МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ ІНТЕРНЕТ-МАГАЗИНУ АВТОТОВАРІВ»**

Виконав: студент 4 курсу, групи 6.1222-з
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)
освітньої програми Комп'ютерні науки
(назва освітньої програми)
Ю.Є. Бдуленко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Миронова Н.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

" 08 " грудня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Бдуленко Юрію Євгеновичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Проєктування та розробка вебзастосунку інтернет-магазину автотоварів

керівник роботи Миронова Наталя Олексіївна, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1742-с

2. Строк подання студентом роботи 15.05.2026

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 вибір технологій
2. Побудова структури вебзастосунку та бази даних
3. Створення та впровадження вебзастосунку

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 05.12.2025

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.	15.12.2025	
2.	Збір вихідних даних.	21.01.2026	
3.	Обробка методичних та теоретичних	16.02.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.03.2026	
5.	Розробка третього розділу.	15.04.2026	
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи бакалавра.	15.05.2026	
7.	Захист кваліфікаційної роботи.	29.05.2026	

Студент _____ Ю.Є.Бдуленко
(підпис) (ініціали та прізвище)

Керівник роботи _____ Н.О.Миронова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г.Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Проєктування та розробка вебзастосунку інтернет-магазину автотоварів»: 105 с., 26 рис., 9 табл., 32 джерела, 3 додатки.

АВТОТОВАРИ, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ЕЛЕКТРОННА КОМЕРЦІЯ, ЗАМОВЛЕННЯ, ІНТЕРНЕТ-МАГАЗИН, КЛІЄНТСЬКА ЧАСТИНА, КОРИСТУВАЧ, СЕРВЕРНА ЧАСТИНА, ТОВАРНИЙ КАТАЛОГ

Об’єкт дослідження – процеси функціонування інтернет-магазину автотоварів в середовищі електронної комерції.

Предмет дослідження – методи, моделі та програмні засоби проєктування і розробки вебзастосунку інтернет-магазину автотоварів з використанням технологій Node.js, Express.js, SQLite, HTML, CSS та JavaScript.

Мета роботи: проєктування та розробка вебзастосунку інтернет-магазину автотоварів, що забезпечує перегляд каталогу, роботу з товарами, користувачами, кошиком, замовленнями та адміністративним керуванням даними.

Методи дослідження – аналіз предметної області, порівняльний аналіз існуючих вебзастосунків, моделювання вимог, UML-моделювання, проєктування бази даних, структурно-функціональний підхід, програмна реалізація, тестування функціональності вебзастосунку.

В кваліфікаційній роботі досліджено особливості функціонування інтернет-магазинів автотоварів, проаналізовано існуючі вебзастосунки відповідного спрямування та визначено основні вимоги до програмного продукту. В процесі виконання роботи, сформовано функціональні та нефункціональні вимоги, розроблено архітектурну структуру вебзастосунку,

побудовано логічну модель бази даних та визначено основні сценарії взаємодії користувачів із системою.

Практична частина роботи передбачає створення вебзастосунку інтернет-магазину автотоварів, що містить клієнтську та серверну частини. Серверну частину реалізовано з використанням Node.js та Express.js, а зберігання даних організовано засобами SQLite. Клієнтська частина створена за допомогою HTML, CSS та JavaScript і забезпечує зручне відображення каталогу, роботу з кошиком, оформлення замовлень та доступ до історії покупок. Додатково реалізовано адміністративну частину для керування товарами та замовленнями.

Результатом роботи є функціональний вебзастосунок, який забезпечує основні операції електронної комерції в сфері продажу автотоварів, підтримує роботу з користувацькими даними, забезпечує обробку замовлень і може бути використаний як основа для подальшого розвитку програмного продукту.

SUMMARY

Bachelor's Qualifying Theses «Design and Development of a Web Application for an Online Auto Parts Store»: 105 pages, 26 figures, 9 tables, 32 references, 3 supplements.

AUTO GOODS, DATABASE, WEB APPLICATION, E-COMMERCE, ORDER, ONLINE STORE, CLIENT SIDE, USER, SERVER SIDE, PRODUCT CATALOG

Object of the study – processes of functioning of an online auto goods store in the e-commerce environment.

Subject of the study – methods, models, and software tools for designing and developing a web application for an online auto goods store using Node.js, Express.js, SQLite, HTML, CSS, and JavaScript technologies.

Aim of the study: design and develop a web application for an online auto goods store that provides catalog browsing, product management, user management, cart functionality, order processing, and administrative data management.

Methods of research – subject area analysis, comparative analysis of existing web applications, requirements modeling, UML modeling, database design, structural and functional approach, software implementation, and functional testing of the web application.

The qualification work examines the features of online auto goods stores, analyzes existing web applications in this field, and defines the main requirements for the software product. During the work, functional and non-functional requirements were formed, the architectural structure of the web application was developed, the logical database model was built, and the main scenarios of user interaction with the system were defined.

The practical part of the work involves the development of a web application for an online auto goods store that includes client-side and server-side components. The server side was implemented using Node.js and Express.js, while data storage

was organized using SQLite. The client side was developed using HTML, CSS, and JavaScript and provides convenient catalog display, cart functionality, order placement, and access to purchase history. In addition, an administrative section was implemented for managing products and orders.

The result of the work is a functional web application that supports the main e-commerce operations in the field of auto goods sales, provides processing of user data and orders, and can be used as a basis for further development of the software product.

ЗМІСТ

Завдання на кваліфікаційну роботу..... 2
4
6
10
1 Аналіз предметної області та вибір технологій12
1.1 Характеристика предметної області інтернет-магазину авто товарів12
1.2 Аналіз існуючих веб-застосунків інтернет-магазинів авто товарів14
1.3 Формування функціональних та нефункціональних вимог до вебзастосунку19
1.4 Обґрунтування вибору технологічного стеку23
1.5 Постановка задачі та визначення основних сценаріїв роботи користувачів25
1.6 27
2 Побудова структури вебзастосунку та бази даних28
28
2.2 Розробка логічної структури бази даних33
37
43
49
2.6 57
3 Створення та впровадження вебзастосунку59
3.1 Розробка серверної частини вебзастосунку з використанням Node.js та Express.js59
3.2 Створення та налаштування бази даних SQLite61
3.3 Впровадження функціоналу роботи з товарами, користувачами та замовленнями65

3.4 Розробка клієнтської частини вебзастосунку засобами HTML, CSS та JavaScript……69
3.5 Забезпечення захисту даних та стабільності роботи вебзастосунку72
3.6 74
75
76
77
Додаток Б Реалізація обробки замовлень у веб-застосунку 94
Додаток В Програмна реалізація клієнтської частини веб-застосунку 99

ВСТУП

Розвиток цифрових технологій та стрімке поширення мережі Інтернет, спричинили суттєві трансформації в сфері торгівлі, що призвело до формування нового етапу розвитку економічних відносин – електронної комерції. В сучасних умовах, інтернет-магазини формують не лише альтернативу традиційній роздрібній торгівлі, але й стають основним каналом взаємодії між продавцем та покупцем. Високий рівень доступності інформаційних технологій, зростання кількості користувачів мережі Інтернет, розвиток логістичної інфраструктури, створюють передумови для активного розвитку онлайн-продажів в різних сферах, зокрема, в сегменті автотоварів.

Актуальність теми дослідження визначається комплексом факторів, пов’язаних з сучасними тенденціями розвитку електронної комерції та специфікою ринку автотоварів. По-перше, зростання кількості транспортних засобів та підвищення середнього віку автомобільного парку зумовлюють стабільний попит на запасні частини, витратні матеріали та аксесуари. Виникає необхідність створення ефективних інструментів для швидкого пошуку, порівняння та придбання відповідної продукції. По-друге, традиційні форми продажу автотоварів не завжди здатні забезпечити достатній рівень доступності асортименту, що особливо актуально для невеликих населених пунктів, де вибір продукції обмежений. Вебзастосунки інтернет-магазинів, дозволяють вирішити зазначену проблему шляхом надання доступу до широкого каталогу товарів незалежно від географічного розташування користувача.

В сфері автотоварів, правильність вибору продукції безпосередньо залежить від точності технічних характеристик, сумісності деталей з конкретними моделями автомобілів та повноти опису. Помилки при підборі можуть призводити до значних фінансових втрат, що зумовлює необхідність створення інформаційних систем, здатних забезпечити коректне представлення даних та підтримку процесу прийняття рішення користувачем.

В процесі використання інтернет-магазинів обробляється значний обсяг персональної інформації користувачів, включно з контактними даними та інформацією про замовлення, що потребує впровадження механізмів захисту даних, контролю доступу та забезпечення цілісності інформації. Одночасно система повинна забезпечувати стабільну роботу навіть за умов значної кількості запитів, що висуває додаткові вимоги до архітектури та реалізації програмного продукту.

