МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «**РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ E-COMMERCE СИСТЕМИ ДЛЯ ПРОДАЖУ АВТОЗАПЧАСТИН З ВИКОРИСТАННЯМ БІБЛІОТЕКИ REACT**»

Виконав: студент 4 курсу, групи 6.1212-пі-2
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.О. Мельник
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Мельнику Андрію Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка клієнтської частини E-Commerce системи для продажу автозапчастин з використанням бібліотеки React

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 01.06.2026 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз предметної області та постановка задачі.
2. Проєктування та обґрунтування архітектури системи.
3. Програмна реалізація та тестування вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 09.12.2025 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	22.12.2025	
2.	Збір вихідних даних.	20.01.2026	
3.	Обробка методичних та теоретичних	12.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.03.2026	
5.	Розробка третього розділу.	18.05.2026	
6.	Оформлення та нормоконтроль	25.05.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	17.06.2026	

Студент _____ А.О. Мельник
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка клієнтської частини E-Commerce системи для продажу автозапчастин з використанням бібліотеки React»: 55 с., 16 рис., 17 табл., 21 джерело.

АВТОРИЗАЦІЯ, АДМІНІСТРУВАННЯ, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ЕЛЕКТРОННА КОМЕРЦІЯ, ЗАМОВЛЕННЯ, КАТАЛОГ, КОШИК, POSTGRESQL, REST API, TYPESCRIPT, NODE.JS.

Об’єкт дослідження – процес онлайн-продажу автомобільних запчастин та компонентів через системи електронної комерції.

Мета роботи: проєктування та програмна реалізація клієнт-серверного вебзастосунку інтернет-магазину автозапчастин із інтегрованою системою перевірки сумісності деталей та адміністративним керуванням складськими залишками.

Методи дослідження – методи системного аналізу предметної області, принципи об’єктно-орієнтованого проєктування та моделювання систем (UML), методи реляційного моделювання баз даних, а також методи сценарного функціонального тестування програмного забезпечення.

Кваліфікаційну роботу присвячено проєктуванню, програмній реалізації та верифікації Fullstack-вебзастосунку інтернет-магазину автозапчастин. У роботі визначено вимоги до системи, описано її клієнт-серверну архітектуру, розроблено логічну модель реляційної бази даних PostgreSQL та формалізовано алгоритми ключових модулів. Програмну реалізацію виконано на базі технологій React (Vite) для фронтенду та Express (Node.js) на TypeScript для серверної частини. Вебзастосунок забезпечує повний цикл взаємодії користувача з системою: авторизацію (JWT), багатокритеріальний пошук і фільтрацію деталей за сумісністю, роботу з кошиком та транзакційне оформлення замовлень, а також містить захищену адмін-панель для керування каталогом і залишками на складі.

SUMMARY

Bachelor’s qualifying paper “Development of the Client-Side of an E-Commerce System for Auto Parts Sales Using the React Library”: 55 pages, 16 figures, 17 tables, 21 references.

AUTHORIZATION, ADMINISTRATION, DATABASE, WEB APPLICATION, E-COMMERCE, ORDER, CATALOG, SHOPPING CART, POSTGRESQL, REST API, TYPESCRIPT, NODE.JS.

The object of research is the process of online sales of automotive parts and components through e-commerce systems.

The objective of the work is the design and software implementation of a client-server web application for an online auto parts store with an integrated parts compatibility verification system and administrative management of warehouse stock.

The research methods include system analysis of the subject area, principles of object-oriented design and systems modeling using UML, relational database modeling, and scenario-based functional software testing.

The bachelor’s thesis is devoted to the design, software implementation, and verification of a Fullstack web application for an online auto parts store. The thesis defines the system requirements, describes its client-server architecture, develops a logical model of the PostgreSQL relational database, and formalizes the algorithms of key modules. The software implementation is performed based on React (Vite) technologies for the frontend and Express (Node.js) on TypeScript for the backend. The web application provides a full cycle of user interaction with the system: authorization (JWT), multi-criteria search and filtration of parts by compatibility, shopping cart functionality, and transactional checkout, as well as contains a secured admin panel for catalog and warehouse inventory management.

ЗМІСТ

Завдання на кваліфікаційну роботу 2
Реферат 4
Summary 5
Вступ 8
1 Аналіз предметної області та постановка задачі 11
1.1 Особливості предметної області та аналіз ринку автозапчастин 11
1.2 Порівняльний аналіз існуючих програмних рішень 13
1.2.1 Універсальні маркетплейси 13
1.2.2 Спеціалізовані каталоги автозапчастин 13
1.2.3 Магазини локальних постачальників 13
1.3 Специфікація вимог до вебзастосунку та постановка задачі 14
1.4 Висновки до розділу 1 17
2 Проєктування та обґрунтування архітектури системи 18
2.1 Обґрунтування вибору технологічного стеку 18
2.2 Інформаційне забезпечення та проєктування бази даних 23
2.3 Структурна архітектура системи та взаємодія компонентів 29
2.4 Алгоритмічне забезпечення функціонування модулів 33
2.5 Висновки до розділу 2 36
3 Програмна реалізація та тестування вебзастосунку 37
3.1 Особливості реалізації клієнтської та серверної частини 37
3.1.1 Клієнтська частина 37
3.1.2 Серверна частина 38
3.2 Опис інтерфейсу користувача та сценаріїв роботи 38
3.3 Верифікація та функціональне тестування вебзастосунку 47
3.4 Оцінка якості та аналіз ризиків програмного забезпечення 48
3.4.1 Додатковий аналіз якості реалізації 48
3.4.2 Аналіз ризиків системи 49

3.4.3 Подальше вдосконалення застосунку 50
3.5 Висновки до розділу 3 50
Висновки 52
Перелік посилань 54

ВСТУП

Актуальність теми. Стрімкий розвиток цифрових технологій та глобальна трансформація ритейлу зумовили високу популярність і стійкий попит на розробку спеціалізованих систем електронної комерції. У сучасних умовах створення вебзастосунків для торгівлі є не просто інструментом розширення ринку збуту, а базовою передумовою забезпечення конкурентоспроможності бізнесу. На відміну від традиційних підходів минулих років, які спиралися на офлайн-взаємодію або громіздкі телефонні замовлення за паперовими каталогами, сучасні вебтехнології дозволяють автоматизувати повний цикл обслуговування клієнта в режимі реального часу.

Сфера електронної комерції автозапчастин має особливу інженерну та логістичну складність, оскільки покупець обирає не просто товар за назвою або ціною, а технічно сумісну деталь для конкретної модифікації автомобіля, покоління, року випуску та коду двигуна. Помилка підбору на етапі пошуку критично збільшує кількість повернень, спричиняє фінансові втрати на повторну доставку, створює надлишкове навантаження на менеджерів та знижує загальний рівень довіри до торговельного майданчика. Виникнення цього феномену зумовлює високі вимоги до архітектури, швидкодії та інформаційного забезпечення профільних вебсистем, що обумовлює актуальність даного проєкту.

Мета дослідження – проєктування та програмна реалізація масштабованого, безпечного вебзастосунку інтернет-магазину автозапчастин, який забезпечує інтелектуальний багатокритеріальний пошук товарів, фільтрацію за технічною сумісністю з транспортними засобами, транзакційне оформлення замовлень та оперативне адміністрування складських залишків.

Досягнення поставленої мети передбачає реалізацію наступних **завдань**:

– проаналізувати предметну область інтернет-торгівлі автозапчастинами та існуючі ринкові аналоги;

- дослідити та обґрунтувати вибір архітектурних рішень та сучасного технологічного стеку для реалізації системи;
- розробити функціональні та нефункціональні вимоги до вебзастосунку, а також специфікувати сценарії взаємодії користувачів;
- спроєктувати структуру реляційної бази даних, компоненти клієнт-серверної архітектури та ключові бізнес-алгоритми системи;
- програмно реалізувати серверний API та динамічний клієнтський інтерфейс вебзастосунку;
- протестувати основні користувацькі сценарії, виконати верифікацію бізнес-правил та провести аналіз технічних ризиків експлуатації продукту.

Об’єкт дослідження – процес онлайн-продажу автомобільних запчастин та компонентів через системи електронної комерції.

Предмет дослідження – архітектурні шаблони, моделі даних, методи та програмні засоби побудови клієнт-серверного вебзастосунку для каталогізації, вибору, транзакційного замовлення та адміністрування товарів автомобільного призначення.

Методи дослідження. Для досягнення поставлених результатів було використано методи системного аналізу предметної області, принципи об’єктно-орієнтованого проєктування та моделювання систем (UML), методи реляційного моделювання баз даних, а також методи сценарного функціонального тестування програмного забезпечення.

Інструментальним засобом реалізації проєкту обрано мову TypeScript та середовище Node.js з фреймворком Express для серверної частини, що дозволило побудувати строго типізоване та стійке до помилок API. Клієнтську частину побудовано на базі компонентної моделі React з інструментарієм Vite, що забезпечує високу швидкість рендерингу інтерфейсу. На відміну від NoSQL рішень, для надійного збереження сильно структурованих та взаємопов’язаних даних (користувачі, автомобілі, сумісність деталей,

замовлення) було обрано реляційну систему управління базами даних PostgreSQL із підтримкою індексування та ACID-транзакцій, що гарантує цілісність даних та унеможливлює продаж недоступних залишків товару. Контейнеризація та розгортання системи реалізовані засобами Docker Compose.

Структура та обсяг роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, загальних висновків та переліку посилань. Загальний обсяг роботи становить 55 сторінок. У переліку посилань 21 використане джерело.

Перший розділ присвячений системному аналізу предметної області, дослідженню специфіки підбору запчастин, порівнянню існуючих ринкових рішень та формуванню технічного завдання на розробку системи. У другому розділі наведено обґрунтування технологічного стеку, побудовано концептуальну та логічну моделі бази даних, проілюстровано сценарії використання за допомогою UML-діаграм та деталізовано архітектурні патерни взаємодії компонентів. В третьому розділі описано особливості програмної реалізації клієнтського та серверного шарів, наведено специфікацію UI-екранів, представлено результати сценарного тестування, а також виконано оцінку якості, аналіз технічних ризиків та визначено напрямки подальшого вдосконалення системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Особливості предметної області та аналіз ринку автозапчастин

Купівля автозапчастин відрізняється від звичайної покупки побутового товару тим, що покупець має врахувати технічну сумісність деталі з автомобілем. На практиці користувачі часто знають марку та модель авто, але не завжди розуміють, чи підходить конкретний артикул до покоління, року випуску, типу кузова або двигуна.

Для продавця проблема полягає у необхідності підтримувати структурований каталог, актуальні залишки, коректні ціни, історію замовлень і швидку обробку звернень. Якщо ці дані зберігаються в розрізнених таблицях або месенджерах, бізнес втрачає керованість, а клієнт не отримує прогнозованого сервісу. На рисунку 1.1 представлено дерево проблем користувачів під час вибору автозапчастин.

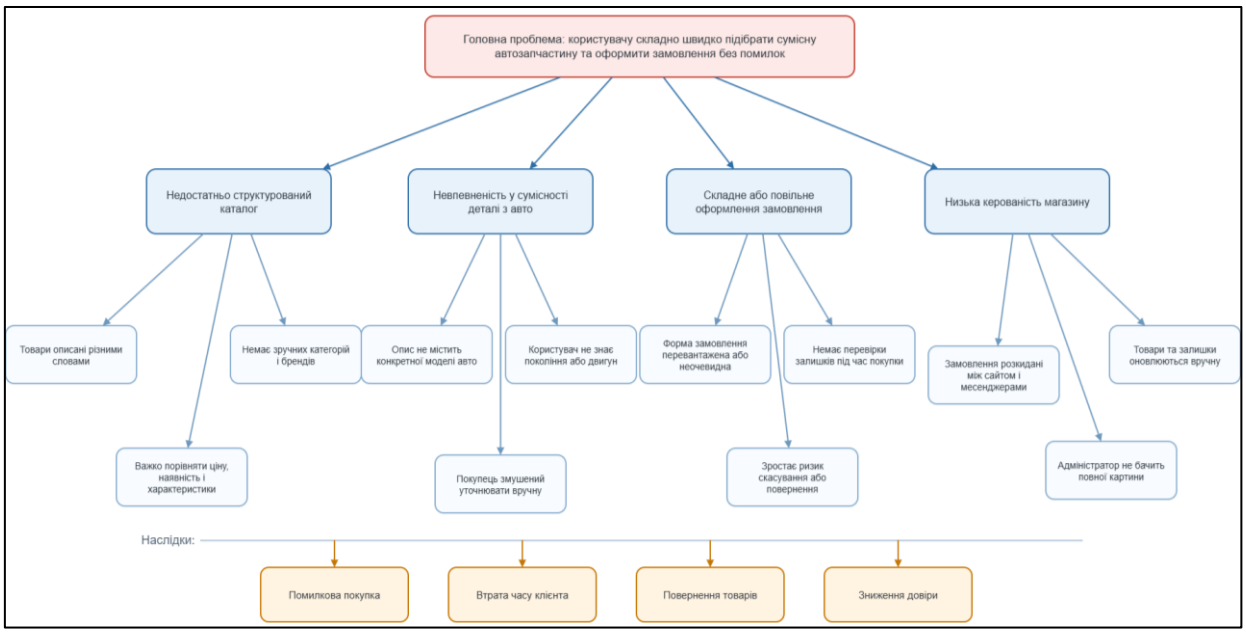


Рисунок 1.1

Розроблюване програмне забезпечення має зменшити кількість ручних уточнень, прискорити пошук товарів, забезпечити прозоре оформлення

замовлення і надати адміністратору інструмент для керування асортиментом та статусами замовлень.

