидко будувати серверну логіку з гарною читаємістю коду. PHP (Laravel) — традиційний варіант, який досі часто використовується в поєднанні з CMS.

Окремої уваги заслуговують CMS — Content Management Systems, або системи керування вмістом. Вони дозволяють створювати сайти без необхідності програмування, орієнтуючись на адміністрування контенту. Прикладами є WordPress, Joomla чи Drupal. Хоча вони значно спрощують створення вебсайтів, їх гнучкість і безпека часто обмежені, що робить їх менш придатними для складних SPA або динамічних онлайн-застосунків.
Таким чином, у контексті розробки повноцінного застосунку сучасного типу, більш перспективними виглядають зв’язки бібліотек типу React із серверними середовищами, як-от Node.js, що забезпечує як продуктивність, так і гнучкість масштабування.

Такий підхід значно спрощує розробку: один і той самий фахівець може працювати і над клієнтською, і над серверною частиною, зменшуючи витрати часу на контекстні перемикання між мовами та парадигмами. Використання

Next.js API Routes дозволяє реалізувати серверну логіку прямо в межах самого застосунку, не вимагаючи окремого сервера або окремого бекенд-фреймворку [2].

Таким чином, вибір технологій був зумовлений прагненням до оптимального балансу між продуктивністю, простотою підтримки, перспективністю розвитку та зручністю реалізації. Це особливо актуально в контексті концептуального експерименту з уявним магазином космічних товарів, де важливішим є не новизна самих технологій, а гармонійна їхня інтеграція у сучасну модель взаємодії користувача з цифровими сервісами.

1.3. Бази даних у веб-розробці

В основі будь-якого сучасного веб-застосунку, який оперує динамічним контентом або зберігає дані користувачів, лежить система керування базами даних (СКБД). Бази даних — це, по суті, основа, на якій тримається логіка збереження, обробки та подальшого використання інформації. Залежно від типу застосунку, очікуваного навантаження, масштабів та потреб гнучкості, обираються ті чи інші технології зберігання даних [21].
У веб-розробці можна виділити два основних підходи до організації баз даних:

Реляційні бази даних — структуровані СКБД, в яких дані зберігаються у вигляді таблиць із чіткими зв’язками між ними (foreign keys). Прикладами таких є MySQL, PostgreSQL, SQLite. Вони ідеально підходять для систем, де важливі чіткі залежності, транзакції, консистентність даних [21].

Нереляційні бази даних (NoSQL) — системи, що відходять від класичної табличної структури й зберігають дані у вигляді документів, пар “ключ-значення” або графів. Найвідомішим представником є MongoDB — документоорієнтована СКБД, яка зберігає дані у форматі, подібному до JSON (BSON) [4]. Такі бази відзначаються гнучкістю, масштабованістю та швидкою інтеграцією з JavaScript-орієнтованими середовищами, як-от Node.js [23].

Розподіл на реляційні та нереляційні системи не є взаємовиключним: кожен тип має свої сильні сторони. Наприклад, PostgreSQL поєднує класичну

SQL-архітектуру з можливістю зберігання JSON-структур, а MongoDB дозволяє індексацію та агрегацію на високому рівні, що наближає її до реляційних аналогів за частиною функціональності [21][4].

У контексті веб-застосунків, зокрема онлайн-магазинів, вибір бази даних часто визначається такими факторами:

Гнучкість структури. В нереляційних БД, наприклад MongoDB, можна зберігати різні типи товарів із різною структурою даних без необхідності створення складних схем [4].

Масштабованість. NoSQL бази легше масштабуються горизонтально — додаючи нові сервери до системи, що важливо при зростанні аудиторії [4].

Швидкість розробки. У стартапах та експериментальних проектах гнучкість NoSQL дозволяє швидше протестувати ідею без потреби витрачати час на складне моделювання схеми [4].

Водночас у класичних проєктах із чіткою структурою — наприклад, бухгалтерських системах чи проєктах, де важлива транзакційна цілісність — доцільніше використовувати реляційні БД [21].

Також не менш важливою є інтеграція з серверною частиною. У JavaScript-орієнтованих стосах на кшталт MERN (MongoDB, Express, React, Node) використання MongoDB виглядає логічним і зручним, оскільки дозволяє працювати з даними у вигляді об’єктів прямо з клієнтської або серверної частини, що мінімізує трансформацію даних [4][10][23].

Окремо варто згадати про ORM/ODM — інструменти для роботи з базами даних через об’єктно-орієнтований інтерфейс. Наприклад, Mongoose для MongoDB чи Prisma для SQL-баз. Вони дозволяють будувати абстракції, забезпечують типізацію (в разі TypeScript), валідацію та зручну побудову запитів без прямого написання SQL.

Підсумовуючи, у сучасній веб-розробці вибір між типами баз даних і конкретними СКБД залежить не лише від технічних вимог, а й від концепції проєкту, темпів його зростання та внутрішньої структури застосунку. У проєктах, що претендують на футуристичний стиль або експериментують з

нетиповими підходами, схильність до гнучких NoSQL-рішень виглядає природною та стратегічно вмотивованою [4][21].

1.4. Використання бібліотек і фреймворків у розробці

У сучасному середовищі фронтенд-розробки важко уявити повноцінний веб-застосунок без залучення спеціалізованих бібліотек та фреймворків. Вони не лише спрощують процес написання коду, а й надають гнучкі інструменти для побудови інтерфейсів, управління станом, маршрутизацією та стилізацією.

Однією з важливих категорій є CSS-бібліотеки, які відповідають за візуальну складову інтерфейсу. На відміну від класичного підходу, де стилі створювались вручну або з використанням препроцесорів (наприклад, SASS), сучасні CSS-рішення орієнтовані на масштабованість, компонентність та швидкість розробки. Вони дозволяють уникнути дублювання коду, зменшують кількість стилістичних помилок та підтримують дизайн-системи.

Серед найбільш популярних рішень:
Tailwind CSS — утилітарна бібліотека, що дозволяє створювати інтерфейси за допомогою готових класів. Такий підхід заохочує інлайн-стилізацію без необхідності писати власні CSS-файли. Tailwind сприяє швидкому прототипуванню, особливо на початкових етапах проєкту, а також добре масштабуються в рамках командної розробки [13].
Styled Components — рішення для написання стилів у JavaScript, яке добре поєднується з React. Воно дозволяє створювати ізольовані стилі для окремих компонентів, що особливо важливо в проєктах зі складною структурою. Крім того, Styled Components підтримує темізацію, автоматичне перейменування класів для уникнення конфліктів і полегшує повторне використання стилізованих елементів [3].

Bootstrap або Material UI — класичні приклади CSS-фреймворків із вбудованими готовими компонентами інтерфейсу. Хоча ці рішення менш гнучкі в порівнянні з Tailwind чи Styled Components, вони залишаються

актуальними завдяки стабільності, підтримці адаптивності та швидкому старту проєкту [9].

Вибір тієї чи іншої бібліотеки залежить не лише від уподобань розробника, але й від вимог до проєкту: обсяг функціоналу, складність дизайну, очікуване навантаження, потреба в адаптивності. У кожному випадку CSS-бібліотеки виконують ключову роль у забезпеченні візуальної цілісності, підтримуваності та ефективності розробки.

Таким чином, грамотне використання CSS-бібліотек дозволяє зменшити час на створення інтерфейсу, покращити досвід користувача та забезпечити високий рівень консистентності в усьому застосунку [13][3].

1.5. Аналіз сучасного ринку веб-рішень для електронної комерції

Сучасний ринок веб-рішень для електронної комерції характеризується стрімким зростанням обсягу онлайн-продажів та високою потребою у гнучких, масштабованих та безпечних системах управління [15]. З огляду на глобальну цифрову трансформацію, зростає попит на інтерфейси, що поєднують інтуїтивність, адаптивність і інтегровану функціональність. Системи будуються з використанням багаторівневої архітектури, де клієнтська та серверна частини взаємодіють через чітко визначені API, що забезпечує модульність, легкість підтримки та зручність оновлення вмісту [21].

До прикладів сучасних рішень можна віднести: Shopify – глобальна платформа для створення інтернет-магазинів, що забезпечує високу масштабованість і широкий спектр інтеграцій із сервісами платежів та логістики. Цей приклад свідчить про важливість централізованого керування асортиментом та зручності користування, що є ключовими вимогами сучасного ринку [19].

BigCommerce – система електронної комерції, орієнтована на великі підприємства, яка дозволяє інтегрувати численні зовнішні сервіси для автоматизації продажів та маркетингу. Її функціональність демонструє

тенденцію до комплексної інтеграції ERP-систем із платформами онлайн-продажів [15].

Magento (Adobe Commerce) – рішення для висококонкурентного ринку, що дозволяє створювати індивідуалізовані платформи з широкою кастомізацією, забезпечуючи гнучкість у роботі з великим асортиментом товарів і адаптивними торговими процесами [15].

Окрім цього, значну увагу приділяється впровадженню механізмів персоналізації користувацького досвіду, зокрема, рекомендаційних систем, які базуються на аналізі поведінкових даних [16]. Інтеграція таких технологій дозволяє оптимізувати онлайн-продажі, підвищити конверсію та забезпечити більш глибоку взаємодію між користувачами і системою. Загалом, сучасні тенденції, зафіксовані на ринку веб-рішень, визначають необхідність створення систем, які здатні швидко адаптуватися до змін – як технічних (масштабованість, продуктивність, інтеграція з хмарними сервісами), так і ринкових (персоналізація, комплексна аналітика, інтеграція з зовнішніми рішеннями) [15]. В цьому контексті, обрані технологічні стеки й підходи забезпечують можливість створення інноваційних платформ, що відповідають сучасним вимогам користувачів і сприяють стратегічній конкурентоспроможності на ринку.

Висновок до розділу

У межах першого розділу було здійснено аналітичний огляд ключових тенденцій та технологічних рішень, що домінують у сфері сучасної веб-розробки, зокрема в контексті електронної комерції. Було проаналізовано особливості архітектурних підходів (зокрема SPA, SSR, JAMstack), інтерфейсні та серверні фреймворки, формати обміну даними, особливості баз даних і структурованого зберігання інформації, а також типові модулі й функціональні блоки, притаманні веб-застосункам e-commerce-сегменту.

Результати аналітики свідчать про загальну тенденцію до зростання гнучкості, продуктивності та адаптивності веб-рішень. Зокрема,

спостерігається переорієнтація на модульні, масштабовані та інтегровані підходи, які передбачають активне використання хмарних сервісів, API-орієнтовану логіку, та мультиканальність взаємодії з користувачами. Висвітлено також сучасні практики організації UI/UX, шляхи оптимізації продуктивності, принципи безпечної роботи з даними та способи інтеграції з платіжними й логістичними сервісами.

На підставі розглянутих матеріалів сформовано теоретичну базу для подальшого обґрунтування архітектури, структури й реалізаційних рішень, що можуть бути застосовані під час проєктування та створення веб-застосунків у сфері електронної комерції, зокрема інноваційного типу.

РОЗДІЛ 2. ВИБІР АРХІТЕКТУРИ ЗАСТОСУНКУ

2.1. Вибір загальної архітектури

Побудова будь-якого сучасного веб-застосунку потребує чіткого визначення архітектурного підходу, який дозволить забезпечити ефективність, гнучкість і надійність системи в умовах постійного розвитку цифрового середовища [21]. Архітектура застосунку визначає, як взаємодіють між собою клієнтська частина, серверна логіка, база даних та інші допоміжні сервіси. Від її вибору залежить зручність масштабування, адаптації до нових вимог та подальшої підтримки програмного продукту.

