

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Веб-технології»)

на тему:

«Вебзастосунок для продажу автотоварів»

Виконав студент групи ВТ-21-1
ОМЕЛЬЧУК Вадим Вікторович

Керівник роботи:
ДУНЄВ Сергій Сергійович

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«14» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ОМЕЛЬЧУК Вадим Вікторович**

Керівник роботи: **ДУНЄВ Сергій Сергійович**

Тема роботи: **«Вебзастосунок для продажу автотоварів»,**

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61с**

Вихідні дані для роботи: **Об'єктом є процес розробки та функціонування веб-додатку для електронної комерції, зокрема в галузі продажу автотоварів.**

Предмет дослідження є процеси проектування, розробки та реалізації вебзастосунку для продажу автотоварів.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Дунєв С.С.	17.02.2025р.	17.02.2025р.
2	Дунєв С.С.	19.02.2025р.	19.02.2025р.
3	Дунєв С.С.	21.02.2025р.	21.02.2025р.

РЕФЕРАТ

Випускна робота бакалавра представляє собою «Вебзастосунок для продажу автотоварів».

Пояснювальна записка до кваліфікованої роботи складається з переліку умовних скорочень, вступу, трьох розділів, висновків, списку використаної літератури та додатку.

Текстова частина викладена на 59 сторінках друкованого тексту. Пояснювальна записка має 3 сторінки додатків. Список використаних джерел містить 34 найменування і займає 4 сторінки. В роботі наведено 25 рисунків та 9 таблиць. Загальний обсяг роботи - 68 сторінок.

В роботі визначені головні завдання для розробки веб-додатку, проаналізовано аналоги продукту та методи розв'язання поставленої задачі, обґрунтовано вибір архітектури системи. Представлено алгоритм роботи програмного комплексу, структуру та взаємодію його компонентів та описана програмна реалізація системи.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, REACT, NEXTJS, API, TYPESCRIPT.

					ІІЗ.КР.Б-121-25-ІІЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Вебзастосунок для продажу автотоварів Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		В.ОМЕЛЬЧУК						
Керівник		С.ДУНЄВ					3	
н. контр.						Житомирська політехніка, група ВТ-21-1		
Зав. каф.								

ABSTRACT

The bachelor's thesis is a "Web application for selling automotive products".

The explanatory note to the qualified work consists of a list of conventional abbreviations, an introduction, three sections, conclusions, a list of used literature and an appendix.

The text part is presented on 59 pages of printed text. The explanatory note has 3 pages of appendices. The list of sources used contains 37 items and occupies 4 pages. The work contains 25 figures and 9 tables. The total volume of the work is 68 pages.

The paper identifies the main tasks for developing a web application, analyzes product analogues and methods for solving the task, justifies the choice of system architecture. The algorithm of the software complex, the structure and interaction of its components are presented, and the software implementation of the system is described.

KEYWORDS: WEB APPLICATION, REACT, NEXTJS, API, TYPESCRIPT

					ИП3.КР.Б-121-25-ИЗ	
						4

РЕФЕРАТ	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ ПРОДАЖУ АВТОТОВАРІВ	10
1.1 Постановка задачі	10
1.2 Аналіз аналогів програмного продукту	11
1.3 Вибір архітектури застосунку	15
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	17
Висновок до першого розділу	20
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБЗАСТОСУНКУ	22
2.1. Визначення варіантів використання і структури застосунку	22
2.2 Проектування бази даних системи	25
2.3. Проектування та реалізація алгоритмів роботи системи	29
2.4 Реалізація застосунку веб-додатку продажу автотоварів	38
Висновок до другого розділу	44
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ЗАСТОСУНКОМ.....	45
3.1. Порядок встановлення та налаштування параметрів системи.	45
3.2 Структура інтерфейсу вебзастосунку.	46
3.3 Тестування роботи програмного комплексу	55
Висновок до третього розділу	58
ВИСНОВКИ	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	60

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- TS – TypeScript
- API – прикладний програмний інтерфейс
- БД – база даних
- ПЗ – програмне забезпечення
- UI – інтерфейс користувача

ВСТУП

Актуальність теми. У сучасних умовах стрімкого розвитку електронної комерції дедалі більше споживачів надають перевагу онлайн-покупкам, зокрема в сфері автомобільних товарів. Власники автомобілів стикаються з необхідністю постійного обслуговування транспортних засобів, що вимагає своєчасного придбання запчастин, технічних рідин, аксесуарів тощо. Однак у традиційних магазинах вибір часто обмежений, а інформація про сумісність товару з конкретним автомобілем - неповна або відсутня. За даними аналітики ринку e-commerce, понад 60% автолюбителів зазначають складнощі з пошуком потрібних деталей для своєї марки авто через брак структурованої інформації. Цифровізація торгівлі дає змогу значно оптимізувати процес підбору автотоварів. Завдяки розвитку веб-технологій, зокрема JavaScript-фреймворків та хмарних баз даних, з'являється можливість створення зручного онлайн-сервісу з гнучкою системою фільтрації за параметрами автомобіля (марка, рік, тип двигуна) та категоріями товарів (моторні оливи, фільтри, кузовні елементи, електроніка тощо). Це дозволяє користувачам швидко знайти сумісні товари, зменшуючи ризик покупки невідповідної запчастини.

Мета роботи — створення веб-додатку для зручного підбору та продажу автотоварів із врахуванням параметрів транспортного засобу, що дозволить автоматизувати процес пошуку та покращити якість обслуговування клієнтів.

Завданням випускної кваліфікаційної роботи бакалавра :

1. Провести аналіз існуючих рішень у сфері онлайн-продажу автотоварів, дослідити їхні функціональні особливості та виявити недоліки.
2. Обґрунтувати вибір технологій для реалізації веб-додатку, зокрема Next.js , Prisma, Postgres, Tailwind CSS [6].
3. Розробити архітектуру веб-застосунку з чіткою структурою бази даних, маршрутизацією, системою фільтрації та категоризації товарів.
4. Реалізувати функціонал реєстрації, авторизації, адміністративну панель для керування товарами та замовленнями.

5. Провести тестування основних сценаріїв роботи веб-додатку, оцінити зручність інтерфейсу та надійність роботи з базою даних.

Об’єкт дослідження — процес онлайн-торгівлі автотоварами з використанням сучасних веб-технологій.

Предмет дослідження — методи та засоби розробки веб-застосунку для структурованого пошуку, управління та реалізації автомобільних товарів.

Методи дослідження - системний аналіз, функціональне моделювання, проектування клієнт-серверної архітектури, об’єктно-орієнтоване програмування, модульне тестування.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ВЕБЗАСТОСУНКУ ПРОДАЖУ АВТОТОВАРІВ

1.1 Постановка задачі

У випускній кваліфікаційній роботі необхідно розробити веб-додаток для продажу автотоварів, який допоможе користувачам швидко знаходити та замовляти необхідні запчастини та аксесуари відповідно до марки, моделі, року випуску та типу двигуна їхнього автомобіля.

Система повинна складатися з двох основних частин: публічної клієнтської частини для покупців та адміністративної панелі для управління каталогом товарів, замовленнями та користувачами.

Розробка застосунку починається з проектування його архітектури. На цьому етапі виконується аналіз функціональних вимог, обґрунтовується вибір технологічного стеку, що включає Next.js [1] для створення фронтенду та API [4], Prisma [2] як ORM-рішення для взаємодії з базою даних, а також PostgreSQL [3] для надійного зберігання структурованої інформації про товари, категорії, замовлення та користувачів.

Користувацький інтерфейс розробляється з урахуванням принципів зручності та доступності.

Передбачено реалізацію гнучкої системи фільтрації товарів за технічними характеристиками автомобіля, зручну навігацію, перегляд описів, фотографій, ціни, а також можливість додавання товарів у кошик та оформлення замовлення. Серверна частина реалізується з використанням Next.js API routes, що дозволяє створити RESTful-інтерфейс для взаємодії з клієнтською частиною. Аутентифікація користувачів забезпечується через NextAuth.js, з можливістю авторизації як звичайних користувачів, так і адміністраторів. ORM

Prisma відповідає за безпечну та ефективну взаємодію з базою даних PostgreSQL, дозволяючи швидко реалізувати CRUD-операції.

Адміністративна панель надає інтерфейс для керування товарами, категоріями, перегляду замовлень та взаємодії з користувачами.

					ІПЗ.КР.Б-121-25-ПЗ	
						10

Завершальним етапом розробки є тестування застосунку, що включає перевірку основного функціоналу, тестування API, перевірку цілісності бази даних та загальної продуктивності веб-додатку у реальних умовах експлуатації. Також передбачено розгортання на хмарному сервері (наприклад, Vercel або Render) та налаштування автоматичного деплою.

Результатом виконання роботи стане повноцінний веб-застосунок, який дозволить користувачам легко та швидко знаходити автотовари, що підходять до їхнього автомобіля, а адміністраторам — ефективно управляти онлайн-продажами. Такий ресурс сприятиме підвищенню зручності покупок у сфері автомобільної торгівлі та цифровізації автосервісу загалом.

1.2 Аналіз аналогів програмного продукту

На розгляді наявні аналоги, які найбільше подібні до теми кваліфікаційної роботи: DOK.UA, forumauto.parts та ATL.ua. Наведемо їх можливості, переваги та недоліки, а також порівняємо з нашим майбутнім застосунком.

DOK.UA. Онлайн-сервіс Dok.ua є одним із провідних рішень на українському ринку автотоварів. Його основною перевагою є широкий асортимент запчастин, охоплюючий понад 12 мільйонів позицій, що дозволяє задовольнити потреби власників більш ніж 230 моделей автомобілів. Завдяки розгалуженій системі фільтрації користувач може зручно підібрати необхідні товари, вказавши марку, модель, рік випуску та тип двигуна. Додаткову зручність забезпечує функція «Гараж», яка дозволяє зберігати дані про автомобіль і використовувати їх для автоматичного підбору товарів.

Суттєвою технологічною перевагою є підтримка пошуку за VIN-кодом, що підвищує точність у виборі деталей і мінімізує ризик помилок. Крім того, сервіс має мобільний додаток, який забезпечує повноцінний доступ до основного функціоналу, включаючи історію замовлень, швидке оформлення покупок та персоналізовані пропозиції. Користувачі також можуть звертатися до фахівців для уточнення сумісності товарів, що додає впевненості під час покупки.

Втім, сервіс не позбавлений недоліків. Незважаючи на можливість використання VIN-коду, помилки при підборі все ж трапляються. Це особливо

					ІПЗ.КР.Б-121-25-ПЗ	
						11

стосується складних деталей, таких як пружини підвіски, де навіть незначне відхилення у специфікації може призвести до неправильного вибору.

Деякі товари не представлені у відкритому каталозі, і їхнє замовлення можливе лише після консультації з менеджером, що уповільнює процес оформлення покупки. Крім того, не всі користувачі здатні самостійно підібрати товар без допомоги консультанта, що створює певну залежність від доступності служби підтримки, особливо у періоди високого навантаження.

У підсумку, Dok.ua — це потужна платформа з великою базою даних, зручними інструментами для пошуку, мобільною підтримкою та експертною допомогою. Водночас, вона вимагає від користувача певної обережності та готовності до взаємодії з технічними фахівцями для досягнення максимально точного результату при виборі запчастин.

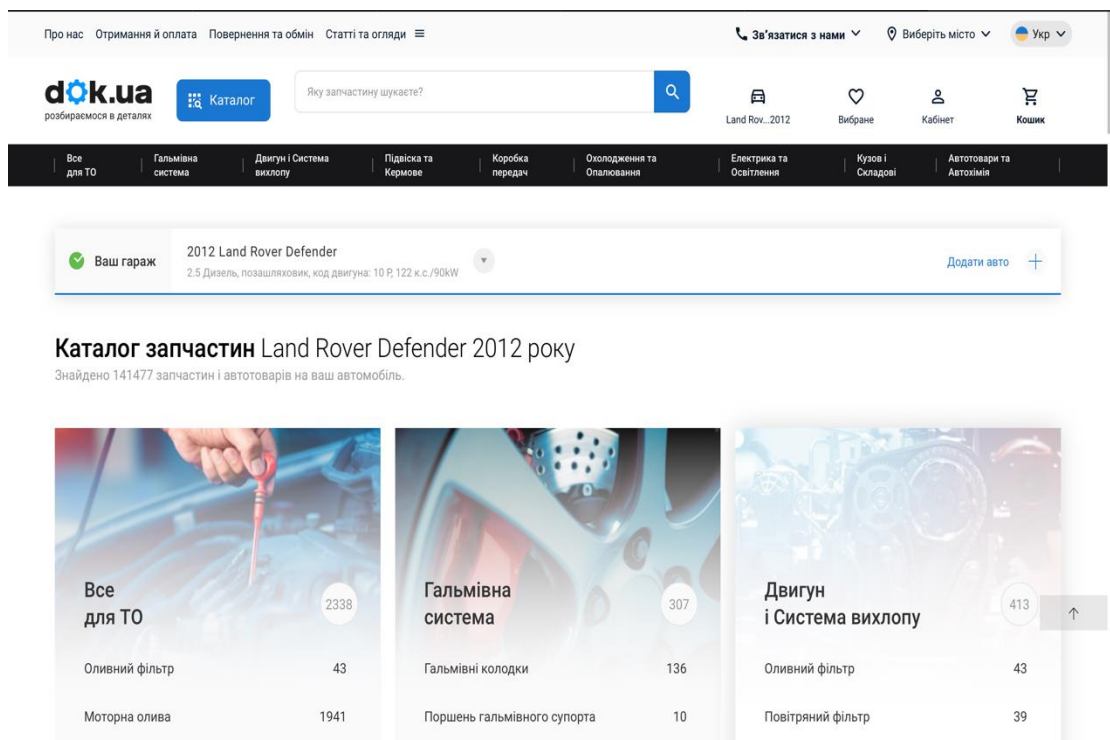


Рис. 1.1 – Головна сторінка dok.ua

Forumauto.parts. Сайт forumauto.parts - це функціональний веб-сервіс для продажу автозапчастин, побудований на платформі PARTS SOFT. Він підтримує пошук товарів за VIN-кодом, артикулом, назвою та параметрами авто, має особистий кабінет, історію замовлень і базову інтеграцію з українськими службами доставки.

					ІПЗ.КР.Б-121-25-ІПЗ	
						12

Технічно сайт реалізовано на класичному стеку з використанням PHP та реляційної СУБД, мовірно MySQL. Робота з базою даних і API постачальників організована стабільно, оновлення наявності та цін відбувається у реальному часі.

Однак інтерфейс виглядає застарілим, із мінімальною мобільною адаптацією та відсутністю сучасних UX-рішень, таких як швидке оновлення кошика чи автопідказки. Сайт виконує свої функції, але візуально і технічно потребує оновлення відповідно до сучасних стандартів онлайн-торгівлі.