Особливості товарної номенклатури. Автозапчастини мають складнішу структуру опису, ніж більшість товарів електронної комерції. Для користувача важливими є не лише ціна, бренд і фотографія, а й артикул, категорія, сумісність із транспортним засобом, технічні характеристики, залишок, умови доставки та наявність відгуків. Якщо будь-який із цих елементів відсутній, покупець змушений звертатися до менеджера або шукати підтвердження на сторонніх ресурсах.

Поведінка покупця під час вибору деталі. Типовий покупець починає пошук із різного рівня впевненості. Один користувач точно знає артикул, другий пам’ятає тільки бренд, третій шукає деталь за категорією, а четвертий орієнтується на автомобіль. Тому інтерфейс не повинен нав’язувати єдиний шлях. Він має підтримувати кілька сценаріїв, які ведуть до одного результату – вибору сумісного товару.

Проблема довіри до онлайн-магазину. У продажу автозапчастин довіра формується з кількох сигналів: зрозумілий опис, наявність зображення, відомий бренд, відгуки, прозора ціна, інформація про склад і нормальний процес оформлення замовлення. Якщо сайт виглядає випадковим набором товарів, користувач не впевнений, що після оплати отримає потрібну деталь.

Операційні потреби продавця. Для продавця інтернет-магазин є не лише вітриною, а й операційним інструментом. Він повинен допомагати контролювати асортимент, залишки, замовлення та статистику. Якщо система не підтримує адміністративні сценарії, бізнес швидко повертається до ручних таблиць і повідомлень у месенджерах.

Критерії якості програмного рішення. Якісне рішення для цієї предметної області повинно бути зрозумілим, швидким, технічно коректним і стійким до помилкових даних. Простий каталог без перевірки сумісності не вирішує основної проблеми, а надто складний професійний каталог може відштовхнути звичайного покупця. Тому важливо знайти баланс між технічною точністю та простотою UX.

1.2 Порівняльний аналіз існуючих програмних рішень

1.2.1 Універсальні маркетплейси

Універсальні маркетплейси надають широкий асортимент товарів і знайомий користувацький досвід: пошук, фільтри, відгуки, доставку та оплату. Їхня перевага полягає у масштабі, довірі до платформи та розвиненій логістиці.

Недоліком таких рішень у контексті автозапчастин є те, що модель сумісності зазвичай не є центральною частиною логіки. Покупець може бачити багато схожих позицій, але йому все одно доводиться перевіряти артикул, уточнювати відповідність або покладатися на опис продавця.

1.2.2 Спеціалізовані каталоги автозапчастин

Спеціалізовані каталоги краще відповідають предметній області, оскільки підтримують підбір за автомобілем, брендом, категорією, артикулом та аналогами. Такі сервіси часто мають складні довідники, історію цін і детальні технічні характеристики.

Водночас надмірна технічність інтерфейсу може бути складною для масового користувача. Якщо система орієнтована переважно на професійних продавців або СТО, звичайному клієнту важче швидко перейти від пошуку до покупки.

1.2.3 Магазини локальних постачальників

Локальні інтернет-магазини мають перевагу в гнучкості, швидкому контакті з клієнтом і можливості адаптувати асортимент під конкретний регіон. Вони можуть швидше змінювати пропозиції, працювати з власними

складами та підтримувати персональні консультації. Основними ризиками є фрагментарність реалізації, слабкий UX, відсутність історії замовлень, ручне оновлення залишків і недостатня автоматизація адміністративних процесів. Саме ці проблеми враховано під час розробки власного застосунку.

1.3 Специфікація вимог до вебзастосунку та постановка задачі

З метою чіткого формування вимог до вебзастосунку та коректної постановки задачі на проектування було проведено аналіз наявного програмного забезпечення у сфері електронної комерції автозапчастин. Порівняння здійснювалося за ключовими критеріями пошуку, обробки сумісності, транзакційності замовлень та архітектурної масштабованості. Зведена характеристика існуючих рішень та розроблюваного застосунку представлена в табл. 1.1.

Таблиця 1.1 – Порівняння існуючих програмних рішень

Критерій	**Маркетплейс**	**Спеціалізований каталог**	**Локальний магазин**	**Розроблюваний застосунок**
Пошук товарів	Широкий, але загальний	Технічний і деталізований	Залежить від реалізації	Пошук за назвою, SKU, категорією, брендом
Сумісність з авто	Обмежена або описова	Розвинена	Часто вручну	Окрема таблиця сумісності товарів і авто
Оформлення замовлення	Стандартизо-ване	Іноді складне	Може бути через форму / дзвінок	Кошик, доставка, оплата, транзакційне списання залишків
Адміністрування	На рівні продавця	Складне	Часто ручне	Статистика, товари, статуси замовлень
Масштабованість	Висока	Висока	Нерівномірна	Docker, API, PostgreSQL, модульна структура

Порівняння показує, що для невеликого або середнього магазину доцільно створити власний вебзастосунок, який не дублює надмірну складність великих каталогів, але зберігає ключові можливості: структурований каталог, фільтрацію, сумісність, оформлення замовлень і адміністративний контроль.

Розроблюване рішення займає проміжну позицію між універсальним маркетплейсом і професійним каталогом: воно має бути достатньо зрозумілим для кінцевого покупця, але технічно коректним для товарної групи автозапчастин.

Отже, попередній аналіз показує, що майбутній застосунок має поєднувати зручність звичайного інтернет-магазину з точністю спеціалізованого каталогу автозапчастин.

Формулювання **функціональних вимог**. Функціональні вимоги визначають базовий спектр можливостей, які повинен забезпечувати розроблюваний вебзастосунок інтернет-магазину автозапчастин. Вони безпосередньо описують регламент дій користувачів різних рівнів доступу (гостей, клієнтів, адміністраторів), а також детерміновану реакцію системи на ці дії.

Для проектованої системи пріоритетними архітектурними завданнями є інтерактивний перегляд каталогу, багатокритеріальний пошук і фільтрація товарних позицій за технічною сумісністю, інформативне відображення карток деталей, безпечне транзакційне оформлення замовлень, гнучка авторизація користувачів та оперативне адміністрування інтернет-магазину. Повний перелік функціональних вимог із визначенням їхнього інженерного пріоритету та базового методу реалізації представлено в таблиці 1.2.

Таблиця 1.2 – Функціональні вимоги до вебзастосунку

Код	**Вимога**	**Пріоритет**	**Реалізація**
FR-01	Система повинна відображати каталог активних товарів із пагінацією.	Високий	GET /api/products
FR-02	Система повинна підтримувати фільтрацію за пошуком, категорією, брендом, авто та ціною.	Високий	SQL WHERE + параметри запиту

Продовження таблиці 1.2

Код	Вимога	Пріоритет	Реалізація
FR-03	Користувач повинен мати змогу переглядати картку товару, характеристики, рейтинг і сумісність.	Високий	GET /api/products/:slug
FR-04	Авторизований користувач повинен мати змогу додавати відгук.	Середній	POST /api/products/:id/reviews
FR-05	Система повинна створювати замовлення з перевіркою залишків.	Високий	Транзакція в orders.service
FR-06	Адміністратор повинен керувати товарами й статусами замовлень.	Високий	Маршрути /api/admin
FR-07	Система повинна підтримувати реєстрацію, вхід, профіль і зміну пароля.	Високий	auth.service + JWT

Формулювання **нефункціональних вимог**. Нефункціональні вимоги описують якісні та системні характеристики вебзастосунку, висуваючи суворі критерії до середовища його функціонування. Вони не визначають конкретні ізольовані функції системи, проте встановлюють граничні параметри безпеки, продуктивності, інженерної надійності, зручності використання (UX/UI) та загальної супроводжуваності архітектури програмного забезпечення.

Враховуючи специфіку предметної області електронної комерції автозапчастин, особливу увагу приділено транзакційній цілісності даних при паралельних запитах та захисту персональних даних користувачів. Зведену специфікацію нефункциональних вимог до програмного продукту з інженерними поясненнями наведено в таблиці 1.3.

Таблиця 1.3 – Нефункціональні вимоги до вебзастосунку

Код	Вимога	Пояснення
NFR-01	Безпека	Паролі зберігаються у вигляді bcrypt-хешів, доступ до адмінки обмежений роллю.
NFR-02	Надійність	Оформлення замовлення виконується в транзакції з блокуванням товарів FOR UPDATE.
NFR-03	Продуктивність	Каталог має пагінацію, індекси за категорією, брендом і пошуком.
NFR-04	Масштабованість	Система розділена на frontend, API та PostgreSQL і може запускатися в Docker.

Продовження таблиці 1.3

Код	**Вимога**	**Пояснення**
NFR-05	Зручність	Інтерфейс орієнтований на швидкий пошук, зрозумілі фільтри і короткий шлях до замовлення.
NFR-06	Супроводжуваність	Backend поділено на маршрути, сервіси, репозиторії, middleware і валідацію.

1.4 Висновки до розділу 1

У першому розділі визначено основну проблему предметної області: покупець автозапчастин повинен швидко знайти не просто дешевий товар, а технічно сумісну деталь для конкретного автомобіля. Аналіз універсальних маркетплейсів, спеціалізованих каталогів і локальних магазинів показав, що кожен підхід має сильні сторони, але жоден повністю не закриває потреби невеликого магазину, який хоче мати власну керовану систему.

Детальний аналіз особливостей товарної номенклатури та операційних потреб продавця дозволив чітко окреслити практичну мету розробки застосунку. Система повинна зменшити кількість ручних уточнень, упорядкувати каталог, підтримати фільтрацію за реальними ознаками товарів, показувати сумісність з автомобілями, контролювати залишки під час замовлення і дати адміністратору інструмент для керування товарами та статусами.

Отже, подальша розробка власного вебзастосунку є обґрунтованою. Він має поєднати простоту інтернет-магазину для покупця з достатньою технічною структурою для предметної області автозапчастин.

2 ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

Вибір технологій для розроблення інтернет-магазину автозапчастин здійснювався з урахуванням реальних задач системи: відображення каталогу, фільтрації товарів, роботи з кошиком, оформлення замовлень, авторизації, адміністрування, збереження структурованих даних і подальшого розгортання. Тому оцінювалася не лише популярність технологій, а й їхня відповідність конкретним сценаріям застосунку.

2.1 Обґрунтування вибору технологічного стеку

React обрано як основну бібліотеку для побудови клієнтської частини, оскільки інтерфейс магазину природно складається з повторюваних компонентів: карток товарів, фільтрів, кнопок, форм, таблиць, блоків статусів і модальних повідомлень. Компонентна модель дозволяє створювати ці елементи один раз і повторно використовувати їх на різних сторінках [1]. Для розроблюваного застосунку це особливо важливо в каталозі. Картка товару може відображатися на головній сторінці, у списку результатів пошуку, у кошику або в адміністративній частині. Якщо логіка картки винесена в окремий компонент, зміна її вигляду не потребує редагування багатьох різних місць.

React також зручно розділяє візуальний шар і роботу з даними. Компонент не повинен знати, як саме backend формує SQL-запит; він отримує готовий об’єкт товару й показує назву, бренд, ціну, рейтинг, залишок і зображення. Це робить frontend легшим для підтримки [2].

Vite використано як інструмент розробки та збирання frontend-частини. Його головна перевага – швидкий запуск dev-середовища і швидке оновлення

сторінки під час змін у коді. Під час розробки каталогу, фільтрів, кошика і checkout це суттєво пришвидшує перевірку інтерфейсу [3].

У production-режимі Vite створює статичну збірку, яку можна віддавати через Nginx. Це добре узгоджується з архітектурою проєкту: frontend після збирання є набором статичних файлів, а backend залишається окремим API-сервісом [4].

Порівняно з Webpack, Vite потребує менше ручного налаштування для типового React-застосунку. Для дипломного проєкту це важливо, оскільки основна увага має бути зосереджена на логіці магазину, а не на складній конфігурації інструментів.

CSS та адаптивна верстка. CSS використовується для побудови візуальної системи застосунку: сітки каталогу, станів кнопок, форм checkout, таблиць адміністратора, адаптивності та читабельності. Для магазину автозапчастин дизайн повинен бути не декоративним, а практичним: користувач має швидко бачити товар, ціну, бренд і доступність.

Адаптивна верстка є обов’язковою, оскільки покупець може шукати деталь зі смартфона, планшета або ноутбука. На мобільному екрані фільтри, кнопки й картка товару повинні залишатися зручними, а текст не має перекривати інші елементи.

Порівняння технологій для реалізації клієнтської частини зведено в таблицю 2.1.

Таблиця 2.1 – Порівняння клієнтських технологій

Технологія	**Переваги**	**Недоліки**	**Оцінка для проєкту**
React	Компонентність, велика екосистема, зручність для SPA та повторного використання UI	Потребує дисципліни в структурі компонентів	Висока
Vue	Простий старт, зрозумілий синтаксис, добра придатність для невеликих застосунків	Менше відповідає вже обраному стеку проєкту	Середня

Продовження таблиці 2.1

Технологія	Переваги	Недоліки	Оцінка для проєкту
Angular	Повний фреймворк, сувора структура, готові архітектурні правила	Надмірний для прототипу магазину	Середня
Vanilla JS	Мінімум залежностей	Складніше підтримувати стан кошика, фільтрів і форм	Низька

Node.js обрано для серверної частини через зручність реалізації HTTP API та ефективну роботу з операціями введення-виведення. Інтернет-магазин часто звертається до бази даних: отримує каталог, перевіряє користувача, створює замовлення, читає довідники, оновлює статуси. Для таких задач Node.js є практичним вибором [5, 6].