Серед найпоширеніших архітектурних рішень, які сьогодні використовуються в розробці веб-застосунків, виділяють монолітну архітектуру, мікросервісну модель та класичну клієнт-серверну архітектуру [21]. Кожна з них має свої переваги в контексті певних бізнес-задач та технічних умов.

Монолітна архітектура передбачає збирання всієї логіки в один нерозділений застосунок. Такий підхід може бути доречним на початкових етапах проєктів з невеликою складністю, однак зі зростанням функціоналу він створює труднощі в оновленні, масштабуванні й незалежному тестуванні окремих частин [21].

Мікросервіси, навпаки, дозволяють розбити функціонал на невеликі незалежні сервіси, які взаємодіють через API. Це підвищує гнучкість, але одночасно збільшує складність у керуванні, синхронізації й забезпеченні безперебійної роботи всіх частин системи [21]. Для застосунків середнього масштабу, з типовим бізнес-процесом і обмеженим числом функцій, впровадження мікросервісів зазвичай є недоцільним.

Найоптимальнішим варіантом для розробки веб-магазину виступає клієнт-серверна архітектура, яка забезпечує чіткий поділ на інтерфейс користувача (frontend) та серверну частину (backend). Такий підхід дозволяє:

- створювати масштабований інтерфейс з динамічною взаємодією;
- централізовано обробляти бізнес-логіку на сервері;
- зручно керувати базою даних та користувацькими запитами;
- впроваджувати адміністративну панель як окремий інтерфейс без дублювання логіки.

Архітектура була обрана з урахуванням потреб звичайного інтернет-магазину: можливість додавання та редагування товарів, обробка замовлень, збереження даних користувачів, швидке оновлення інтерфейсу, адаптація під мобільні пристрої [21]. Застосунок реалізовано як багатокомпонентну систему, де клієнтська частина відповідає за взаємодію з користувачем, а серверна — за обробку запитів, роботу з базою даних і захист персональних даних.

Таким чином, вибір клієнт-серверної архітектури дозволяє забезпечити як технічну стабільність, так і подальшу гнучкість у розробці, розширенні й підтримці веб-застосунку.

2.2. Вибір клієнтської та серверної технології

Після визначення загальної архітектурної моделі веб-застосунку, наступним логічним етапом стало обґрунтування вибору інструментів реалізації як клієнтської, так і серверної частин. Оскільки мова йде про веб-магазин із динамічним наповненням, очікуваною інтерактивністю й потребою забезпечення належної оптимізації під пошукові системи, було обрано рішення на користь використання фреймворку Next.js [2].

З технічної точки зору, Next.js є надбудовою над React, яка надає гнучкий інструментарій для створення застосунків з підтримкою як клієнтського, так і серверного рендерингу [2]. Такий підхід дозволяє поєднати переваги обох моделей: з одного боку, забезпечується швидке завантаження початкової сторінки та належна індексація в пошукових системах (що особливо важливо для публічного застосунку), з іншого — реалізується

плавна навігація між сторінками без повного перезавантаження, що покращує користувацький досвід [2].

Серед альтернатив, які розглядалися, варто згадати такі інструменти як:

- Create React App — зручний стартовий інструмент для SPA-застосунків, але він не передбачає серверного рендерингу, що робить його менш ефективним з точки зору SEO.
- Vue.js або Nuxt.js — теж популярні технології, проте їхня екосистема менш поширена в середовищі, орієнтованому на React-розробку, і тому вони поступилися в даному виборі.
- Angular — потужний фреймворк, що вимагає значно більшого обсягу налаштувань, а отже, витрат часу на розгортання проєкту.

Окрему увагу слід звернути на те, що Next.js надає можливість реалізовувати обробку запитів безпосередньо всередині застосунку через API-роути [2]. Таким чином, серверна логіка може бути частково інтегрована в межах єдиного репозиторію та розгортання, без потреби окремого сервера на базі Express чи NestJS. Це не лише спрощує структуру, але й знижує вартість підтримки проєкту в довгостроковій перспективі.

З огляду на потребу в ефективному зберіганні даних (каталогу товарів, категорій, замовлень тощо), обрано MongoDB — документно-орієнтовану базу даних, що добре поєднується з JavaScript-екосистемою та забезпечує гнучку структуру зберігання [4].

Отже, вибір Next.js у поєднанні з MongoDB був продиктований низкою як технічних, так і організаційних чинників. Такий стек технологій дозволяє досягти належного балансу між швидкістю розробки, зручністю підтримки та функціональністю, що відповідає сучасним вимогам до веб-застосунків комерційного типу [2][4].

2.3. Вибір бібліотек для стилізації інтерфейсу

Для стилізації інтерфейсу клієнтської частини застосунку було обрано бібліотеку Styled Components [3]. Цей інструмент дозволяє писати CSS

безпосередньо в JavaScript, що спрощує структуру проєкту, дозволяє інкапсулювати стилі в межах компонентів і уникати конфліктів імен [3]. Рішення було ухвалено також з міркувань особистого професійного розвитку — для ознайомлення з технологією CSS-in-JS у реальному проєкті. Використання Styled Components дало змогу організувати стилі в єдиному середовищі з логікою компонентів React, що спростило підтримку коду та забезпечило гнучкість при роботі з темами або умовною стилізацією [3]. Для створення адміністративної частини було використано TailwindCSS — утилітарну CSS-бібліотеку, яка забезпечує високу швидкість верстки та компактність коду [13]. Tailwind дає змогу безпосередньо в HTML-розмітці задавати стилі, що значно прискорює розробку інтерфейсів із повторюваними елементами.

Обидві частини застосунку мали різні вимоги: публічна частина потребувала більшої гнучкості та чистоти стилів, тоді як адмін-панель — швидкості та простоти. Такий розподіл технологій виявився ефективним для обох завдань.

Висновок до розділу

У другому розділі було здійснено послідовне технічне обґрунтування архітектурних і технологічних рішень, що стали основою для реалізації веб-застосунку типу інтернет-магазину. На основі попереднього аналітичного огляду обрано архітектурну модель клієнт-серверного типу, яка виявилася найбільш релевантною для застосунку середнього рівня складності з типовими e-commerce-функціями.

Під час аналізу можливих архітектурних варіантів (моноліт, мікросервіси, клієнт-сервер) було виявлено, що мікросервісна модель не є оптимальною з огляду на ресурсоємність підтримки, а моноліт — надмірно обмежений для проєкту, що передбачає розділення відповідальностей. Клієнт-серверна архітектура дозволила досягти балансу між централізацією логіки та незалежністю інтерфейсів.

Особливу увагу приділено вибору технологій для фронтенду й бекенду. Аналіз сучасних інструментів (Next.js, Create React App, Angular, Nuxt.js тощо) засвідчив переваги Next.js як фреймворку, що поєднує зручність React-екосистеми з можливістю серверного рендерингу, що важливо для SEO-оптимізації та швидкого завантаження сторінок. Крім того, можливість реалізації API-логіки безпосередньо в межах Next.js спростила інфраструктуру розгортання, уникнувши потреби у сторонньому серверному середовищі.

Як база даних обрано MongoDB — документно-орієнтовану систему, яка добре узгоджується з динамічною структурою e-commerce-даних і з JavaScript-екосистемою загалом. Вона забезпечує гнучкість у моделюванні даних та високу масштабованість, що є важливим у контексті майбутнього розширення функціоналу.

Окрему роль відіграло рішення щодо стилізації інтерфейсу. Публічна частина застосунку реалізована з використанням Styled Components, що дозволило інкапсулювати стилі в межах компонентів і підвищити керованість дизайном. Для адміністративної панелі обрано TailwindCSS — утилітарну бібліотеку, що значно прискорила процес верстки завдяки готовим класам і зрозумілій структурі.

Таким чином, вибір архітектури та технологічного стеку було здійснено з урахуванням реальних потреб проєкту, перспектив масштабування, зручності підтримки та актуальних тенденцій у галузі веб-розробки. Результатом стало формування узгодженого, сучасного та ефективного технічного підґрунтя для реалізації функціонального й адаптивного веб-застосунку.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1. Адміністративна панель застосунку

Як вже було вирішено раніше, архітектура веб-застосунку буде мати дворівневу структуру, яка умовно поділяється на адміністративну та клієнтську частини. Такий поділ сприяє ізоляції функціональних обов'язків, спрощує обслуговування системи й покращує безпеку даних. У цьому підрозділі буде поетапно розглянуто технічні рішення, що лягли в основу реалізації адміністративної частини.

3.1.1. Постановка цілі та етапів реалізації

У процесі проєктування архітектури адміністративної панелі веб-застосунку першочерговим завданням стало визначення її функціонального призначення та логіки внутрішньої організації. Панель мала забезпечити повний контроль над ключовими елементами інформаційної моделі магазину: товарами, категоріями, замовленнями та відгуками. Водночас необхідно було передбачити інтеграцію базових аналітичних механізмів, що дозволяють відстежувати прибутковість продуктів і динаміку їхньої популярності.

Відповідно до цього, розроблення адміністративної частини здійснювалося поетапно, із поступовим нарощуванням функціональності та деталізації інтерфейсу. Першим етапом стало створення уніфікованого шаблону, який би давав змогу створювати нові сторінки магазину без великої кількості повторюваного коду, а також виконував функцію перевірки користувача на кожній окремій сторінці сайту.

Наступним кроком стала реалізація функціональних сторінок відповідно до логіки предметної області. Було передбачено створення інтерфейсів для:

– керування товарною номенклатурою (з можливістю додавання, редагування, видалення товарів, завантаження зображень, задання цін, описів, категорій та властивостей);

– формування структури категорій, з підтримкою властивостей та вкладеності;

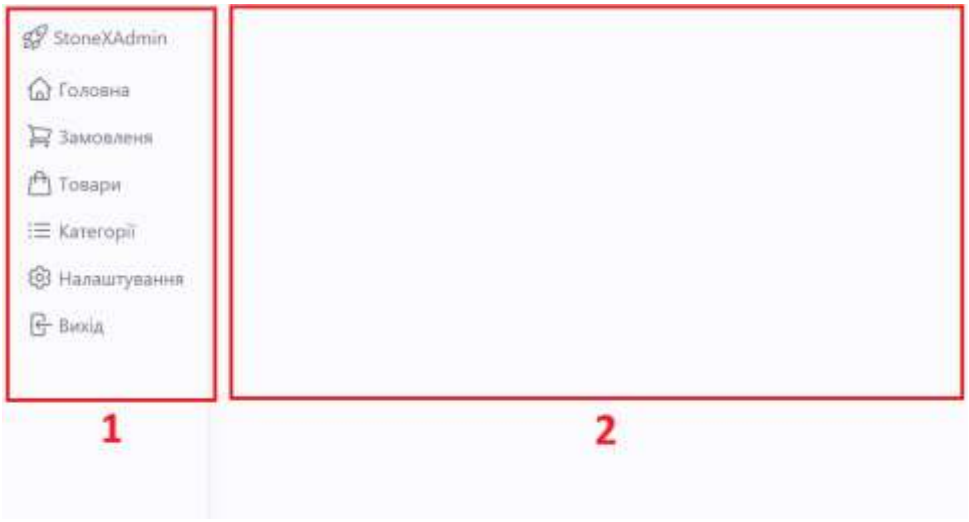
– перегляду й аналізу замовлень, включно з відображенням інформації про склад кожного замовлення та його покупця;

– відображення основних статистичних індикаторів, зокрема загального прибутку, кількості замовлень та визначення найуспішнішого продукту.