Метою кваліфікаційної роботи є проєктування та розробка вебзастосунку інтернет-магазину автотоварів з використанням сучасних веб-технологій, що забезпечує реалізацію основних бізнес-процесів електронної комерції та ефективну взаємодію користувача із системою.

Об’єктом дослідження є процеси функціонування інтернет-магазинів автотоварів в середовищі електронної комерції.

Предметом дослідження є методи та засоби розробки веб-застосунків з використанням технологій Node.js, Express.js, SQLite, HTML, CSS та JavaScript.

Для досягнення поставленої мети потрібно виконати наступні завдання:

– провести аналіз предметної області інтернет-магазинів автотоварів;
– дослідити існуючі вебзастосунки та визначити їх особливості;
– сформувати функціональні та нефункціональні вимоги до системи;
– обґрунтувати вибір технологічного стеку для реалізації вебзастосунку;
– розробити структуру вебзастосунку та бази даних;
– реалізувати серверну частину системи;
– впровадити функціонал роботи з товарами, користувачами та замовленнями;
– розробити клієнтську частину вебзастосунку;
– забезпечити захист даних та стабільність роботи системи.

Практичне значення отриманих результатів полягає в створенні функціонального вебзастосунку інтернет-магазину автотоварів, який може бути використаний як прототип реальної комерційної системи або як основа для подальшого вдосконалення та масштабування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Характеристика предметної області інтернет-магазину авто товарів

Розвиток електронної комерції істотно змінив структуру роздрібної торгівлі в світовому та національному масштабах, перетворивши онлайн-канал на один з провідних способів взаємодії між продавцем та покупцем. За узагальненими оцінками аналітичних агентств, в середині 2020-х років, частка інтернет-продажів в світовому роздрібному обігу перевищила п’яту частину, а сукупний обсяг онлайн-торгівлі обчислюється трильйонами доларів США на рік. Подібна динаміка, свідчить про стійку зміну споживчої поведінки, коли пошук інформації про товар, порівняння пропозицій та безпосереднє придбання, дедалі частіше відбуваються в цифровому середовищі. За даними профільних досліджень, обсяг онлайн-продажів в Україні протягом останніх років зростає стрімкими темпами, а кількість інтернет-покупців обчислюється мільйонами осіб, що формує стійкий попит на дистанційні канали придбання товарів.

Сегмент автотоварів, посідає значне місце в структурі електронної комерції з огляду на специфіку автомобільного ринку та особливості експлуатації транспортних засобів. В європейських країнах, середній вік легкових автомобілів перевищує дванадцять років, що означає, регулярну потребу в технічному обслуговуванні, ремонті та заміні комплектуючих. Подібна ситуація характерна і для України, де значна частина автопарку складається з машин із тривалим терміном використання. За умов постійної експлуатації, попит на запасні частини, витратні матеріали, аксесуари та засоби догляду зберігає стабільний характер, а обсяг ринку післяпродажного обслуговування автомобілів залишається значним. Перехід частини зазначеного попиту в онлайн-середовище, зумовлюється зручністю доступу до широкого

асортименту, можливістю швидкого порівняння пропозицій і скороченням часових витрат на пошук потрібної продукції [1].

Предметна область інтернет-магазину автотоварів вирізняється великою різноманітністю номенклатури та складною структурою асортименту. На подібних торгівельних майданчиках представлено запасні частини для різних систем автомобілів, витратні матеріали, автохімію, аксесуари для салону та кузова, елементи зовнішнього та внутрішнього тюнінгу, інструменти для обслуговування, а також, сезонні товари. Кожна група продукції має власні технічні характеристики, стандарти виготовлення та умови застосування. Особливу роль відіграє сумісність деталей з конкретними марками, моделями, роками випуску та модифікаціями транспортних засобів, оскільки помилка під час добору зазвичай призводить до неможливості використання придбаної позиції за призначенням, тому інформаційний супровід товару набуває не меншого значення, ніж наявність позиції на складі.

Онлайн-формат продажу автотоварів поступово змінює традиційну модель взаємодії між покупцем та продавцем, коли за умов офлайн-торгівлі значна частина консультацій покладалася на персонал торговельної точки, тоді як в цифровому середовищі, функцію первинного інформування виконують описи, характеристики, фотоматеріали та довідкові матеріали. Покупець отримує можливість самостійно порівнювати параметри продукції різних виробників, аналізувати цінові пропозиції та обирати оптимальний варіант без прив’язки до місця розташування магазину. Подібна зміна ролей посилює вимоги до якості представленої інформації та до зручності її сприйняття, оскільки саме від зрозумілості та повноти опису залежить рішення про придбання [2].

Асортимент спеціалізованих офлайн-магазинів в невеликих населених пунктах зазвичай обмежений, тоді як онлайн-платформи забезпечують доступ до значно ширшого вибору продукції незалежно від місця проживання покупця. Логістичні сервіси та служби доставки, компенсують фізичну віддаленість між складом та споживачем, що робить дистанційну форму придбання практично рівноцінною традиційній за рівнем доступності. В поєднанні з поширенням

безготівкових розрахунків і зростанням довіри до електронних платежів, наведені умови сприяють подальшому розширенню частки онлайн-каналу в структурі продажу автотоварів.

Для власників автомобілів, характерним є прагнення отримати деталь або витратний матеріал в найкоротший термін за прийнятною ціною з гарантією відповідності технічним вимогам. Онлайн-середовище надає інструменти для швидкого пошуку за назвою або артикулом, для відбору за технічними параметрами та для порівняння пропозицій різних постачальників. В результаті, формується модель споживання, орієнтована не на прив’язаність до конкретної торговельної точки, а на поєднання ціни, доступності та інформаційної прозорості пропозиції [3].

Інтернет-магазини автотоварів є важливою складовою сучасного ринку автомобільного післяпродажного обслуговування. Стійке зростання електронної комерції в світі та в Україні, значний обсяг попиту на запасні частини та супутню продукцію, а також, потреба споживачів в швидкому доступі до широкого асортименту формують об’єктивні передумови для подальшого розвитку онлайн-каналу в зазначеній сфері. Область продажу автотоварів в цифровому середовищі, характеризується поєднанням великої кількості товарних позицій, високих вимог до інформаційної точності та орієнтації на зручність для покупця, що визначає її специфіку порівняно з багатьма іншими сегментами електронної торгівлі.

1.2 Аналіз існуючих веб-застосунків інтернет-магазинів авто товарів

Ринок електронної комерції в сфері автотоварів, сформував низку спеціалізованих веб-застосунків, що орієнтуються на різні сегменти споживачів, масштаби асортименту та моделі взаємодії з покупцем. Дослідження подібних платформ, дозволяє простежити усталені підходи до організації каталогів, подання технічної інформації та підтримки процесу вибору продукції, а також,

виявити характерні обмеження, пов’язані з перевантаженням інтерфейсу або недостатньою гнучкістю пошукових інструментів.

Платформа Autodoc належить до найбільших європейських онлайн-магазинів автозапчастин і супутньої продукції та орієнтується на масовий сегмент ринку [4]. Головна сторінка вебзастосунку Autodoc представлена на рисунку 1.1.

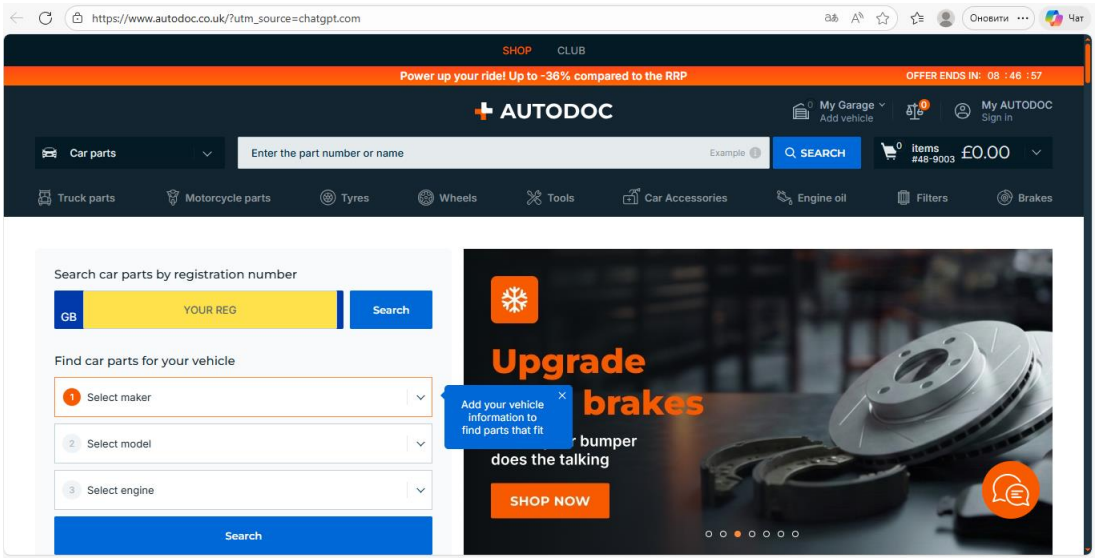


Рисунок 1.1 – Головна сторінка вебзастосунку Autodoc

Зміст головної сторінки побудований навколо швидкого доступу до каталогу, в якому представлено продукцію для легкових автомобілів, мотоциклів і комерційного транспорту. Структура ресурсу передбачає поділ за марками та моделями, а також, використання ідентифікаційних параметрів автомобіля, що дозволяє пов’язати вибір товару з конкретним транспортним засобом. Значна кількість позицій супроводжується технічними характеристиками, фотографіями та додатковими описами, що створює умови для попередньої оцінки відповідності деталі ще до оформлення замовлення. Подібна насиченість інформацією сприяє зниженню ризику помилкового придбання та підвищує впевненість покупця у власному виборі. Разом з тим, інтерфейс ресурсу перевантажений елементами навігації, рекламними блоками та допоміжними модулями, що ускладнює сприйняття сторінок під час першого знайомства з

платформою. Для користувачів без досвіду роботи з великими каталогами автозапчастин, орієнтація в структурі розділів може потребувати додаткового часу, а пошук оптимального варіанта нерідко супроводжується необхідністю багаторазового уточнення параметрів.