Express використано як фреймворк маршрутизації. Він забезпечує просту модель REST API, підтримку middleware, обробку JSON, інтеграцію з CORS, Helmet і логуванням. У проєкті маршрути поділено на `auth`, `catalog`, `orders` і `admin`, що робить серверну частину зрозумілою [7, 8].

NestJS міг би бути альтернативою, але для поточного масштабу він додає більше службової структури. Fastify має переваги у продуктивності, проте Express достатній для дипломного прототипу і простіший для пояснення. Тому вибір Express є збалансованим.

TypeScript підвищує надійність backend-коду завдяки статичній типізації. У системі є багато структурованих об’єктів: користувач, товар, фільтри каталогу, замовлення, позиція замовлення, JWT-користувач, payload checkout. Типи допомагають контролювати ці структури ще до запуску програми.

У сервісі замовлень типізація особливо важлива, тому що там обробляються товари, кількість, користувач, контактні дані, доставка, оплата й підсумкова сума. Помилка в структурі таких даних може призвести до

некоректного замовлення, тому TypeScript є не просто зручністю, а засобом зменшення ризику.

JavaScript дозволив би реалізувати таку саму функціональність, але з меншим контролем. Для дипломної роботи TypeScript краще демонструє інженерний підхід і робить код зрозумілішим для подальшої підтримки.

PostgreSQL обрано як основну базу даних, оскільки предметна область має чіткі реляційні зв'язки. Товар належить до категорії та бренду, замовлення містить багато позицій, відгук належить товару, а сумісність товарів і автомобілів є зв'язком багато-до-багатьох [9, 10].

Для магазину критично важливі транзакції. Під час оформлення замовлення система повинна створити запис у `orders`, додати позиції в `order_items` і зменшити залишки товарів. Якщо будь-який крок завершується помилкою, усі зміни мають бути скасовані. PostgreSQL добре підтримує такі сценарії. Додатковою перевагою PostgreSQL є JSONB. Характеристики автозапчастин можуть відрізнятися залежно від категорії: для фільтра важливі одні параметри, для гальмівного диска – інші, для амортизатора – ще інші. JSONB дозволяє зберігати гнучкі специфікації без руйнування основної схеми [11].

Zod, JWT, bcrypt та middleware безпеки. Zod використано для валідації вхідних даних. Frontend може перевіряти форму, але backend не повинен довіряти клієнту повністю. Саме сервер має перевірити email, пароль, кількість товару, статус замовлення, ціну та обов'язкові поля доставки [14, 15].

Bcrypt використано для хешування паролів. Паролі не зберігаються у відкритому вигляді, що є базовою вимогою безпеки. JWT використано для автентифікації: після входу користувач отримує токен і передає його під час запитів до профілю, історії замовлень або адміністративних маршрутів [16].

Middleware `requireAuth` і `requireAdmin` реалізують контроль доступу. Звичайний користувач може переглядати каталог і створювати замовлення, але не може змінювати товари або статуси замовлень. Це розділення ролей є необхідним для будь-якого магазину з адміністративною частиною.

Docker Compose використано для повторюваного запуску системи. Проєкт складається з PostgreSQL, API, frontend і gateway. Якщо налаштовувати ці частини вручну, на різних комп’ютерах легко отримати різні середовища. Контейнеризація робить запуск передбачуваним [12, 13].

Для дипломного проєкту це також показує готовність до практичного розгортання. Застосунок можна перенести на сервер, задати змінні середовища і запустити сервіси в однаковій структурі. Це важливо, тому що програмний продукт повинен бути не лише написаний, а й придатний до використання.

Порівняння технологій для реалізації серверної частини наведено у таблиці 2.2. Підсумкове обґрунтування вибору технологій наведено у таблиці 2.3.

Таблиця 2.2 – Порівняння серверних технологій

Технологія	**Призначення**	**Аргументи вибору**	**Ризики**
Express	REST API	Проста маршрутизація, middleware, швидка розробка	Потрібна архітектурна дисципліна
TypeScript	Мова реалізації	Типи для DTO, сервісів, репозиторіїв	Потрібне налаштування компіляції
PostgreSQL	База даних	Транзакції, зовнішні ключі, індекси, JSONB	Потрібне коректне проєктування схеми
Zod	Валідація	Явні правила перевірки вхідних даних	Схеми треба підтримувати разом із вимогами
Docker	Розгортання	Повторюване середовище для API, БД і frontend	Потрібні коректні змінні середовища

Таблиця 2.3 – Обраний стек технологій

Компонент	**Обраний засіб**	**Причина вибору**	**Чому не основна альтернатива**
Інтерфейс	React	Компонентність, повторне використання UI, зручність для SPA	Angular надмірний, Vanilla JS складніший для підтримки
Збірка	Vite	Швидкий dev-режим і проста production-збірка	Webpack потребує більше конфігурації
API	Express	Простий REST-шар, middleware, достатня гнучкість	NestJS важчий для масштабу прототипу

Продовження таблиці 2.3

Компонент	Обраний засіб	Причина вибору	Чому не основна альтернатива
Мова	TypeScript	Контроль DTO, сервісів, репозиторіїв і помилок	JavaScript має слабший контроль структури
База даних	PostgreSQL	Зв’язки, транзакції, індекси, JSONB	NoSQL менш придатний для цілісності замовлень
Безпека	bcrypt, JWT, Zod	Хешування паролів, токени, валідація API	Ручні перевірки складніше підтримувати
Розгортання	Docker Compose	Повторюваний запуск усіх сервісів	Ручне налаштування важче відтворити

Таким чином, для frontend обрано React, Vite і CSS, оскільки вони забезпечують компонентність, швидку розробку, адаптивність і зручну production-збірку.

Для backend обрано Node.js, Express, TypeScript і PostgreSQL. Цей стек відповідає вимогам до REST API, типізації, транзакційності, структурованих зв’язків і гнучких характеристик товарів. Додатково використано Zod, bcrypt, JWT і middleware для валідації, автентифікації та контролю доступу.

Docker Compose забезпечує повторюваний запуск і підготовку системи до розгортання. Отже, обраний стек є логічним і достатнім для вебзастосунку інтернет-магазину автозапчастин, оскільки покриває ключові задачі: каталог, фільтри, замовлення, користувачів, адміністративні функції та базу даних.

2.2 Взаємодії користувачів із системою та проєктування бази даних

На основі проведеного аналізу визначено коло цільових користувачів вебзастосунку, до якого належать покупці автозапчастин (гості та зареєстровані клієнти) та адміністратори магазину.

Для покупця критично важливими є швидкість пошуку, наявність фільтрів, зрозуміла картка товару, можливість оформлення замовлення без зайвих кроків і доступ до історії покупок. Для адміністратора ключовими

визначено керування асортиментом, деактивація товарів, перегляд замовлень, зміна статусів і швидке отримання статистики. На рівні системи в цілому пріоритетними є коректність транзакцій, валідація даних, захист адміністративних маршрутів і контроль залишків.

Для формалізації взаємодії користувачів із системою розроблено сценарії використання. Зведений перелік базових прецедентів наведено в таблицю 2.4.

Таблиця 2.4 – Основні сценарії використання

ID	**Назва**	**Актор**	**Короткий опис**	**Результат**
UC-1	Пошук товару	Покупець	Користувач вводить запит або обирає фільтри за категорією, брендом, ціною чи автомобілем.	Список релевантних товарів
UC-2	Оформлення замовлення	Покупець	Користувач додає товари в кошик, вводить контактні дані, доставку й оплату.	Створене замовлення
UC-3	Авторизація	Клієнт	Користувач реєструється або входить у систему, отримує JWT і доступ до профілю.	Активна сесія
UC-4	Адміністрування	Адміністратор	Адміністратор переглядає статистику, товари й замовлення, змінює статуси.	Актуальні дані магазину

Для детального аналізу внутрішньої логіки функціонування системи, визначення чіткої послідовності кроків взаємодії акторів із програмним забезпеченням, а також опису реакції серверної частини на користувацькі події розроблено розгорнуті сценарії прецедентів. Покроковий алгоритм виконання базових операцій, початкові умови, очікувані результати та альтернативні логічні маршрути обробки помилок для кожного виділеного сценарію (UC-1–UC-4) представлено у відповідних специфікаціях (див. табл. 2.5 – 2.8).

Таблиця 2.5 – Специфікація UC-1: пошук товару

Ідентифікатор	**UC-1**
Актори	Покупець, гість або авторизований користувач.
Початкові умови	Користувач відкрив головну сторінку або каталог. Система має доступ до довідників категорій, брендів, автомобілів і активних товарів.
Очікуваний результат	Користувач отримує список товарів, які відповідають запиту або вибраним фільтрам, і може перейти до картки конкретного товару.
Порядок дій	1. Користувач відкриває каталог. 2. Вводить пошуковий запит або обирає категорію. 3. За потреби додає бренд, автомобіль, мінімальну або максимальну ціну. 4. Система передає параметри на API `/api/products`. 5. Backend формує SQL-запит з урахуванням фільтрів. 6. Frontend відображає список товарів, рейтинг, ціну, наявність і пагінацію.
Альтернативні шляхи	Якщо товарів не знайдено, система показує порожній стан і пропонує змінити фільтри. Якщо API недоступний, відображається повідомлення про помилку завантаження.

Таблиця 2.6 – Специфікація UC-2: оформлення замовлення

Ідентифікатор	**UC-2**
Актори	Покупець, гість або авторизований користувач.
Початкові умови	У кошику є хоча б один товар. Товари активні, а їх залишок більший або дорівнює вибраній кількості.
Очікуваний результат	Система створює замовлення, додає позиції замовлення, зменшує залишки товарів і повертає підтвердження .
Порядок дій	1. Користувач додає товар у кошик. 2. Переходить до оформлення. 3. Вводить ім’я, email, телефон, місто, адресу. 4. Обирає спосіб доставки та оплати. 5. Frontend надсилає checkout-запит. 6. Backend валідовує дані через Zod. 7.Сервіс замовлень у транзакції перевіряє товари `FOR UPDATE`. 8. Система створює `orders`, `order_items` і списує stock.
Альтернативні шляхи	Якщо товару недостатньо на складі, система повертає помилку 409 і не створює замовлення.

Таблиця 2.7 – Специфікація UC-3: авторизація користувача

Ідентифікатор	**UC-3**
Актори	Клієнт, адміністратор.
Початкові умови	Користувач має обліковий запис або відкрив форму реєстрації. Backend доступний, база даних працює.

Продовження таблиці 2.7

Ідентифікатор	**UC-3**
Очікуваний результат	Користувач отримує JWT-токен і доступ до функцій, які відповідають його ролі.
Порядок дій	1. Користувач відкриває форму входу або реєстрації. 2. Вводить email і пароль. 3. Backend перевіряє формат даних. 4. Для входу система знаходить користувача за email. 5. bcrypt порівнює пароль із хешем. 6. У разі успіху backend створює JWT. 7. Frontend зберігає токен і використовує його для захищених запитів.
Альтернативні шляхи	Якщо email не знайдено або пароль неправильний, система повертає помилку авторизації. Якщо користувач не має ролі адміністратора, доступ до `/api/admin` забороняється.

Таблиця 2.8 – Специфікація UC-4: адміністрування магазину

Ідентифікатор	**UC-4**
Актори	Адміністратор.
Початкові умови	Адміністратор авторизований, JWT дійсний, роль користувача дорівнює `admin`.
Очікуваний результат	Адміністратор переглядає статистику, керує товарами, переглядає замовлення і змінює їх статуси.
Порядок дій	1. Адміністратор входить у систему. 2. Відкриває адміністративний модуль. 3. Middleware перевіряє токен і роль. 4. Адміністратор переглядає статистику або список замовлень. 5. За потреби змінює статус замовлення. 6. За потреби створює, редагує або деактивує товар. 7. Backend зберігає зміни в PostgreSQL.
Альтернативні шляхи	Якщо токен відсутній, система повертає 401. Якщо користувач не адміністратор, система повертає 403. Якщо товар або замовлення не знайдено, повертається 404.

Схема бази даних. База даних є окремим структурним елементом системи, оскільки саме вона забезпечує цілісність каталогу, користувачів, замовлень і відгуків. Схема побудована так, щоб уникати дублювання: бренди, категорії та автомобілі зберігаються в окремих таблицях, а товари лише посилаються на них. Замовлення також винесені в окремі таблиці `orders` і `order_items`, що дозволяє зберігати історію покупки навіть після зміни ціни або деактивації товару (див. рис. 2.1).