Паралельно було реалізовано механізм захищеного доступу до панелі керування, що базується на перевірці сесії через авторизацію з використанням стороннього провайдера. Це дозволило створити безпечне середовище, доступне лише уповноваженим особам. Усі зазначені етапи формували логічну послідовність у реалізації функціональної архітектури адміністративної частини, що забезпечила гнучкість, масштабованість і відповідність інтерфейсу завданням веб-застосунку в цілому.

3.1.2. Каркас інтерфейсу адміністративної панелі

Як вже було зазначено, початковим технічним кроком у реалізації адміністративної частини веб-застосунку стало створення каркасної структури інтерфейсу. Метою цього етапу було формування єдиного шаблону, який би забезпечував послідовне відображення контенту, централізоване керування навігацією та зручну організацію доступу до різних функціональних сторінок. Каркас інтерфейсу був реалізований на основі компонентного підходу, властивого фреймворку Next.js [3]. Центральну роль відіграє компонент Layout, який інкапсулює спільні для всіх сторінок елементи — зокрема, навігаційне меню, область динамічного контенту (children), логотип, а також логіку перевірки автентифікації користувача (Рис. 3.1.).



(Рис. 3.1. Каркас інтерфейсу. 1 - бічне меню з навігацією "Nav"; 2 - область основного контенту)

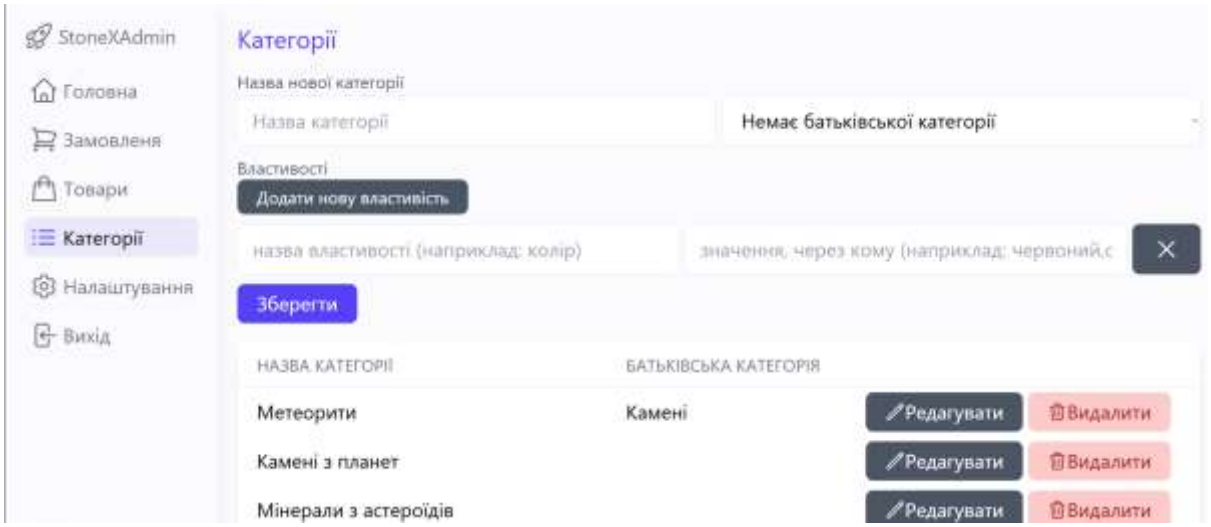
Усі підключені сторінки відображаються в межах цього шаблону, що забезпечує єдиний вигляд і поведінку адміністративної панелі. Для візуальної побудови інтерфейсу було використано утилітарну бібліотеку стилів Tailwind CSS. Вона дозволила швидко й гнучко реалізувати адаптивний дизайн, що зберігає цілісність як на великих екранах, так і на мобільних пристроях. Основні елементи каркасу — такі як відступи, кольори фону, скруглення, тіні та сітки — були налаштовані за допомогою Tailwind-класів, що сприяє читабельності й модульності коду. Однією з ключових функцій каркасу стала реалізація механізму автентифікації. Усі внутрішні сторінки адміністративної панелі захищені перевіркою сесії, яка здійснюється за допомогою функції useSession() з бібліотеки NextAuth [5]. Якщо користувач не авторизований, система перенаправляє його на сторінку входу, де передбачено вхід через Google-акаунт. Така інтеграція стороннього провайдера автентифікації спрощує управління доступом, підвищуючи рівень безпеки. Бічне навігаційне меню, реалізоване у вигляді окремого компонента Nav, дозволяє перемикатися між основними розділами панелі: товарами, категоріями, замовленнями, статистикою. Для мобільної версії меню реалізовано у вигляді адаптивного

висувного вікна, що відкривається через кнопку-гамбургер. Таким чином, забезпечено зручний доступ до функціоналу незалежно від типу пристрою. Каркас також передбачає чітке структурування вмісту сторінки: кожна з них має заголовок, розміщений у верхній частині, під яким знаходиться основна зона — таблиці, форми або графіки залежно від функціонального призначення. Така організація створює логічну й послідовну взаємодію користувача з інтерфейсом, мінімізуючи когнітивне навантаження та підвищуючи зручність керування. Реалізація каркасу створила основу для подальшого розгортання функціональних модулів панелі, зберігаючи єдність стилістики, зручність навігації та надійність доступу.

3.1.3. Сторінки керування категоріями та планетами

Сторінка керування категоріями (/categories) є фундаментальною для подальшого формування товарного асортименту, оскільки кожен товар обов’язково належить до певної категорії, що визначає перелік його параметрів (властивостей). Тому реалізація саме цієї сторінки стала першим кроком у побудові системи керування товарним контентом.

Сторінка складається з форми для додавання або редагування категорії та таблиці наявних категорій. Таблиця відображає назву категорії та, за наявності, батьківську категорію. В останньому стовпці містяться кнопки редагування й видалення (Рис 3.2).



(Рис. 3.2 Сторінка керування категоріями)

Форма створення категорії передбачає заповнення поля назви та визначення одного або кількох параметрів — властивостей. Кожна властивість має назву та перелік допустимих значень, що вводяться через кому. Наприклад, для категорії “Камені” можна додати властивість “Структура” зі значеннями “Щільна,Пориста,Зерниста,Вкраплена”.

Складність реалізації полягала у динамічному формуванні форми властивостей. З цією метою було створено механізм, який дозволяє додавати, редагувати та видаляти властивості без перезавантаження сторінки. Всі зміни зберігаються через API-запити до бази MongoDB, яка зберігає кожну категорію як окремий документ із полями name та properties.

Зважаючи на потребу в інтеграції категорій з формою створення товарів, формат збереження властивостей був уніфікований: кожна властивість має структуру { name: String, values: [String] }. Це дозволяє згодом у формі товару генерувати поля вибору значень на основі обраної категорії.

Таким чином, сторінка керування категоріями створила структурну основу, без якої неможлива була б логічна побудова товарної системи, і стала першим повноцінним модулем функціонального інтерфейсу адміністративної частини застосунку.

Окремо слід зазначити про сторінку керування планетами (/planets). Її функціональність схожа зі сторінкою керуванням категоріями, але має низку специфічних особливостей, пов’язаних із природою самих об'єктів, які вона обслуговує. Планета в контексті цього веб-застосунку не є простою категорією, а виступає як окремий інформаційний елемент із більш розширеним набором даних. Кожна планета містить такі поля, як назва, опис, відстань від Сонця, а також додаткове поле для фото планети, якщо воно буде потрібно, що може бути корисною для тематичного супроводу товарів або інтеграції з освітніми чи навігаційними модулями.

Сторінка керування планетами також складається з таблиці вже доданих записів та форми створення або редагування планети. У формі передбачено поля для введення назви, стислого опису, числового значення відстані від Сонця (у мільйонах кілометрів) та фото (Рис. 3.3.). Користувач має змогу додавати або змінювати записи без перезавантаження сторінки — через асинхронні API-запити.

(Рис 3.3. Сторінка керування планетами)

З технічного боку реалізація цієї сторінки спирається на ті самі принципи, що й сторінка категорій: компоненти форми створено з урахуванням контролю стану через React, валідації полів та подальшого збереження даних у MongoDB. Однак, оскільки структура планети не передбачає вкладеності або зв’язків між об’єктами, маніпуляції з даними є дещо простішими: кожна планета зберігається як окремий документ з фіксованими полями, без додаткових вкладених масивів.

Таким чином, сторінка керування планетами виконує не лише утилітарну функцію додавання й редагування даних, але й закладає основу для тематичної персоналізації товарного контенту — через можливість пов’язувати товари з планетами, формувати спеціальні добірки або фільтрувати асортимент за космічними асоціаціями. У перспективі це

дозволяє гнучко масштабувати функціонал адміністративної панелі та розширювати контентну базу без потреби в глибоких архітектурних змінах.

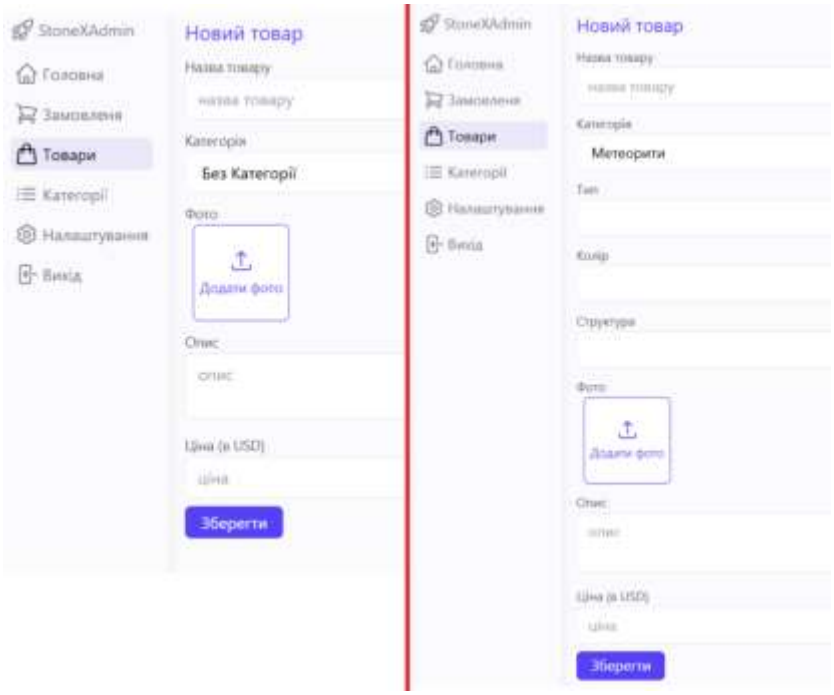
3.1.4. Сторінка керування товарами

Наступним логічним кроком створення повноцінної адміністративної панелі було реалізація сторінки для керування товарами. Її основна мета — забезпечити зручне середовище для створення, редагування та перегляду товарів, що в майбутньому будуть відображатися на клієнтській частині застосунку.

Сторінка, на зразок сторінки з категоріями, складається з двох підкомпонентів: форми для додавання або редагування конкретного товару та списку наявних товарів. Список реалізовано у вигляді таблиці, як складається з трьох стовпців: назви товару, знижки та блоку кнопок для редагування та видалення відповідного товару, та призначення знижки до товару (Рис. 3.4.). Дані отримуються з бази даних. Комунікація з базою здійснюється через окремий обробник, який обробляє методи GET і POST у залежності від дії.