Інтернет-магазин OnlineCarParts, також працює на європейському ринку та пропонує широкий вибір автозапчастин і витратних матеріалів [5]. Головна сторінка вебзастосунку OnlineCarParts представлена на рисунку 1.2.

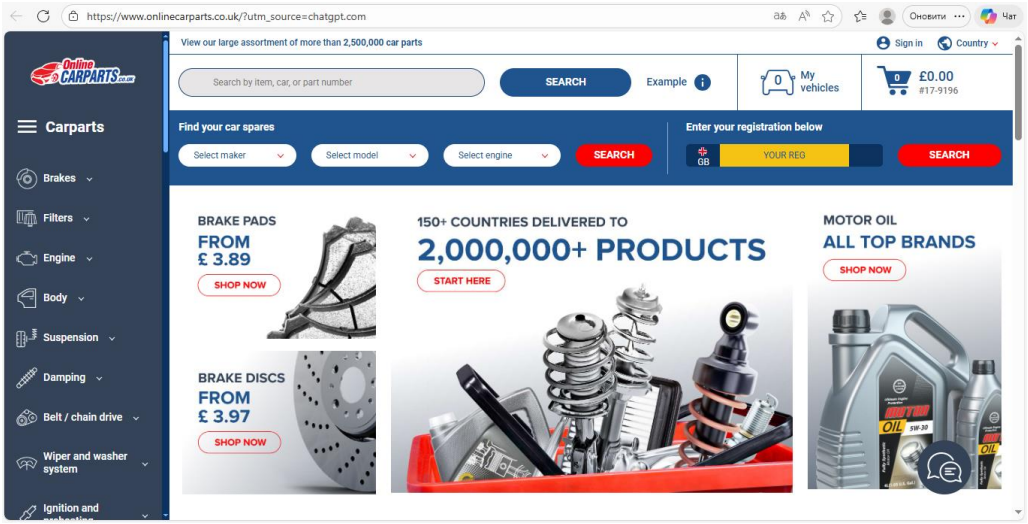


Рисунок 1.2 – Головна сторінка вебзастосунку OnlineCarParts

Зовнішнє оформлення платформи характеризується більш стриманим підходом до подання інформації, де основний акцент зроблено на пошуковому рядку та доступі до каталогу. Навігаційна структура виглядає простішою в порівнянні з великими універсальними майданчиками, що позитивно впливає на швидкість переходу до потрібного розділу. Користувач отримує можливість оперативно перейти до перегляду товарів за базовими параметрами без тривалого ознайомлення з інтерфейсом. Подібна організація сприяє зручності виконання типових дій, пов’язаних з пошуком та переглядом позицій. Водночас, обмежена кількість інструментів деталізованого відбору за технічними характеристиками, знижує ефективність використання ресурсу у випадках, коли необхідно підібрати специфічну деталь для рідкісної модифікації автомобіля. За відсутності точного артикулу або чіткого уявлення про потрібні параметри

процес вибору може вимагати значно більшої кількості кроків та переходів між розділами.

Український ринок представлений, зокрема, веб-застосунком АвтоТО, що спеціалізується на продажі оригінальних запчастин і аналогів для широкого переліку марок автомобілів [6]. Головна сторінка вебзастосунку АвтоТО представлено на рисунку 1.3.

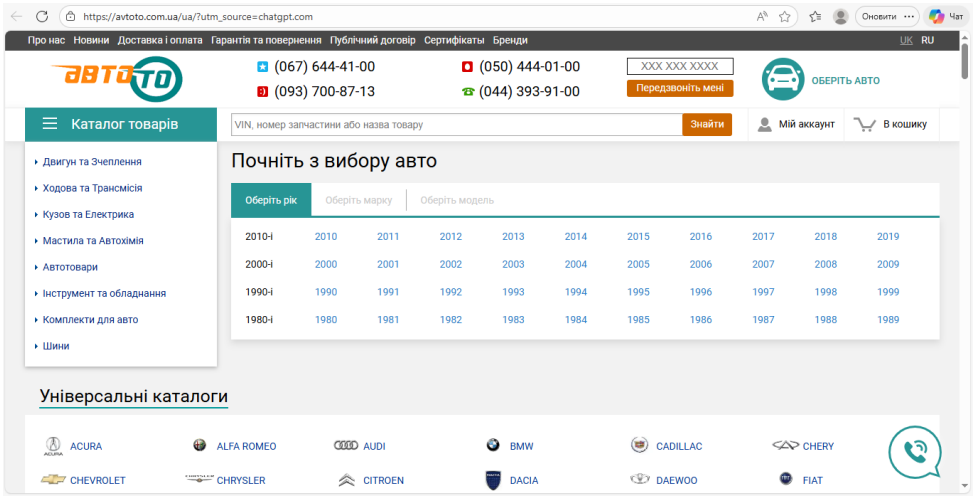


Рисунок 1.3 – Головна сторінка вебзастосунку АвтоТО

Характерною рисою зазначеного ресурсу, виступає орієнтація на точність добору продукції, що реалізується через використання ідентифікаційних параметрів автомобіля, зокрема пошуку за VIN-кодом. Подібний підхід дозволяє пов’язати запит покупця з конкретною конфігурацією транспортного засобу та істотно зменшити ймовірність невідповідності придбаної деталі. Каталог супроводжується технічними описами та базовими візуальними матеріалами, що формує достатній рівень інформативності для прийняття рішення про купівлю. Разом з тим, візуальне оформлення сторінок та логіка розміщення елементів інтерфейсу орієнтовані переважно на користувачів з попереднім досвідом пошуку автозапчастин. Для менш підготовленої аудиторії, процес навігації може здаватися складним, оскільки частина функціональних можливостей не має очевидних візуальних підказок, а загальна структура сторінок тяжіє до

утилітарного стилю з мінімальним акцентом на зручність первинного ознайомлення.

Платформа Signeda виступає прикладом ресурсу, орієнтованого на надзвичайно широкий асортимент кузовних деталей, елементів підвіски та супутньої продукції [7]. Головна сторінка вебзастосунку Signeda представлена на рисунку 1.4.

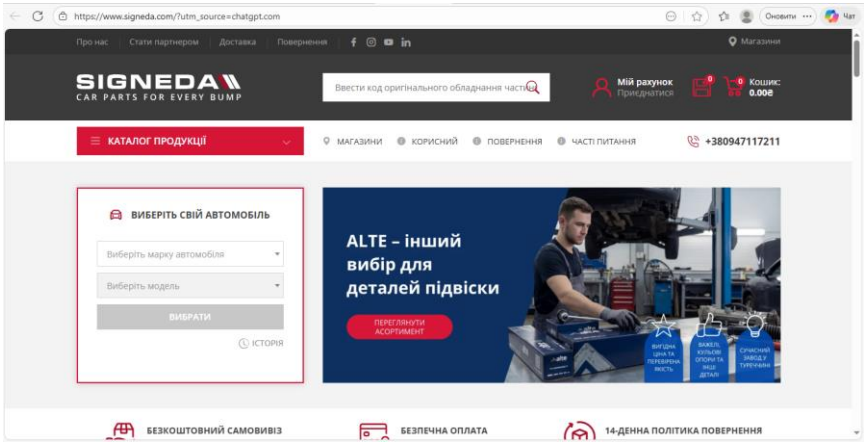


Рисунок 1.4 – Головна сторінка вебзастосунку Signeda

Масштаб каталогу створює передумови для охоплення значної частини потреб власників різних марок та моделей автомобілів, а наявність великої кількості товарних позицій, дозволяє розглядати платформу як універсальний майданчик для пошуку комплектуючих. Інформаційне наповнення сторінок, включно з описами та зображеннями, сприяє попередньому ознайомленню з продукцією та порівнянню пропозицій. Водночас, значний обсяг асортименту за відсутності достатньо гнучких механізмів деталізованої фільтрації ускладнює процес пошуку для користувачів, які не мають точних ідентифікаційних даних потрібної деталі. В подібних ситуаціях, вибір може перетворюватися на послідовність багаторазових уточнень і переходів між розділами, що знижує загальну зручність роботи з ресурсом.