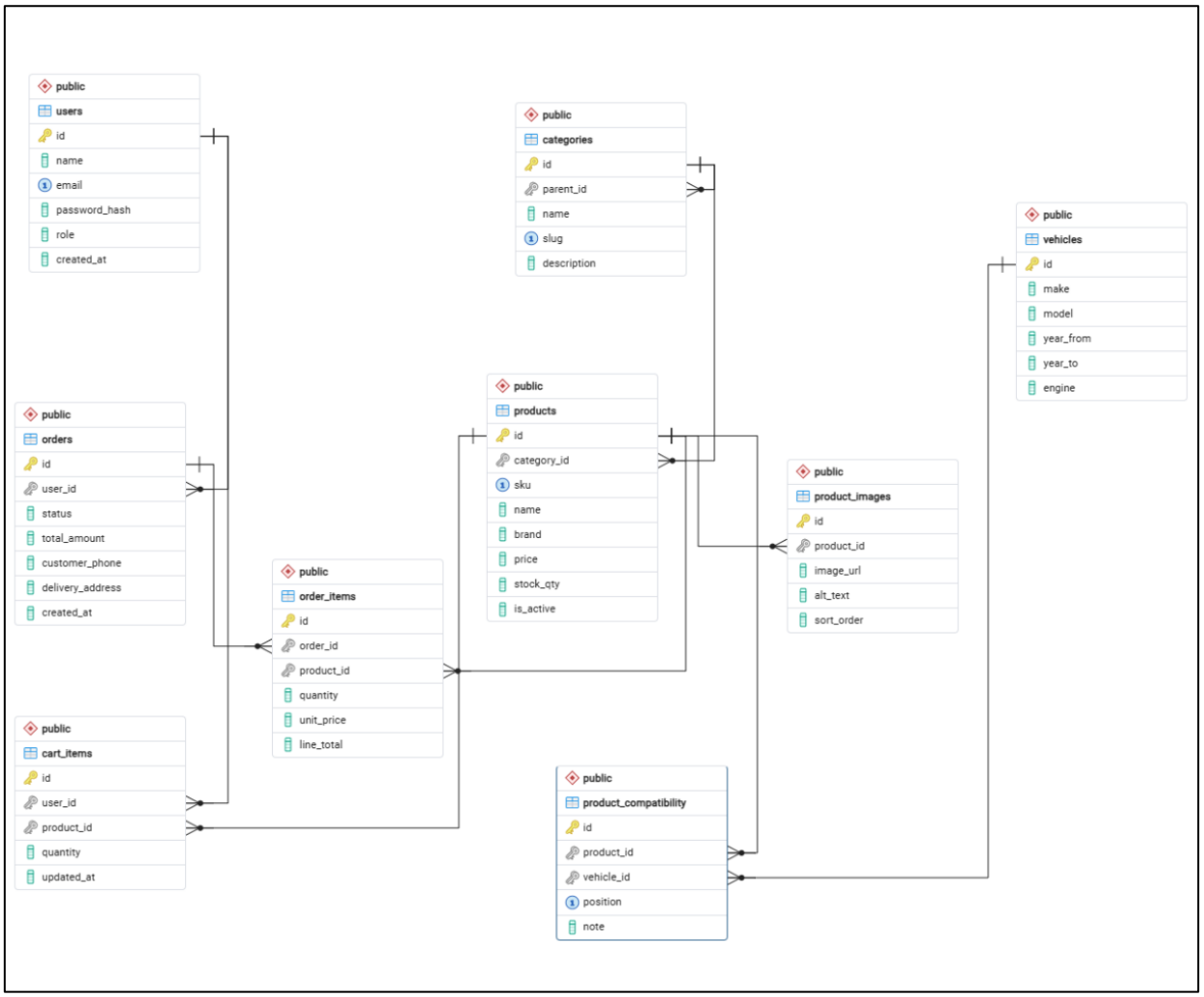


Рисунок 2.1 – Логічна модель бази даних

Схема бази даних побудована навколо сутності `products`, яка з'єднана з категоріями, брендами, відгуками, замовленнями та автомобілями. Така структура дозволяє розвивати каталог без повної перебудови бази. Наприклад, можна додати нові категорії або бренди без зміни коду, якщо API вже читає ці довідники з таблиць. Детальний опис структурної організації кожної реляційної сутності, її функціонального призначення та суміжних зв'язків наведено в таблиці 2.9.

Таблиця `product_compatibility` реалізує зв'язок багато-до-багатьох між товарами і транспортними засобами. Це важливе рішення, оскільки одна деталь може підходити до кількох моделей, а один автомобіль має багато сумісних деталей. Збереження цього зв'язку окремо дає змогу уникнути дублювання товарів.

Таблиця 2.9 – Схема бази даних застосунку

Таблиця	**Основні поля**	**Призначення**	**Зв’язки**
users	id, name, email, password_hash, phone, role	Зберігання користувачів і адміністраторів	orders.user_id, reviews.user_id
categories	id, slug, name, description	Довідник категорій автозапчастин	products.category_id
brands	id, slug, name, country	Довідник виробників	products.brand_id
vehicles	id, make, model, generation, year_from, year_to, body	Довідник автомобілів для сумісності	product_compatibility. vehicle_id
products	id, sku, slug, name, price, stock, specifications	Основний каталог товарів	categories, brands, reviews, order_items
product_compatibility	product_id, vehicle_id, engine_code, notes	Зв’язок товарів з автомобілями	products, vehicles
orders	id, user_id, customer data, delivery_method, payment_method, status, total	Замовлення клієнтів	users, order_items
order_items	id, order_id, product_id, sku, name, price, quantity	Позиції конкретного замовлення	orders, products
reviews	id, product_id, user_id, author, rating, text	Відгуки та оцінки товарів	products, users

З метою детального аналізу інформаційного наповнення системи та визначення точного складу атрибутів, які проходять обов’язкову валідацію на рівні бізнес-сервісів (зокрема при транзакційному оформленні замовлень і хешуванні конфіденційних даних), у таблиці 2.10 систематизовано конкретну інформацію, що підлягає довгостроковому збереженню.

Таблиця 2.10 – Збережена інформація в базі даних

Таблиця	**Призначення**	**Ключові поля**
users	Облікові записи покупців та адміністраторів	id, name, email, password_hash, role
categories	Категорії автозапчастин	id, slug, name, description

Продовження таблиці 2.10

Таблиця	Призначення	Ключові поля
brands	Виробники товарів	id, slug, name, country
vehicles	Автомобілі для перевірки сумісності	make, model, generation, year_from, year_to
products	Каталог товарів	sku, slug, price, stock, specifications, image_url
product_compatibility	Зв’язок товарів з автомобілями	product_id, vehicle_id, engine_code
orders	Замовлення клієнтів	customer data, delivery, payment, status, total
order_items	Позиції замовлення	sku, name, price, quantity
reviews	Відгуки про товари	product_id, user_id, rating, text

2.3 Структурна архітектура системи та взаємодія компонентів

Система побудована як клієнт-серверний вебзастосунок. Клієнтська частина відповідає за візуальну взаємодію з користувачем: перегляд каталогу, фільтрацію, сторінку товару, кошик, оформлення замовлення, профіль і адміністративні екрани. Серверна частина надає REST API, виконує бізнес-логіку, перевіряє права доступу та взаємодіє з базою даних. Загальну концептуальну схему розподілу компонентів, логіку взаємодії клієнта з сервером та рівні абстракції програмного комплексу відображено на рисунку 2.2.

У backend-проєкті виділено маршрути `auth`, `catalog`, `orders` та `admin`. Такий поділ зменшує зв’язаність і робить систему зрозумілішою для підтримки: каталог обслуговує товари і довідники, авторизація керує користувачами й JWT, замовлення працюють із транзакціями, а адміністративний модуль контролює товари, статуси та статистику.

Розроблений застосунок є інтернет-магазином автозапчастин, орієнтованим на покупця, який хоче швидко знайти потрібну деталь, перевірити її основні характеристики та оформити замовлення. На відміну від

звичайної вітрини товарів, система враховує специфіку предметної області: товари мають категорії, бренди, артикул, залишок, зображення, характеристики, рейтинг і зв’язок із сумісними автомобілями.



Рисунок 2.2 – Загальна структура системи

Головна сторінка виконує роль стартової точки взаємодії. Вона має швидко пояснити користувачу, що перед ним магазин автозапчастин, і спрямувати його до каталогу або популярних категорій. Для такої системи головна сторінка не повинна бути лише рекламним блоком: її практична функція – скоротити шлях до пошуку товару. Тому важливими елементами є помітний перехід до каталогу, короткі переваги магазину, товарні категорії та приклади позицій.

Каталог є центральним екраном застосунку. Саме тут користувач взаємодіє з фільтрами за категорією, брендом, автомобілем, ціною та сортуванням. Якщо результатів багато, пагінація дозволяє не завантажувати весь асортимент одразу. Якщо результатів немає, інтерфейс має показати зрозумілий порожній стан, щоб користувач змінив фільтри, а не сприйняв систему як несправну.

Картка товару призначена для прийняття рішення про покупку. Вона повинна показувати назву, бренд, SKU, ціну, стару ціну за наявності, залишок, опис, специфікації, сумісність з автомобілями та відгуки. Для автозапчастин

це особливо важливо, оскільки користувач купує не просто товар, а технічно придатну деталь.

Кошик і checkout реалізують перехід від вибору до замовлення. Користувач перевіряє товари, кількість і суму, після чого вводить контактні дані, місто, адресу, спосіб доставки та оплати. На сервері цей процес завершується транзакцією: система перевіряє наявність товарів, створює замовлення, додає позиції та зменшує залишки.

Адміністративна частина потрібна для керування магазином. Адміністратор переглядає статистику, замовлення, товари, змінює статуси, створює або редагує позиції та деактивує недоступні товари. Таке розділення ролей дозволяє одному застосунку обслуговувати два різні сценарії: зручну покупку для клієнта і контроль операцій для власника магазину.

Для практичної реалізації описаної архітектури та забезпечення повноцінного Fullstack-зв’язку між клієнтським інтерфейсом та серверними сервісами було спроєктовано та декомпозивано систему маршрутизації API. Специфікацію розроблених інтерфейсів взаємодії, методів передачі даних та їхнього функціонального призначення наведено в таблиці 2.11.

Таблиця 2.11 – Основні API-маршрути системи

Група	**Метод**	**Маршрут**	**Призначення**
Auth	POST	/api/auth/register	Створення користувача і видача токена
Auth	POST	/api/auth/login	Перевірка облікових даних
Auth	GET	/api/auth/me	Отримання поточного користувача
Catalog	GET	/api/products	Пошук і фільтрація товарів
Catalog	GET	/api/products/:slug	Повна картка товару
Catalog	POST	/api/products/:id/reviews	Додавання відгуку
Orders	POST	/api/orders	Оформлення замовлення
Orders	GET	/api/orders/my	Історія замовлень користувача
Admin	GET	/api/admin/stats	Статистика магазину
Admin	PATCH	/api/admin/orders/:id/status	Зміна статусу замовлення

Маршрути каталогу. Маршрути каталогу відповідають за читання довідників і товарів. Запити `/categories`, `/brands` і `/vehicles` повертають

опорні дані для фільтрів. Запит `/products` приймає параметри пошуку, категорії, бренду, автомобіля, ціни, сортування і пагінації. Це дає змогу frontend будувати гнучкий каталог без дублювання фільтраційної логіки на клієнті.

Окремий маршрут отримання товару за slug повертає розширену інформацію: базові поля, категорію, бренд, рейтинг, кількість відгуків і сумісні автомобілі. Для користувача це означає, що сторінка товару може бути достатньо повною, а для розробника – що логіка збирання даних знаходиться на сервері.

Маршрути авторизації. Авторизація складається з реєстрації, входу, отримання поточного користувача і зміни пароля. Під час реєстрації пароль хешується, а під час входу порівнюється збережений хеш. Після успішної перевірки сервер видає JWT, який клієнт використовує для подальших запитів.

Перевага такого підходу полягає у відсутності серверної сесії, що спрощує масштабування API. Водночас токен повинен мати обмежений строк дії, а секрет підпису має зберігатися у змінних середовища, а не в коді. У проєкті це враховано через `JWT_SECRET` і `JWT_EXPIRES_IN`.

Маршрути замовлень. Маршрут створення замовлення використовує optional authentication: користувач може оформити покупку як гість або як авторизований клієнт. Якщо токен присутній, замовлення прив’язується до користувача, що дозволяє відображати історію у кабінеті. Якщо токена немає, система все одно зберігає контактні дані покупця. Маршрут `/orders/my` вимагає авторизації і повертає замовлення поточного користувача. Це розділяє публічні та приватні дії: створити замовлення може ширше коло користувачів, але перегляд особистої історії доступний лише власнику.

Адміністративні маршрути. Адміністративний router спочатку застосовує `requireAuth`, а потім `requireAdmin`. Така послідовність означає, що сервер спершу встановлює особу користувача, а потім перевіряє його роль. Усі вкладені маршрути автоматично отримують цей захист, що зменшує ризик випадково залишити метод відкритим.

Адмінка містить статистику, список замовлень, список товарів, зміну статусів, створення, редагування і деактивацію товарів. Деактивація замість фізичного видалення є логічно правильнішою для магазину: старі замовлення не повинні втрачати посилання на товарну позицію.

Обробка помилок. У серверній частині використовується централізований error handler. Це означає, що маршрути можуть кидати `AppError` з кодом і повідомленням, а відповідь клієнту формується в одному місці. Такий підхід зменшує дублювання `try/catch` і робить поведінку API передбачуваною.

Для користувача правильна обробка помилок важлива не менше, ніж успішні сценарії. Якщо товар не знайдено, пароль неправильний або залишку недостатньо, система повинна повернути зрозумілий статус і повідомлення. Це дає frontend можливість показати коректний стан інтерфейсу.

2.4 Алгоритмічне забезпечення функціонування модулів

Найважливішими алгоритмами роботи вебзастосунку є алгоритм авторизації користувача та алгоритм оформлення замовлення. Саме ці процеси безпосередньо пов’язані з обробленням персональних даних, перевіркою коректності введеної інформації та зміною стану системи. Авторизація забезпечує ідентифікацію користувача, перевірку його облікових даних і надання доступу відповідно до ролі. Оформлення замовлення відповідає за перевірку кошика, контроль залишків товарів, створення записів у базі даних і передавання замовлення на подальшу обробку адміністратором.

Для зменшення кількості помилкових станів у зазначених алгоритмах застосовуються валідація вхідних даних, перевірка прав доступу, оброблення виняткових ситуацій і повернення зрозумілих повідомлень користувачу. Такий підхід підвищує надійність системи, запобігає некоректному створенню замовлень і забезпечує безпечну роботу з обліковими записами користувачів.