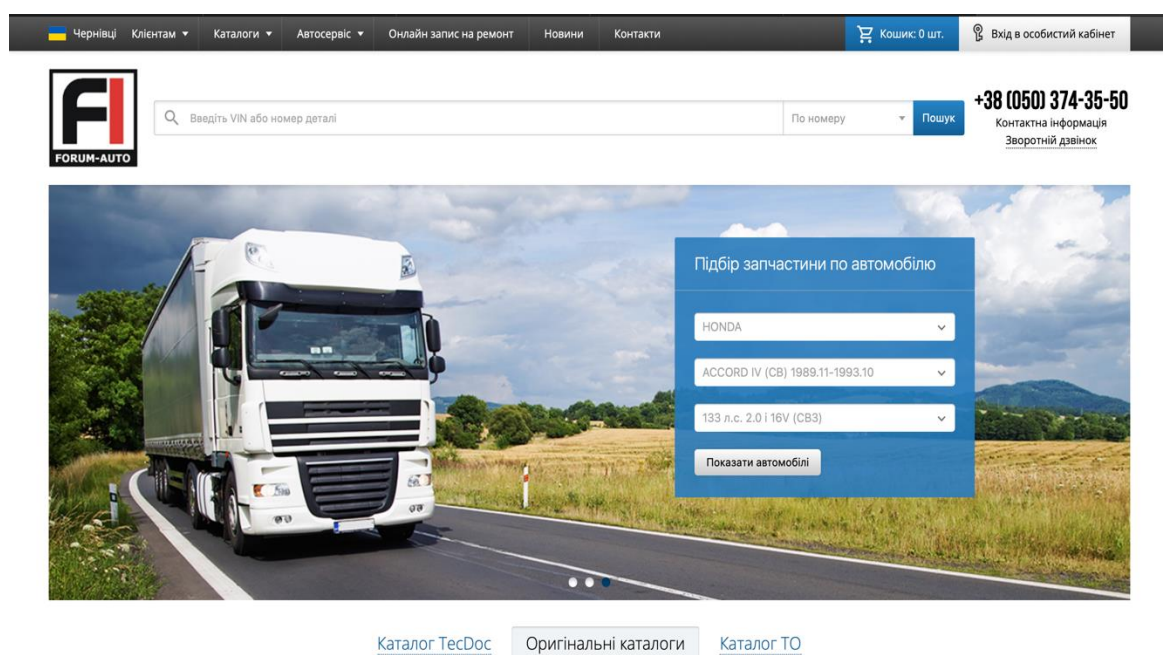


Рис. 1.2 – Головна сторінка forumauto.parts

ATL.ua. Сайт ATL.ua - функціональний онлайн-магазин автотоварів із пошуком за VIN-кодом, артикулом і параметрами авто. Підтримує особистий кабінет, історію замовлень, фільтрацію товарів і доставку по Україні.

Технічно платформа стабільна, швидко оновлює ціни та залишки через API постачальників. Проте інтерфейс виглядає застарілим, має слабку мобільну адаптацію та мінімум сучасних UX-елементів.

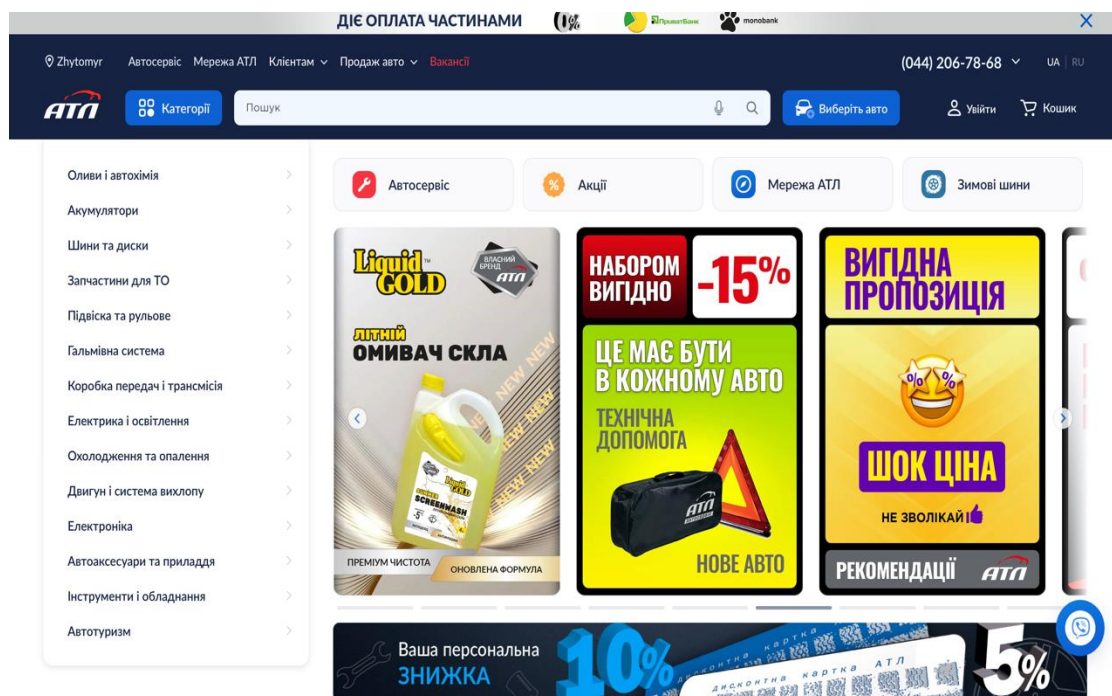


Рис. 1.3 – Головна сторінка atl.ua

Таблиця 1.1

Порівняльна таблиця наявних рішень на ринку

Критерій	Dok.ua	Forumauto.parts	Atl.ua
Фільтрація	Зручна	Базова	Детальна
Особистий кабінет	Повнофункціональний	Базовий	Є
Доставка	Тільки доставки поштою	По Україні та самовивіз	Тільки доставки поштою
Мобільна адаптація	Мобільний додаток	Базова	Слабка
Дизайн	Сучасний	Застарілий	Застарілий
Оновлення даних	Швидке	Повільне	Повільне
Загальна оцінка	++++	++	++

Сайт Dok.ua вирізняється сучасним і зручним інтерфейсом, продуманою фільтрацією та функціональним особистим кабінетом, що робить користування комфортним. Підтримка мобільного додатку та швидке оновлення даних забезпечують актуальність інформації і зручність покупок. Це робить Dok.ua найбільш технологічно просунутим серед трьох платформ.

Forumauto.parts та ATL.ua пропонують більш базовий функціонал із застарілим дизайном і слабкою мобільною адаптацією. Оновлення інформації на цих сайтах відбувається повільніше, що може створювати незручності для користувачів.

В цілому, Dok.ua краще підходить для тих, хто цінує сучасний і швидкий сервіс, тоді як Forumauto.parts і ATL.ua більше орієнтовані на стабільність і базові можливості без додаткових інновацій.

1.3 Вибір архітектури застосунку

Розробка веб-додатку для продажу автотоварів: архітектурні рішення та обґрунтування вибору технологій

Розробка сучасного веб-додатку для продажу автотоварів вимагає ретельного планування архітектури, яка повинна відповідати вимогам продуктивності, масштабованості, безпеки, надійності та забезпечувати зручність як для кінцевого користувача, так і для команди підтримки й розробників. В умовах динамічного розвитку електронної комерції критично важливо забезпечити гнучку платформу, здатну ефективно обробляти великі обсяги запитів, даних та інтегруватися з різноманітними зовнішніми сервісами.

З цією метою було обрано клієнт-серверну архітектуру з чітким розділенням фронтенд- та бекенд-частин, що дозволяє масштабувати компоненти незалежно один від одного, покращувати продуктивність системи й спрощувати розробку нових функцій.

Клієнтська частина реалізована з використанням Next.js — потужного фреймворку на базі React, який поєднує переваги серверного рендерингу (SSR) і статичної генерації сторінок (SSG). SSR забезпечує швидке завантаження першого екрану, що є критично важливим для користувачів із повільним

					ІПЗ.КР.Б-121-25-ПЗ	
						15

з'єднанням або при першому вході на сайт. SSG, у свою чергу, зменшує навантаження на сервер завдяки попередній генерації часто використовуваних сторінок, таких як головна, сторінки категорій чи окремих товарів.

Завдяки React [5] можливо легко будувати повторно використовувані UI-компоненти та реалізовувати сучасний підхід до керування станом застосунку. У проєкті можуть бути використані бібліотеки, React Context API або інші рішення залежно від складності бізнес-логіки та кількості взаємодіючих компонентів.

Фронтенд також оптимізований під мобільні пристрої (адаптивна верстка) і має швидкий інтерфейс за рахунок застосування динамічного імпорту компонентів, lazy loading, індексації пошуковими системами завдяки SSR, а також зручну навігацію через маршрутизацію, реалізовану в Next.js.

Серверна логіка побудована на базі Prisma ORM у поєднанні з PostgreSQL — потужною реляційною базою даних, що підтримує складні реляційні зв'язки, транзакції та високий рівень надійності. Prisma забезпечує простий і безпечний доступ до бази даних завдяки системі типів, автоматичній генерації моделей, міграцій та автозаповнення в IDE.

PostgreSQL дозволяє ефективно працювати з великими обсягами структурованих даних, реалізовувати фільтрацію за кількома параметрами, підтримує індексацію, збережені процедури, представлення (views), а також розширення (наприклад, pg_trgm для повнотекстового пошуку або PostGIS у разі потреби геолокаційних даних).

Серед особливостей реалізації слід зазначити ретельне моделювання даних, з урахуванням ієрархічної структури категорій, атрибутів товарів, їх варіантів, сумісності з певними моделями автомобілів (марка, рік, тип двигуна тощо). Це дозволяє гнучко організувати пошук, фільтрацію та сортування товарів у каталозі, забезпечуючи швидкий відгук системи навіть при великій кількості записів.

Обмін даними між клієнтом і сервером організований через REST API або GraphQL (залежно від потреб проєкту). REST є зручним для стандартних CRUD-

операцій, тоді як GraphQL забезпечує гнучкіший механізм запитів, особливо коли потрібно зменшити кількість запитів і передавати лише необхідні поля.

Для захисту даних користувачів та безпеки операцій впроваджено механізм аутентифікації та авторизації на базі JWT (JSON Web Tokens). Такий підхід дозволяє реалізувати безпечну систему сесій без потреби постійно зберігати дані на сервері, що позитивно впливає на масштабованість і продуктивність додатку. Додатково застосовуються такі практики, як шифрування паролів за допомогою bcrypt, захист від CSRF, обмеження частоти запитів (rate limiting) та перевірка прав доступу до приватних ресурсів.

Оскільки веб-додаток повинен обслуговувати велику кількість користувачів і постійно зростати, було враховано потенціал горизонтального масштабування. Наприклад, фронтенд може бути розгорнутий через CDN (наприклад, Vercel або Netlify), а серверна частина — у вигляді контейнерів Docker, що спрощує розгортання в хмарних середовищах (AWS, Google Cloud, DigitalOcean). Для збереження стану системи й логування використовуються сервіси моніторингу.

Вибрана архітектура дозволяє створити масштабований, продуктивний і безпечний веб-додаток для продажу автотоварів, який відповідає сучасним вимогам електронної комерції. Вона забезпечує гнучкість у розробці, легкість масштабування, а також високий рівень користувацького досвіду завдяки швидкому інтерфейсу та точному підбору товарів через гнучку систему фільтрації. Ці рішення створюють основу для довготривалої підтримки, подальшого розвитку системи та інтеграції з новими технологіями й сервісами.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для розробки веб-додатку з продажу автотоварів було обрано сучасний і збалансований стек технологій, який відповідає вимогам продуктивності, гнучкості, безпеки, а також забезпечує зручність підтримки й подальшого масштабування системи. Такий підхід дозволяє ефективно реалізувати

повноцінну платформу для електронної комерції, що адаптується до змін ринку та очікувань користувачів.

Основою клієнтської частини став Next.js — прогресивний фреймворк, що базується на React. Його ключова перевага — підтримка серверного рендерингу (SSR) та статичної генерації сторінок (SSG), що дозволяє досягти високої швидкості завантаження інтерфейсу, що є критично важливим для покращення досвіду користувача, а також значно підвищує ефективність індексації сайту пошуковими системами [23].

SSR дає змогу формувати сторінки на сервері перед відправкою клієнту, що забезпечує швидкий доступ до вмісту навіть за повільного інтернет-з'єднання. SSG, у свою чергу, ідеально підходить для сторінок, які не змінюються часто — наприклад, опис товару, сторінки брендів, категорій тощо. Таким чином, Next.js забезпечує оптимальне поєднання динамічності й продуктивності.

Завдяки React та архітектурі компонентів стало можливим створення інтерактивних UI-елементів, які легко масштабуються та повторно використовуються. Крім того, Next.js дозволяє налаштовувати маршрутизацію, lazy loading та обробку запитів, що позитивно впливає на загальну продуктивність застосунку.

Для взаємодії з базою даних обрано Prisma ORM, яка забезпечує типізований, безпечний і зручний доступ до даних. Завдяки Prisma значно спрощується створення, зміна та підтримка структури бази даних. Інструмент дозволяє легко реалізовувати складні реляційні запити, а також автоматизує міграції, що зменшує ймовірність людських помилок під час оновлення схеми. У парі з Prisma використовується PostgreSQL — потужна реляційна система управління базами даних, яка давно зарекомендувала себе у високонавантажених проєктах. Вона підтримує складні SQL-запити, транзакції, індексацію, JSON-типи, розширення (таких як pg_trgm для пошуку) і є стійкою до великих обсягів даних. Це дозволяє ефективно опрацьовувати великі каталоги автотоварів з

численними характеристиками, зв'язками та атрибутами, як-от марка, модель, рік випуску, тип двигуна тощо.

Серверна логіка реалізована на Node.js — платформі, що дозволяє виконувати JavaScript-код на сервері та має чудову інтеграцію з Prisma і Next.js. Для побудови API застосовуються легкі та продуктивні фреймворки (наприклад, Express або Fastify), які забезпечують швидку обробку HTTP-запитів, просте підключення middleware, логування, обробку помилок та зручне структурування коду.

Між клієнтською частиною і сервером реалізовано інтерфейс обміну даними через REST або GraphQL, що дозволяє забезпечити гнучку, типізовану та незалежну логіку доступу до ресурсів, оптимізувати кількість запитів та покращити зручність інтеграції зі сторонніми сервісами — платіжними шлюзами, логістичними API, аналітичними платформами тощо.

Для забезпечення безпеки користувачів впроваджено систему аутентифікації та авторизації на основі JWT (JSON Web Tokens). Цей механізм дозволяє підтримувати безпечну роботу сесій без необхідності зберігати дані на сервері, а також забезпечує масштабованість при зростанні кількості користувачів і запитів.

Під час розробки використовуються популярні середовища розробки (зокрема, Visual Studio Code), які підтримують інтеграцію з Git, дебагінг, автодоповнення, перевірку синтаксису тощо. Код контролюється через систему керування версіями Git, що дозволяє працювати в команді, відслідковувати зміни та створювати надійні гілки розробки [7].

Для забезпечення безперервної інтеграції та доставки змін (CI/CD) використовуються сервіси на зразок GitHub Actions, GitLab CI або Jenkins, які дозволяють автоматизувати тестування, збірку та деплой нових версій проєкту [27]. Це значно підвищує стабільність продукту, зменшує людський фактор і пришвидшує реліз оновлень.

Вимоги до локального середовища розробки не є надмірно високими. Проте для комфортної роботи рекомендується використовувати персональні

					ІІЗ.КР.Б-121-25-ІІЗ	
						19

комп'ютери або ноутбуки з процесором не нижче Intel Core i5 (або аналог AMD Ryzen 5), не менше 8 ГБ оперативної пам'яті та SSD-накопичувачем, який забезпечує швидке зчитування і запис даних, особливо при роботі з базою та збірці фронтенду.