Результати аналізу розглянутих вебзастосунків, свідчить про наявність спільних тенденцій в сфері онлайн-торгівлі автотоварами, пов’язаних з прагненням охопити максимально широкий асортимент і забезпечити доступ до

нього через цифровий каталог. Помітними залишаються відмінності в підходах до організації навігації, подання технічної інформації та підтримки процесу добору продукції. Частина платформ робить акцент на точності підбору через використання ідентифікаційних параметрів автомобіля, інші орієнтуються на спрощення інтерфейсу та швидкий доступ до каталогу.

1.3 Формування функціональних та нефункціональних вимог до вебзастосунку

Формування вимог в контексті предметної області інтернет-магазину автотоварів, ґрунтується на необхідності забезпечити повний цикл онлайн-покупки, коректну інформаційну підтримку вибору продукції, надійне опрацювання замовлень та керування асортиментом. Вимоги поділяються на функціональні, що визначають склад операцій і поведінку системи, та нефункціональні, що регламентують властивості якості, безпеки, продуктивності, надійності й зручності використання. Узгоджений набір вимог, забезпечує однозначність подальшої реалізації та створює основу для контролю відповідності програмного продукту очікуванням користувачів та бізнес-процесам онлайн-торгівлі.

Функціональні вимоги охоплюють користувацькі можливості для неавторизованого відвідувача, зареєстрованого покупця та адміністративної ролі [8]. Для відвідувача, передбачається доступ до каталогу з переглядом категорій і підкатегорій, відкриттям картки товару з описом, характеристиками, ціною, наявністю та фотоматеріалами, а також, можливість виконувати пошук за назвою або артикулом і застосовувати фільтрацію за ключовими параметрами, що підтримують добір. Додатково визначається підтримка сортування результатів за ціною, популярністю або новизною, що підвищує керованість перегляду асортименту при великій кількості позицій. Функції роботи з кошиком включають додавання товарів, зміну кількості, видалення позицій, обчислення

проміжної суми замовлення з урахуванням кількості, збереження стану кошика під час навігації сайтом.

Для зареєстрованого покупця, вводиться вимога реєстрації облікового запису з перевіркою унікальності електронної пошти, а також, автентифікації з підтримкою сесійної роботи. Профіль користувача має забезпечувати перегляд і зміну контактних даних, а історія замовлень повинна відображати створені заявки з датою, складом, сумою та станом виконання. Процедура оформлення замовлення визначається як послідовність операцій, де користувач підтверджує склад кошика, вводить дані отримувача, обирає спосіб доставки та оплати, після чого отримує підтвердження про створення замовлення з присвоєнням ідентифікатора. Для підвищення якості сервісу до функціональних вимог також належить інформування користувача про зміни статусу замовлення в персональному кабінеті, а за наявності відповідних технічних можливостей – через повідомлення електронною поштою.

Адміністративна функціональність визначається необхідністю керувати довідниками та операційними даними магазину. Роль адміністратора повинна мати доступ до створення, редагування та деактивації товарів, керування категоріями та брендами, завантаження та оновлення зображень товарів, встановлення цін та наявності [9]. Обробка замовлень в адміністративній частині включає перегляд переліку заявок, деталізацію складу, зміну статусів виконання, фіксацію коментарів для внутрішнього використання та контроль загальних показників за період.

Нефункціональні вимоги формують цільові показники якості та обмеження, дотримання яких забезпечує стабільну роботу вебзастосунку. Вимоги до продуктивності передбачають швидке завантаження основних сторінок і результатів пошуку, коректну роботу фільтрації на типовому обсязі каталогу та стабільну обробку паралельних запитів. Вимоги до надійності охоплюють стійкість до помилок введення, коректне відновлення після збоїв на рівні запиту, узгодженість даних замовлення та кошика під час виконання критичних операцій. Вимоги до безпеки включають зберігання паролів лише у

вигляді криптографічних хешів, захист сесій і токенів автентифікації, контроль доступу до адміністративних маршрутів, протидію типовим вебзагрозам, базову валідацію вхідних даних для зменшення ризиків ін’єкцій та підміни параметрів. Вимоги до зручності використання передбачають логічну навігацію, передбачувану поведінку елементів інтерфейсу, читабельне подання характеристик товарів, зрозумілий шлях від пошуку до оформлення замовлення, коректну адаптацію інтерфейсу для різних розмірів екрана. Вимоги до сумісності визначають підтримку сучасних браузерів і коректну роботу без залежності від вузькоспеціалізованих плагінів. Вимоги до супроводжуваності охоплюють структурованість серверної частини, відокремлення логіки доступу до даних, зрозумілий підхід до конфігурації середовища та можливість подальшого розширення функцій без руйнування базових сценаріїв. Діаграма варіантів використання представлена на рисунку 1.5

Наведена діаграма варіантів використання відображає основні напрями взаємодії між користувачами та функціональними можливостями вебзастосунку в рамках інтернет-торгівлі автотоварами. Виокремлення ролей відвідувача, покупця та адміністратора дозволяє зафіксувати різні рівні доступу до функцій системи та підкреслити відмінності між інформаційним переглядом асортименту, здійсненням покупок та керуванням даними магазину. Подібне розмежування ролей забезпечує логічну узгодженість сценаріїв та створює основу для подальшого проєктування механізмів автентифікації та авторизації.

Сценарії, пов’язані з роботою каталогу, пошуком та фільтрацією продукції, відображають базовий рівень взаємодії, що передує прийняттю рішення про купівлю. Включення варіантів використання, пов’язаних з керуванням кошиком та оформленням замовлення, демонструє перехід від інформаційної взаємодії до виконання транзакційних операцій. Подібна послідовність відповідає типовій логіці поведінки покупця в онлайн-середовищі, де попередній перегляд і порівняння пропозицій поступово змінюються етапами підтвердження вибору та фіксації наміру придбання [10].



Рисунок 1.5 – Діаграма варіантів використання вебзастосунку інтернет-магазину автотоварів

Окремий блок сценаріїв, пов’язаний з профілем користувача та історією замовлень, відображає сервісну складову взаємодії після завершення покупки. Наявність подібних можливостей, формує довготривалі відносини між покупцем і торговельною платформою, оскільки забезпечує доступ до накопиченої інформації та спрощує повторні звернення до сервісу. Адміністративні сценарії, в свою чергу, фіксують внутрішні процеси керування асортиментом та замовленнями, що дозволяє розглядати вебзастосунок не лише як інтерфейс для покупця, а й як інструмент організації операційної діяльності магазину.

1.4 Обґрунтування вибору технологічного стеку

Вибір технологічного стеку для створення вебзастосунку інтернет-магазину автотоварів визначається необхідністю поєднати інформативне подання великого за обсягом каталогу, стабільну обробку користувацьких запитів та надійне зберігання даних в контексті єдиної узгодженої програмної платформи. Особливого значення набувають швидкість відгуку інтерфейсу, передбачуваність поведінки сервісу під час виконання типових операцій, можливість підтримувати та розширювати функціональність без радикальної перебудови архітектури. Доцільним є застосування поширених та добре апробованих технологій, що мають широку екосистему інструментів, стабільну документаційну підтримку та значний досвід використання у веб-проєктах торговельного спрямування.

Клієнтський рівень вебзастосунку орієнтується на відображення структури каталогу, представлення карток товарів, забезпечення пошуку та фільтрації, підтримку інтерактивних елементів взаємодії з користувачем. Реалізація зазначених завдань, потребує чіткого розмежування між структурою сторінок, їх візуальним оформленням та динамічною поведінкою. Саме тому, доцільним є використання поєднання HTML, CSS та JavaScript, де кожна технологія виконує визначену роль у формуванні інтерфейсу та забезпеченні коректної взаємодії між користувачем і системою. Узагальнені характеристики технологій клієнтського рівня подано в таблиці 1.1.