Для формалізації логіки розгалужень, циклічних перевірок та послідовності взаємодії клієнтської частини з серверними сервісами під час ідентифікації користувача було розроблено відповідну модель поведінки, графічне представлення якої наведено на рисунку 2.3.

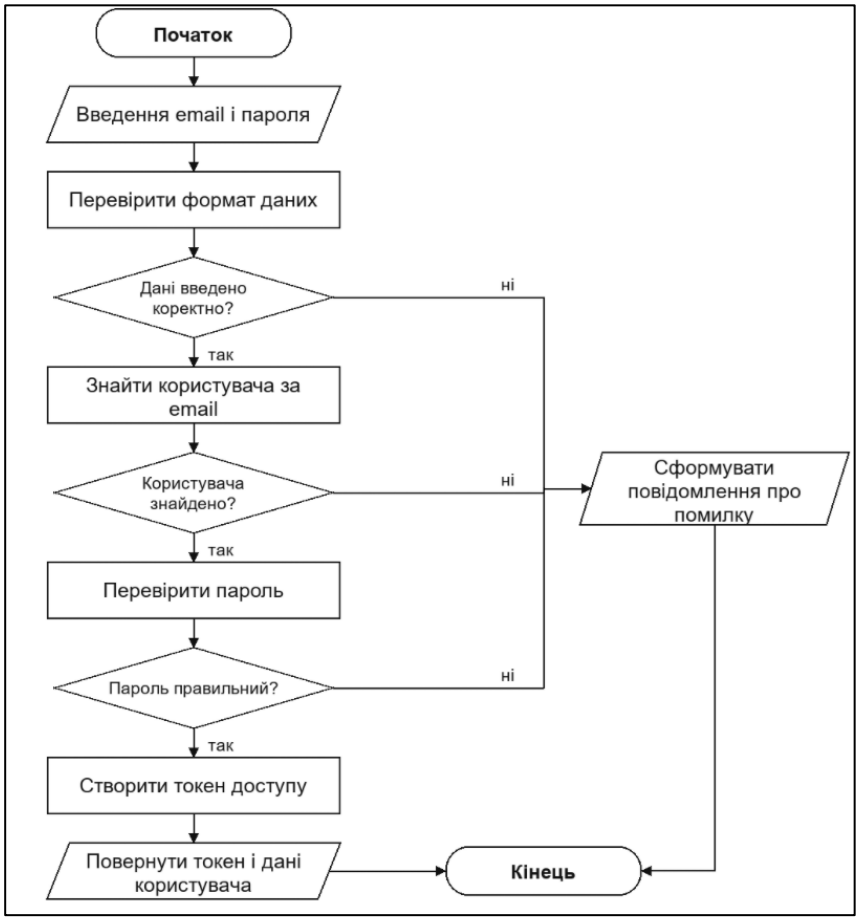


Рисунок 2.3 – Блок-схема алгоритму авторизації

Алгоритм авторизації користувача починається з введення email і пароля. Після цього система перевіряє коректність формату введених даних. Якщо дані заповнено неправильно, користувачу повертається повідомлення про помилку. Якщо формат даних коректний, система виконує пошук користувача за email, перевіряє наявність облікового запису та порівнює введений пароль зі збереженим значенням. У разі успішної перевірки створюється токен доступу, який повертається клієнтській частині разом з основними даними користувача. Якщо користувача не знайдено або пароль неправильний, система формує повідомлення про помилку авторизації.

Іншим критично важливим процесом, що потребує суворого контролю ізольованості та атомарності операцій на рівні СКБД при паралельних запитах, є алгоритм обробки купівель. Покрокову послідовність дій, логіку перевірки складських запасів та фіксації транзакційних даних відображено на рисунку 2.4.

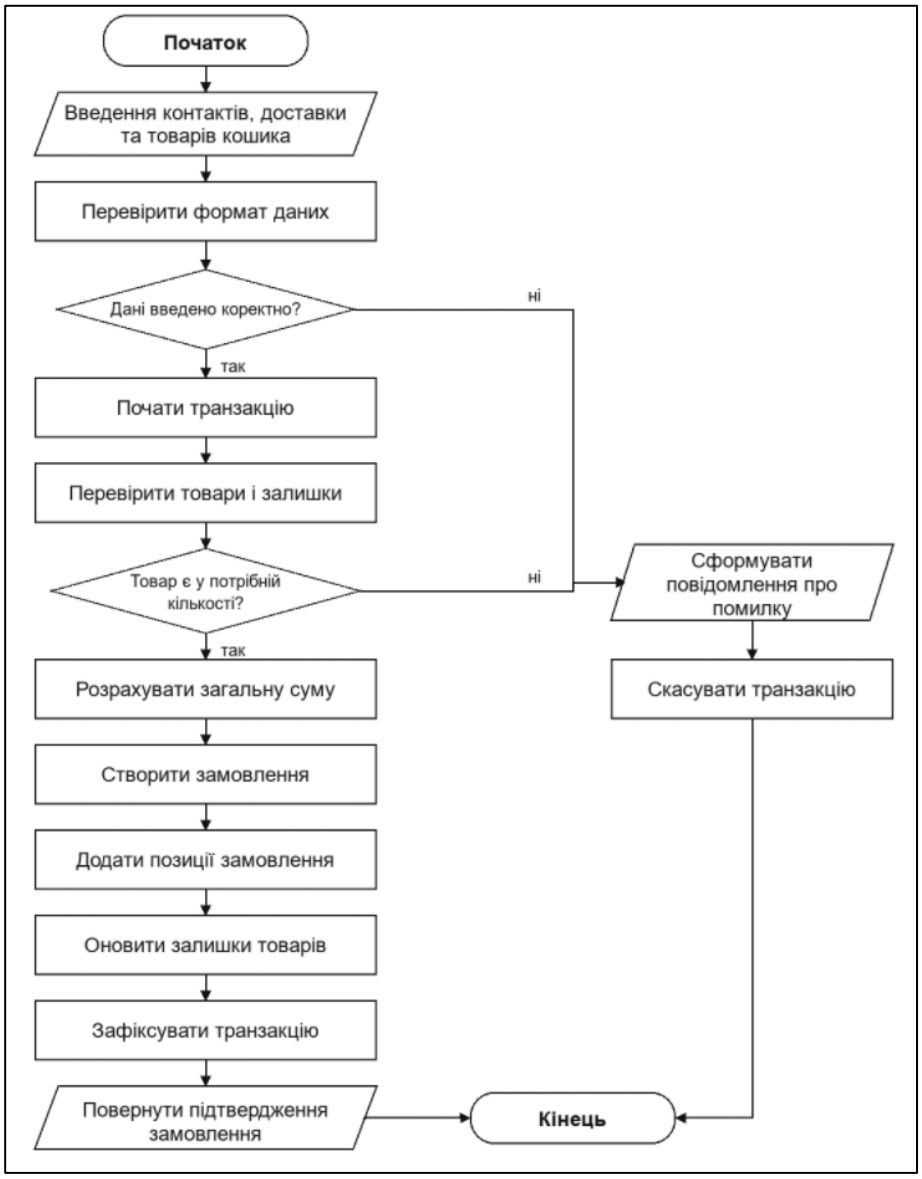


Рисунок 2.4 – Блок-схема алгоритму оформлення замовлення

Алгоритм оформлення замовлення починається з введення контактних даних, параметрів доставки та переліку товарів із кошика. На першому етапі система перевіряє правильність заповнення форми. Якщо дані некоректні, користувач отримує повідомлення про помилку. Якщо дані введено

правильно, система починає транзакцію, перевіряє товари та їх залишки на складі. За наявності потрібної кількості товарів розраховується загальна сума, створюється замовлення, додаються позиції замовлення та оновлюються залишки товарів. Після цього транзакція фіксується, а користувач отримує підтвердження замовлення. Якщо під час перевірки виявлено відсутність товару або недостатню кількість на складі, система формує повідомлення про помилку та скасовує транзакцію. Це дозволяє уникнути некоректного списання товарів і забезпечує цілісність даних у базі.

2.5 Висновки до розділу 2

У другому розділі було виконано проєктування та обґрунтування архітектури вебзастосунку інтернет-магазину автозапчастин. На основі аналізу вимог детально обґрунтовано вибір технологічного стеку (React, Node.js, Express, PostgreSQL, Docker).

Розроблено структурну архітектуру системи та схему взаємодії компонентів, визначено ключові API-маршрути для груп функцій Auth, Catalog, Orders та Admin, а також реалізовано централізований механізм обробки помилок. Сформовано інформаційне забезпечення системи: спроєктовано логічну структуру бази даних PostgreSQL, включаючи таблицю сумісності деталей та автомобілів product_compatibility. Окрему увагу приділено алгоритмічному забезпеченню функціонування модулів, зокрема розроблено блок-схеми алгоритмів авторизації користувачів та транзакційного оформлення замовлень із контролем складських залишків.

Отже, у розділі сформовано повну проєктну основу вебзастосунку, яка забезпечує високу надійність, безпеку та є базою для подальшої програмної реалізації системи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

3.1 Особливості реалізації клієнтської та серверної частини

3.1.1 Клієнтська частина

Клієнтська частина реалізує основний шлях покупця: від перегляду товарів до оформлення замовлення. Інтерфейс побудований навколо каталогу, де користувач бачить товарні картки із зображенням, назвою, брендом, ціною, наявністю та рейтингом.

Для зменшення когнітивного навантаження фільтри згруповано за природними ознаками: категорія, бренд, автомобіль, ціновий діапазон і сортування. Такий підхід робить систему зрозумілою навіть для користувача, який не має глибоких знань про артикули.

Важливу роль у сприйнятті інформації відіграє графічне наповнення картки товару, оскільки наявність візуальних маркерів допомагає покупцеві швидше ідентифікувати потрібну деталь. Специфіку візуального відображення та приклади зображень товарів, використаних у каталозі розроблюваного вебзастосунку, наведено на рисунку 3.1.

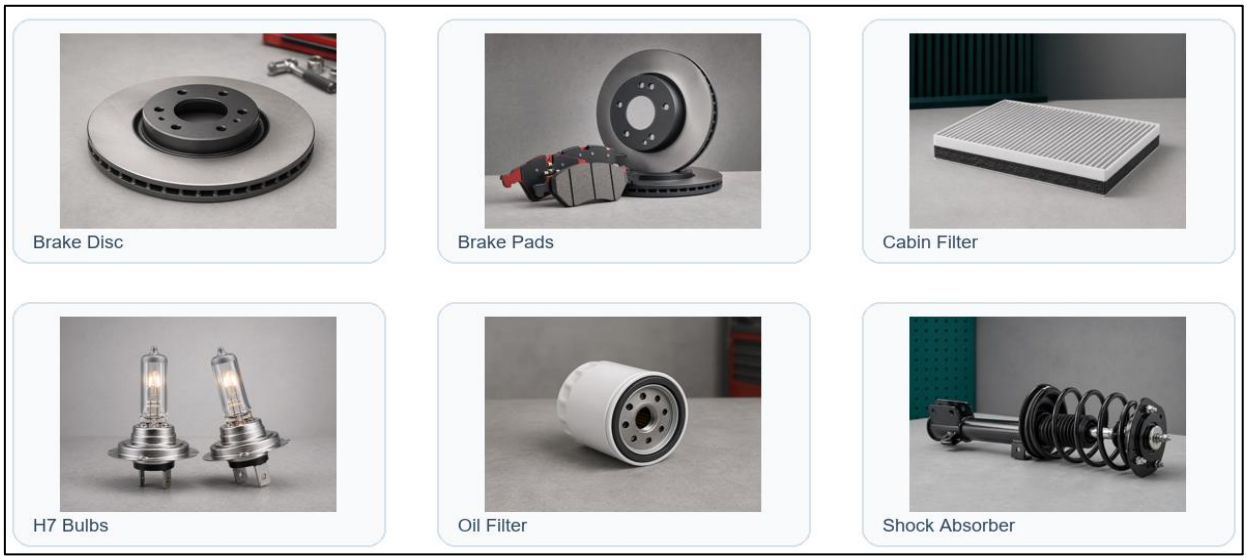


Рисунок 3.1 – Приклади зображень товарів, використаних у каталозі

3.1.2 Серверна частина

Серверна частина складається з маршрутів, middleware, сервісів, репозиторіїв і схем валідації входять даних за допомогою бібліотеки Zod. Middleware requireAuth перевіряє токен JWT і додає об’єкт користувача до запиту, а requireAdmin обмежує адміністративні дії, перевіряючи відповідність ролі admin.

Репозиторії повністю інкапсулюють SQL-запити до бази даних PostgreSQL. Наприклад, модуль каталогу використовує складні запити з об’єднанням таблиць (JOIN) до категорій і брендів, одночасно обчислюючи середній рейтинг девайса та загальну кількість залишених відгуків, а також забезпечує підтримку пагінації на рівні СКБД. Сервіс замовлень містить у собі ізольовану бізнес-логіку перевірки залишків та відкриття транзакцій, яку винесено з шару контролерів і маршрутів для забезпечення чистоти архітектури.

3.2 Опис інтерфейсу користувача та сценаріїв роботи

Інтерфейс розробленого застосунку має прикладний характер: він не перевантажений декоративними блоками і фокусується на діях користувача. На першому плані перебувають пошук, фільтри, перелік товарів, ціна, наявність і перехід до детальної інформації [20].

Картка товару виконує роль точки прийняття рішення. Вона має показувати назву, бренд, характеристики, сумісні автомобілі, відгуки та доступність. Для автозапчастин це важливо, оскільки покупець часто ухвалює рішення не лише за ціною, а за відповідністю та довірою до виробника.