(Рис. 3.4. Список товарів)

Форма редагування товару доступна за натисканням кнопки “Додати новий товар” та побудована з урахуванням таких полів: назва (title), опис (description), ціна (price), зображення (images), категорія (category) та властивості (properties). Вибір категорії реалізовано через випадаючий список, а доступні властивості змінюються динамічно залежно від обраної категорії. Завдяки цьому адміністратор може точно задавати параметри товару відповідно до його типу (Рис 3.5.).



(Рис 3.5. Форма додавання товару без обраної категорії та з обраною категорією відповідно.)

Функціонал завантаження зображень побудовано з використанням зовнішньої бібліотеки для обробки файлів. Користувач має змогу завантажити одне або кілька зображень, які миттєво відображаються у вигляді мініатюр. Зображення зберігаються у віддаленому сховищі, посилання на які вносяться до бази даних.

Валідація введених даних реалізована на клієнтському рівні: обов’язковими є назва, ціна та фото. Після збереження форми користувач перенаправляється назад до таблиці зі списком товарів.

Цей етап завершив формування повноцінного модуля для керування товарною номенклатурою.

3.1.5. Сторінка перегляду замовлень

Сторінка з переглядом замовлень є ключовим елементом адміністративної панелі, оскільки забезпечує доступ до даних, що безпосередньо відображають активність клієнтів і результативність функціонування магазину. Основна мета цієї сторінки — надати

адміністраторам можливість оперативно отримувати інформацію про зроблені покупки, структуру кожного замовлення.

На етапі реалізації першочергово було створено серверний запит до бази даних для отримання масиву об’єктів замовлень. Кожен об’єкт містить масив line_items, де кожен елемент включає quantity (кількість одиниць), price_data.unit_amount (вартість у копійках) і product_data.name (назву товару). Через відсутність попередньо обчисленого підсумкового поля total, загальна вартість кожного замовлення розраховується перемоноженням кількості на ціну.

На рівні інтерфейсу замовлення виводяться у вигляді таблиці, де кожен рядок відповідає одному замовленню. Виводиться базова інформація: дата, чи оплачено вже замовлення, отримувач, а також список товарів, які було замовлено (Рис 3.6.).

(Рис 3.6. Таблиця замовлень)

Загалом, реалізація цієї сторінки дозволила забезпечити адміністративному інтерфейсу функціональну завершеність у контексті керування повним циклом товарообігу — від створення товарів до обробки замовлень.

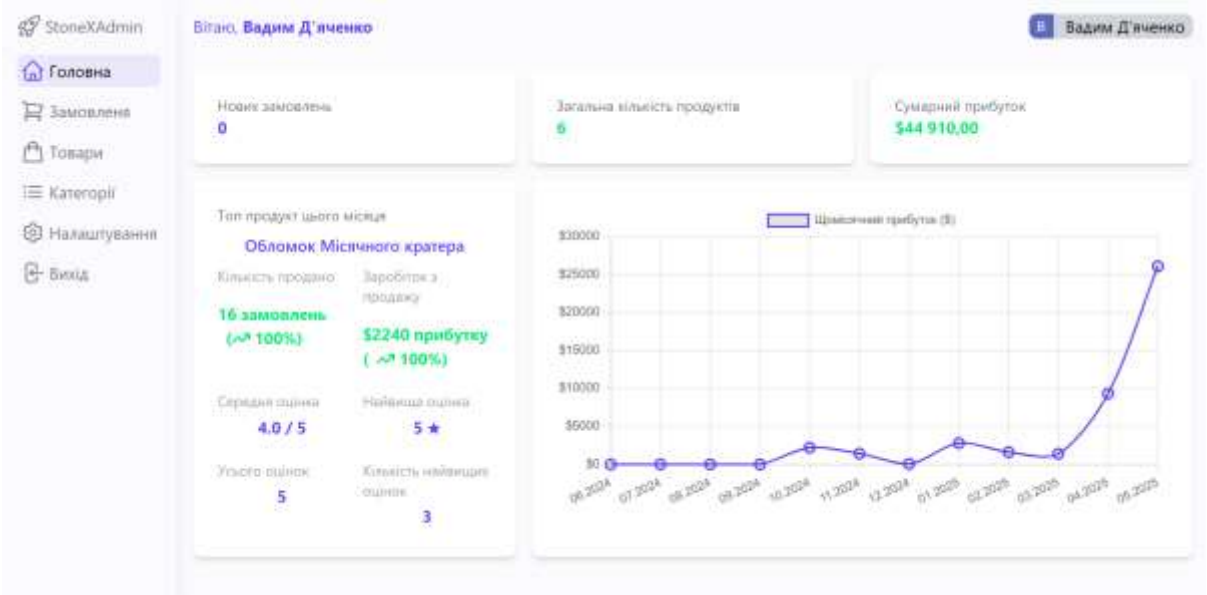
3.1.6. Сторінка аналітики та статистики

Головна сторінка адміністративної панелі виконує функцію інтерактивного аналітичного центру, де зосереджено ключові метрики діяльності магазину. Її основним призначенням є забезпечення адміністратора узагальненою інформацією про стан замовлень, прибутковість, динаміку продажів і продуктивність окремих товарів.

На етапі проєктування було прийнято рішення реалізувати окремий API-обробник /api/stats/profit, який обчислює та повертає агреговані показники: кількість нових замовлень, сумарний прибуток, а також приріст цих показників у порівнянні з попереднім місяцем. Для обчислення використовуються поля line_items, де враховується кількість проданих одиниць і ціна за одиницю. Після обчислення значення конвертуються у зручний формат для відображення.

Окрему увагу приділено модулю з топ-продуктом місяця. На основі замовлень визначається товар, який було придбано найбільшу кількість разів, з обрахуванням загального прибутку, середньої оцінки користувачів (на основі збережених відгуків), а також кількості максимальних оцінок. Отримані дані передаються на клієнт, де формуються візуальні блоки.

Інтерфейс реалізовано у вигляді адаптивної сітки з картками, де кожна картка відповідає за конкретний тип аналітики: загальний прибуток, кількість замовлень, приріст, і топ-продукт. Такий підхід дозволяє швидко зчитувати важливу інформацію навіть при поверхневому перегляді сторінки. Оновлення даних відбувається в реальному часі при кожному зверненні до сторінки (Рис. 3.7.).



(Рис 3.7. Головна сторінка)

Таким чином, реалізація цієї сторінки забезпечує зручну початкову точку для адміністрування — без необхідності переходу на інші вкладки користувач отримує найважливішу інформацію про діяльність застосунку в компактному та зручному вигляді. Також з завершенням реалізації цієї сторінки було створено основний функціонал адміністративної панелі, що дало змогу переходити до створення користувацької частини.

3.2. Клієнтська частина застосунку

З завершенням розробки адміністративної панелі застосунку відкрилась також можливість почати розробку клієнтської частини, так як попередні етапи, крім візуально зручних інструментів керування магазином, також формували “невидимий” каркас усього магазину та систему взаємодій товарів, категорій та замовлень. Та незважаючи на готову систему взаємодій, все ще стоїть задача інтегрувати її в користувацький досвід.

3.2.1. Постановка цілей

Реалізація клієнтської частини веб-застосунку передбачає створення логічно структурованого набору сторінок, кожна з яких виконує окрему

функцію в межах інтерфейсу для покупців. Метою є забезпечення зручної навігації, візуальної привабливості, функціональності та інтерактивності, необхідних для ефективного ознайомлення з асортиментом товарів, їхньої оцінки й подальшого придбання.

У межах користувацького інтерфейсу було визначено такі цільові сторінки:

1. Головна сторінка, яка відображає останні додані товари, а також спеціально обраний товар (топ або рекомендований) з короткою інформацією про нього;
2. Сторінка всіх товарів, що дозволяє користувачеві переглядати повний перелік доступної продукції, з базовими параметрами відображення;
3. Сторінка мапи Сонячної системи, яка б створювала ефект присутності у космічному просторі через 3D-карту, що демонструє розташування планет.
4. Сторінка окремого товару, на якій надається повна інформація: зображення, опис, ціна, список характеристик, а також функціонал коментування й оцінювання;
5. Кошик, який відображає обрані користувачем товари, дозволяє змінювати кількість, видаляти позиції та переходити до оформлення замовлення;
6. Сторінка облікового запису, що передбачає перегляд історії замовлень.

Таким чином, побудова клієнтської частини орієнтована на послідовну реалізацію низки сторінок, кожна з яких відповідає за окремий аспект взаємодії користувача з платформою магазину.

3.2.2. Головна сторінка

Головна сторінка є першою точкою взаємодії користувача з веб-застосунком і виконує функцію навігаційного та візуального ядра інтерфейсу. Її структура покликана забезпечити швидкий доступ до основної інформації

про магазин, ознайомлення з останніми товарами та акцентування уваги на продуктах, які потенційно можуть зацікавити користувача.

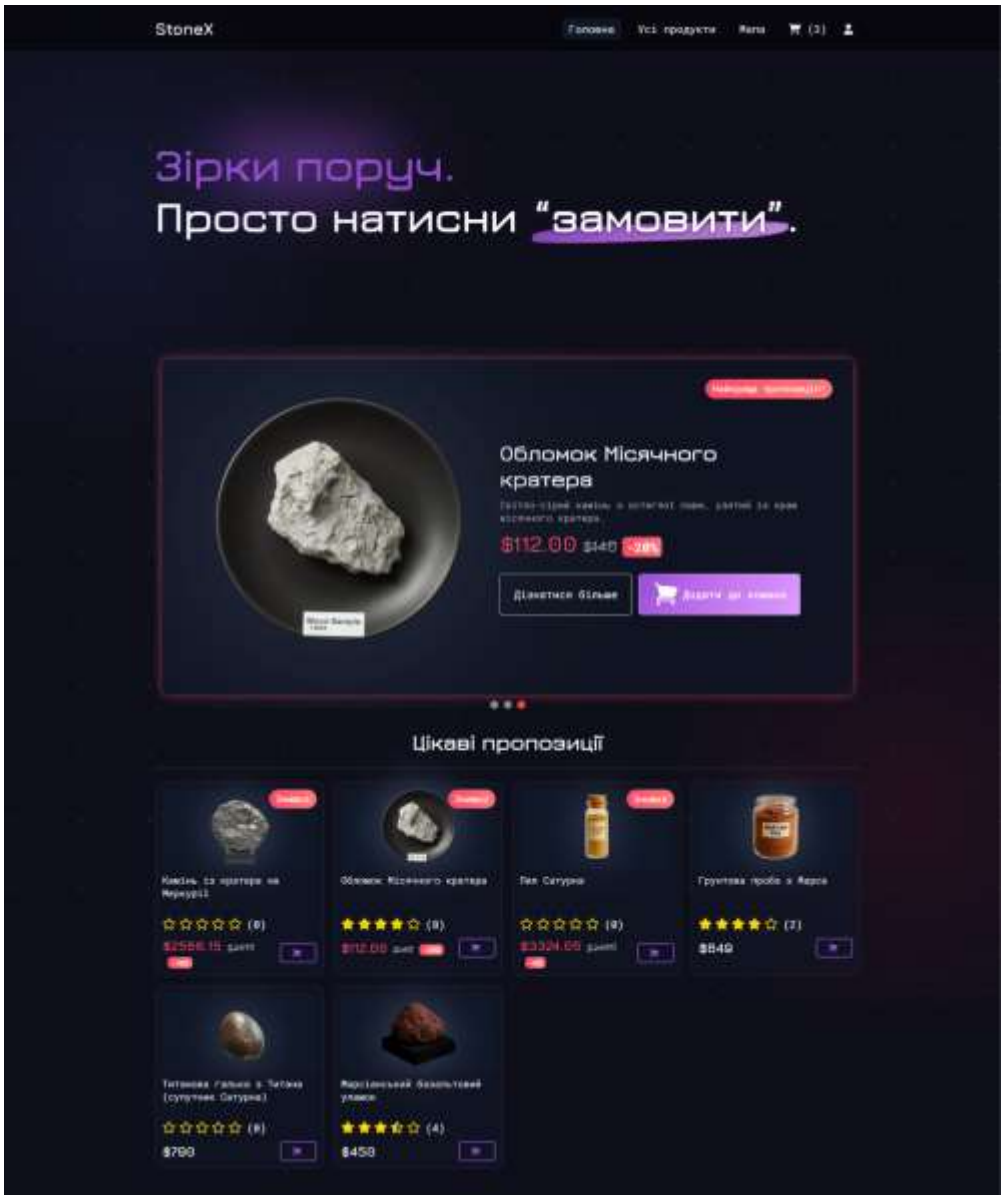
У межах реалізації було визначено три ключові складові головної сторінки:

1. Частина зі слоганом, покликана звернути на себе увагу користувача при першому вході на сторінку, а також мотивування користувача до можливих покупок. Як слоган було обрано фразу: “Зірки поруч. Просто натисни “замовити””. Ця фраза як найкраще розкриває концепцію застосунку для користувачів, показуючи що проект є свого роду “мостом” між користувачем та зорями.

2. Блок з найкращими пропозиціями, де виводиться окремі продукти, які мають найбільшу знижку на даний момент. Цей блок виконує в першу чергу промоційну роль, даючи користувачу розуміння того, який продукт можна придбати найвигідніше. Список цих продуктів формується за допомогою запиту на сервер через getServerSideProps, з окремим фільтром та сортуванням, де повертаються лише товари які мають знижку, та сортуються від найбільшої. Після отримання відповіді з серверу кожен елемент списку формується в окремий компонент “картки”, в якому перелічена уся важлива інформація про певний продукт, така як назва, опис, ціна (яка вже змінена з урахуванням знижки) та безпосередньо сама знижка. Блок має функцію “слайдера”, водночас на сторінці демонструється лише один продукт, але є можливість гортати картки з товарами, а також картки самі будуть змінюватися з плином часу.

3. Блок цікавих товарів, який в свою чергу має схожу роль із попереднім, але продукти в цьому блоці розташовані у вигляді сітки, тож одразу користувач може подивитися усі цікаві пропозиції. Також, на відміну від попереднього блоку, в цьому представленні і останнє додані товари, які теоретично можуть також зацікавити користувачів, особливо тих що активно слідкують за оновленнями магазину.

Загальний вигляд головної сторінки був розроблений з урахуванням адаптивності: компонування автоматично підлаштовується під розмір екрана, забезпечуючи комфортне використання на різних пристроях. Основний акцент зроблено на візуальній привабливості, мінімалістичному стилі та логічній ієрархії блоків (Рис. 3.8.).



(Рис. 3.8. Головна сторінка)

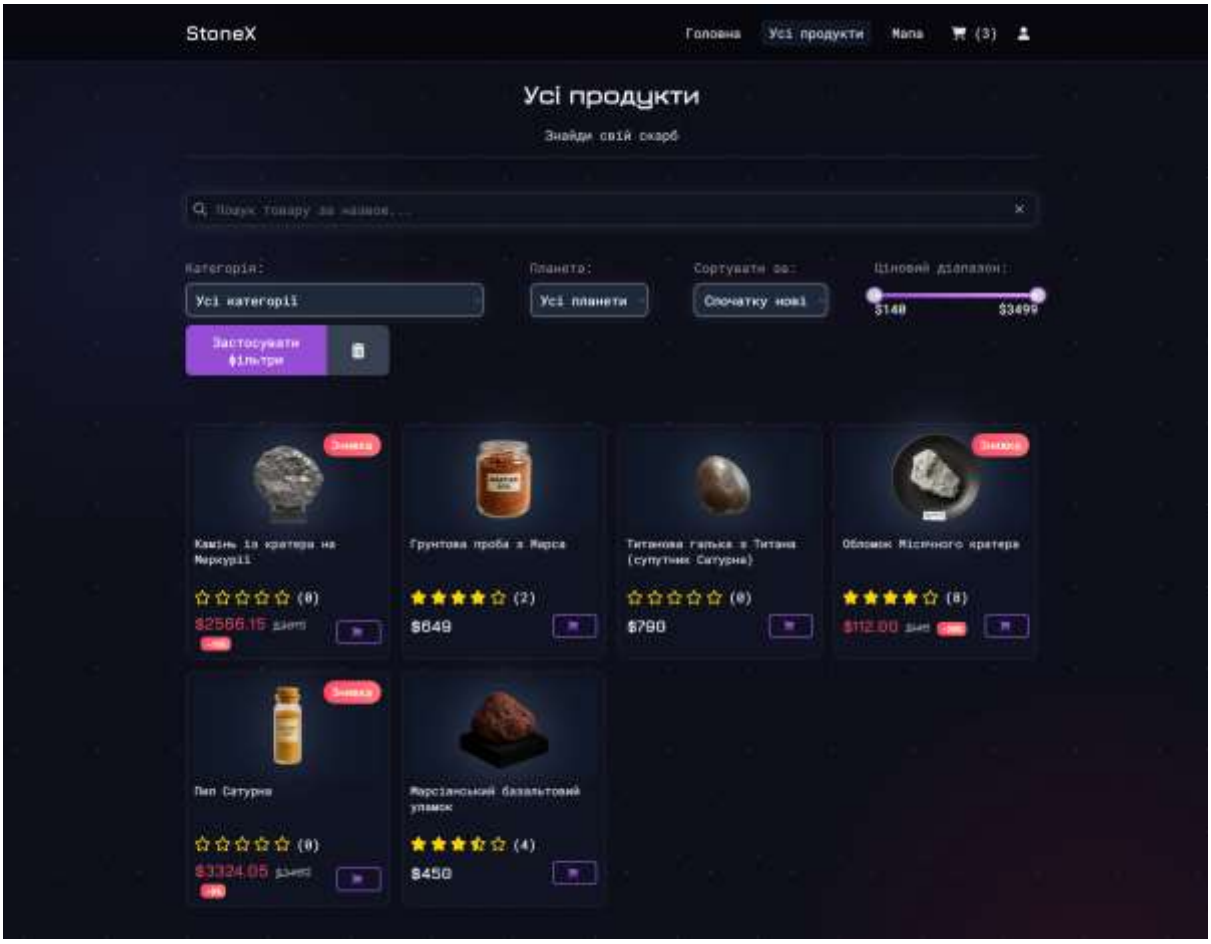
Таким чином, головна сторінка виконує роль презентаційного інструменту, який сприяє орієнтації користувача в магазині та стимулює подальшу взаємодію з іншими сторінками застосунку.

3.2.3. Сторінки з усіма товарами

Серед основних структурних елементів користувацької частини веб-застосунку важливе місце посідає сторінка, призначена для систематичного представлення наявного асортименту — йдеться, зокрема, про сторінку, яка відображає усі товари. Ця сторінка реалізує логіку попередньої навігації, забезпечуючи користувачеві широкий огляд продукції магазину до моменту переходу на детальнішу сторінку конкретного товару. Вона відіграє роль своєрідного посередницького шару між інтерфейсом загального призначення (такого як головна сторінка) та спеціалізованими модулями (наприклад, кошиком чи відгуками).

Сторінка, що виводить усі товари, є важливою складовою візуального ознайомлення з наповненням магазину. Вона реалізована як окрема маршрутна одиниця (/products) і містить перелік товарів, а також можливість сортувати і фільтрувати наявні товари за обраними критеріями. Товари виводяться у вигляді адаптивної сітки карток, кожна з яких містить зображення, назву та ціну. Відповідна верстка реалізована з використанням компонентів із власним стилем, що забезпечує як естетичну привабливість інтерфейсу, так і його зручність для взаємодії (Рис. 3.9.).

Особливої уваги заслуговує реалізований функціонал пошуку за назвою товару та фільтрація, що дозволяє користувачеві виокремлювати продукцію за ключовими словами, категоріями, походженням, а також сортувати товари за ціновим діапазоном. Усі види фільтрації інтегровано безпосередньо в інтерфейс сторінки та сполучено з логікою getServerSideProps, що забезпечує динамічну вибірку з бази даних у відповідь на зміну параметрів запиту. Таким чином, результат формується на сервері відповідно до актуального стану пошукового терміну. Для покращення користувацького досвіду реалізовано візуальний індикатор завантаження (спіннер), що з’являється у момент виконання запиту. Це дозволяє уникнути ефекту "порожнього екрану" та зберегти відчуття безперервної взаємодії.



(Рис. 3.9. Вигляд сторінки з переліком продуктів)

Пошуковий інтерфейс також доповнено іконкою збільшувального скла — універсальним символом пошуку, розташованим ліворуч у текстовому полі. Це не лише підсилює візуальну чіткість функції, але й підвищує інтуїтивність її використання, що особливо важливо в умовах швидкої взаємодії та високих очікувань сучасного користувача.

Кнопку підтвердження обраних фільтрів було поділено на дві частини: перша відповідає за безпосереднє застосування фільтрів, а друга — за їхнє видалення. Таке розташування є інтуїтивно зрозумілим і дозволяє заощадити місце на сторінці, що спрощує процес верстки.

Таким чином, сторінка відіграє одну з головних ролей у процесі навігації та фільтрації. Вона створює умови для швидкого доступу до релевантного сегменту товарів, мінімізують часові витрати на пошук і водночас

забезпечує більш глибоку структурованість контенту. З погляду користувацького досвіду, це сприяє ефективному й приємному ознайомленню з асортиментом магазину, підвищуючи як зручність використання інтерфейсу, так і загальний рівень задоволеності від взаємодії з веб-застосунком.

3.2.4. Сторінка з мапою Сонячної системи.

Сторінка з мапою Сонячної системи є природним доповненням до сторінки з усіма товарами, оскільки надає інтерактивний простір для візуалізації походження товарів та підсилює тематичний контекст магазину. Після переходу на цю сторінку користувач стикається із коротким завантаженням, під час якого на екрані відображається логотип магазину та індикатор завантаження (Рис. 3.10.). Як тільки ресурси підвантажуються, користувач бачить тривимірну мапу зоряної системи, реалізовану за допомогою бібліотеки Three.js, яка робить можливим створення 3Д-сцен у браузері (Рис.3.11.) [24].