Таблиця 1.1 – Характеристика технологій клієнтського рівня

Технологія	Основне призначення	Ключові можливості	Роль у вебзастосунку
HTML	Структурування вмісту сторінок	Семантична розмітка, ієрархія елементів, підтримка доступності	Формує каркас сторінок каталогу, карток товарів і сервісних розділів
CSS	Візуальне оформлення	Керування макетом, адаптивність, типографіка, компонування	Забезпечує цілісне стилістичне подання та коректне відображення на різних екранах
JavaScript	Інтерактивна поведінка	Обробка подій, асинхронні запити,	Реалізує пошук, фільтрацію, роботу з кошиком і перевірку

		динамічне оновлення інтерфейсу	введених даних

Застосування HTML дозволяє побудувати семантично коректну структуру сторінок, що полегшує підтримку проєкту та сприяє зрозумілості розмітки для подальшого супроводу. CSS відповідає за формування візуальної ідентичності ресурсу, забезпечуючи єдині принципи оформлення та адаптацію до різних розмірів екрана, що має суттєве значення з огляду на поширеність мобільного доступу до інтернет-магазинів. JavaScript, надає можливість реалізувати динамічну поведінку інтерфейсу, що проявляється у швидкому оновленні результатів пошуку, зміні стану кошика без перезавантаження сторінки та взаємодії з сервером у фоновому режимі. Сукупне використання зазначених технологій формує клієнтський рівень, орієнтований на зручність сприйняття інформації та ефективність виконання типових дій користувача.

Серверний рівень вебзастосунку, покликаний забезпечити обробку HTTP-запитів, реалізацію бізнес-логіки, контроль доступу та взаємодію з базою даних. Для виконання зазначених функцій доцільним є використання середовища Node.js [10] разом з фреймворком Express.js, що надає лаконічні засоби організації маршрутів і проміжної обробки запитів. Зберігання структурованих даних пропонується реалізувати за допомогою SQLite [11-12], що відповідає вимогам до простоти розгортання та достатньої продуктивності для проєктів з помірним навантаженням. Узагальнені характеристики серверних технологій наведено в таблиці 1.2.

Таблиця 1.2 – Характеристика серверних технологій

Технологія	Призначення	Характерні властивості	Застосування у проєкті
Node.js	Середовище виконання JavaScript на сервері	Подієва модель, неблокувальний ввід-вивід, єдина мова для клієнта і сервера	Обробка запитів, реалізація бізнес-логіки, взаємодія з базою даних
Express.js	Вебфреймворк для Node.js	Спрощена маршрутизація, підтримка проміжного ПЗ, модульна організація	Побудова API, керування маршрутами та доступом
SQLite	Вбудована реляційна база даних	Файлова модель зберігання, простота налаштування,	Зберігання відомостей про товари, користувачів

		підтримка SQL	і замовлення

Використання Node.js на серверному рівні забезпечує уніфікацію мови програмування в межах проєкту, що спрощує підтримку коду та знижує складність взаємодії між клієнтською та серверною частинами. Express.js доповнює зазначене середовище чіткою структурою маршрутизації та можливістю організації проміжної логіки, пов’язаної з перевіркою доступу та обробкою даних запитів. SQLite, в свою чергу, відповідає потребам у зберіганні структурованих даних без необхідності розгортання окремого серверного рішення для керування базою даних [12]. Відповідність вимог предметної області та можливостей технологічного стеку наведено в таблиці 1.3.

Таблиця 1.3 – Відповідність вимог предметної області та можливостей технологічного стеку

Вимога	Пояснення відповідності
Подання великого каталогу продукції	HTML і CSS формують структуроване та адаптивне відображення, JavaScript забезпечує динамічну взаємодію
Підтримка пошуку та фільтрації	JavaScript реалізує оновлення інтерфейсу без перезавантаження, Node.js обробляє запити до даних
Обробка замовлень і керування станами	Express.js організовує маршрути та логіку, SQLite зберігає транзакційні відомості
Простота розгортання та супроводження	Файлова модель SQLite та екосистема Node.js зменшують складність інфраструктури
Можливість розвитку функціональності	Модульна організація серверної частини дозволяє розширювати проєкт без порушення базової логіки

Поєднання HTML, CSS і JavaScript забезпечує передбачувану та адаптивну поведінку клієнтського рівня, тоді як зв’язка Node.js з Express.js разом з SQLite, формує серверну основу, орієнтовану на стабільну обробку запитів і збереження даних.

1.5 Постановка задачі та визначення основних сценаріїв роботи користувачів

Метою виконання роботи є створення вебзастосунку інтернет-магазину автотоварів, призначеного для підтримки процесів онлайн-продажу запасних частин, витратних матеріалів та супутньої продукції для автомобілів. Зміст поставленої задачі полягає в побудові програмного рішення, що забезпечує впорядковане подання каталогу, коректну обробку запитів користувачів, формування та супровід замовлень, централізоване зберігання та актуалізацію даних про товари, користувачів і операції купівлі. Вебзастосунок має функціонувати як єдина інформаційна система, де поєднуються клієнтський інтерфейс та серверна логіка з використанням реляційної моделі даних.

Постановка задачі передбачає реалізацію доступу до каталогу автотоварів з можливістю перегляду категорій та окремих позицій, отримання детальної інформації про характеристики та вартість продукції, виконання пошуку за назвою або ідентифікаційними ознаками і застосування відбору за визначеними параметрами. Окрему частину функціональності становить формування переліку вибраних товарів для подальшого оформлення замовлення, фіксація складу покупки та забезпечення коректного обчислення підсумкової вартості. Вебзастосунок має підтримувати реєстрацію та автентифікацію користувачів, збереження відомостей про створені замовлення і надання доступу до інформації про їх поточний стан та історію виконання.

Визначення основних сценаріїв роботи користувачів ґрунтується на поділі ролей і функцій в межах системи. Відвідувач взаємодіє з вебзастосунком через перегляд асортименту, ознайомлення з картками товарів і використання пошукових засобів для уточнення запитів. Покупець, який пройшов процедуру автентифікації, додатково виконує операції, пов’язані з формуванням кошика, підтвердженням вибраних позицій, введенням даних отримувача та створенням замовлення з подальшим контролем перебігу його виконання. Адміністративний користувач працює з системою через сценарії, пов’язані з наповненням і коригуванням каталогу, підтриманням актуальності цін і наявності, опрацюванням замовлень з фіксацією змін їх станів відповідно до етапів обробки.

Сформульована задача визначає межі функціональності програмного продукту та задає напрям подальших етапів проєктування та реалізації. Чітке окреслення сценаріїв взаємодії користувачів з вебзастосунком забезпечує узгодженість між вимогами до системи, структурою майбутнього рішення та практичною реалізацією інтерфейсу і серверної частини, що створює підґрунтя для побудови цілісної та послідовної системи онлайн-продажу автотоварів.

1.6 Висновки до розділу 1

В процесі виконання першого розділу, було здійснено ґрунтовний аналіз предметної області інтернет-магазинів автотоварів, що дозволило визначити специфіку даного сегмента електронної комерції, пов’язану з широким асортиментом продукції, необхідністю врахування технічної сумісності та високими вимогами до інформаційного наповнення. Дослідження існуючих вебзастосунків дало можливість виявити основні підходи до організації каталогів, навігації та пошуку товарів, а також встановити наявні недоліки, пов’язані зі складністю інтерфейсу або недостатньою гнучкістю фільтрації.

Сформовано функціональні та нефункціональні вимоги, які визначають поведінку системи, її продуктивність, безпеку та зручність використання. Обґрунтовано вибір технологічного стеку, який забезпечує ефективну реалізацію вебзастосунку. Отримані результати створюють основу для подальшого проєктування структури системи та її реалізації.

2 ПОБУДОВА СТРУКТУРИ ВЕБЗАСТОСУНКУ ТА БАЗИ ДАНИХ

2.1 Опис архітектурної організації вебзастосунку інтернет-магазину

Архітектурна організація вебзастосунку інтернет-магазину автотоварів визначає спосіб взаємодії між основними складовими системи, порядок обробки запитів користувачів та принципи розподілу відповідальності між клієнтською та серверною частинами. Побудова архітектури спрямована на забезпечення узгодженості між інтерфейсом, бізнес-логікою та рівнем зберігання даних, що дозволяє досягти стабільної роботи системи під час виконання типових сценаріїв, пов’язаних з переглядом каталогу, пошуком продукції та оформленням замовлень.

Вебзастосунок інтернет-магазину автотоварів, варто реалізовувати на основі клієнт-серверної моделі, де браузер користувача виступає як середовище відображення інтерфейсу, тоді як сервер відповідає за обробку запитів, виконання бізнес-логіки та взаємодію з базою даних. Подібна організація дозволяє відокремити процеси відображення інформації від процесів її обробки, що сприяє підвищенню гнучкості та керованості системи. Клієнтська частина забезпечує формування запитів до сервера, обробку відповідей та оновлення інтерфейсу без необхідності повного перезавантаження сторінок, що позитивно впливає на швидкість взаємодії користувача з системою [14].

Серверна частина побудована з використанням середовища Node.js та фреймворку Express.js, що дозволяє організувати маршрутизацію запитів, реалізувати логіку обробки даних та забезпечити взаємодію з базою даних SQLite. В серверній частині, варто виділити декілька логічних рівнів, серед яких рівень маршрутизації, рівень бізнес-логіки та рівень доступу до даних. Рівень маршрутизації відповідає за прийом HTTP-запитів та їх спрямування до відповідних обробників, бізнес-логіка виконує перевірку коректності вхідних даних, обчислення вартості замовлення та інші операції, пов’язані з обробкою

запитів, а рівень доступу до даних забезпечує взаємодію з базою даних, включно з виконанням запитів на вибірку, додавання та оновлення записів.