Що стосується серверної інфраструктури, рекомендується використовувати хмарні платформи (AWS, Google Cloud, Azure) або VPS з підтримкою масштабування, автоматичними бекапами, захищеними середовищами виконання. Базовими параметрами серверу мають бути щонайменше 2-4 ядра CPU, 4 ГБ оперативної пам'яті та швидкий SSD-диск, особливо при розгортанні бази даних і API.

Таким чином, обраний стек технологій та інфраструктурні рішення утворюють надійне підґрунтя для створення сучасного веб-додатку у сфері електронної комерції. Вони забезпечують баланс між високою продуктивністю, безпекою, гнучкістю і вартістю підтримки, що робить проєкт ефективним, стабільним і перспективним для масштабування та розвитку в довгостроковій перспективі.

Висновок до першого розділу

У першому розділі було підкреслено актуальність створення веб-додатку для продажу автотоварів у зв'язку зі зростанням потреби користувачів у зручних і швидких цифрових сервісах для вибору та придбання автомобільних комплектуючих. Окреслено основні цілі проєкту — розробка інтуїтивно зрозумілого інтерфейсу, реалізація ефективного пошуку та фільтрації товарів, а також створення надійної системи управління замовленнями і користувачами.

Проведений аналіз сучасних технологічних рішень дозволив обрати оптимальну архітектуру з використанням Next.js для фронтенду, Prisma ORM і PostgreSQL [9] для бекенду. Такий стек забезпечує високу продуктивність, масштабованість, зручність у розробці та безпеку обробки даних, що є важливими чинниками для реалізації складного каталогу автотоварів з великою кількістю параметрів.

Також визначено основні вимоги до інструментальних засобів і апаратного забезпечення, які дозволяють створити комфортні умови для розробки та підтримувати стабільну роботу додатку у продакшені. Загалом, сформована міцна теоретична та технічна база для подальшої розробки веб-додатку, який відповідатиме сучасним потребам ринку та очікуванням користувачів.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБЗАСТОСУНКУ

2.1. Визначення варіантів використання і структури застосунку

Розроблена система являє собою веб-додаток, що поєднує в собі взаємодію користувачів різних ролей: звичайних користувачів та адміністраторів. Для наглядного представлення взаємодії між компонентами системи було створено діаграму варіантів використання яка зображена на рисунку 2.1.

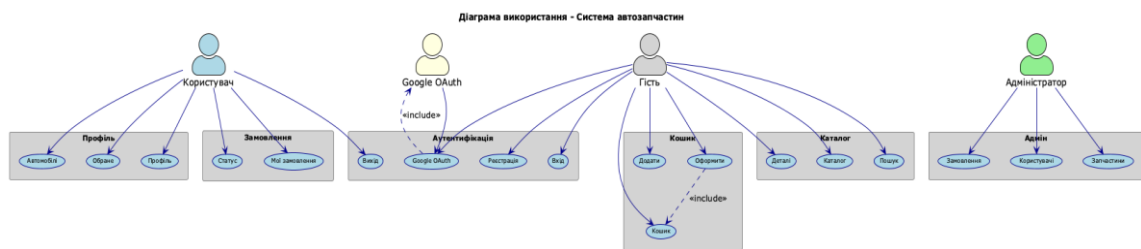


Рисунок 2.1 - Діаграма варіантів використання

Діаграма варіантів використання включає два основні актори:

Користувачі є основними учасниками роботи застосунку. Користувачі взаємодіють із системою для того, щоб отримати інформацію про товари.

Адміністратори є користувачами з розширеним правом доступом і відповідають за адміністрування застосунку, модерацію та підтримку якості даних.

Функціональні вимоги звичайних користувачів:

Звичайні користувачі веб-застосунку повинні мати змогу здійснювати повноцінну взаємодію з платформою в межах купівлі автотоварів. Це передбачає можливість переглядати каталог продукції, фільтрувати товари за технічними параметрами свого автомобіля (марка, модель, рік випуску, тип двигуна), шукати необхідні запчастини за ключовими словами чи категоріями, а також переглядати детальну інформацію про кожен товар. Користувачі мають змогу реєструватися, авторизовуватися в системі, додавати товари до кошика, оформлювати замовлення, відслідковувати статус доставки.

Вимоги адміністраторів:

Адміністративна частина застосунку повинна надати адміністраторам повний контроль над вмістом і процесами системи. Через захищену панель керування адміністратор може додавати, редагувати й видаляти товари, оновлювати ціни, керувати категоріями та технічними характеристиками продукції. Також передбачено управління користувачами: блокування облікових записів, надання або обмеження доступу до певного функціоналу, перегляд історії активності. Адміністратори мають доступ до списку замовлень, аналітики продажів, логістичних звітів і можливість вручну змінювати статус доставки. Крім того, реалізовано механізми для виявлення помилок, ведення логів і моніторингу продуктивності системи.

Бізнес-вимоги:

Бізнес-вимоги до веб-застосунку передбачають створення надійної, масштабованої та комерційно ефективної онлайн-платформи для продажу автотоварів. Застосунок має забезпечувати швидке обслуговування великої кількості користувачів, бути готовим до зростання кількості товарних позицій і постійного оновлення залишків. Платформа повинна підтримувати інтеграцію з зовнішніми сервісами доставки, платіжними системами та складськими обліковими програмами. Також важливо передбачити можливість впровадження маркетингових механізмів, таких як знижки, акції, промокоди, система бонусів і автоматичні розсилки. Сайт повинен бути адаптивним, SEO-оптимізованим і мати інтуїтивно зрозумілий інтерфейс для кінцевого споживача.

Функціональні вимоги:

Функціональна частина системи охоплює реалізацію усіх основних можливостей як для користувача, так і для адміністратора. Це включає управління товарами, замовленнями, профілями користувачів, а також фільтрацію й сортування продукції за заданими критеріями. Важливо, щоб структура даних дозволяла легко знаходити потрібну інформацію, а інтерфейс передбачав швидку взаємодію без зайвих переходів. Окрім цього, функціональні вимоги охоплюють інтеграцію з API служб доставки, платіжних систем і засобів відстеження замовлень. Додаткові можливості включають систему сповіщень,

					ІПЗ.КР.Б-121-25-ПЗ	
						23

керування відгуками, а також збереження обраних товарів і автоматичну генерацію рекомендованих позицій.

Нефункціональні вимоги:

До нефункціональних вимог належить забезпечення високої швидкодії, стабільності, безпеки та доступності системи. Застосунок має швидко реагувати на дії користувача, бути доступним 24/7 без суттєвих простоїв, мати зручну мобільну адаптацію та відповідати сучасним вимогам до UI/UX [21]. Усі дані повинні зберігатися з використанням захищених протоколів передачі (HTTPS), а доступ до облікових записів захищений токенами (JWT). Важливим є впровадження механізмів резервного копіювання бази даних, логування подій та масштабування інфраструктури у разі зростання навантаження. Додаток також повинен бути легко розширюваним без порушення поточної функціональності.

Архітектура системи:

Застосунок реалізується за принципом клієнт-серверної архітектури. Фронтенд відповідає за взаємодію з користувачем, візуалізацію інтерфейсу, надсилання запитів і обробку отриманих даних. Серверна частина обробляє логіку роботи із замовленнями, товарами, користувачами та зовнішніми сервісами, а також здійснює валідацію даних. Компоненти системи ізольовано за ролями, що дозволяє уникати конфліктів, а також полегшує підтримку і тестування. База даних PostgreSQL використовується для централізованого зберігання всієї інформації, а ORM Prisma [28] дозволяє гнучко працювати з нею.

Клієнтська архітектура:

Клієнтська частина реалізована з використанням фреймворку Next.js [26], який забезпечує як серверний рендеринг сторінок, так і динамічні інтерактивні компоненти на стороні браузера. Такий підхід покращує швидкодію додатку, особливо при першому завантаженні сторінок, і позитивно впливає на результати SEO. Компоненти побудовані на базі React, а для керування станом застосунку використовуються React Context або Redux Toolkit. Передбачена клієнтська маршрутизація, адаптивна верстка для всіх розмірів екранів та реалізація

приватних маршрутів для захисту персоналізованих сторінок. Комунікація з сервером здійснюється через REST [10] або GraphQL API.

2.2 Проектування бази даних системи

Для зберігання інформації в застосунку було обрано PostgreSQL [11], яка в співпраці з Prisma ORM [22] забезпечує ефективну та безпечну роботу з базою даних

База даних складається з 7 основних колекцій:

Таблиця 2.1.

Призначення колекцій бази даних

Назва колекції	Призначення
User	Зберігання профілів користувачів
UserCar	Зберігає задане авто користувача
Car	Авто для яких запчастини
Part	Запчастини до авто
FavoritePart	Містить обрані деталі користувачів
Order	Замовлення яке робить користувач
OrderItem	Зберігає конкретні товари в замовленні

Структуру бази даних представлено на рисунку 2.2, де відображено 7 основних колекцій документів з відповідно визначеними зв'язками. Вся система вдало продумана та оптимізована для безпечного зберігання всіх можливих даних, які з'являються в процесі взаємодії користувачів з застосунком та розподіленням на колекції відповідно до типу даних, що необхідно зберігати.

ER-діаграма бази даних - Система автозапчастин

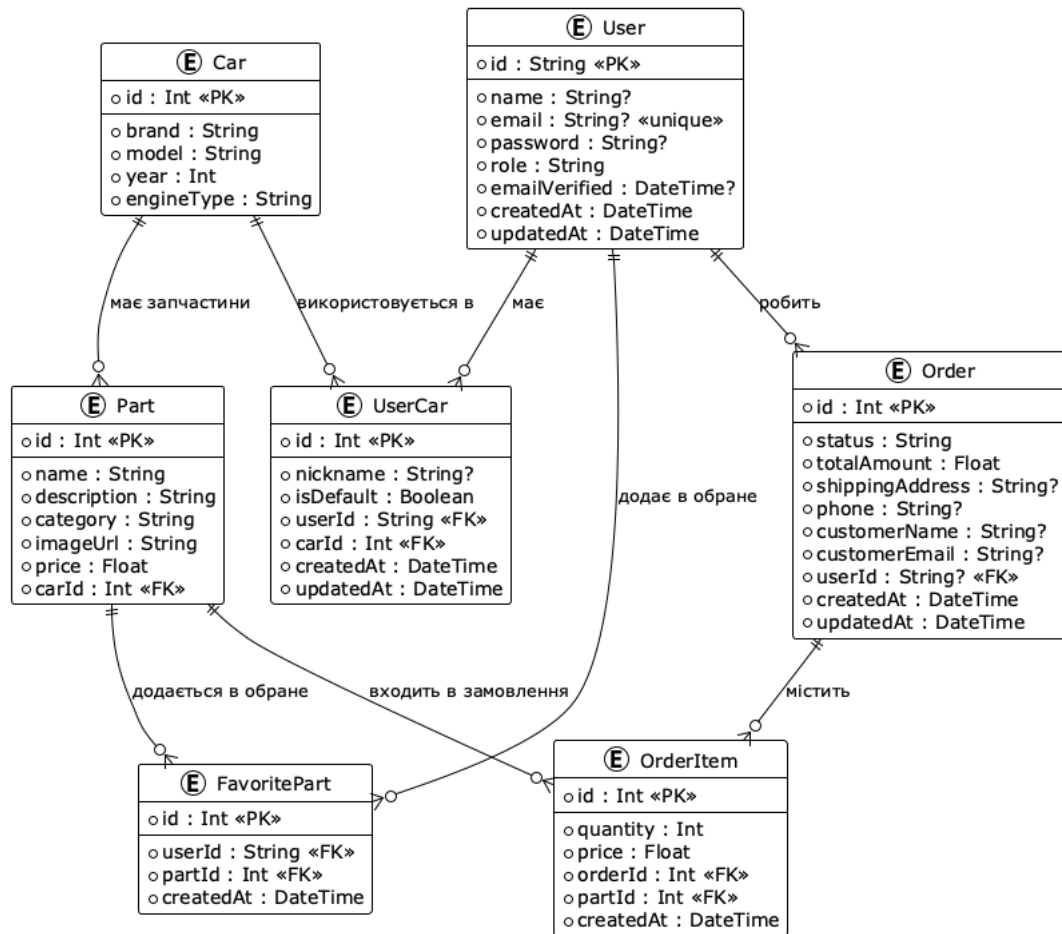


Рис. 2.2 - Діаграма бази даних

Таблиця 2.2

Колекція User

Поле	Тип	Призначення
id	string, CUID	Унікальний ідентифікатор
email	String?, unique	Електронна адреса
name	String?	Ім'я (опціонально)
password	string?	Пароль
role	string	Роль користувача
emailVerified	dateTime?	Дата підтвердження пошти
updatedAt	dateTime	Дата оновлення користувача
createdAt	dateTime	Дата створення користувача

Колекція Users відповідає за збереження даних про користувача, ролі та персональні налаштування.

Таблиця 2.3

Колекція Car

Поле	Тип	Призначення
id	int	Унікальний ідентифікатор
brand	string	Марка авто
model	string	Модель авто
year	int	Рік виробництва
engineType	string	Тип двигуна

Колекція Car зберігає в собі список всіх автомобілів з їх характеристики.

Таблиця 2.4

Колекція UserCar

Поле	Тип	Призначення
id	int PK	Унікальний ідентифікатор
nickname	string?	Особиста назва авто
isDefault	boolean	Перевірка чи є основна
userId	string	Посилання на користувача
carId	int	Посилання на авто
createdAt	dateTime	Коли додав авто користувач
updatedAt	dateTime	Коли змінено інформацію

Колекція UserCar для того щоб зареєстровані користувачі могли для зручності додавати свої авто.

Таблиця 2.5

Колекція Part

Поле	Тип	Призначення
id	int	Унікальний ідентифікатор
name	string	Назва запчастини
description	string	Опис
category	string	Тип запчастини
imageUrl	string	Зображення
price	float	Ціна
carId	Int FK	Для визначення деталі до авто

Колекція Part зберігає в собі всі запчастин для автомобіля.

Таблиця 2.6

Колекція Order

Поле	Тип	Призначення
id	int	Унікальний ідентифікатор
status	string	Статус замовлення
totalAmount	float	Загальна вартість
shippingAdress	string	Повна адреса доставки
phone	string	Телефон для зв'язку
customerName	string	Ім'я замовника
customerEmail	string	Пошта замовника
userId	string	Посилання на користувача
createdAt	dateTime	Створено статус замовлення
updatedAt	dateTime	Оновлено статус замовлення

Колекція Order для того щоб користувачі могли робити замовлення.

Таблиця 2.7

Колекція OrderItem

Поле	Тип	Призначення
id	string	Унікальний ідентифікатор
quantity	string	Кількість запчастин
price	string	Фіксована ціна на момент створення замовлення
orderId	string	Посилання на замовлення
partId	string	Посилання на запчастину
createdAt	dateTime	Коли додано елемент до замовлення

Колекція OrderItem відповідає за конкретні запчастини в замовлення, їх кількість ціну .

Таблиця 2.8

Колекція FavoritePart

Поле	Тип	Призначення
id	id	Унікальний ідентифікатор
userId	string	Посилання на користувача
partId	int	Посилання на запчастину
createdAt	dateTime	Час коли було додано

Колекція FavoritePart зберігає обрані товари тільки зареєстрованих користувачів.