Адміністративний інтерфейс має інший фокус: він повинен бути щільнішим, швидшим для сканування і придатним для повторюваних операцій. Тому основними елементами є таблиці, статуси, форми редагування

та статистичні показники. З метою систематизації архітектури інтерфейсу та відображення взаємозв’язку між потребами користувачів і функціональними компонентами системи розроблено матрицю відповідності, яку наведено в таблиці 3.1.

Таблиця 3.1 – Відповідність UI-екранів користувацьким задачам

Екран	**Основна задача**	**Очікуваний результат**
Головна	Швидко перейти до каталогу або популярних категорій	Користувач починає пошук
Каталог	Знайти товар за фільтрами	Скорочений список релевантних товарів
Картка товару	Перевірити сумісність і характеристики	Рішення про додавання в кошик
Кошик	Перевірити позиції та суму	Підготовка до оформлення
Checkout	Передати контакти, доставку й оплату	Створене замовлення
Профіль	Переглянути дані та історію	Контроль власних замовлень
Адмінка	Керувати товарами й замовленнями	Оперативне адміністрування

Для підтвердження працездатності розробленого вебзастосунку було проведено повне натурне тестування та перевірку основних користувацьких сценаріїв. На відповідних скриншотах (рис. 3.2 – 3.11) детально наведено роботу графічного інтерфейсу інтернет-магазину автозапчастин: авторизацію, реєстрацію, перегляд особистого кабінету, роботу з каталогом, кошиком, оформленням замовлення та адміністративною частиною.

На першому етапі було перевірено сторінку входу до особистого кабінету. Користувач має можливість ввести email та пароль, після чого система виконує перевірку облікових даних. Якщо користувач ще не має акаунта, він може перейти до форми реєстрації. Такий підхід забезпечує простий і зрозумілий шлях входу до системи без зайвих дій. Візуальну реалізацію форми авторизації показано на рисунку 3.2.

Форма реєстрації містить основні поля, необхідні для створення облікового запису: ім’я, телефон, email та пароль. Наявність обов’язкових полів дозволяє запобігти створенню неповних записів користувачів. Після

успішної реєстрації користувач може авторизуватися та отримати доступ до особистого кабінету. Графічний вигляд форми створення нового облікового запису наведено на рисунку 3.3.

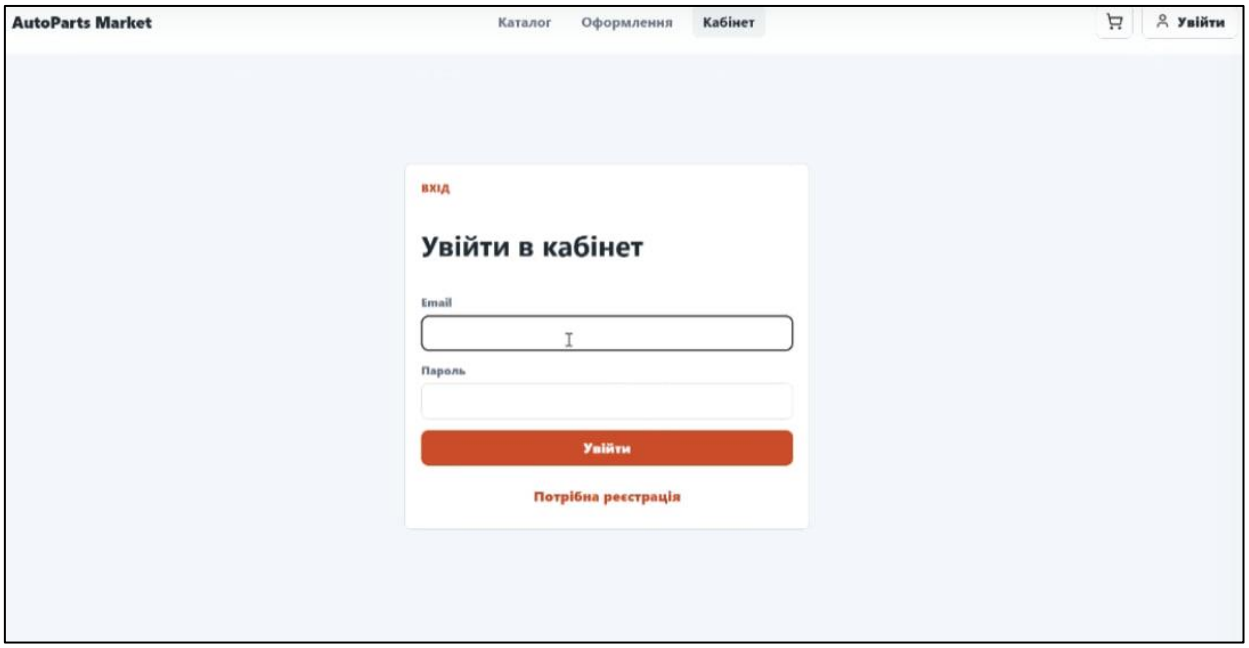


Рисунок 3.2 – Сторінка входу

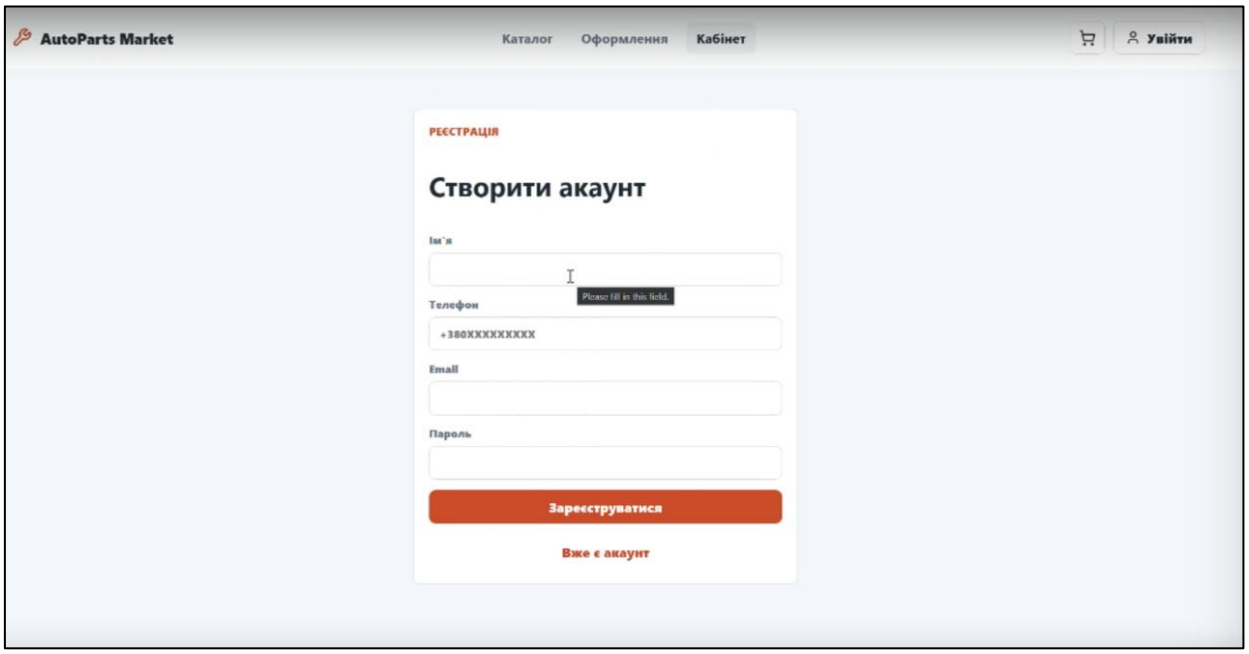


Рисунок 3.3 – Реєстрація користувача

В особистому кабінеті відображаються дані користувача: ім’я, email і номер телефону. Також реалізовано блок зміни пароля, у якому потрібно

ввести поточний пароль, новий пароль і підтвердження нового пароля. Це дозволяє користувачу самостійно керувати безпекою свого облікового запису. Окремо в кабінеті передбачено розділ історії замовлень, де користувач може переглядати свої попередні покупки. Приклад візуалізації сторінки профілю користувача відображено на рисунку 3.4.

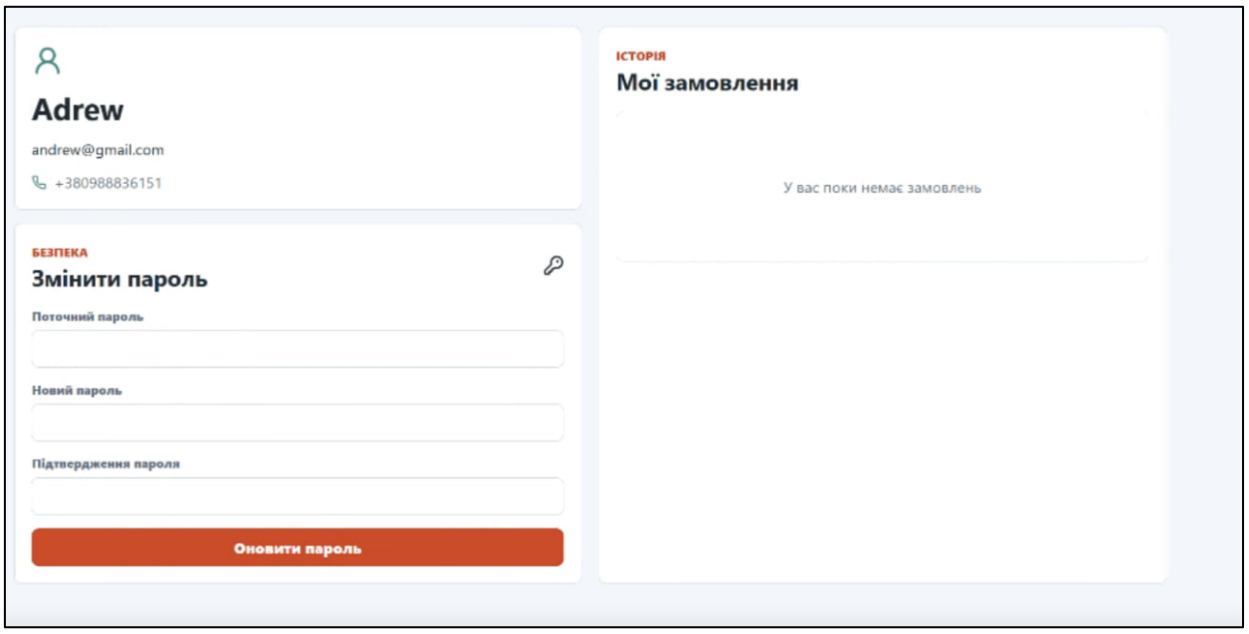


Рисунок 3.4 – Особистий кабінет

Центральним елементом вебзастосунку є каталог товарів. У каталозі відображаються картки автозапчастин із фотографією, назвою, брендом, категорією, коротким описом, ціною, старою ціною за наявності, залишком і рейтингом. Ліва панель містить фільтри за пошуковим запитом, категорією, брендом, сумісністю з автомобілем, ціновим діапазоном і сортуванням. Така структура полегшує пошук потрібної деталі та дозволяє користувачу швидко звузити список товарів. Загальний інтерфейс сторінки пошуку та фільтрації товарів наведено на рисунку 3.5.

Після додавання товарів до кошика користувач може переглянути вибрані позиції у боковій панелі, яка відкривається без переходу на окрему сторінку. У кошику відображаються назви товарів, зображення, ціна за одиницю, вибрана кількість, кнопки збільшення або зменшення кількості,

кнопка видалення позиції та загальна сума замовлення. Завдяки цьому покупець може швидко перевірити склад майбутньої покупки, не втрачаючи контекст перегляду каталогу або картки товару.

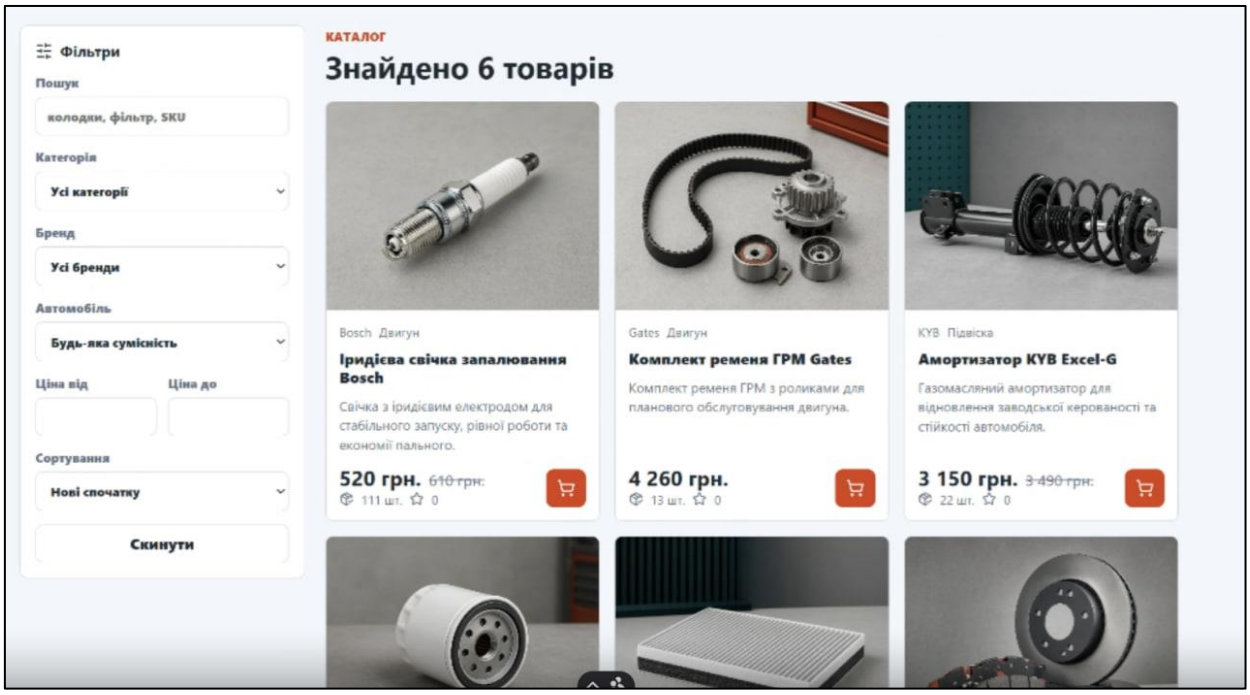


Рисунок 3.5 – Каталог товарів

Користувач має можливість змінювати кількість товарів безпосередньо в кошику, після чого система автоматично перераховує підсумкову суму. Якщо певна позиція була додана помилково, її можна видалити окремою кнопкою. Така реалізація особливо важлива для інтернет-магазину автозапчастин, оскільки покупець може одночасно обирати кілька різних деталей і повинен контролювати їхню кількість, сумісність та загальну вартість перед оформленням замовлення. Роботу інтерфейсу висувного модуля кошика з обраними деталями продемонстровано на рисунку 3.6.