База даних SQLite використовується як централізоване сховище інформації про товари, користувачів та замовлення [15]. Вона забезпечує збереження структурованих даних у вигляді таблиць з визначеними зв’язками, що дозволяє ефективно виконувати операції пошуку, фільтрації та агрегування інформації. Використання вбудованої реляційної системи керування базами даних спрощує розгортання вебзастосунку та зменшує вимоги до інфраструктури, що є доцільним для проєктів з помірним навантаженням. Діаграма компонентів представлена на рисунку 2.1.

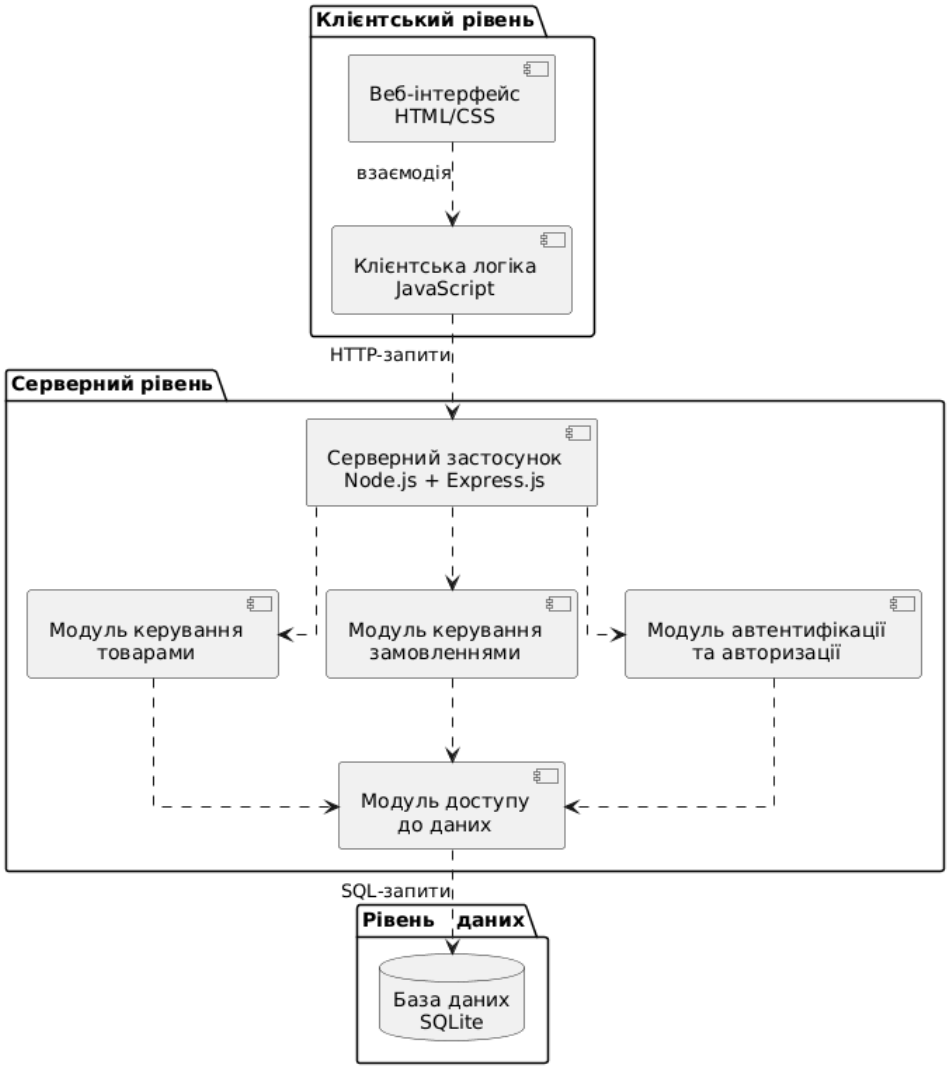


Рисунок 2.1 – Діаграма компонентів вебзастосунку інтернет-магазину автотоварів

Діаграма компонентів відображає внутрішню логіку побудови вебзастосунку та демонструє, яким чином функціональні частини системи взаємодіють одна з одною під час виконання запитів користувача. В структурі програмного рішення, чітко простежується розподіл на клієнтський рівень, серверний рівень та рівень зберігання даних. Дане архітектурне розмежування, має принципове значення для інтернет-магазину автотоварів, оскільки система повинна одночасно забезпечувати зручне відображення великого каталогу, підтримувати запити на пошук і фільтрацію, опрацьовувати операції з кошиком і замовленнями та зберігати значний обсяг структурованої інформації без втрати цілісності.

Клієнтська частина охоплює засоби візуального подання інтерфейсу та модулі, що відповідають за динамічну поведінку сторінок в браузері користувача. Розмітка HTML формує семантичний каркас сторінок, стилі CSS забезпечують зовнішнє оформлення, а JavaScript реалізує інтерактивні механізми, пов’язані з пошуком товарів, зміною вмісту кошика, перевіркою введених даних та надсиланням асинхронних запитів до сервера, що дозволяє не перевантажувати браузер надмірною логікою опрацювання даних і водночас, підтримувати достатній рівень швидкодії під час виконання стандартних дій користувача. Для інтернет-магазину автомобільних товарів, приведена властивість є особливо важливою, оскільки користувач часто працює з великою кількістю товарних позицій, переходить між категоріями, застосовує фільтри та очікує негайного оновлення результатів [16].

Серверна частина на діаграмі компонентів представлена трьома взаємопов’язаними модулями, серед яких виокремлюються маршрутизація, бізнес-логіка та доступ до даних. Маршрутизація виконує функцію прийому HTTP-запитів та спрямування їх до відповідних обробників залежно від призначення запиту. Саме на рівні маршрутів визначається, чи стосується звернення користувача перегляду каталогу, відкриття картки товару, авторизації, додавання позиції до кошика або оформлення замовлення. Після надходження запиту керування передається до рівня бізнес-логіки, де виконується основне

опрацювання. На зазначеному рівні, відбувається перевірка коректності вхідних даних, аналіз прав доступу, формування правил обробки запиту, обчислення підсумкової суми замовлення, а також, підготовка відповіді, що буде повернута до клієнтської частини [17].

Рівень доступу до даних відіграє роль проміжної ланки між бізнес-логікою та базою даних. Його призначення полягає у формуванні запитів до таблиць, зчитуванні результатів, зміні наявних записів та створенні нових сутностей. Дане відокремлення дозволяє уникнути прямого змішування прикладної логіки з SQL-операціями, що підвищує структурованість програмного коду й полегшує його подальший супровід. Для вебзастосунку інтернет-магазину автотоварів наведене рішення є виправданим, оскільки система працює з різними групами даних, серед яких товари, категорії, бренди, користувачі, кошики та замовлення. Якщо логіка роботи з базою даних не ізольована в окремому модулі, ускладнюється підтримка цілісності системи та зростає ймовірність помилок під час розширення функціональності.

Важливою перевагою компонентної організації є підвищення модульності програмного продукту. Кожен блок виконує обмежене коло функцій, що дозволяє локалізувати зміни в межах окремої частини системи без критичного впливу на інші складові. Наприклад, модифікація правил пошуку товарів або уточнення механізму фільтрації можуть бути реалізовані на рівні бізнес-логіки без суттєвого втручання у структуру клієнтського інтерфейсу. Аналогічно, зміни в способі доступу до бази даних не потребують перебудови маршрутизації, якщо зовнішній інтерфейс взаємодії між модулями зберігає узгоджену структуру. Для системи торговельного призначення, подібна властивість має практичну цінність, оскільки інтернет-магазин зазвичай розвивається поступово, спочатку реалізується базовий набір функцій, а надалі додаються нові сервіси, пов’язані з обробкою замовлень, аналітикою, персоналізацією або інтеграцією з зовнішніми платформами [18].

Концентрація перевірки прав доступу, валідації вхідних параметрів та контролю коректності операцій на серверному рівні знижує ризик некоректного

опрацювання запитів, що надходять з клієнтської частини. Для інтернет-магазину автотоварів, представлене рішення особливо важливе в сценаріях, пов’язаних з авторизацією, зміною персональних даних, керуванням товарами та оформленням замовлень. Окрім того, ізольований доступ до бази даних унеможливлює прямий вплив браузерної частини на структуру збережених відомостей, що позитивно впливає на цілісність інформаційного середовища.

Поряд з логічною організацією програмного рішення не менш важливим є уявлення про фізичне розташування його складових. Навіть за умови добре структурованої компонентної побудови система повинна функціонувати в реальному середовищі, де клієнтський пристрій, веб-сервер та база даних взаємодіють через визначені канали обміну даними. Діаграма розгортання вебзастосунку інтернет-магазину наведена на рисунку 2.2.