2.3. Проєктування та реалізація алгоритмів роботи системи

У цьому розділі розглядається діаграма діяльності, які є невід'ємною для процесів розробки програмного забезпечення. Ці діаграми схожі на звичайні блок-схеми і активно використовуються при створенні застосунків на React Native.

UML діаграми діяльності дозволяють описувати конкретні варіанти використання мобільного застосунку, ілюструючи процеси в системі. Їх також можна використовувати для опису бізнес-процесів в аналізі харчування, що допомагає точно окреслити порядок подій і умови, необхідні для виконання подій.

Зображена діаграма (рис.2.3) діяльності реєстрації та авторизації користувача в застосунку.

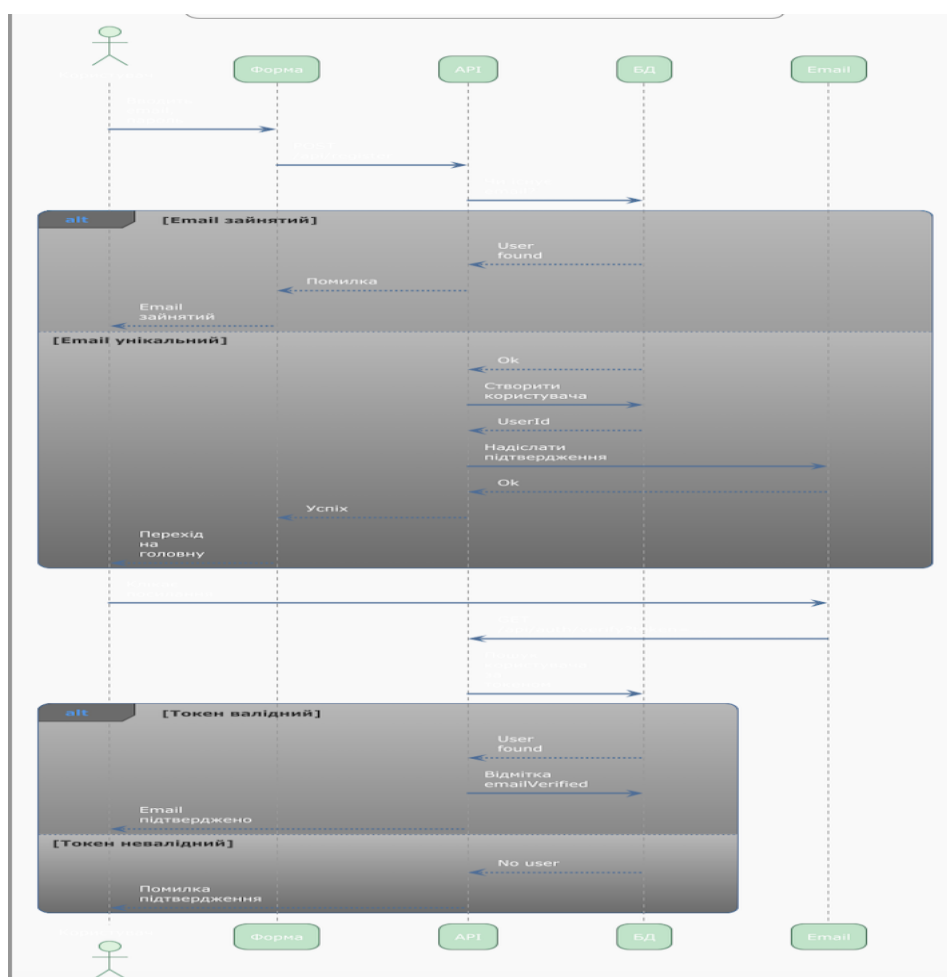


Рис. 2.3 - Діаграма діяльності реєстрації та авторизації користувача

Алгоритм аутентифікації та авторизації

Система аутентифікації реалізована на основі NextAuth.js [24] з інтеграцією PostgreSQL бази даних. Процес входу користувача починається з введення email адреси та пароля у відповідні поля форми. Система виконує пошук користувача в базі даних за унікальною email адресою. При знаходженні запису відбувається порівняння хешованого пароля з використанням bcrypt бібліотеки, яка забезпечує безпечне хешування з сіллю. У разі успішної перевірки створюється JWT токен через NextAuth.js, який містить інформацію про користувача та його роль. Сесія зберігається в cookies браузера для подальшої авторизації запитів. Після успішного входу система перенаправляє користувача на головну сторінку додатка.

Процес реєстрації нових користувачів включає валідацію email адреси на унікальність та відповідність стандартному формату. Пароль користувача хешується з використанням bcrypt з 12 раундами солі для забезпечення максимальної безпеки. Створюється новий запис в таблиці User з базовими даними користувача. Система відправляє email для підтвердження адреси, що дозволяє верифікувати справжність контактних даних. Після успішної реєстрації користувач автоматично входить в систему без необхідності повторного введення облікових даних.

Система авторизації базується на middleware, який перевіряє JWT токен на кожному запиті до захищених ресурсів. Користувачі з роллю "admin" отримують повний доступ до адміністративної панелі, включаючи управління товарами, замовленнями та користувачами. Користувачі з роллю "user" мають доступ до особистого кабінету, історії замовлень та функцій обрання товарів. Гості, які не авторизовані в системі, можуть переглядати каталог товарів та додавати товари в кошик, але для оформлення замовлення необхідно надати контактні дані.

Алгоритм роботи з кошиком

Функціональність кошика реалізована з використанням локального зберігання даних та React Context для управління станом. При додаванні товару в кошик система спочатку перевіряє наявність запису в каталозі через запит до бази даних. Після підтвердження існування товару створюється або

оновлюється запис в localStorage браузера, що дозволяє зберігати кошик між сесіями користувача. Структура даних кошика включає ідентифікатор запчастини, кількість, ціну, назву та посилання на зображення. Система автоматично оновлює лічильник товарів в заголовку сторінки та перераховує загальну суму кошика. Всі зміни в кошику відображаються в реальному часі через CartContext [17], який забезпечує синхронізацію стану між компонентами.

Оновлення кількості товарів в кошику включає валідацію нової кількості з перевіркою мінімального значення, яке не може бути менше одиниці. При зміні кількості система оновлює відповідний запис в localStorage та перераховує загальну вартість всіх товарів. Інтерфейс користувача автоматично оновлюється через CartContext, що забезпечує консистентність відображення даних. При видаленні товару з кошика система повністю видаляє запис з localStorage та оновлює всі пов'язані показники.

Алгоритм оновлення замовлення

Процес оформлення замовлення починається з валідації даних форми, яка включає перевірку обов'язкових полів таких як адреса доставки, контактний телефон та ім'я замовника. Система використовує Zod [12] схеми для валідації даних на серверній стороні, що забезпечує надійність та безпеку обробки інформації. Після успішної валідації створюється новий запис в таблиці Order з базовою інформацією про замовлення та статусом "pending". Всі товари з кошика користувача додаються в таблицю OrderItem з фіксованою ціною на момент створення замовлення, що захищає від зміни цін після оформлення. Система автоматично розраховує загальну суму замовлення на основі кількості та цін всіх товарів.

Після створення замовлення кошик користувача очищається, а система відправляє email підтвердження з детальною інформацією про замовлення. Користувач перенаправляється на сторінку підтвердження, де може переглянути деталі замовлення та отримати номер для відстеження. Для гостей, які не авторизовані в системі, створюється замовлення без зв'язку з користувачем, а контактні дані зберігаються в спеціальних полях customerName та customerEmail.

Гості можуть відстежувати статус замовлення за його номером без необхідності створення облікового запису.

Інтеграція з платіжними системами реалізована через webhook'и [13], які обробляють повідомлення про статус оплати від платіжного провайдера. При отриманні підтвердження оплати система автоматично оновлює статус замовлення та відправляє повідомлення користувачу про успішну оплату. Вся інформація про транзакції логується для подальшого аналізу та аудиту.

Алгоритм пошуку та фільтрації

Система пошуку запчастин реалізована з використанням PostgreSQL [14] повнотекстового пошуку та ILIKE операторів для нечутливого до регістру пошуку. Алгоритм пошуку починається з парсингу пошукового запиту користувача, який може містити ключові слова, назви запчастин або опис. Система виконує пошук по назві запчастини з використанням ILIKE оператора, що дозволяє знаходити часткові збіги. Додатково виконується пошук по полю опису для знаходження запчастин, які можуть не містити шукані ключові слова в назві. Пошук по категорії виконується з точним збігом для забезпечення релевантності результатів.

Результати пошуку об'єднуються з унікальністю для виключення дублікатів та сортуються за релевантністю. Система фільтрації дозволяє користувачам обмежувати результати за категорією запчастин, діапазоном цін та характеристиками автомобіля. Фільтри комбінуються через логічну операцію AND, що забезпечує точність результатів. Результати пошуку піддаються пагінації з відображенням 20 товарів на сторінку для оптимізації продуктивності та зручності користувача.

SQL запити оптимізовані з використанням JOIN операцій для отримання інформації про автомобіль разом з запчастинами, що зменшує кількість запитів до бази даних. Система використовує індекси для швидкого пошуку по ключових полях, що значно прискорює виконання запитів при великій кількості даних.

Алгоритм роботи з обраним

Функціональність обраних товарів дозволяє авторизованим користувачам зберігати посилання на цікаві запчастини для подальшого перегляду. При додаванні товару в обране система спочатку перевіряє авторизацію користувача, оскільки ця функція доступна тільки зареєстрованим користувачам. Далі виконується перевірка унікальності комбінації користувача та запчастини для запобігання дублювання записів. При відсутності дублікату створюється новий запис в таблиці FavoritePart з поточною датою та часом.

Інтерфейс користувача автоматично оновлюється після додавання товару в обране, змінюючи зовнішній вигляд кнопки на активний стан. Система відображає повідомлення про успішне додавання товару в обране для підтвердження дії користувача. При видаленні товару з обраного система знаходить відповідний запис за комбінацією користувача та запчастини та повністю видаляє його з бази даних.

Відображення списку обраних товарів включає запит всіх записів FavoritePart для конкретного користувача з JOIN операцією з таблицею Part для отримання повної інформації про запчастини. Результати групуються за категоріями для зручності навігації та відображення. Користувач може додавати товари з обраного безпосередньо в кошик, що забезпечує швидкий доступ до часто використовуваних запчастин.

Алгоритм адмін-панелі

Адміністративна панель надає повний контроль над функціональністю додатка для користувачів з роллю "admin". Управління замовленнями включає отримання всіх замовлень з пагінацією для ефективної роботи з великою кількістю даних. Система дозволяє фільтрувати замовлення за статусом, датою створення та конкретним користувачем для швидкого пошуку необхідної інформації. При зміні статусу замовлення система автоматично відправляє email повідомлення користувачу про оновлення та логує дію адміністратора для аудиту.

Управління товарами реалізовано через повний набір CRUD операцій, що дозволяє створювати, редагувати, переглядати та видаляти запчастини. Система

підтримує завантаження зображень через multer middleware з автоматичною валідацією типів файлів та розмірів. Всі дані проходять валідацію з перевіркою позитивних значень цін та обов'язкових полів. Адміністратори можуть оновлювати категорії запчастин та зв'язки з автомобілями, а також виконувати масові операції для ефективного управління каталогом.

Управління користувачами включає перегляд списку всіх зареєстрованих користувачів з можливістю зміни їх ролей між "user" та "admin". Система дозволяє блокувати та розблоковувати акаунти користувачів для контролю доступу до додатка. Адміністратори можуть переглядати повну історію замовлень конкретного користувача для аналізу поведінки та надання підтримки. Система підтримує експорт даних користувачів у різних форматах для зовнішнього аналізу та звітності.

Алгоритм обробки помилок

Система валідації даних реалізована на кількох рівнях для забезпечення надійності та безпеки. Frontend валідація використовує HTML5 атрибути для базової перевірки даних безпосередньо в браузері користувача. JavaScript валідація виконується перед відправкою даних на сервер для попередньої перевірки та надання миттєвого зворотного зв'язку. Backend валідація реалізована через Zod схеми, які забезпечують строгу типізацію та валідацію всіх вхідних даних на серверній стороні.

Обробка помилок валідації включає структуровані повідомлення про помилки, які надають користувачу зрозумілу інформацію про проблеми з введеними даними. Система обробки помилок API реалізована через try-catch блоки в кожному API роуті для перехоплення та обробки неочікуваних помилок. Всі помилки логуються в консоль сервера з детальною інформацією про контекст виникнення для подальшого аналізу та виправлення.

Система повертає структуровані помилки з відповідними HTTP статус кодами для правильного оброблення на клієнтській стороні. Глобальний error boundary в React забезпечує перехоплення помилок на рівні компонентів та відображення fallback UI для критичних помилок. Обробка мережевих помилок

включає перевірку статусу відповіді від сервера та реалізацію retry логіки для тимчасових помилок мережі.

Система кешування даних в localStorage дозволяє працювати з додатком в offline режимі для критичних функцій таких як перегляд каталогу та кошика. Користувачі отримують зрозумілі повідомлення про проблеми з мережею та рекомендації щодо подальших дій.

Алгоритм оптимізації продуктивності

Система кешування реалізована на кількох рівнях для максимального прискорення роботи додатка. Кешування каталогу товарів в Redis дозволяє швидко отримувати популярні дані без запитів до основної бази даних. Результати пошуку кешуються для повторних запитів, що значно зменшує навантаження на базу даних. Зображення товарів обслуговуються через CDN для швидкої доставки контенту користувачам з різних географічних локацій.

Статичні дані кешуються в браузері користувача для зменшення кількості мережових запитів та прискорення завантаження сторінок. Система інвалідації кешу автоматично оновлює кешовані дані при зміні інформації в базі даних для забезпечення актуальності контенту. Оптимізація зображень включає автоматичне зменшення розміру завантажених зображень до оптимальних розмірів для веб-використання.

Система конвертує зображення в сучасний WebP формат для зменшення розміру файлів при збереженні якості. Автоматичне створення thumbnail'ів для списків товарів дозволяє швидко завантажувати сторінки каталогу. Lazy loading зображень реалізований для завантаження контенту тільки при необхідності, що прискорює початкове завантаження сторінок.

Responsive images забезпечують доставку оптимальних розмірів зображень для різних пристроїв та розмірів екрану. Оптимізація запитів до бази даних включає використання індексів для швидкого пошуку по ключових полях. JOIN запити замість окремих запитів зменшують кількість звернень до бази даних та прискорюють отримання пов'язаних даних.

Пагінація для великих списків забезпечує ефективне використання пам'яті та швидке відображення результатів. Використання Prisma include для eager loading дозволяє отримувати пов'язані дані в одному запиті замість окремих запитів. Система моніторингу повільних запитів допомагає ідентифікувати та оптимізувати проблемні місця в роботі з базою даних.

Алгоритми безпеки

Захист від SQL ін'єкцій реалізований через використання Prisma ORM [15], який автоматично параметризує всі запити до бази даних. Система валідує всі вхідні дані перед обробкою для запобігання виконання зловмисного коду. Екранування спеціальних символів забезпечує безпечне відображення користувацького контенту без ризику виконання HTML або JavaScript коду.

Обмеження довжини вхідних полів запобігає атакам через переповнення буферів та зменшує ризик DoS атак. Система логування підозрілих запитів дозволяє відстежувати потенційні спроби атак та реагувати на них вчасно. Захист від XSS атак включає автоматичне екранування HTML в React компонентах для запобігання виконання зловмисного JavaScript коду.