(Рис. 3.10. Екран завантаження)

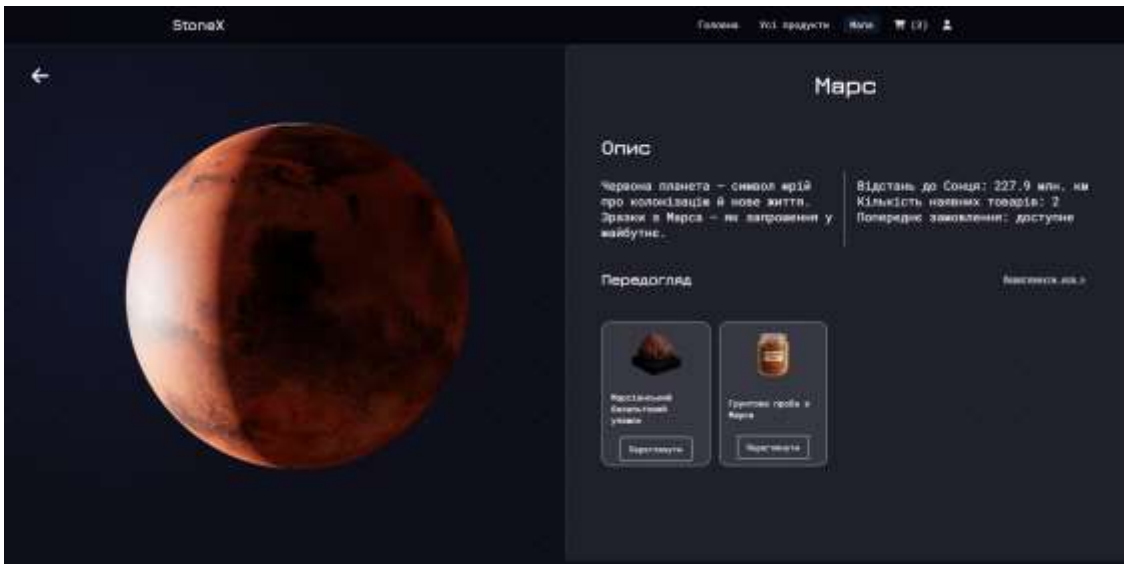
(Рис. 3.11. Мапа Сонячної системи)

У центрі віртуальної мапи розташоване Сонце — єдиний нерухомий об’єкт, навколо якого обертаються всі планети. Положення кожної планети визначається на основі даних із сервісу *api.le-systeme-solaire.net:* відстань від Сонця враховано відповідно до фактичних орбітальних радіусів, але для спрощення візуалізації планети, за допомогою окремої математичної функції, розміщено на ідеальних колах. Кожна планета має невеликий індикатор та підпис із назвою: індикатор підкреслює її на орбіті, а текст над планетою точно повторює рух, щоб користувач міг одразу ідентифікувати об’єкт навіть при русі камери. Також в правому верхньому куті сцени розташований перемикач режиму камери, яки дозволяє як дивитися на мапу з будь якого кута, так і зафіксувати її на місці.

При наведенні курсора на планету її індикатор та назва підсвічуються, що робить взаємодію інтуїтивною. Якщо користувач натискає на певну планету, камера автоматично плавно наближається і розташовує обрану планету у лівій частині екрана. Після затримки менше однієї секунди у правій частині екрана з’являється бічна панель із детальною інформацією про обрану

планету: назва, короткий опис та перелік доступних у магазині товарів, пов’язаних із цією планетою (Рис. 3.12.). У цьому блоці також присутня кнопка «Переглянути усе», яка перенаправляє користувача на сторінку /products і одночасно застосовує фільтрацію за обраною планетою, що забезпечує негайний доступ до відповідного асортименту.

Для оптимізації роботи з даними про планети при завантаженні мапи виконується запит до зовнішнього API, а позиції та орбітальні радіуси перетворюються у внутрішню систему координат, де масштабовані значення відповідають умовному візуальному розміру області перегляду. У візуалізації використовується методика «billboard» для підписів планет, що гарантує їхню завжди читабельну орієнтацію незалежно від кута огляду.



(Рис.3.12. Вигляд сторінки при виборі планети)

Інтерактивні елементи сторінки, зокрема камери та орбіти, налаштовані таким чином, щоб забезпечити максимальну плавність рухів. Виклик «OrbitControls» із бібліотеки @react-three/drei дозволяє користувачеві обертати картинку в довільні боки, доки користувач не вибере конкретну планету.

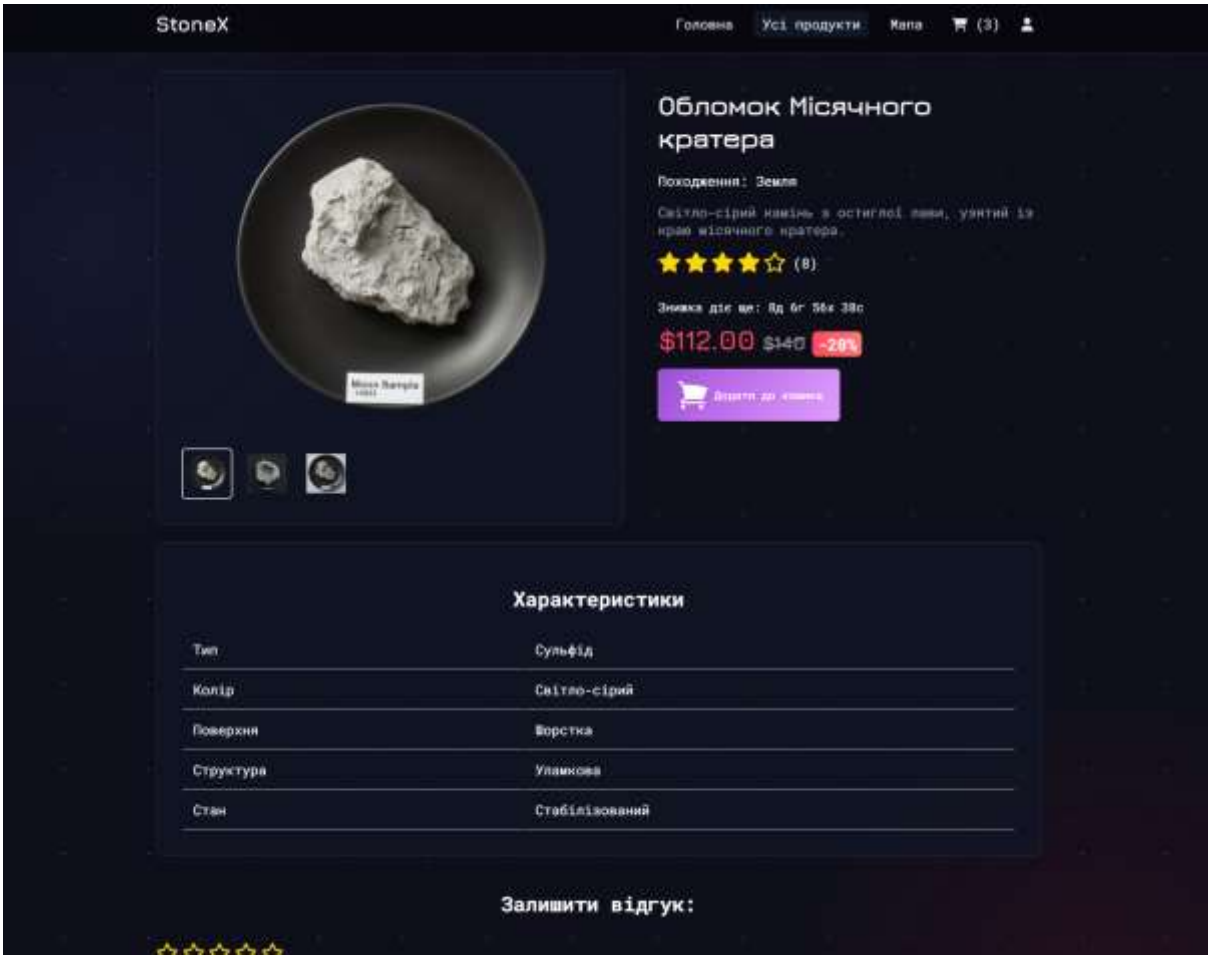
Таким чином, сторінка з мапою Сонячної системи служить не тільки привабливою візуальною частиною, але й функціональним інструментом, який безпосередньо доповнює сторінку з усіма товарами. Вона дозволяє

користувачу отримати глибше уявлення про походження кожного каменю чи метеорита, створює емоційне занурення у тематику магазину та забезпечує швидкий перехід до фільтрованого переліку товарів відповідно до обраної планети. Це підвищує зручність навігації, сприяє кращому розумінню асортименту і загалом покращує користувацький досвід роботи з інтернет-магазином.

3.2.5. Сторінка окремого товару

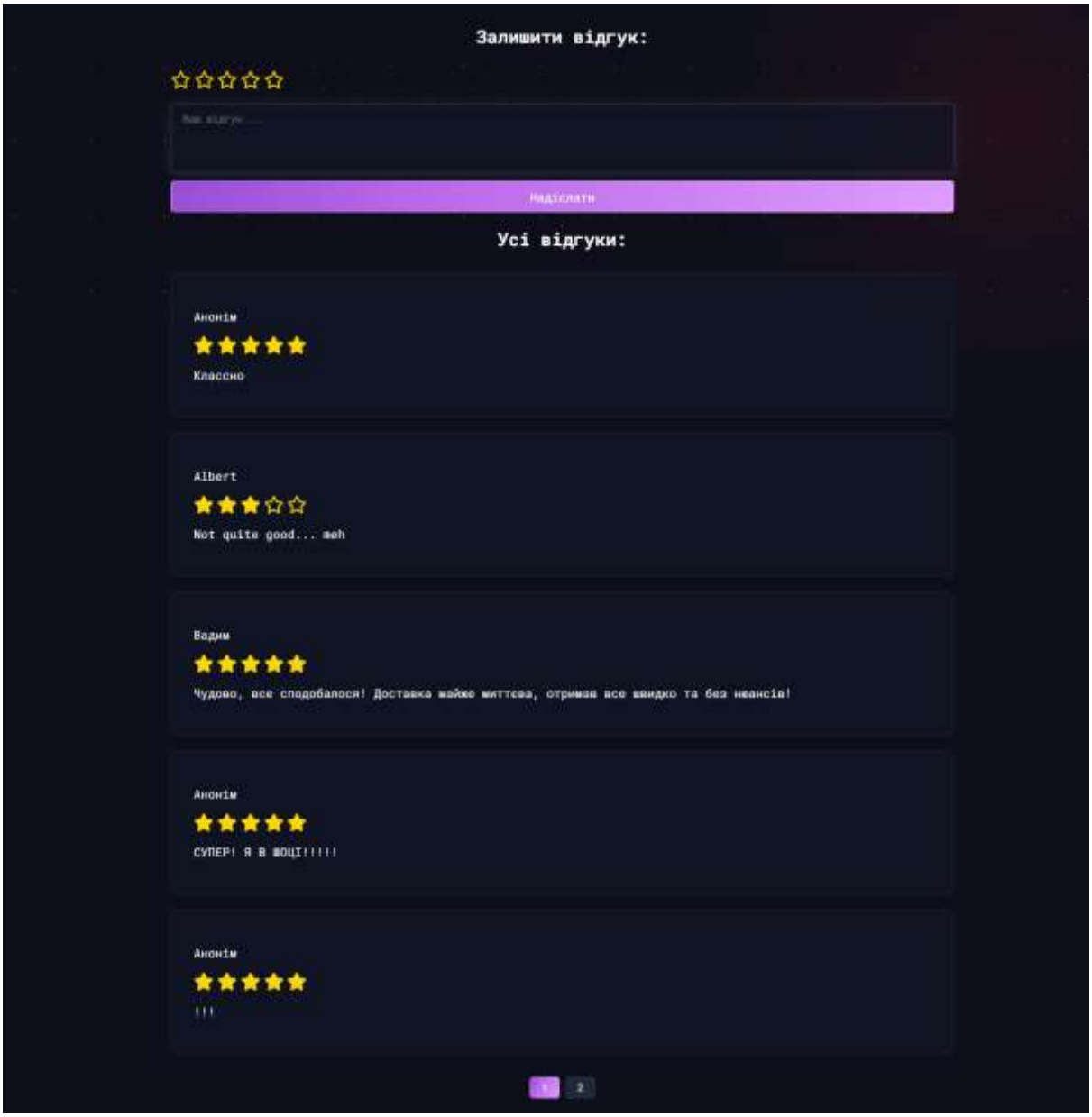
Однією з ключових складових веб-застосунку є сторінка окремого товару, яка виконує не лише інформативну функцію, а й виступає в ролі головної точки взаємодії користувача з конкретною одиницею продукції. Її значущість полягає в тому, що саме тут відбувається переосмислення товару не як частини загального асортименту, а як повноцінного інформаційного об'єкта зі своєю унікальною структурою, характеристиками та відгуками. У цьому сенсі сторінка відіграє роль інтерфейсного посередника між абстрактною пропозицією й особистим вибором, поєднуючи на одному екрані візуальні, описові, аналітичні та інтерактивні компоненти (Рис. 3.13.).