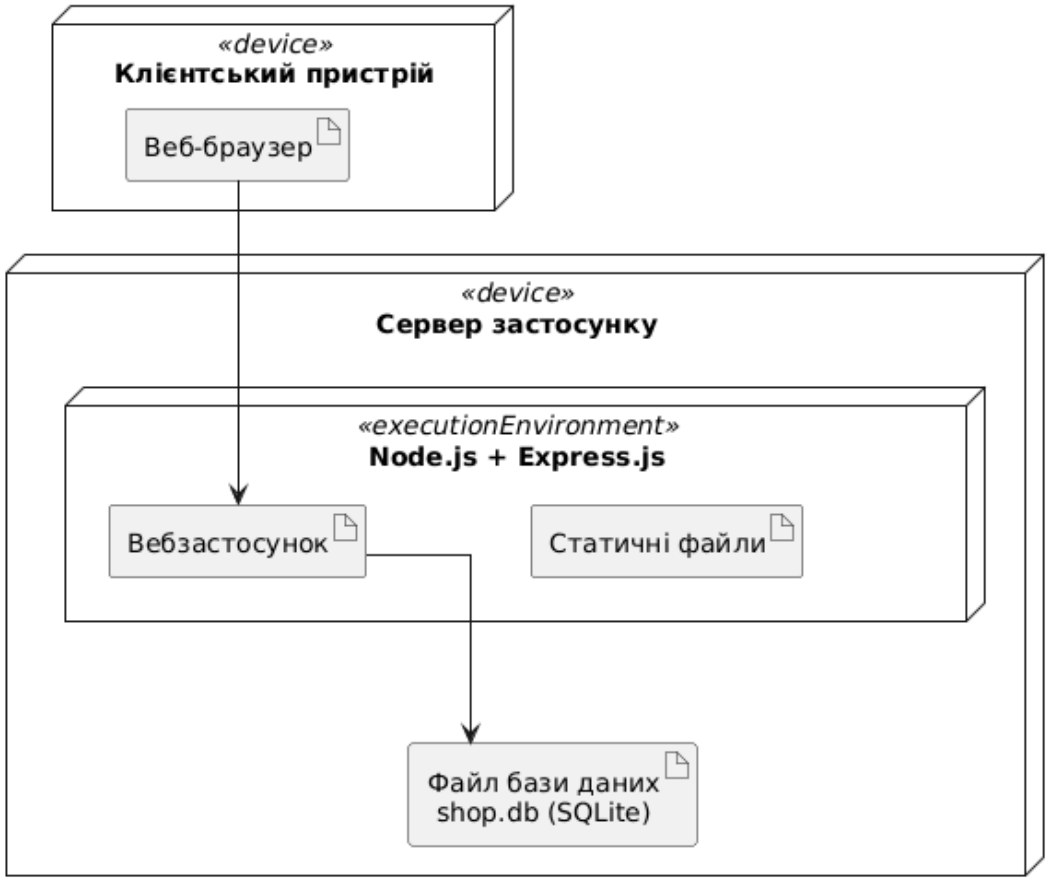


Рисунок 2.2 – Діаграма розгортання вебзастосунку інтернет-магазину автотоварів

Діаграма розгортання відображає фізичну структуру системи, де клієнтський пристрій взаємодіє з сервером через мережеві протоколи, а сервер забезпечує доступ до бази даних, що дозволяє централізувати обробку запитів і зберігання інформації, та підвищує керованість системи та спрощує контроль за її роботою. Взаємодія між клієнтом та сервером, здійснюється за допомогою HTTP або HTTPS, що забезпечує передачу даних в стандартизованому форматі, тоді як обмін між сервером і базою даних відбувається через виконання SQL-запитів [19].

Логічна структура визначає розподіл функцій між компонентами, тоді як фізична структура відображає спосіб їх розгортання та взаємодії в реальному середовищі. Узгодженість зазначених рівнів дозволяє забезпечити передбачувану поведінку системи, мінімізувати затримки під час обробки запитів та створити умови для подальшого масштабування у разі зростання навантаження.

Загальна архітектурна організація вебзастосунку інтернет-магазину автотоварів базується на чіткому розмежуванні функцій між клієнтською частиною, серверною логікою та рівнем зберігання даних.

2.2 Розробка логічної структури бази даних

Формування логічної структури бази даних для вебзастосунку інтернет-магазину автотоварів є ключовим етапом проєктування, оскільки саме від коректності побудови моделі даних залежить стабільність функціонування системи, узгодженість інформації та ефективність виконання запитів. База даних виконує функцію централізованого сховища, в якому накопичуються відомості про асортимент продукції, користувачів, їх взаємодію з системою та результати здійснених транзакцій. Її структура повинна забезпечувати не лише збереження даних, але й можливість їх швидкого отримання, оновлення та контролю цілісності.

Побудова логічної моделі здійснюється на основі аналізу предметної області, що включає процеси продажу автотоварів, структуру каталогу, взаємодію користувачів з вебзастосунком та механізм оформлення замовлень. Особливістю даної предметної області є наявність значної кількості товарних позицій з різними характеристиками, а також потреба в гнучкому механізмі фільтрації та пошуку, що обумовлює необхідність створення структури даних, в якій інформація про товари організована таким чином, щоб забезпечити ефективну обробку запитів навіть за умови збільшення обсягу каталогу.

Логічна структура бази даних формується через виділення основних сутностей, що відображають об’єкти реального середовища, та встановлення зв’язків між ними відповідно до характеру їх взаємодії. До таких сутностей належать категорії, товари, користувачі, замовлення та позиції замовлення. Кожна з зазначених сутностей, має власний набір атрибутів, що описують її властивості, а також бере участь у визначених зв’язках із іншими сутностями [20].

Сутність категорії призначена для структуризації асортименту продукції. Вона дозволяє об’єднувати товари за функціональним призначенням або типом, що значно спрощує навігацію по каталогу. В рамках логічної моделі, кожна категорія має унікальний ідентифікатор і назву, що використовується для відображення в користувацькому інтерфейсі. Взаємодія між категоріями і товарами будується за принципом ієрархічної залежності, де одна категорія може включати велику кількість товарів.

Сутність товару є центральною в структурі бази даних, оскільки саме вона відображає основний об’єкт продажу. Атрибути товару повинні містити не лише базову інформацію, таку як назва, опис і ціна, але й додаткові характеристики, що мають значення для предметної області автотоварів. До даних характеристик можуть належати бренд, сумісність з моделями автомобілів, технічні параметри та інші властивості, що впливають на вибір користувача. Наявність поля, яке відповідає за кількість доступних одиниць, дозволяє контролювати складські

залишки та уникати ситуацій, коли замовлення оформлюється на відсутній товар.

Сутність користувача забезпечує збереження інформації про осіб, які взаємодіють з веб-застосунком. В межах логічної моделі передбачається зберігання ідентифікаційних і контактних даних, а також параметрів, необхідних для автентифікації, що дозволяє реалізувати персоналізовану взаємодію з системою, включно з доступом до історії замовлень та можливістю повторного використання збережених даних під час оформлення покупки. Важливим аспектом є забезпечення унікальності облікових записів, що досягається за рахунок використання унікальних значень електронної пошти або іншого ідентифікатора.

Сутність замовлення відображає факт здійснення покупки та містить узагальнену інформацію про транзакцію. До її складу входять дані про користувача, дату створення, загальну вартість і поточний стан обробки. Збереження статусу замовлення дозволяє відстежувати етапи його виконання, що має важливе значення як для користувача, так і для адміністратора системи. Структура замовлення передбачає зв’язок з користувачем, який його створив, що реалізується через зовнішній ключ.

Деталізація складу замовлення здійснюється за допомогою сутності позиції замовлення. Вона виконує роль зв’язувальної ланки між замовленням і товарами та дозволяє відобразити, які саме товари входять до конкретної покупки, в якій кількості та за якою ціною. Використання окремої сутності для збереження позицій замовлення забезпечує можливість формування складних замовлень, що містять декілька товарів, а також, дозволяє зберігати історичні значення цін, незалежно від подальших змін у каталозі.

Зв’язки між сутностями визначають логіку взаємодії об’єктів і забезпечують узгодженість даних. Зв’язок між категорією та товаром має характер «один до багатьох», що відображає включення кількох товарів до однієї категорії. Зв’язок між користувачем і замовленням відповідає принципу «один до багатьох», оскільки один користувач може створювати декілька замовлень

протягом часу. Замовлення і позиції замовлення пов’язані аналогічним чином, що дозволяє формувати набір товарів в межах однієї покупки. Взаємодія між позицією замовлення та товаром має характер «багато до одного», оскільки одна товарна позиція може входити до складу різних замовлень. ER-діаграма логічної структури бази даних інтернет-магазину наведена на рисунку 2.3.