Валідація вхідних даних на серверній стороні забезпечує додатковий рівень захисту навіть при обході клієнтської валідації. Використання Content Security Policy обмежує виконання JavaScript коду тільки з довірених джерел. Sanitization користувацького контенту очищує потенційно небезпечний HTML код перед збереженням в базу даних.

Захист від DOM-based XSS включає валідацію всіх даних перед вставкою в DOM структуру сторінки. Захист від CSRF атак реалізований через використання CSRF токенів, які перевіряються при кожному POST запиті. Система перевіряє Origin header для запобігання міжсайтових запитів з недовірених джерел.

Налаштування SameSite cookies забезпечує захист від CSRF атак через обмеження передачі cookies між різними сайтами. Валідація реферера додатково перевіряє джерело запиту для підтвердження його легітимності. Логування

підозрілих запитів допомагає ідентифікувати потенційні атаки та реагувати на них.

Алгоритм моніторингу та логування

Система логування реалізована для повного відстеження роботи додатка та діагностики проблем. Логування всіх API запитів включає інформацію про метод, URL, статус відповіді та час виконання для аналізу продуктивності. Помилки логуються з повним stack trace для швидкої ідентифікації причин та місць виникнення проблем.

Дії користувачів логуються для аналізу поведінки та покращення користувацького досвіду. Адміністративні дії реєструються з детальною інформацією про зміни для забезпечення аудиту та відстеження відповідальності. Система ротації лог файлів автоматично архівує старі логи та обмежує використання дискового простору.

Моніторинг продуктивності включає відстеження часу відповіді API для ідентифікації повільних запитів та оптимізації проблемних місць. Моніторинг використання пам'яті допомагає запобігти витокам пам'яті та забезпечити стабільну роботу додатка. Відстеження кількості помилок дозволяє швидко реагувати на проблеми та зменшувати час простою системи.

Моніторинг активності користувачів надає інформацію про популярні функції та шляхи навігації для покращення інтерфейсу. Система алертів автоматично повідомляє про критичні проблеми через email або інші канали комунікації для швидкої реакції команди підтримки.

Аналітика додатка включає відстеження популярних товарів для оптимізації каталогу та маркетингових стратегій. Аналіз шляхів користувачів допомагає зрозуміти поведінку та покращити навігацію по додатку. Відстеження конверсії кошика в замовлення надає важливу інформацію про ефективність продажів.

Аналіз пошукових запитів допомагає оптимізувати каталог товарів та покращити алгоритми пошуку. Система генерації звітів для адміністрації надає

детальну інформацію про продажі, популярні товари та поведінку користувачів для прийняття обґрунтованих бізнес-рішень.

Надзвичайно важливим етапом у проектуванні системи є розробка діаграми послідовності, яка служить ключовим інструментом для детального відображення взаємодії між різними елементами системи в часовому аспекті. Діаграма послідовності дозволяє наочно продемонструвати порядок виконання операцій, обмін повідомленнями між компонентами та логіку роботи всієї системи. Для даного веб-додатку була розроблена загальна діаграма послідовності (додаток Б).

2.4 Реалізація застосунку веб-додатку продажу автотоварів

Система веб-додатку для продажу автотоварів містить у собі широкий набір функціональних модулів, що забезпечують повноцінну роботу платформи. Серед них — модуль аутентифікації та авторизації користувачів, система управління базою даних товарів, механізми фільтрації та пошуку за технічними параметрами авто, інтеграція з платіжними та логістичними сервісами, адміністративна панель для керування контентом, система обробки замовлень, модулі обробки зображень товарів, а також аналітичні інструменти для збору статистики продажів і поведінки користувачів.

Архітектура програмного забезпечення організована у вигляді ключових груп компонентів. Система включає зовнішні інтеграції (API платіжних і логістичних сервісів), допоміжні утилітарні модулі (валидація, логування, форматування даних), систему маршрутизації та навігації, користувацькі інтерфейси (сторінки каталогу, товару, замовлення, профілю), окремі компоненти для обробки запитів до бази даних і кешування, модулі введення та редагування даних (форми, фільтри), а також базові UI-елементи — кнопки, картки товарів, сповіщення та інші повторно використовувані інтерфейсні компоненти.

Така структурна організація забезпечує гнучкість, повторне використання коду, модульність та спрощує супровід і масштабування системи.

Підсистема конфігурації та ініціалізації. Підсистема конфігурації та ініціалізації в системі відповідає за централізоване управління параметрами всіх її підсистем. Для цього використовується єдиний конфігураційний модуль, який забезпечує типізований доступ до налаштувань і виконує валідацію конфігураційних даних, що дозволяє підтримувати цілісність і коректність параметрів на всіх рівнях системи.

В контексті аутентифікації цей модуль містить конфігурацію, яка включає адаптер для роботи з базою даних через Prisma, а також список провайдерів аутентифікації. Зокрема, реалізовано механізм авторизації через обробку облікових даних користувача — email та пароль. Логіка авторизації перевіряє наявність введених даних, шукає користувача у базі за email, а потім порівнює введений пароль із захешованим паролем у базі даних за допомогою безпечної криптографічної бібліотеки bcrypt.

У разі успішної валідації повертається об'єкт користувача з основними властивостями, такими як ідентифікатор, електронна пошта, ім'я та роль користувача, що надалі використовується для керування правами доступу.

Для підтримки сесій використовується стратегія на основі JWT (JSON Web Tokens), що дозволяє зберігати інформацію про користувача у токени без необхідності зберігати стан на сервері. За допомогою callback-функцій забезпечується оновлення токена інформацією про роль користувача, а також передачу цих даних до сесії, що дозволяє підтримувати актуальні права доступу під час взаємодії користувача з системою.

Таким чином, підсистема конфігурації та ініціалізації забезпечує надійну, безпечну та централізовану організацію процесу аутентифікації і управління параметрами системи, що сприяє гнучкості, підтримці масштабованості та полегшує подальший розвиток додатку (Додаток А).

Локальне зберігання. Для забезпечення збереження стану кошика між сесіями використовується локальне сховище браузера — localStorage. Це дозволяє користувачам зберігати інформацію про вибрані товари навіть після оновлення сторінки або закриття браузера. У відповідній логіці застосовується

стан, який містить масив товарів у кошику, а також функції для додавання та видалення товарів.

Функція додавання товару оновлює поточний стан, спочатку копіюючи існуючий масив кошика. Далі вона перевіряє, чи є у кошику товар із заданим ідентифікатором. Якщо такий товар вже існує, кількість збільшується на передане значення. Якщо ж товар новий, він додається до масиву. Після оновлення масиву функція серіалізує його у формат JSON і зберігає у `localStorage`, що гарантує збереження актуального стану між сесіями.

Аналогічним чином функція видалення товару фільтрує масив, виключаючи товар з вказаним ідентифікатором, після чого оновлений масив також записується у `localStorage`. Завдяки такому підходу забезпечується синхронізація стану кошика з локальним сховищем, що дозволяє підтримувати його в актуальному стані при будь-яких змінах.

В результаті, використання `localStorage` дає можливість користувачам зберігати вибрані товари в кошику протягом тривалого часу, покращуючи зручність і досвід взаємодії з веб-додатком без необхідності постійного звернення до сервера.

```
const useCart = () => {  
  const [cart, setCart] = useState<CartItem[]>([]);  
  const addToCart = useCallback((part: Part, quantity: number = 1) => {  
    setCart(prev => {  
      const newCart = [...prev];  
      const existingIndex = newCart.findIndex(item => item.partId === part.id);  
  
      if (existingIndex >= 0) {  
        newCart[existingIndex].quantity += quantity;  
      } else {  
        newCart.push({ partId: part.id, quantity, part });  
      }  
  
      localStorage.setItem('cart', JSON.stringify(newCart));  
      return newCart;  
    });  
  }, []);  
  
  const removeFromCart = useCallback((partId: number) => {
```

```

setCart(prev => {
  const newCart = prev.filter(item => item.partId !== partId);
  localStorage.setItem('cart', JSON.stringify(newCart));
  return newCart;
});
}, []);

return { cart, addToCart, removeFromCart };

```

Лістинг 2.1 – Ініціалізація системних компонентів

Управління станом. React Context API використовується для управління глобальним станом кошика у веб-додатку. Це дозволяє зберігати інформацію про товари, додані користувачем, і робить її доступною у будь-якому компоненті без необхідності передавати дані через пропси на багатьох рівнях. Контекст створюється з типом, який описує структуру даних і функцій, що будуть доступні, а початкове значення встановлюється як `undefined`, щоб запобігти використанню контексту без провайдера.

Компонент-провайдер обгортає частину додатку, яка потребує доступу до кошика, приймаючи дочірні елементи. Він зберігає масив товарів у стані за допомогою хука `useState`, а також надає функції для додавання, видалення та очищення кошика. Значення контексту передається через провайдер, що дозволяє дочірнім компонентам отримувати доступ до актуального стану кошика і керувати ним.

Функція додавання товару приймає об'єкт товару та кількість і виконує оновлення стану, враховуючи, чи є цей товар вже у кошику. Якщо товар присутній, його кількість збільшується на вказане значення, а якщо ні — він додається як новий елемент. Це запобігає дублюванню товарів у кошику та дозволяє коректно відслідковувати кількість кожного з них.

Хоча у наведеному коді відсутні функції видалення товарів та очищення кошика, вони зазвичай включаються до складу контексту для повноцінного керування станом. Також не реалізована логіка збереження стану у локальному сховищі чи синхронізації з сервером, а також обробка можливих помилок або обмежень при зміні кількості товарів.

Загалом, застосування React Context API дозволяє централізовано та ефективно управляти станом кошика, підвищуючи читабельність коду та зручність розробки, особливо при використанні типізації, що знижує ризики помилок. Такий підхід забезпечує гнучкість і масштабованість інтерфейсу користувача.

```
export const CartContext = createContext<CartContextType | undefined>(undefined);

export const CartProvider: React.FC<{ children: React.ReactNode }> = ({ children }) => {
  const [cart, setCart] = useState<CartItem[]>([]);

  const addToCart = useCallback((part: Part, quantity: number = 1) => {
    setCart(prev => {
      const existing = prev.find(item => item.partId === part.id);
      if (existing) {
        return prev.map(item =>
          item.partId === part.id
            ? { ...item, quantity: item.quantity + quantity }
            : item
        );
      }
      return [...prev, { partId: part.id, quantity, part }];
    });
  }, []);

  return (
    <CartContext.Provider value={{ cart, addToCart, removeFromCart, clearCart }}>
      {children}
    </CartContext.Provider>
  );
};
```

Лістинг 2.2 – Система управління глобальним станом.

Каталог запчастин реалізовано як окрему підсистему, яка відповідає за зберігання, пошук, фільтрацію та відображення автозапчастин. Вона підтримує категоризацію, бренди, а також сумісність із різними моделями автомобілів. Каталог інтегрується з підсистемою кошика, замовлень та адміністративною панеллю для забезпечення повного циклу роботи з товарами.

					ІПЗ.КР.Б-121-25-ПЗ	
						42

Підсистема кошика дозволяє користувачам додавати, видаляти та змінювати кількість товарів у кошику. Вона зберігає стан кошика як для авторизованих, так і для неавторизованих користувачів, забезпечуючи зручність користування сервісом. Дані з кошика передаються до підсистеми оформлення замовлення для подальшої обробки.

Оформлення та обробка замовлень здійснюється окремою підсистемою, яка приймає замовлення, перевіряє валідність даних, створює відповідні записи у базі даних, а також відправляє підтвердження на електронну пошту користувача. Вона зберігає статуси замовлень і взаємодіє з підсистемами користувачів, кошика та каталогу.

Адміністративна панель є важливою підсистемою, що дозволяє адміністраторам керувати товарами, категоріями, замовленнями та користувачами. Вона має окремий інтерфейс із підвищеними правами доступу та взаємодіє з усіма основними підсистемами для ефективного управління даними.

Підсистема завантаження файлів відповідає за додавання зображень товарів та аватарів користувачів. Вона перевіряє типи та розмір файлів, зберігає їх у публічній директорії та інтегрується з каталогом і профілем користувача. Пошук і фільтрація реалізовані як окрема підсистема, що забезпечує швидкий пошук по каталогу, а також фільтрацію за різними параметрами, такими як категорія, ціна чи бренд. Вона працює як на стороні клієнта, так і на сервері, і тісно пов'язана з підсистемою каталогу.

Окрему увагу приділено **підсистемі безпеки**, яка захищає API, перевіряє права доступу, запобігає SQL-ін'єкціям, XSS та CSRF-атакам. Вона впроваджує CORS, безпечні заголовки та валідацію даних, працюючи у зв'язці з усіма підсистемами додатку.

Завдяки чіткій структурі та взаємодії підсистем, веб-додаток є модульним, масштабованим і зручним для подальшої підтримки та розвитку.

Висновок до другого розділу

У процесі написання другого розділу було розроблено архітектурну діаграму веб-застосунку для продажу автотоварів, а також діаграми компонентів, які відображають повну структуру системи, включаючи клієнтську, серверну та інтеграційну частини. Сформовано дев'ять основних функціональних підсистем, що забезпечують повноцінне функціонування веб-платформи — від взаємодії з користувачем до обробки замовлень та керування адміністративним контентом.

Особливу увагу приділено побудові бази даних на основі PostgreSQL [19] з використанням Prisma ORM, що дозволяє ефективно обробляти дані про товари, користувачів, замовлення та фільтрувальні параметри. Також спроектовано систему авторизації з JWT [16], інтеграцію з платіжними системами, механізм фільтрації за технічними характеристиками авто, а також реалізовано логіку адміністративної панелі, що забезпечує контроль над усіма елементами контенту та аналітики.

На завершальному етапі реалізовано повнофункціональний веб-застосунок з усіма запланованими можливостями, включаючи автентифікацію користувачів, адаптивний інтерфейс, пошук і сортування товарів, оформлення замовлень та керування ними, що повністю відповідає вимогам сучасної e-commerce системи у сфері автотоварів.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ЗАСТОСУНКОМ

3.1. Порядок встановлення та налаштування параметрів системи.

Система користувача повинна відповідати базовим технічним вимогам, необхідним для стабільної роботи веб-додатку з продажу автотоварів у сучасних браузерах. Веб-застосунок розрахований на десктопні та мобільні пристрої з підтримкою JavaScript, сучасних CSS-стандартів і безпечного з'єднання через HTTPS.

Для комфортної роботи на персональному комп'ютері рекомендується наявність сучасного процесора (не нижче Intel Core i3 або аналогічного AMD), щонайменше 4 ГБ оперативної пам'яті, а також актуальний браузер (Google Chrome, Mozilla Firefox, Safari або Microsoft Edge останніх версій). З боку мобільних пристроїв підтримуються Android 8.0+ та iOS 12+, при цьому мінімальна конфігурація повинна включати не менше 2 ГБ оперативної пам'яті, стабільне підключення до Інтернету та активований JavaScript у браузері.

Діаграма розгортання слугує для візуального представлення взаємодії компонентів системи. На ній (рисунок 3.1) зображено архітектуру веб-додатку у вигляді клієнт-серверної моделі, що охоплює фронтенд, бекенд та зовнішні сервіси. Клієнтська частина, реалізована на базі Next.js [20], взаємодіє з серверною частиною через REST API або GraphQL-запити. Сервер, реалізований на Node.js, відповідає за обробку бізнес-логіки, автентифікацію, керування замовленнями й запитами до бази даних.