Розробка цієї сторінки передбачала створення багаторівневої логіки, яка забезпечує динамічне відображення даних та безперебійну інтеракцію. Центральним елементом, який забезпечує передачу актуальної інформації про товар, є функція серверного рендерингу, що реалізована за допомогою спеціального механізму, який дозволяє передати в компонент уже сформовані дані з бази. Завдяки цьому сторінка одразу містить усю необхідну інформацію: назву, опис, зображення, перелік технічних характеристик, ціну, а також — за наявності — знижку та відлік до її завершення. Цей підхід підвищує продуктивність застосунку, дозволяє уникнути затримок у відображенні контенту та забезпечує сумісність із пошуковими системами.



(Рис. 3.13. Загальний вигляд сторінки з окремим товаром)

Сторінка також містить інтегровану систему оцінювання, що виконує не лише функцію збору рецензій, а й формує узагальнений портрет товару з позиції користувача (Рис. 3.14.). Візуалізація середнього рейтингу, динамічне оновлення відгуків і підтримка асинхронного надсилання нових оцінок — усе це сприяє створенню відкритого комунікативного простору всередині застосунку, де кожен покупець може не тільки споживати інформацію, а й формувати її. Взаємодія з системою відгуків є прозорою й негайною: користувач має змогу вибрати кількість зірок, залишити текстовий коментар і миттєво побачити свій відгук на сторінці, що створює відчуття прямої присутності та діалогу. Якщо ж користувач авторизований, то його коментар буде підписаний його іменем.



(Рис 3.14. Вигляд секції коментарів)

Окремої уваги заслуговує адаптивна структура представлення характеристик товару. Перелік властивостей реалізовано як таблицю зі зручною семантичною структурою, що дозволяє швидко ознайомитися з технічними або унікальними параметрами об'єкта. Цей блок не є статичним — він генерується на основі динамічних даних, що надходять із бази, і адаптується відповідно до вмісту кожного товару. Таким чином, сторінка не лише презентує товар, а й підкреслює його індивідуальність, уникаючи шаблонності та надлишкової уніфікації.

Ще одним важливим елементом є модуль додавання товару до кошика, який виконує роль сполучної ланки між оглядовою частиною сторінки та процедурою оформлення замовлення. Натискання відповідної кнопки активує контекстну функцію додавання, яка оновлює глобальний стан кошика. Цей механізм уможливлює побудову інтуїтивно зрозумілої взаємодії, що не вимагає перезавантаження сторінки чи переходу до іншого розділу. Водночас застосунок фіксує дію користувача у внутрішньому контексті, зберігаючи її як частину сеансового досвіду.

Візуально-компонентна структура сторінки сформована таким чином, щоб уникнути як перевантаження, так і недомовленості. Усі логічні блоки — зображення, текстовий опис, ціна, функціонал додавання, рейтинг, відгуки, характеристики — мають власні візуальні межі й просторові маркери, що сприяє сприйняттю інформації й орієнтації в межах сторінки. Важливо, що стилізація компонентів ґрунтується на підході модульного оформлення, коли всі елементи мають автономні стилі, недоступні за межами свого контексту. Це забезпечує легкість у підтримці, масштабованість інтерфейсу та зменшує ризики конфліктів між стилями.

Отже, сторінка окремого товару у структурі веб-застосунку постає не просто як елемент відображення одиничної позиції, а як складне функціональне середовище, що інтегрує в собі кілька сценаріїв користувацької поведінки: ознайомлення, оцінювання, коментування, взаємодію з кошиком. У такий спосіб вона стає точкою персоналізованого контакту з системою, у якому зосереджено логіку взаємного обміну — між клієнтом, інтерфейсом і базою даних. Саме через призму цього багатошарового функціоналу вона відіграє роль ключового посередника в користувацькому шляху — від абстрактного зацікавлення до конкретного рішення про придбання.

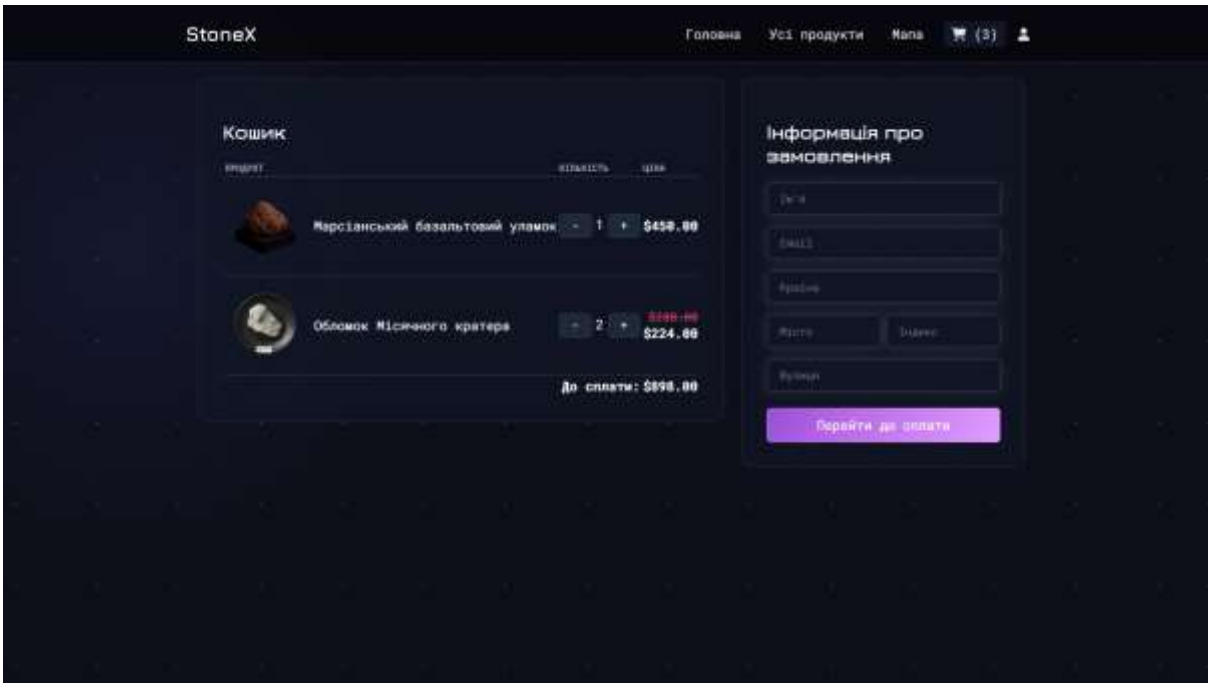
3.2.5. Кошик та оплата

Сторінка кошика у веб-застосунку виконує багатофункціональну роль, поєднуючи в собі кілька рівнів логіки: від візуального відображення вибраних товарів до ініціювання процесу покупки з інтеграцією зовнішньої платіжної системи. Вона не лише акумулює інформацію про споживацькі вподобання, а й перетворює цю інформацію на конкретну транзакцію, що має юридичну й економічну вагу. Саме тому в її структурі особливо важливо забезпечити стабільну взаємодію між фронтендом і серверною логікою, синхронізовану обробку стану та надійність передачі чутливих даних, таких як контактна інформація та адреса доставки.

На рівні архітектури реалізація кошика спирається на контекстне API React, що дозволяє централізовано зберігати й обробляти список ідентифікаторів обраних товарів упродовж усієї сесії. Це рішення усуває потребу у дублюванні логіки на різних сторінках застосунку та дає змогу змінювати стан кошика — додавати, прибирати або очищувати вміст — з будь-якої частини інтерфейсу. Також контекст зберігає методи, за допомогою яких ці зміни ініціюються. Сам перелік товарів, у свою чергу, оновлюється при кожній зміні масиву cartProducts, коли на сервер надсилається запит до API /api/cart. На основі отриманого масиву об’єктів формується таблиця з детальною інформацією про кожен продукт, що забезпечує користувачеві наочність і можливість контролю над кожним елементом покупки.

Особливістю реалізації є динамічне підрахування кількості кожного товару в кошику, а також врахування знижки кожного товару. Замість зберігання кількості як окремого атрибуту в об'єкті, система вираховує кількість повторень ідентифікатора продукту в масиві, що дозволяє спростити структуру даних та забезпечити гнучкість при додаванні й видаленні позицій (Рис. 3.15.). Кожна така операція відображається без необхідності оновлення сторінки, а користувач візуально спостерігає за змінами через адаптивну таблицю. Загальна сума розраховується на клієнтському боці у вигляді

підсумовування значень price, помножених на відповідну кількість. Таким чином, кожна зміна кількості товару негайно оновлює й фінансовий підсумок, що створює ефект "живої" взаємодії.