Якщо кошик порожній, система показує відповідний порожній стан із повідомленням «Кошик порожній» і кнопкою переходу до каталогу. Такий інтерфейс не залишає користувача без пояснення, а одразу пропонує наступну дію. Це покращує зручність використання застосунку. Графічний вигляд інтерфейсу при відсутності доданих товарів наведено на рисунку 3.7.

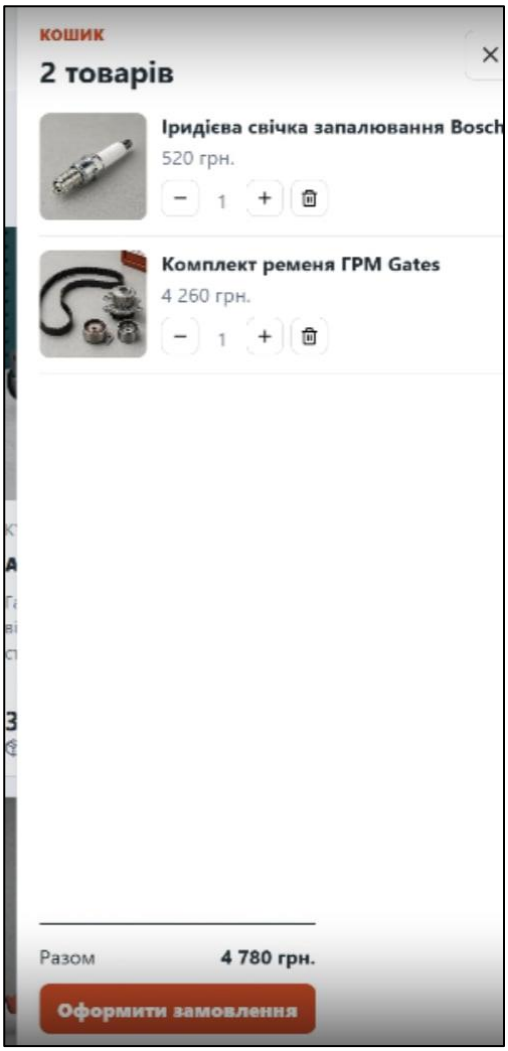


Рисунок 3.6 – Кошик замовлення

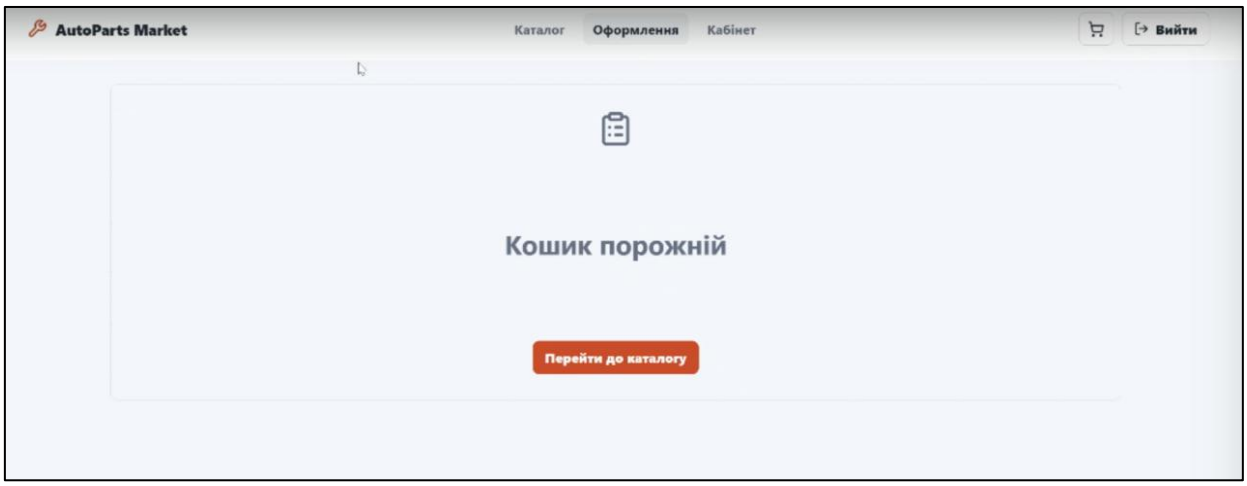


Рисунок 3.7 – Порожній кошик

Під час оформлення замовлення користувач переходить до сторінки checkout. На цій сторінці автоматично підставляються дані авторизованого

користувача: ім’я, email і телефон. Додатково користувач заповнює місто, адресу доставки, спосіб доставки, спосіб оплати та коментар до замовлення. Праворуч від форми відображається склад замовлення з переліком товарів і підсумковою сумою. Після натискання кнопки підтвердження система створює замовлення. Інтерфейс форми оформлення та перевірки кінцевих даних замовлення наведено на рисунку 3.8.

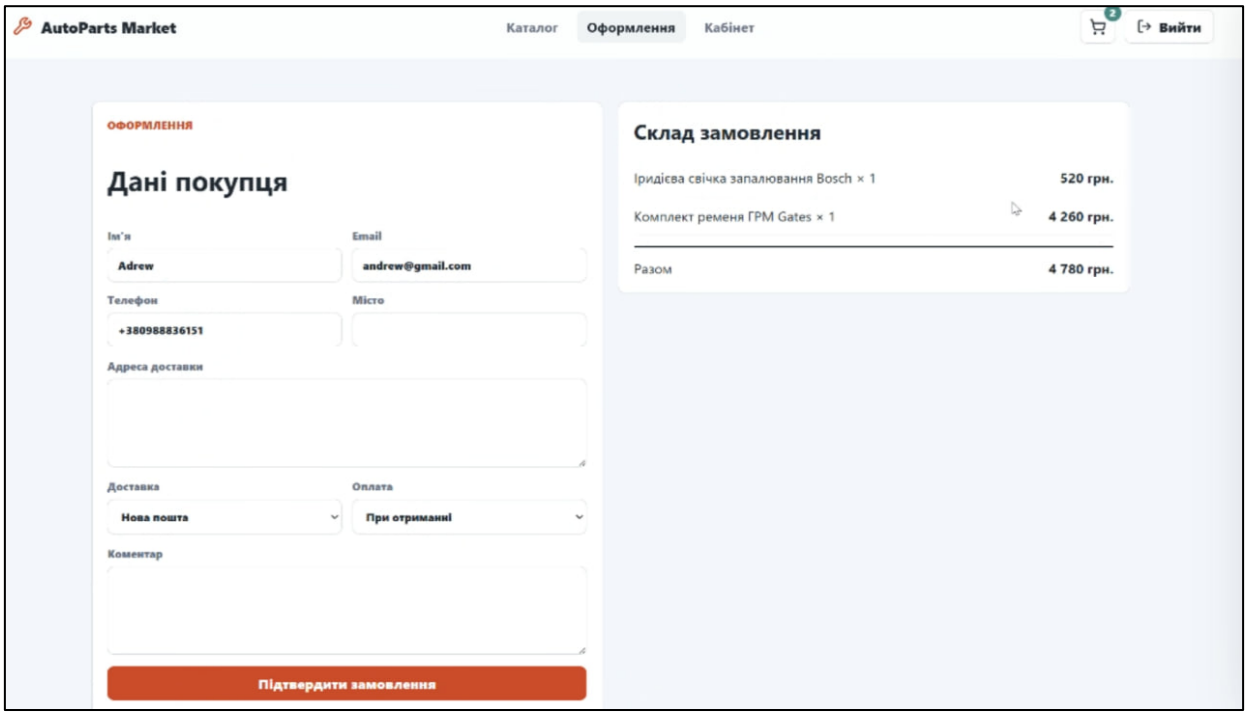


Рисунок 3.8 – Особистий кабінет з замовленням

Після успішного оформлення замовлення воно з’являється в історії користувача. У списку відображаються номер замовлення, дата, статус і загальна сума. Це підтверджує, що сценарій створення замовлення працює коректно: товари додаються до кошика, дані передаються на сервер, замовлення зберігається і відображається в кабінеті покупця. Візуальне представлення списку створених користувачем покупок наведено на рисунку 3.9.

Також було перевірено адміністративну частину вебзастосунку. В адмін-панелі адміністратор може переглядати останні замовлення, бачити клієнта, дату, статус і суму. Для кожного замовлення доступна зміна статусу,

наприклад: «Новий», «Зібрано», «Відправлено» або «Скасовано». Це дозволяє адміністратору контролювати процес обробки замовлень без прямого втручання в базу даних. Інтерфейс робочого місця адміністратора з переліком замовлень відображено на рисунку 3.10.



Рисунок 3.9 – Мої замовлення



Рисунок 3.10 – Адмін-панель замовлень

Окремо в системі реалізовано інтерфейс керування товарами, призначений для адміністратора магазину. За допомогою цього модуля адміністратор може додавати нові товари до каталогу, вказуючи основні параметри: SKU, назву, slug, поточну ціну, стару ціну, залишок на складі, категорію, бренд, посилання на зображення, опис і технічні характеристики. Наявність таких полів дає змогу зберігати не лише базову інформацію про товар, а й дані, необхідні для коректного відображення товарної картки та подальшої фільтрації в каталозі.

Праворуч від форми додавання розміщено список уже наявних товарів каталогу. Для кожної позиції відображається назва, ціна, залишок на складі та кнопка видалення. Такий підхід дозволяє адміністратору швидко перевіряти поточний асортимент, видаляти неактуальні позиції та оперативно оновлювати товарні дані. Завдяки цьому інтерфейс керування товарами спрощує підтримку каталогу, зменшує кількість ручних дій і допомагає забезпечувати актуальність інформації для покупців. Інтерфейс додавання, редагування та контролю складських залишків продукції представлено на рисунку 3.11.

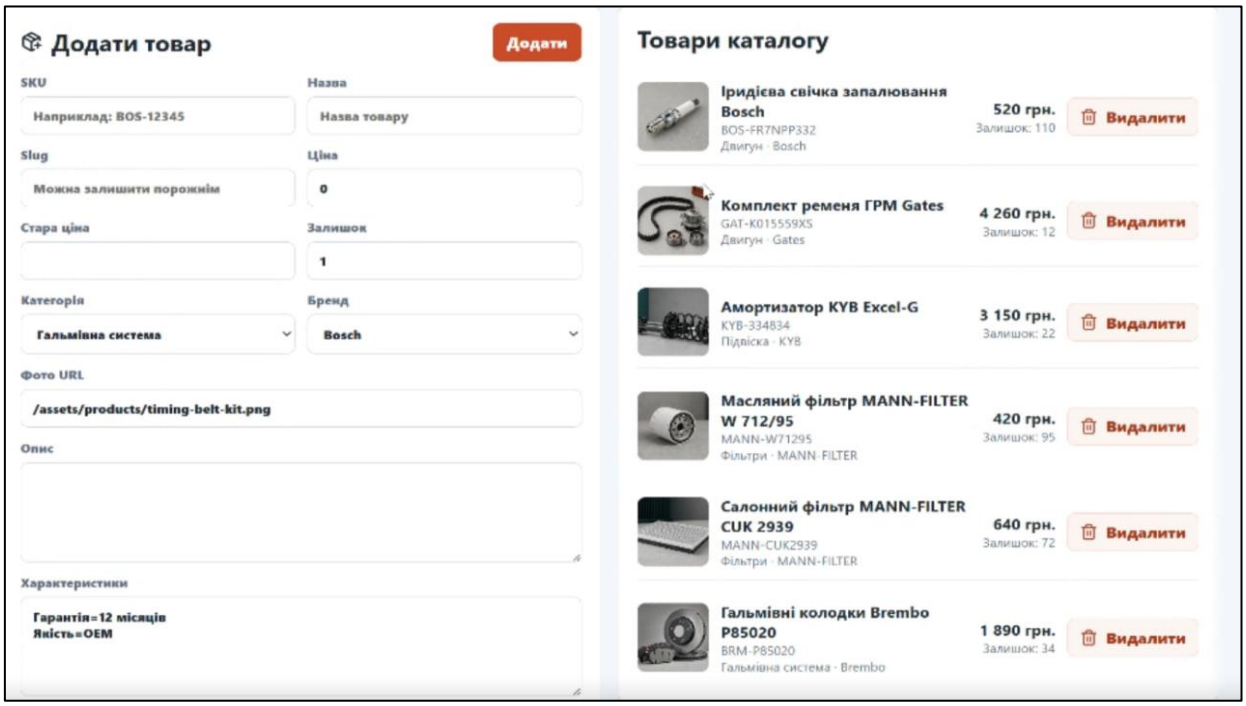


Рисунок 3.11 – Додавання товару та залишок на складі

Отже, наведені скриншоти демонструють роботу основних модулів вебзастосунку: авторизації, реєстрації, особистого кабінету, каталогу, кошика, оформлення замовлення, історії покупок і адміністративного керування. Перевірка показала, що система забезпечує повний базовий цикл роботи інтернет-магазину автозапчастин: від вибору товару користувачем до створення замовлення та його подальшої обробки адміністратором. Це підтверджує практичну працездатність розробленого програмного продукту

та відповідність основним функціональним вимогам, сформульованим у попередніх розділах.