Серверна логіка використовує PostgreSQL [21] як основне сховище інформації, а ORM Prisma забезпечує зручний і типізований доступ до даних. Для автентифікації та авторизації застосовується механізм JWT. Також система може бути розгорнута на хмарній інфраструктурі (наприклад, Vercel [8] або Railway) з підтримкою масштабування. Таким чином, веб-додаток забезпечує безпечну і продуктивну взаємодію між користувачем, інтерфейсом і серверними ресурсами в реальному часі.

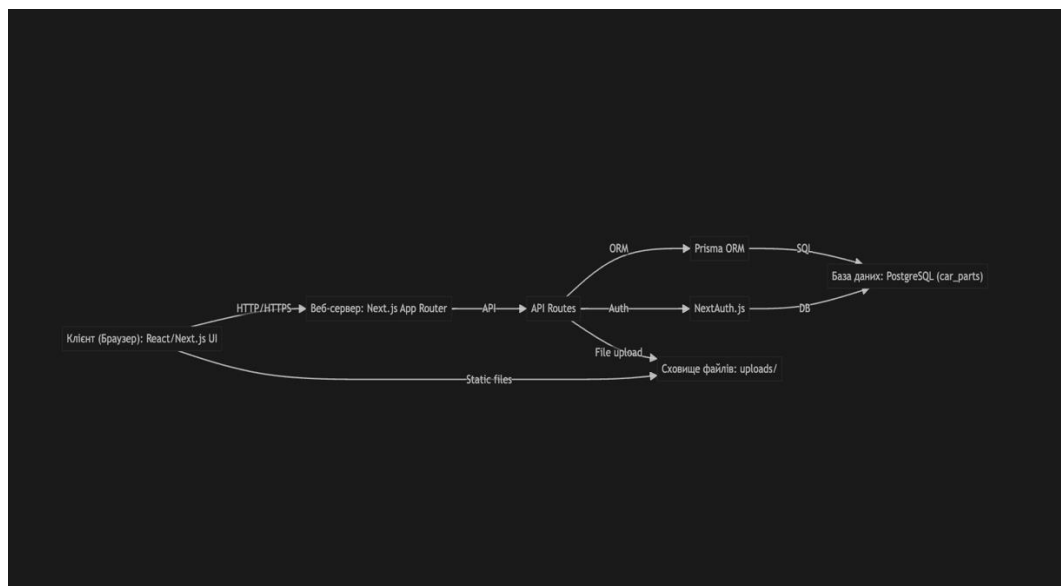


Рисунок 3.1 - Діаграма розгортання застосунку

3.2 Структура інтерфейсу вебзастосунку.

Головна сторінка веб-додатку яку користувач бачить коли заходить на сайт. (рисунок 3.2).

Автозапчастини

Головна | Про нас | Кошик | Увійти |

Реєстрація

Оберіть ваш автомобіль

Виберіть марку, модель, рік та тип двигуна, щоб знайти підходящі запчастини

Виберіть характеристики вашого автомобіля

Оберіть марку, модель, рік та тип двигуна для пошуку підходящих запчастин

Марка *

Модель *

Рік *

Тип двигуна *

Оберіть марку

Оберіть модель

Оберіть рік

Оберіть двигун

* Обов'язкові поля. Заповніть всі поля для пошуку запчастин.

Пошук

Пошук за назвою, описом.

Пошук

Категорія

Всі

Ціна

0

-

100€

Застосувати ціну

Сортування

Нові

Весь асортимент

Гальмівна система

Кузов

Двигун

Гальмівна система

Рисунок 3.2 – Головна сторінка

На головній сторінці користувач може перейти відразу до вибору авто для якого необхідна деталь або ж за допомогою фільтрів та пошуку обрати конкретну деталь в незалежності від конкретного авто.

Також є можливість перегляду всього асортименту товарів проскроливши в низ спочатку виводиться 10 товарів, але при можливості можна натиснувши кнопку переглянути ще 10 товарів (рисунк 3.4).

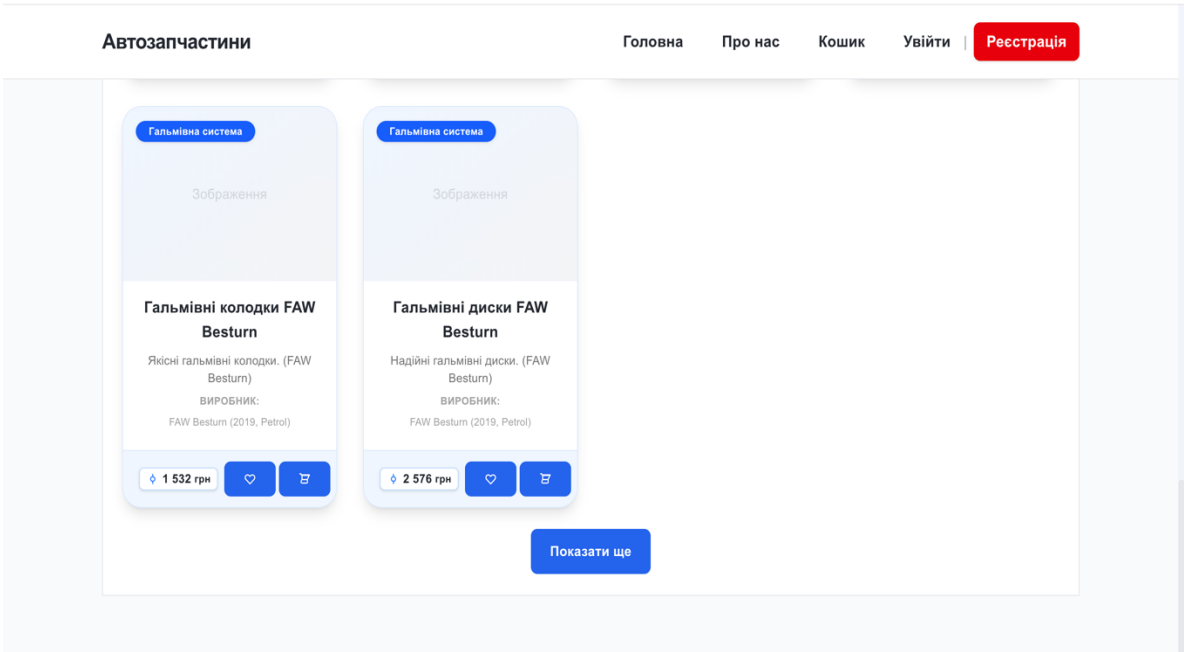


Рисунок 3.3 – Весь асортимент

Натиснувши на сторінку “Про нас” відображається інформація про компанію (рисунк 3.4).

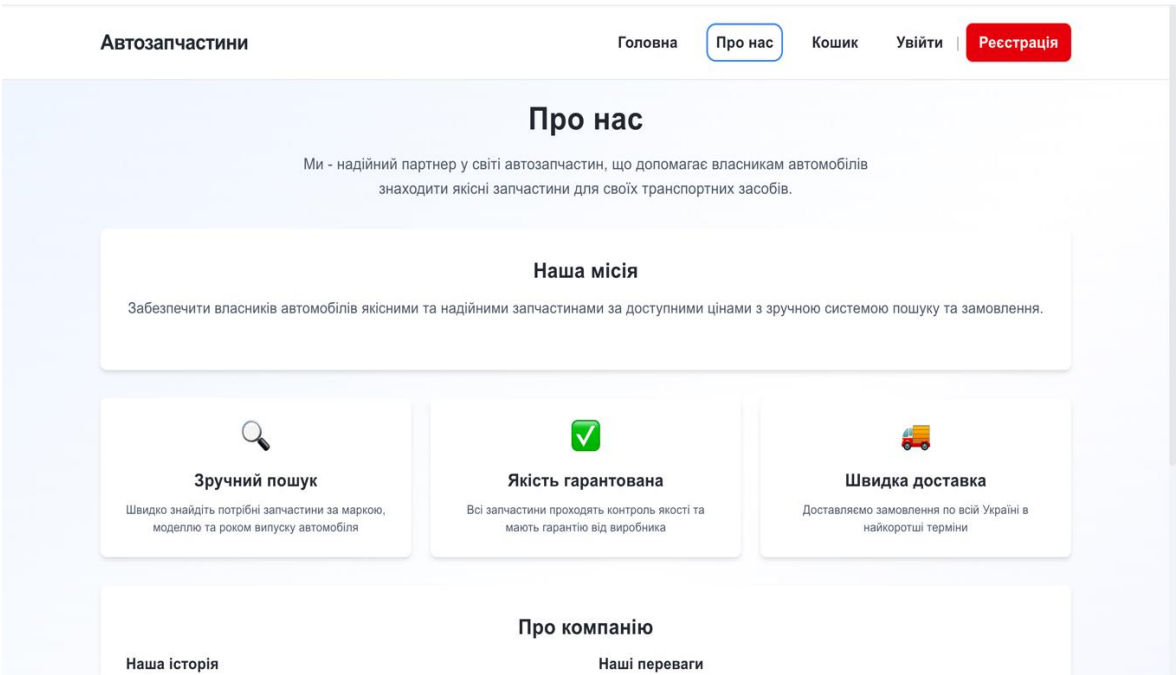


Рисунок 3.4 – Сторінка “Про нас”

При додаванні деталі в кошик висвічується примітка що товар додано (рисунок 3.5) та в кошику з'являється додана деталь. (рисунок 3.6)

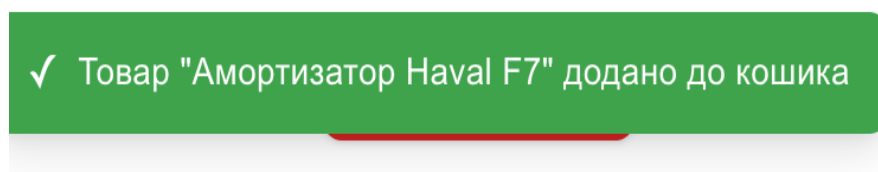


Рисунок 3.5 – Примітка про додавання в кошик.

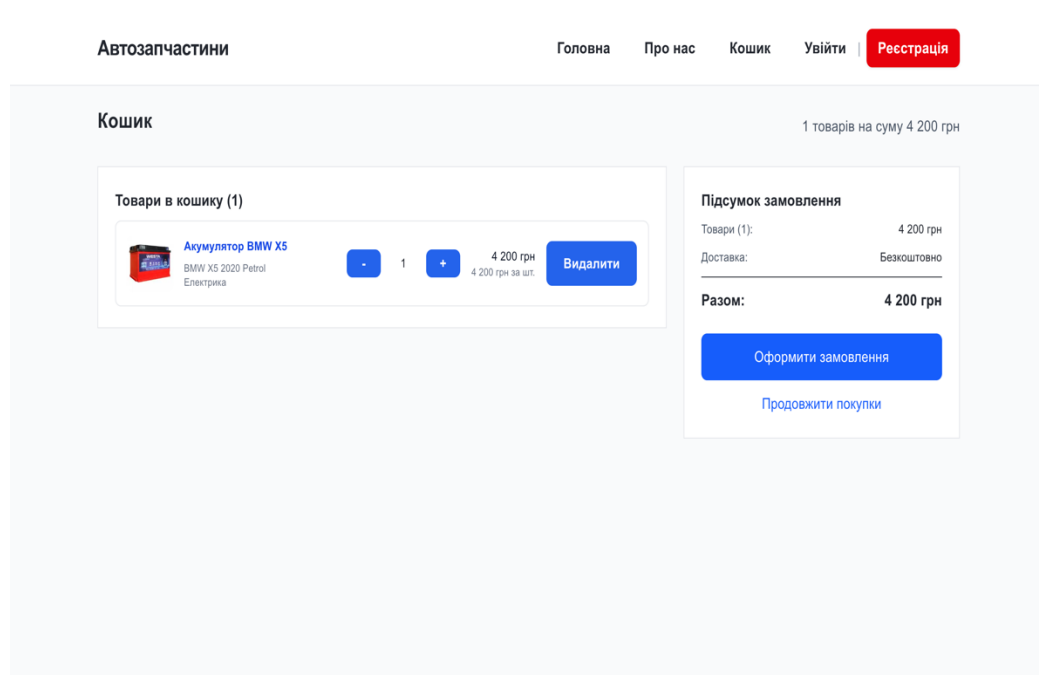


Рисунок 3.6 – Сторінка “Кошик”.

Кошиком може користуватись як і авторизований користувач а також може і не авторизований.

При виборі конкретного авто користувачу доступні категорії товарів за якими є деталі (рисунок 3.7) натиснувши на обрану категорію користувач переходить на сторінку зі списком деталей (рисунок 3.8):

Автозапчастини

Головна
Про нас
Кошик
Увійти

Реєстрація

Виберіть характеристики вашого автомобіля

Оберіть марку, модель, рік та тип двигуна для пошуку підходящих запчастин

Марка *

Acura

Модель *

MDX

Рік *

2017

Тип двигуна *

Petrol

✓
Обрано: Acura MDX (2017, Petrol)

* Обов'язкові поля. Заповніть всі поля для пошуку запчастин.

Обраний автомобіль

Acura MDX (2017, Petrol)

Обрано

Категорії запчастин для обраного автомобіля

Кузов

Двигун

Рисунок 3.7 – Вибір авто.

Автозапчастини

Головна
Про нас
Кошик
Увійти

Реєстрація

Двигун

Запчастини для Acura MDX (2017, Petrol)

Знайдено 1 деталей

Назад до вибору автомобіля

Двигун

Зображення

Ремінь ГРМ Acura MDX

Ремінь газорозподільного механізму. (Acura MDX)

ВИБРОБНИК:

Acura MDX (2017, Petrol)

1 877 грн

Рисунок 3.8 – Список авто за конкретними параметрами авто.

Також користувач може авторизуватися або зареєструватися на сайті (рисунок 3.8-3.9).

Автозапчастини

Головна | Про нас | Кошик | Увійти | **Реєстрація**

Вхід

Email

you@email.com

Пароль

Введіть пароль

Увійти

Немає акаунта? [Зареєструватися](#)

Рисунок 3.8 – Сторінка входу.

Автозапчастини

Головна | Про нас | Кошик | Увійти | **Реєстрація**

Реєстрація

Ім'я

Ваше ім'я

Email

you@email.com

Пароль

Введіть пароль

Зареєструватися

Вже є акаунт? [Увійти](#)

Рисунок 3.9 – Сторінка реєстрації.

Для користувачів які зареєстровані та увійшли на сайт є можливість додавати свої авто (рисунок 3.10) та додавати в обране деталі (рисунок 3.11):

Автозапчастини

Головна
Про нас
Кошик
Мій профіль
Вийти

Мій профіль

t@gmail.com

Мої автомобілі

Улюблені деталі

Мої замовлення

Мої збережені автомобілі

У вас поки немає збережених автомобілів

Додати новий автомобіль

Марка

Аudi

Модель

A4

Рік

2018

Двигун

Diesel

Назва (необов'язково)

Наприклад: Моя BMW

Обрано: Audi A4 (2018, Diesel)

Додати автомобіль

Скасувати

Рисунок 3.10 – Додавання авто користувача.