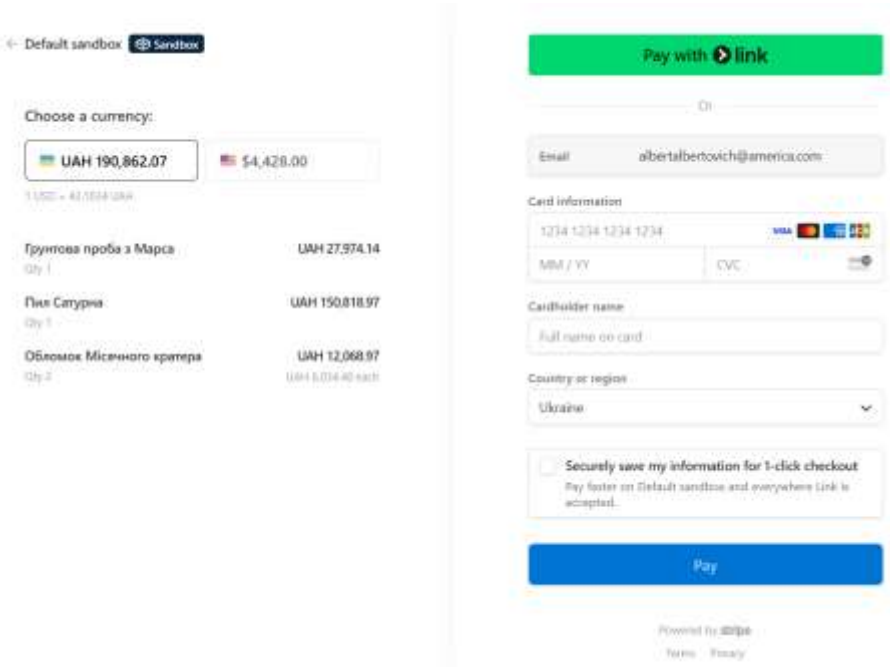


(Рис. 3.15. Кошик та форма контактної інформації)

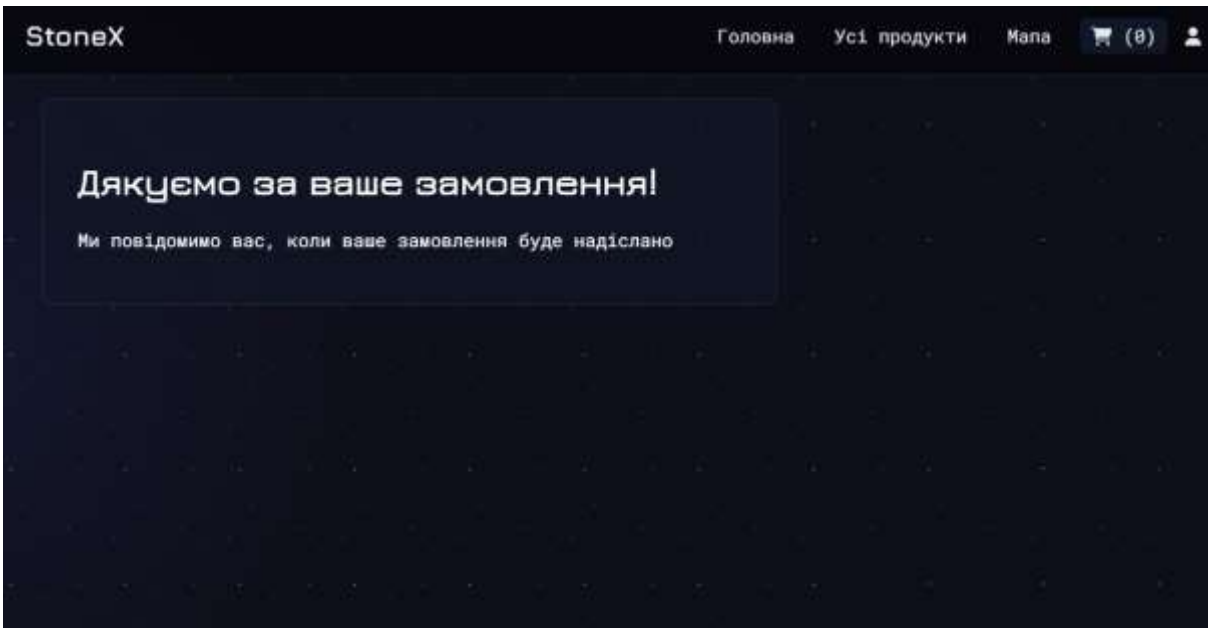
У правому стовпчику компонування інтерфейсу розміщується форма введення персональних даних. Цей компонент не є ізольованим: він прив’язаний до авторизаційного модуля через використання useSession, що дозволяє підставити ім’я та електронну адресу користувача без його додаткових дій. Така інтеграція зі службою аутентифікації підвищує зручність використання та скорочує час, необхідний для оформлення замовлення. Окрім того, форма містить окремі поля для країни, міста, поштового індексу, вулиці та номера будинку — усе, що потрібно для доставки. Поля мають двобічне зв’язування з локальним станом, що уможливлює реактивне оновлення та перевірку введених значень.

Завершальною ланкою у взаємодії є функція goToPayment, яка обробляє усі зібрані дані й надсилає їх POST-запитом до ендпоінту /api/checkout. У відповідь очікується URL, на який користувача буде перенаправлено для

здійснення оплати. Це демонструє інтеграцію із зовнішньою системою, зокрема платіжною платформою, яка реалізована через Stripe (Рис. 3.15.). Після успішного платежу механізм обробляє параметр success у рядку URL, завдяки чому ініціюється очищення кошика через метод clearCart, а користувачеві виводиться підтвердження зі словами вдячності та поясненням щодо подальших дій (Рис. 3.16.).



(Рис. 3.15. Вікно оплати провайдера Stripe)



(Рис. 3.16. Повідомлення про вдале створення замовлення)

Таким чином, логіка кошика формує цілу автономну підсистему в межах застосунку, в якій тісно переплетено локальний стан, контекстну архітектуру, асинхронну взаємодію з API та зовнішні сервіси. Усе це вимагає високої узгодженості між компонентами інтерфейсу, чіткого розділення відповідальностей та уважного підходу до обробки стану. Завдяки цим механізмам сторінка кошика виконує не лише роль функціонального перегляду вибраних товарів, а й слугує шлюзом між наміром користувача та фактичним комерційним актом, фіксованим у базі даних замовлень.

Висновок до розділу

У третьому розділі було детально проаналізовано ключові аспекти практичної реалізації веб-застосунку магазину космічних товарів, починаючи від загальної архітектури інтерфейсу користувача й логіки серверної взаємодії до структури бази даних, маршрутизації та реалізації адміністративної панелі. Особливу увагу приділено функціональній побудові як клієнтських, так і адміністративних сторінок, із фокусом на ефективну обробку запитів, збереження інформації про товари, категорії, властивості та замовлення.

Окрім класичних елементів інтернет-магазину, як-от каталог, кошик і система керування контентом, веб-застосунок оснащено розвиненою функціональністю для формування рейтингу товарів на основі відгуків користувачів. Такий підхід не лише забезпечує соціальне підтвердження якості продукції, а й сприяє зростанню довіри до бренду. Реалізовано зручний механізм адміністрування: додавання й редагування товарів, категорій, властивостей, перегляд замовлень — усе це здійснюється у захищеному середовищі з підтримкою автентифікації через NextAuth.

Важливою інноваційною складовою проєкту стала сторінка з інтерактивною мапою Сонячної системи, що використовує дані з відкритих API та технології тривимірної візуалізації (Three.js). Вона поєднує естетику з

функціональністю: дозволяє користувачеві дослідити планети, дізнатися коротку інформацію про них і перейти безпосередньо до фільтрованого списку товарів, пов’язаних з обраним небесним тілом. Такий підхід сприяє емоційному зануренню в тему космосу, посилює зв’язок між наративною частиною сайту та його комерційною складовою, а також демонструє можливості сучасного вебу у сфері візуалізації складних концепцій.

Загалом, проєкт вирізняється високим ступенем технічної готовності, продуманим візуальним оформленням і тематичною цілісністю. Унікальна концепція магазину — продаж рідкісних космічних артефактів — дозволяє позиціонувати застосунок як інноваційний і майже безконкурентний продукт на ринку. Враховуючи обрані технології, архітектуру й можливість масштабування, розроблений веб-застосунок має потенціал до комерційного запуску та подальшого розвитку як платформи нового покоління.

ВИСНОВОК

У рамках кваліфікаційної роботи було реалізовано веб-застосунок магазину космічних товарів, який поєднує в собі функціональність електронної комерції та сучасні підходи до дизайну й архітектури веб-програм. Система охоплює клієнтську частину з інтерфейсом для користувача, а також адміністративну панель для керування товарами, категоріями та замовленнями. Всі компоненти розроблені з урахуванням принципів масштабованості, модульності та ефективної взаємодії з базою даних MongoDB.

Проєктна реалізація дозволила значно поглибити практичні навички у сфері сучасної веб-розробки: було опановано роботу з фреймворком Next.js, бібліотеками Styled Components та Tailwind CSS, системою автентифікації NextAuth, а також з REST API для обміну даними між клієнтською частиною й сервером. Особливу увагу було приділено формуванню кошика, логіці обробки замовлень і реалізації інтерфейсів, які відповідають поточним вимогам до UX/UI.

Результати роботи демонструють не лише технічну реалізованість заявленого функціоналу, але й потенціал для подальшого розвитку системи. До перспективних напрямків модернізації належить впровадження фільтрації товарів, системи знижок, рекомендаційного механізму на основі історії покупок, а також розширення адміністративного інтерфейсу для аналітики продажів.

Загалом, виконаний проєкт засвідчив здатність застосовувати набуті знання в умовах реальної задачі, показав комплексне розуміння процесів створення, оптимізації та підтримки сучасних веб-застосунків. Отримані результати можуть бути основою для подальших досліджень у сфері електронної комерції, а сама система — використана як шаблон для розробки аналогічних застосунків у суміжних галузях.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Що таке веб-застосунок? Основи та приклади – URL: https://www.itechart.com/blog/web-application/ (дата звернення: 10.10.2024)

2. Next.js Documentation – URL: https://nextjs.org/docs (дата звернення: 12.10.2024)

3. Styled Components – офіційна документація – URL: https://styled-components.com/docs (дата звернення: 15.10.2024)

4. MongoDB – офіційний сайт – URL: https://www.mongodb.com (дата звернення: 20.10.2024)

5. NextAuth.js – документація – URL: https://next-auth.js.org (дата звернення: 25.10.2024)

6. Зберігання зображень у хмарі – що обрати розробнику – URL: https://blog.uploadcare.com/cloud-image-storage/ (дата звернення: 02.11.2024)

7. GitHub – URL: https://github.com (дата звернення: 10.11.2024)

8. Stack Overflow – URL: https://stackoverflow.com/questions (дата звернення: 15.11.2024)

9. Mozilla Developer Network – URL: https://developer.mozilla.org/uk/ (дата звернення: 18.11.2024)

10. Документація React – URL: https://reactjs.org (дата звернення: 20.11.2024)

11. OpenAI API Reference – URL: https://platform.openai.com/docs (дата звернення: 30.11.2024)

12. Stripe для розробників – URL: https://stripe.com/docs (дата звернення: 03.12.2024)

13. Використання Tailwind CSS – URL: https://tailwindcss.com/docs (дата звернення: 10.12.2024)

14. Що таке REST API: простою мовою – URL: https://habr.com/ru/articles/454258/ (дата звернення: 15.12.2024)

15. Тенденції електронної комерції у 2025 – URL: https://www.shopify.com/blog/ecommerce-trends (дата звернення: 20.01.2025)

16. Майбутнє інтерфейсів: футуризм у веб-дизайні – URL: https://uxdesign.cc/future-ui-trends (дата звернення: 01.02.2025)

17. Документація Cloudinary – URL: https://cloudinary.com/documentation (дата звернення: 10.02.2025)

18. Google Fonts – URL: https://fonts.google.com (дата звернення: 22.02.2025)

19. How to Build a Modern Web Storefront – URL: https://vercel.com/blog/building-modern-storefronts (дата звернення: 05.03.2025)

20. Космічні товари: що реально купити сьогодні? – URL: https://spacenews.com/space-merch-market-2025/ (дата звернення: 15.03.2025)

21. Сучасні архітектури веб-додатків – URL: https://web.dev/learn/architecture/ (дата звернення: 01.04.2025)

22. Розробка інтерфейсів з UX у фокусі – URL: https://uxplanet.org/web-interface-principles (дата звернення: 22.04.2025)

23. Node.js Documentation – URL: https://nodejs.org/en/docs (дата звернення: 02.05.2025)

24. Three.js Documentation - URL: https://threejs.org/docs (дата звернення: 02.05.2025)