3.3 Верифікація та функціональне тестування вебзастосунку

Тестування виконувалося за сценарним принципом: для кожної ключової функції визначено вхідні дані, очікуваний результат і критерій успішності. Особливу увагу приділено негативним сценаріям, оскільки саме вони показують, чи система коректно захищає дані та бізнес-правила [19, 21].

Для комплексної оцінки працездатності розробленого програмного продукту, валідації інтеграційних маршрутів API та перевірки механізмів розмежування ролей було сформовано перелік тестових випадків (Test Cases). Деталізовану специфікацію проведених випробувань, параметрів запитів та підсумкових статусів успішності виконання процедур верифікації наведено в таблиці 3.2.

Таблиця 3.2 – Тестові сценарії

ID	Сценарій	Вхідні дані	Очікуваний результат	Статус
TC-01	Отримання списку категорій	GET /api/categories	JSON-масив категорій	Пройдено
TC-02	Пошук товару за текстом	search=filter	Повертаються релевантні товари	Пройдено
TC-03	Фільтрація за брендом	brand=bosch	У списку лише товари Bosch	Пройдено
TC-04	Реєстрація нового користувача	Коректні name/email/password	201 Created, JWT	Пройдено
TC-05	Вхід з неправильним паролем	email + помилковий пароль	401 або повідомлення про помилку	Пройдено
TC-06	Створення замовлення	Контакти + позиції кошика	Замовлення створено, залишок зменшено	Пройдено
TC-07	Недостатній залишок	quantity більше stock	409 Conflict, склад не змінено	Пройдено

Продовження таблиці 3.2

ID	Сценарій	Вхідні дані	Очікуваний результат	Статус
ТС-08	Доступ до адмінки без ролі	JWT звичайного користувача	403 Forbidden	Пройдено
ТС-09	Зміна статусу замовлення	PATCH /api/admin/orders/:id/status	Оновлений статус	Пройдено

Результати тестування показують, що основні маршрути, бізнес-правила та обмеження доступу працюють узгоджено. Найбільш критичним місцем є оформлення замовлення, тому воно реалізоване через транзакцію, яка не допускає часткового створення даних.

3.4 Оцінка якості та аналіз ризиків програмного забезпечення

3.4.1 Додатковий аналіз якості реалізації

Оцінка супроводжуваності. Супроводжуваність проєкту визначається чітким розділенням архітектурних шарів у backend-частині на директорії routes, services, repositories, middleware, validation і utils. Кожна директорія має зрозумілу зону відповідальності, завдяки чому нові функції можна легко додавати без виникнення побічних ефектів у суміжних модулях.

Оцінка безпеки. Базові механізми безпеки повністю інтегровані в систему: пакет Helmet встановлює захисні HTTP-заголовки, політика CORS обмежує походження сторонніх запитів, бібліотека bcrypt надійно захищає паролі за допомогою солі та хешування, а авторизація заснована на токенах JWT. Вхідна валідація схем через Zod мінімізує ризики ін’єкцій та неочікуваних збоїв.

Оцінка продуктивності. На рівні API швидкість роботи підтримується обов’язковою пагінацією та фільтрацією безпосередньо на стороні СКБД.

Створення індексів у PostgreSQL за категорією, брендом та ідентифікаторами зв’язків є ефективною стартовою оптимізацією для роботи з великими обсягами товарів.

Оцінка користувацького досвіду (UX). Шлях покупця спрощено до мінімуму логічних кроків: пошук – кошик без зміни контексту екрана – checkout-форма із автоматичним заповненням контактів. Інформування про технічну сумісність запчастин з автомобілями безпосередньо на картці товару підвищує конверсію та знижує ризик помилкового замовлення.

Для автозапчастин особливо важливо показувати сумісність до покупки, а не після неї. Якщо користувач бачить відповідні автомобілі на картці товару, він має більше впевненості. Якщо сумісність невідома, система має чесно спонукати до уточнення, а не приховувати невизначеність.

3.4.2 Аналіз ризиків системи

Для забезпечення стабільної експлуатації вебзастосунку, мінімізації потенційних технічних збоїв та захисту критично важливих бізнес-процесів інтернет-магазину автозапчастин було проведено комплексне дослідження потенційних загроз. На основі цього аналізу було сформовано матрицю технічних ризиків та визначено превентивні інженерні заходи щодо їх нівелювання.

Повний перелік ідентифікованих ризиків, можливих деструктивних наслідків для системи та засобів їхнього запобігання наведено в таблиці 3.3.

Таблиця 3.3 – Ризики та способи їх зменшення

Ризик	**Наслідок**	**Спосіб зменшення**
Некоректні залишки	Продаж недоступного товару	Транзакції, блокування рядків, аудит змін складу
Слабкий пароль адміністратора	Несанкціонований доступ	Політика складності, 2FA, rate limiting

Продовження таблиці 3.3

Ризик	Наслідок	Спосіб зменшення
Застарілі зображення	Низька довіра до товару	Модерація медіа та автоматична оптимізація
Повільний пошук	Втрата користувача	Індекси, повнотекстовий пошук, кеш довідників
Помилка сумісності	Повернення товару	Розширений довідник авто, перевірка VIN, ручна модерація

3.4.3 Подальше вдосконалення застосунку

На основі проведеного аналізу визначено пріоритетні напрямки розвитку програмного продукту:

– додати автоматизований імпорт товарних позицій із файлів CSV/XLSX для великих постачальників;
– реалізувати повноцінний підбір автозапчастин за VIN-кодом автомобіля;
– інтегрувати онлайн-еквайринг через платіжний шлюз (наприклад, LiqPay / Stripe);
– розширити аналітичний модуль для адміністратора (звіти з продажів, середній чек, ходові товари);
– додати автоматичні сповіщення користувачів про зміну статусів замовлень через Email/SMS.

3.5 Висновки до розділу 3

У третьому розділі було виконано повну програмну реалізацію, верифікацію та комплексну оцінку якості розробленого вебзастосунку інтернет-магазину автозапчастин.

Під час аналізу особливостей реалізації підтверджено ефективність клієнтської частини на базі компонентної моделі React та серверної логіки на Express і TypeScript, яка ізолює роботу з базою даних через шар репозиторіїв. Опис користувацького інтерфейсу та сценаріїв роботи за допомогою реальних скриншотів системи продемонстрував успішне виконання наскрізного процесу: від пошуку деталі в каталозі з використанням гнучких фільтрів до транзакційного створення замовлення та його менеджменту в адмін-панелі.

Етапи верифікації та функціонального тестування за допомогою сформованих сценаріїв (ТС-01–ТС-09) підтвердили повну відповідність системи бізнес-правилам та вимогам безпеки як для позитивних, так і для негативних запитів (обробка помилок авторизації, блокування доступу, контроль недостатньої кількості товару на складі). Аналіз показників супроводжуваності, безпеки й UX, разом із виявленням потенційних ризиків, підтвердив інженерну зрілість розробленого прототипу та його готовність до розгортання та подальшого масштабування.

ВИСНОВКИ

У кваліфікаційній роботі на основі проведених досліджень вирішено актуальну науково-практичну задачу, що полягає у проєктуванні та розробці сучасного масштабованого вебзастосунку для автоматизації процесів торгівлі та підбору автомобільних запчастин.

Під час виконання аналізу предметної області та формування вимог було детально досліджено специфіку та складність процесів ідентифікації автомобільних компонентів. На основі проведеного порівняльного аналізу універсальних маркетплейсів, спеціалізованих каталогів та локальних систем торгівлі визначено ключові проблеми користувачів, серед яких найкритичнішими є складність самостійного підбору та відсутність прозорої перевірки сумісності деталей. Це дозволило обґрунтувати доцільність створення інтегрованого інтернет-магазину та сформувати технічне завдання, яке враховує виявлені галузеві особливості.

На етапі проєктування архітектури та інформаційного забезпечення системи було побудовано гнучку клієнт-серверну структуру вебзастосунку. У роботі теоретично та практично обґрунтовано вибір сучасного технологічного стеку, що включає компонентну модель React з інструментарієм Vite для фронтенду, середовище Node.js та Express для бекенду, мову TypeScript для забезпечення статичної типізації, СКБД PostgreSQL для збереження реляційних даних, а також Docker Compose для контейнеризації. В межах цього етапу розроблено інформаційне забезпечення, деталізовано прецеденти використання за допомогою діаграм Use Case та формалізовано алгоритми функціонування ключових бізнес-процесів, зокрема пагінації, кошика та транзакційного оформлення замовлень.

У процесі програмної реалізації та верифікації спроєктовані інженерні рішення було успішно втілено у готовий програмний продукт із розділенням на чіткі архітектурні шари, такі як маршрути, сервіси, репозиторії та middleware. Для забезпечення високого рівня надійності впроваджено

комплексні механізми безпеки на основі JWT, bcrypt, HTTP-заголовків Helmet та валідації вхідних даних Zod, а також проведено оптимізацію продуктивності на рівні бази даних через створення індексів. Розробка та виконання серії сценарних тестів дозволили верифікувати працездатність системи як за позитивних, так і за негативних сценаріїв, включаючи обробку конфліктів залишків та обмеження прав доступу, а проведений аналіз технічних ризиків дозволив визначити дієві методи їх мінімізації.

Головним практичним результатом роботи є створення повністю функціонального та працездатного прототипу інтернет-магазину автозапчастин. Система успішно забезпечує наскрізний шлях клієнта, починаючи від інтелектуального пошуку й багатокритеріальної фільтрації деталей за сумісністю і завершуючи формуванням замовлення з автоматичним перерахунком залишків на складі та його оперативною модерацією через адміністративну панель.

Поставлену в кваліфікаційній роботі мету повністю досягнуто. Розроблена архітектура є гнучкою, безпечною та супроводжуваною, що дозволяє використовувати її як готову інженерну основу для автоматизації підприємств малого та середнього бізнесу в сфері автоторгівлі з перспективою подальшого впровадження модулів VIN-підбору, онлайн-еквайрингу через платіжні шлюзи, автоматизованого парсингу прайс-листів постачальників та CRM-аналітики.

ПЕРЕЛІК ПОСИЛАНЬ

1. Describing the UI. *React Documentation*. URL: https://react.dev/learn/describing-the-ui (дата звернення: 04.03.2026).

2. Managing State. *React Documentation*. URL: https://react.dev/learn/managing-state (дата звернення: 09.03.2026).

3. Getting Started. *Vite Documentation*. URL: https://vite.dev/guide/ (дата звернення: 12.03.2026).

4. Env Variables and Modes. *Vite Documentation*. URL: https://vite.dev/guide/env-and-mode (дата звернення: 15.03.2026).

5. About this documentation. *Node.js Documentation*. URL: https://nodejs.org/api/documentation.html (дата звернення: 19.03.2026).

6. HTTP. *Node.js Documentation*. URL: https://nodejs.org/api/http.html (дата звернення: 22.03.2026).

7. Routing. *Express Documentation*. URL: https://expressjs.com/en/guide/routing.html (дата звернення: 25.03.2026).

8. Using middleware. *Express Documentation*. URL: https://expressjs.com/en/guide/using-middleware.html (дата звернення: 28.03.2026).

9. The SQL Language. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/current/tutorial-sql.html (дата звернення: 01.04.2026).

10. Foreign Keys. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/current/tutorial-fk.html (дата звернення: 03.04.2026).

11. JSON Types. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/current/datatype-json.html (дата звернення: 05.04.2026).

12. Dockerfile reference. *Docker Documentation*. URL: https://docs.docker.com/reference/dockerfile/ (дата звернення: 08.04.2026).

13. Compose file reference. *Docker Documentation*. URL: https://docs.docker.com/reference/compose-file/ (дата звернення: 11.04.2026).

14. Introduction. *Zod Documentation*. URL: https://zod.dev/ (дата звернення: 14.04.2026).

15. Defining schemas. *Zod Documentation*. URL: https://zod.dev/api (дата звернення: 16.04.2026).

16. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT). *IETF*. 2015. URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата звернення: 18.04.2026).

17. Overview of HTTP. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Overview (дата звернення: 20.04.2026).

18. HTTP request methods. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Methods (дата звернення: 22.04.2026).

19. Testing. *MDN Web Docs*. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Testing (дата звернення: 24.04.2026).

20. Ecommerce User Experience. *Nielsen Norman Group*. URL: https://www.nngroup.com/reports/ecommerce-user-experience/ (дата звернення: 27.04.2026).

21. OWASP Web Security Testing Guide. *OWASP Foundation*. URL: https://owasp.org/www-project-web-security-testing-guide/ (дата звернення: 29.04.2026).