Автозапчастини

Головна
Про нас
Кошик
Мій профіль
Вийти

Мій профіль

t@gmail.com

Мої автомобілі

Улюблені деталі

Мої замовлення

Улюблені деталі

Оновити

1 деталей

Гальмівні колодки Bugatti Chiron

×

Bugatti Chiron (2021)
Гальмівна система • 1360 грн

Рисунок 3.11 – Улюблені деталі користувача.

Також користувач може переглядати свої замовлення, для цього для початку йому потрібно зробити замовлення (рисунок 3.12):

ІНІ

Головна

Про к

Доставка

2

Оплата

Інформація про доставку

Повне ім'я *

Омельчук Вадим

Email *

vt@gmail.com

Номер телефону *

09977755554

Адреса доставки *

Покровська 135

Місто *

Житомир

Поштовий індекс

10031

Продовжити до оплати

✓ Доставка

Оплата

Оплата

Метод оплати

Готівкою при отриманні

⚠ Оплата готівкою при отриманні

- Оплата здійснюється при отриманні товару
- Додаткова комісія за накладений платіж не стягується
- Перевірте товар перед оплатою
- Можна відмовитися від покупки при отриманні

Загальна сума: 1 360 грн

* Сума для оплати при отриманні

Назад

Підтвердити замовлення

Рисунок 3.12 – Оформлення замовлення.

✓

Замовлення підтверджено!

Дякуємо за ваше замовлення. Номер замовлення: #1

Оплата готівкою при отриманні

- Наш менеджер зв'яжеться з вами для уточнення деталей
- Оплата здійснюється при отриманні товару
- Можна перевірити товар перед оплатою

Мої замовлення

Продовжити покупки

Рисунок 3.13 – Замовлення підтверджено

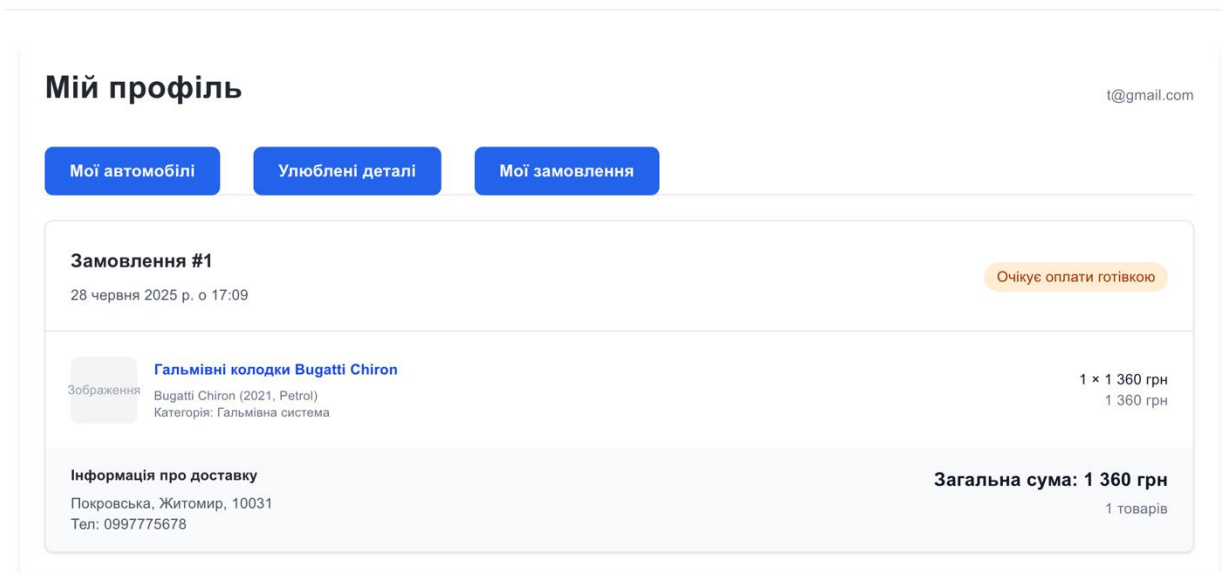


Рисунок 3.14 – Сторінка “Мої замовлення”

Тепер в “Мої замовлення” відображається наше замовлення.

Також для адміністратора є адмін-панель (рисунок 3.15): в якій він може: додавати, змінювати та видаляти товар та категорії товару, а також змінювати стан замовлення (рисунок 3.16).

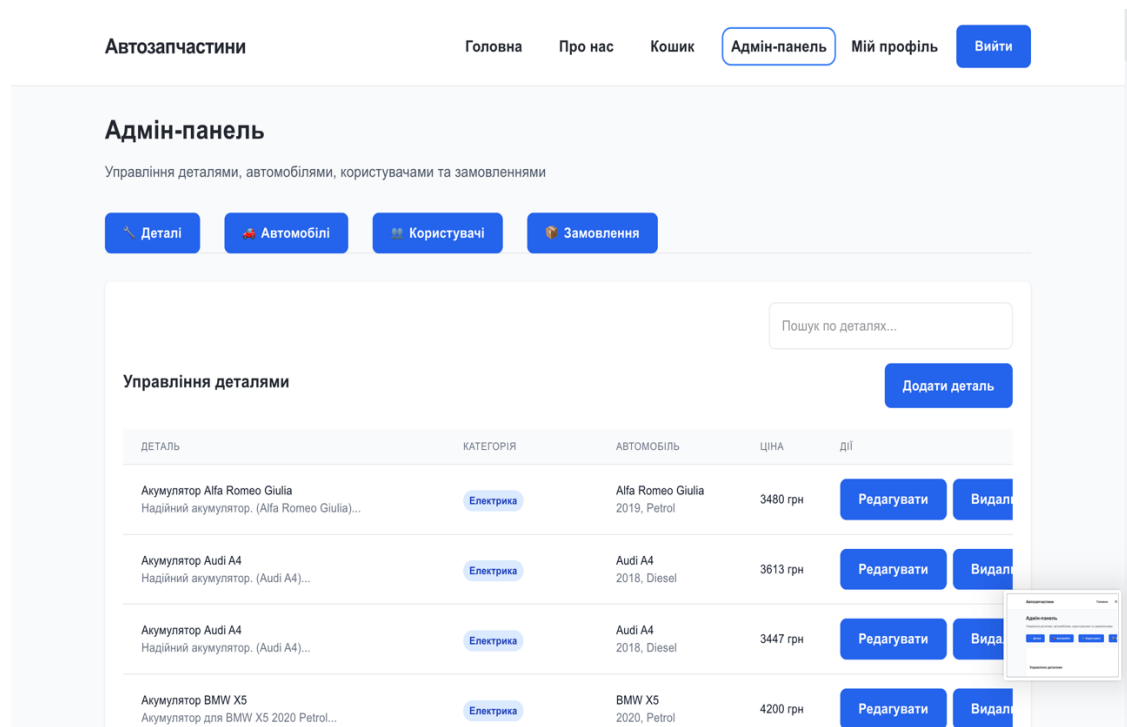


Рисунок 3.13 – Панель адміністратора.

Управління замовленнями

Оновити

Пошук по замовленнях...

Замовлення #1

28.06.2025, 17:09

Очікує оплати готівкою

Інформація про клієнта

Ім'я: Омельчук Вадим
 Email: vt211@gmail.com
 Телефон: 0997775678
 ID користувача: smcgb448r0000onv974vcjivs

Доставка

Адреса:
 Покровська, Житомир, 10031

Очікує підтвердження

✓ Очікує оплати готівкою

Підтверджено

Відправлено

Доставлено

Скасовано

Товари



Гальмівні коподки Bugatti Chiron
 Кількість: 1 x 1 360 грн

1 360 грн

Загальна сума: 1 360 грн

Рисунок 3.14 – Перегляд замовлень на панелі адміністратора.

Також адміністратор має можливість редагувати деталі та є пошук на кожній вкладці.

Редагувати деталь

Назва *

Акумулятор Alfa Romeo Giulia

Категорія *

Електрика

Автомобіль *

Alfa Romeo Giulia (2019, Petrol)

Ціна (грн) *

3480

Зображення деталі *

Вибрати файл

Видалити



/uploads/upload_1751120371588.webp
 Дозволені формати: JPEG, PNG, WebP. Максимальний розмір: 5MB

Опис *

Надійний акумулятор. (Alfa Romeo Giulia)

Оновити

Скасувати

Рисунок 3.15 – Редагування товару.

Деталь оновлено

Рисунок 3.16 – Підтвердження про вдале редагування.

Адміністратор може змінювати роль користувача (рисунок 3.17).

					ІПЗ.КР.Б-121-25-ІПЗ	54

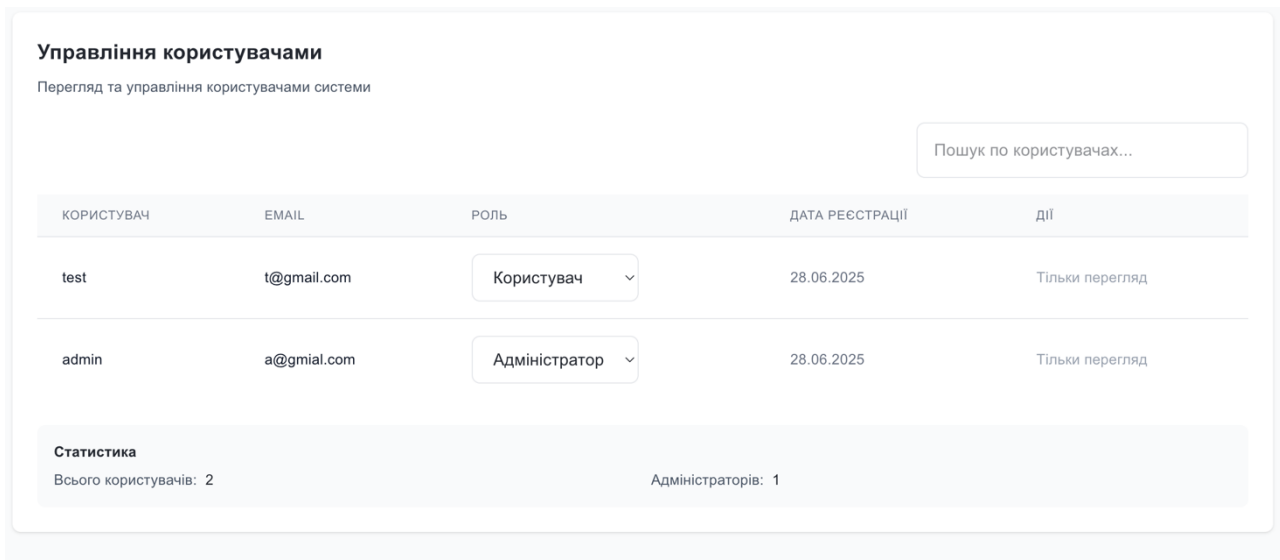


Рисунок 3.17 – Перегляд користувачів на панелі адміністратора.

3.3 Тестування роботи програмного комплексу

Тестування веб-додатку [31] є критично важливим етапом розробки для забезпечення стабільної роботи та високої якості користувацького досвіду. Було проведене комплексне тестування всіх функціональних можливостей системи.

Функціональне тестування

Тест 1: Вхід до облікового запису за email і паролем

Передумови: Користувач ще не авторизований у системі.

Кроки: Він відкриває форму входу, вводить адресу електронної пошти testuser@mail.com та пароль Secure123!, після чого натискає кнопку “Увійти”.

Очікуваний результат: Система повинна впустити користувача до акаунту та показати головну сторінку з категоріями.

Фактичний результат: Користувач потрапляє до свого профілю та бачить головну сторінку з категоріями.

Тест 2: Реєстрація нового користувача

Передумови: Користувач знаходиться на сторінці реєстрації.

Кроки: Він вводить email newuser@mail.com та пароль UserPass2024 у відповідні поля та натискає кнопку підтвердження.

Очікуваний результат: Має бути створено новий обліковий запис, після чого користувача перенаправляє на форму входу.

Фактичний результат: Обліковий запис створено, з'являється форма авторизації.

Тест 3: Фільтрація запчастин за параметрами авто

Передумови: На сторінці доступні фільтри для підбору авто.

Кроки: Користувач обирає Volkswagen як марку, вводить модель Passat, обирає 2014 рік випуску та тип двигуна Diesel, після чого підтверджує фільтрацію.

Очікуваний результат: Сайт має показати категорії товарів, які відповідають обраному авто.

Фактичний результат: Відображаються відповідні блоки з категоріями, наприклад: масла, фільтри, електроніка.

Тест 4: Перехід у категорію “Фільтри”

Передумови: Користувач завершив підбір авто й бачить категорії товарів.

Кроки: Він натискає на категорію “Фільтри”, після чого очікує завантаження сторінки.

Очікуваний результат: Повинна з'явитися вітрина з товарами категорії, включаючи фото, назву та ціну.

Фактичний результат: На сторінці з'являється перелік запчастин з фото, описом та вартістю.

Тест 5: Додавання авто в адмін-панелі

Передумови: Адміністратор перебуває в панелі керування автомобілями.

Кроки: Він заповнює форму з даними Toyota, Corolla, 2017, Hybrid і натискає кнопку додавання.

Очікуваний результат: Нове авто повинно з'явитися в таблиці нижче форми.

Фактичний результат: Запис з новим автомобілем з'являється одразу в таблиці.

Тест 6: Видалення запчастини адміністратором

Передумови: Відкрита сторінка управління запчастинами.

					ІПЗ.КР.Б-121-25-ПЗ	
						56

Кроки: Адміністратор знаходить запчастину з ID 5, натискає кнопку “Видалити” і підтверджує дію.

Очікуваний результат: Запчастина повинна зникнути без перезавантаження сторінки.

Фактичний результат: Товар видаляється з інтерфейсу одразу після підтвердження.

Тест 7: Відображення категорій на головній сторінці

Передумови: Користувач уже авторизований.

Кроки: Він переходить на головну сторінку та переглядає область категорій.

Очікуваний результат: Очікується показ блоків із назвами категорій автозапчастин.

Фактичний результат: Блоки категорій з назвами та зображеннями успішно виводяться на головній сторінці.

Тест 8: Перехід на сторінку товару

Передумови: Користувач відкрив певну категорію запчастин.

Кроки: Він натискає на назву конкретного товару.

Очікуваний результат: Повинна відкритися сторінка з детальною інформацією про товар.

Фактичний результат: Відкривається сторінка з описом, фото і ціною вибраного товару.

Тест 9: Відображення помилки при невірному вході

Передумови: Користувач вводить неправильні облікові дані.

Кроки: Він вводить email wrong@mail.com і пароль wrongpass, потім натискає 'Увійти'.

Очікуваний результат: Система повинна повідомити про помилку авторизації.

Фактичний результат: З'являється повідомлення про неправильні дані.

Тест 10: Перевірка доступу до адмін-панелі без авторизації

Передумови: Користувач не авторизований у системі.

Кроки: Він вводить адресу /admin у браузері.

Очікуваний результат: Повинен бути автоматичний редирект на сторінку входу або повідомлення про заборону доступу.

Фактичний результат: Система перекидає користувача на екран входу без доступу до адмін-панелі.

Висновок до третього розділу

У цьому розділі були визначені технічні характеристики, яким повинен відповідати пристрій користувача для забезпечення стабільної та ефективної роботи веб-застосунку. Розроблено діаграму розгортання, що візуалізує архітектуру системи: клієнтська частина реалізована за допомогою Next.js [18], серверна логіка — на базі Node.js із використанням Prisma для доступу до бази даних PostgreSQL, а також передбачено взаємодію з зовнішніми API для оновлення інформації про автозапчастини.

Докладно описано складові інтерфейсу користувача веб-застосунку: сторінка авторизації, каталог товарів з фільтрами за маркою, роком випуску та типом двигуна [30], сторінка з інформацією про запчастину, кошик, обране, сторінка оформлення замовлення та панель адміністратора для керування контентом.

Проведено комплексне тестування програмного забезпечення, що включає функціональні тести, перевірку інтерфейсу, навантажувальне тестування та перевірку на вразливості.

У результаті виконаної роботи реалізовано повноцінний веб-застосунок для продажу автотоварів, який повністю відповідає функціональним вимогам і забезпечує користувачам швидкий та зручний доступ до необхідних деталей з використанням сучасних веб-технологій.

ВИСНОВКИ

У процесі виконання випускної кваліфікаційної роботи бакалавра було розроблено та реалізовано веб-застосунок, призначений для продажу автотоварів. У рамках дослідження було проведено аналіз предметної області, вивчено функціональні можливості існуючих онлайн-магазинів автозапчастин, що дозволило визначити ключові потреби користувачів та сформулювати основні вимоги до функціоналу майбутнього програмного продукту.

У результаті порівняльного аналізу архітектурних рішень було обрано технологічний стек, що оптимально відповідає вимогам до масштабованості, швидкодії та зручності підтримки. В якості основи клієнтської частини було використано фреймворк Next.js, що дозволяє реалізувати SSR і забезпечити швидке завантаження сторінок, тоді як для серверної логіки застосовано Next.js [25] з ORM Prisma. Дані зберігаються у базі PostgreSQL, що дозволяє гнучко працювати зі складною структурою автотоварів. Передбачена також можливість інтеграції з зовнішніми API для отримання актуальної інформації про запчастини.

Було створено діаграми варіантів використання, компонентів, активностей та структури розгортання. Проведено детальне моделювання основних сценаріїв використання системи, включаючи фільтрацію товарів за параметрами авто, оформлення замовлення, керування обліковим записом, а також адміністрування бази товарів.

Результатом кваліфікаційної роботи став повністю функціональний веб-застосунок, який дозволяє користувачам зручно здійснювати пошук та придбання автозапчастин, а адміністраторам — ефективно керувати вмістом каталогу. Реалізована система є гнучкою до розширення, що відкриває перспективи для подальшого розвитку, зокрема: впровадження системи рекомендацій, синхронізація з обліковими системами СТО, розширення функціоналу кошика, підтримка декількох мов інтерфейсу та адаптація під мобільні пристрої.

					ІПЗ.КР.Б-121-25-ПЗ	
						59

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Next.js – Building Full-Stack Applications [Електронний ресурс]. – Режим доступу: <https://nextjs.org/docs>
2. Prisma ORM Official Docs [Електронний ресурс]. – Режим доступу: <https://www.prisma.io/docs>
3. PostgreSQL: The World's Most Advanced Open Source Relational Database [Електронний ресурс]. – Режим доступу: <https://www.postgresql.org/>
4. MDN Web Docs: REST APIs [Електронний ресурс]. – Режим доступу: <https://developer.mozilla.org/en-US/docs/Glossary/REST>
5. React – Official Documentation [Електронний ресурс]. – Режим доступу: <https://reactjs.org/docs/getting-started.html>
6. Tailwind CSS – Utility-First CSS Framework [Електронний ресурс]. – Режим доступу: <https://tailwindcss.com/docs>
7. GitHub – Autoparts Marketplace App Example [Електронний ресурс]. – Режим доступу: <https://github.com/example/autoparts-app>
8. REST vs GraphQL – A Comparison [Електронний ресурс]. – Режим доступу: <https://www.apollographql.com/blog/graphql-vs-rest/>
9. Supabase vs PostgreSQL: What's the Difference? [Електронний ресурс]. – Режим доступу: <https://supabase.com/blog/supabase-vs-postgresql>
10. Designing a Car Parts E-commerce Architecture [Електронний ресурс]. – Режим доступу: <https://dev.to/articles/car-parts-app-architecture>
11. Axios – Promise Based HTTP Client [Електронний ресурс]. – Режим доступу: <https://axios-http.com/docs/intro>
12. Zod – TypeScript-first schema validation [Електронний ресурс]. – Режим доступу: <https://zod.dev/>
13. React Hook Form Documentation [Електронний ресурс]. – Режим доступу: <https://react-hook-form.com/>
14. PostgreSQL Indexing Best Practices [Електронний ресурс]. – Режим доступу: <https://www.cybertec-postgresql.com/en/indexing-tutorial/>

15. Prisma & PostgreSQL Schema Design [Электронный ресурс]. – Режим доступа: <https://www.prisma.io/docs/concepts/components/prisma-schema>
16. JWT Authentication with Next.js [Электронный ресурс]. – Режим доступа: <https://blog.logrocket.com/jwt-authentication-next-js/>
17. Building a Shopping Cart in Next.js [Электронный ресурс]. – Режим доступа: <https://dev.to/build-nextjs-cart>
18. Internationalization in Next.js [Электронный ресурс]. – Режим доступа: <https://nextjs.org/docs/advanced-features/i18n-routing>
19. PostgreSQL Performance Tuning Guide [Электронный ресурс]. – Режим доступа: <https://www.percona.com/blog/performance-tuning-postgresql/>
20. OWASP Guide to Secure Web Apps [Электронный ресурс]. – Режим доступа: <https://owasp.org/www-project-top-ten/>
21. UX Patterns for Automotive Marketplaces [Электронный ресурс]. – Режим доступа: <https://uxdesign.cc/ux-in-auto-ecommerce>
22. Prisma Relations and Joins Explained [Электронный ресурс]. – Режим доступа: <https://www.prisma.io/docs/concepts/components/prisma-client/relation-queries>
23. SEO Optimization for E-commerce Apps [Электронный ресурс]. – Режим доступа: <https://moz.com/learn/seo/what-is-seo>
24. NextAuth.js – Authentication for Next.js [Электронный ресурс]. – Режим доступа: <https://next-auth.js.org/>
25. File Upload in Next.js using Cloudinary [Электронный ресурс]. – Режим доступа: https://cloudinary.com/blog/nextjs_image_upload
26. Dockerizing a Next.js App with PostgreSQL [Электронный ресурс]. – Режим доступа: <https://blog.docker.com/docker-nextjs-postgres/>
27. CI/CD for Next.js on Vercel [Электронный ресурс]. – Режим доступа: <https://vercel.com/docs/concepts/overview/what-is-vercel>
28. Prisma Migrations and Seed Data [Электронный ресурс]. – Режим доступа: <https://www.prisma.io/docs/guides/database/developing-with-prisma-migrate>

29. Testing a Full-Stack Next.js App [Електронний ресурс]. – Режим доступу: <https://testing-library.com/docs/react-testing-library/intro/>
30. PostgreSQL JSONB for Storing Flexible Data [Електронний ресурс]. – Режим доступу: <https://www.postgresqltutorial.com/postgresql-json/>
31. Implementing Sorting and Filtering with Prisma [Електронний ресурс]. – Режим доступу: <https://www.prisma.io/docs/concepts/components/prisma-client/filtering-and-sorting>
32. Загальна архітектура веб-застосунку автозапчастин на Next.js [Електронний ресурс]. – Режим доступу: <https://example.com/autoparts-app-architecture>

ДОДАТКИ

					ПЗ.КР.Б-121-25-ПЗ	
						63

Лістинг фрагменту коду відповідального за додавання деталей в базу даних:

```

import { NextRequest, NextResponse } from 'next/server'
import { prisma } from '../../../lib/prisma'
import { getServerSession } from 'next-auth'
import { authOptions } from '../../../lib/auth'

export async function GET(request: NextRequest) {
  try {
    const { searchParams } = new URL(request.url)
    const all = searchParams.get('all')
    const brand = searchParams.get('brand')
    const model = searchParams.get('model')
    const year = searchParams.get('year')
    const engine = searchParams.get('engine')
    const category = searchParams.get('category')
    const search = searchParams.get('search') || ''
    const manufacturer = searchParams.get('manufacturer') || ''
    const priceMin = parseFloat(searchParams.get('priceMin') || '0')
    const priceMax = parseFloat(searchParams.get('priceMax') || '100000')
    const sort = searchParams.get('sort') || 'newest'

    // Якщо запитують всі частини (для селекторів), повертаємо без пагінації
    if (all === 'true') {
      const parts = await prisma.part.findMany({
        include: {
          car: true
        },
        orderBy: {
          name: 'asc',
        },
      })
      return NextResponse.json(parts)
    }

    // Для всіх інших запитів застосовуємо всі фільтри разом
    const page = parseInt(searchParams.get('page') || '1')
    const limit = parseInt(searchParams.get('limit') || '10')
    const offset = (page - 1) * limit
  }
}

```

```

// Формуємо where-умову для фільтрів
const where: any = {
  AND: [
    search ? {
      OR: [
        { name: { contains: search, mode: 'insensitive' } },
        { description: { contains: search, mode: 'insensitive' } }
      ]
    } : {},
    category ? { category } : {},
    manufacturer ? { manufacturer } : {},
    { price: { gte: priceMin, lte: priceMax } },
    brand ? { car: { brand } } : {},
    model ? { car: { model } } : {},
    year ? { car: { year: parseInt(year) } } : {},
    engine ? { car: { engineType: engine } } : {},
  ]
}

// Формуємо orderBy для сортування
let orderBy: any = { id: 'desc' }
if (sort === 'price_asc') orderBy = { price: 'asc' }
if (sort === 'price_desc') orderBy = { price: 'desc' }
if (sort === 'name_asc') orderBy = { name: 'asc' }
if (sort === 'name_desc') orderBy = { name: 'desc' }
if (sort === 'newest') orderBy = { id: 'desc' }

const parts = await prisma.part.findMany({
  where,
  skip: offset,
  take: limit,
  include: {
    car: true
  },
  orderBy
})

return NextResponse.json(parts)
} catch (error) {
  console.error('Parts API error:', error)
  return NextResponse.json(
    { error: 'Internal server error' },
    { status: 500 }
  )
}

```

```

    )
  }
}

export async function POST(request: NextRequest) {
  try {
    const session = await getServerSession(authOptions)

    if (!session?.user || (session.user as any).role !== 'admin') {
      return NextResponse.json(
        { error: 'Unauthorized' },
        { status: 401 }
      )
    }

    const { name, description, category, price, imageUrl, carId } = await
request.json()

    // Валідація даних
    if (!name || !description || !category || !price || !imageUrl || !carId)
{
      return NextResponse.json(
        { error: 'Всі поля обов\'язкові' },
        { status: 400 }
      )
    }

    if (price <= 0) {
      return NextResponse.json(
        { error: 'Ціна повинна бути більше 0' },
        { status: 400 }
      )
    }

    // Перевірка чи існує автомобіль
    const car = await prisma.car.findUnique({
      where: { id: parseInt(carId) }
    })

    if (!car) {
      return NextResponse.json(
        { error: 'Автомобіль не знайдено' },
        { status: 404 }
      )
    }
  }
}

```

					ІІЗ.КР.Б-121-25-ІІЗ	
						66

```

    )
  }

  // Перевірка на унікальність назви для цього автомобіля
  const existingPart = await prisma.part.findFirst({
    where: {
      name: name.trim(),
      carId: parseInt(carId)
    }
  })

  if (existingPart) {
    return NextResponse.json(
      { error: 'Деталь з такою назвою вже існує для цього автомобіля' },
      { status: 409 }
    )
  }

  const part = await prisma.part.create({
    data: {
      name,
      description,
      category,
      imageUrl,
      price: parseFloat(price),
      carId: parseInt(carId)
    },
    include: {
      car: true
    }
  })

  if (error?.code === 'P2002') {
    return NextResponse.json(
      { error: 'Деталь з такою назвою вже існує для цього автомобіля' },
      { status: 409 }
    )
  }

  return NextResponse.json(
    {}
  )
}

```

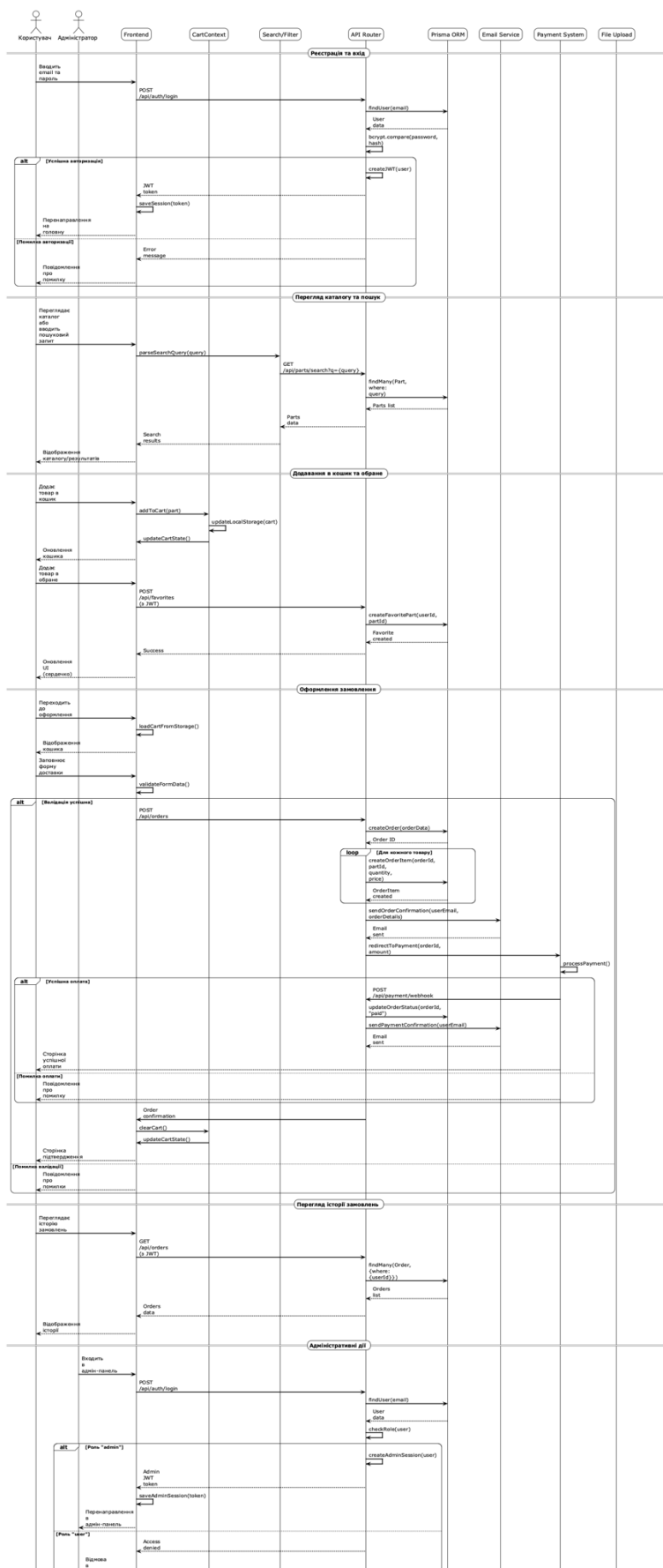


Рис. Загальна діаграма послідовності