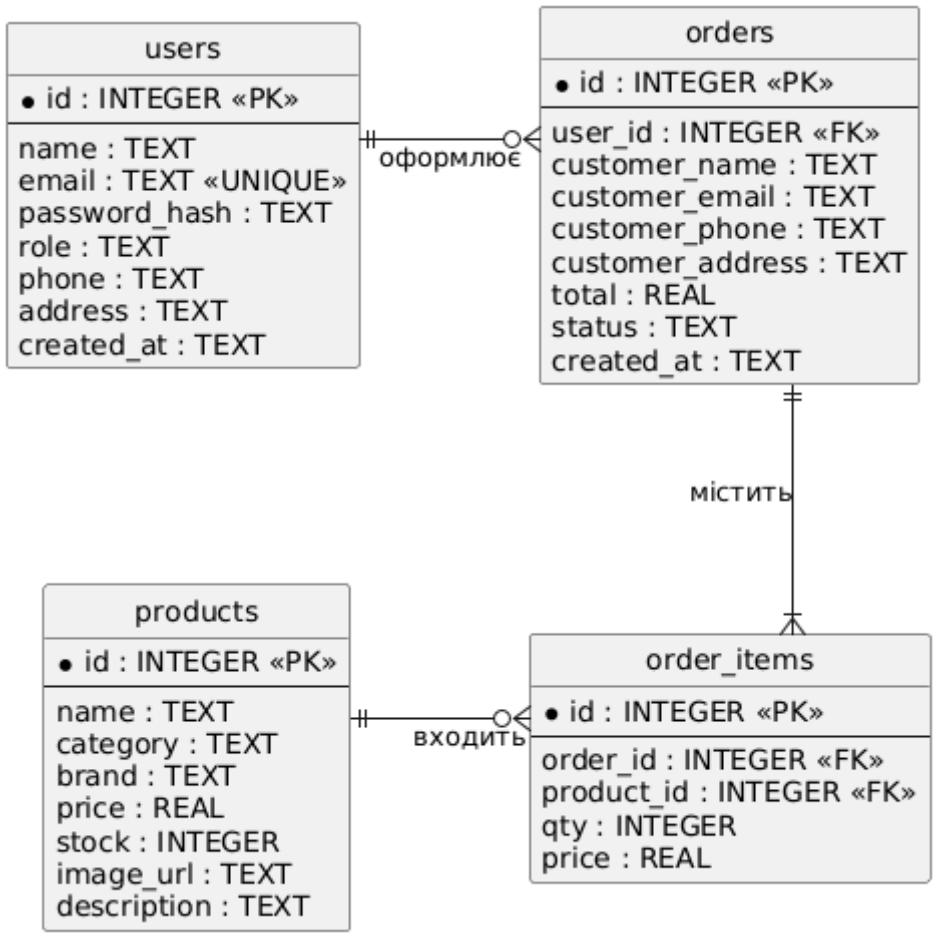


Рисунок 2.3 – ER-діаграма логічної структури бази даних інтернет-магазину автотоварів

Наведена ER-діаграма демонструє узгоджену модель даних, в якій кожна сутність має визначене місце та функціональне призначення. Визначення первинних ключів забезпечує унікальність записів, тоді як зовнішні ключі встановлюють залежності між таблицями, що дозволяє підтримувати цілісність даних на рівні бази. Розглянута організація унеможливлює виникнення

суперечностей між пов’язаними записами та гарантує коректність результатів запитів [21].

Окрему увагу необхідно приділити нормалізації структури бази даних, що спрямована на усунення надлишковості та забезпечення логічної узгодженості інформації. Розподіл даних між сутностями дозволяє уникнути дублювання однакових відомостей і спрощує їх оновлення. Наприклад, інформація про товар зберігається лише в одній таблиці, а в складі замовлення використовується посилання на відповідний запис, що забезпечує узгодженість даних і зменшує обсяг пам’яті, необхідної для їх зберігання.

2.3 Характеристика сутностей, атрибутів та зв’язків бази даних

Побудова логічної структури бази даних набуває практичного змісту лише після детального опису сутностей, їх атрибутів і характеру взаємозв’язків. Саме на рівні характеристик таблиць, закладаються правила збереження інформації, обмеження цілісності та умови коректної обробки даних в процесі роботи вебзастосунку інтернет-магазину автотоварів. Для торговельної системи подібного типу важливим є не лише перелік таблиць, а й точне визначення призначення кожного поля, його ролі в бізнес-процесах та зв’язку з іншими елементами моделі.

В структурі бази даних, доцільно виділити п’ять основних сутностей: категорію, товар, користувача, замовлення та позицію замовлення. Категорія виконує функцію класифікації асортименту та забезпечує логічне групування продукції. Товар відображає одиницю каталогу, доступну для перегляду та купівлі. Користувач зберігає персональні та облікові відомості, необхідні для авторизації та оформлення замовлень. Замовлення фіксує факт покупки та її загальні параметри [22]. Позиція замовлення деталізує склад покупки, зв’язуючи окреме замовлення з конкретними товарами. Узагальнена характеристика зазначених сутностей подана в таблиці 2.1.

Таблиця 2.1 – Призначення основних сутностей бази даних

Сутність	Призначення у системі	Основний зміст даних
Category	Класифікація товарного асортименту	Назва категорії, ідентифікатор
Product	Подання товарної позиції в каталозі	Назва, опис, ціна, залишок, належність до категорії
User	Збереження даних зареєстрованого покупця	Ім’я, електронна пошта, пароль, телефон, адреса
Order	Фіксація факту оформлення покупки	Дата, сума, стан виконання, посилання на користувача
OrderItem	Деталізація складу замовлення	Посилання на замовлення, посилання на товар, кількість, ціна

Сутність Category використовується для впорядкування асортименту за функціональними або предметними ознаками. Для інтернет-магазину авто товарів, подібна класифікація має особливе значення, оскільки користувачі часто починають роботу з каталогом саме з переходу до групи продукції, а не з точного пошуку конкретного артикулу. Наявність окремої таблиці категорій дозволяє не дублювати назви груп в записах товарів та підтримувати єдину довідкову структуру каталогу.

Сутність Product виконує центральну роль в моделі даних, оскільки через неї описується основний об’єкт продажу. Саме в ній зосереджуються відомості, які відображаються на сторінці товару, використовуються в пошуку, беруть участь у фільтрації та формують зміст замовлення. Опис товару повинен поєднувати комерційні характеристики, зокрема назву і ціну, з даними, важливими для прийняття рішення про покупку, включно з описом та інформацією про залишок на складі.

Сутність User забезпечує облік зареєстрованих покупців. Наявність окремої таблиці користувачів надає можливість реалізувати персоналізовану роботу з веб-застосунком, збереження історії замовлень, повторне використання контактних даних та обмеження доступу до особистої інформації. Облікові поля

в межах зазначеної сутності мають зберігатися з урахуванням базових вимог безпеки, зокрема пароль повинен бути представлений не у відкритому вигляді, а у формі криптографічного хешу [22].

Сутність Order призначена для фіксації завершеного факту оформлення покупки. Вона не дублює повний зміст товарних позицій, а містить загальні параметри операції, користувача, дату створення, суму та стан виконання. Збереження окремої таблиці замовлень надає можливість впорядковано вести облік транзакцій, контролювати обробку заявок адміністративною частиною та формувати історію покупок для кожного зареєстрованого покупця.

Сутність OrderItem деталізує склад замовлення і виконує роль проміжної таблиці між замовленням та товаром. Саме в ній зберігаються відомості про кількість кожної позиції та її ціну на момент оформлення покупки. Винесення складу замовлення в окрему таблицю є обов’язковим для коректної побудови моделі, оскільки одне замовлення може містити декілька товарів, а один і той самий товар може бути присутнім в різних замовленнях.

Важливим аспектом характеристики сутностей є опис атрибутів, що формують зміст записів кожної таблиці. Атрибути повинні бути достатніми для підтримки всіх основних сценаріїв функціонування системи, але водночас, не створювати надмірного дублювання. В таблиці 2.2 наведено деталізований опис основних полів, їх типів та призначення.

Таблиця 2.2 – Характеристика атрибутів основних сутностей бази даних

Сутність	Атрибут	Тип даних	Призначення
Category	Id	INTEGER	Первинний ключ категорії
Category	Name	TEXT	Назва категорії товарів
Product	Id	INTEGER	Первинний ключ товару
Product	Name	TEXT	Назва товарної позиції
Product	description	TEXT	Текстовий опис товару
Product	Price	REAL	Поточна ціна товару
Product	Stock	INTEGER	Кількість одиниць у наявності
Product	category_id	INTEGER	Зовнішній ключ на таблицю категорій

Продовження таблиці 2.2

User	Id	INTEGER	Первинний ключ користувача
User	Name	TEXT	Ім’я та прізвище покупця
User	Email	TEXT	Унікальна адреса електронної пошти
User	Password	TEXT	Хеш пароля для авторизації
User	Phone	TEXT	Контактний номер телефону
User	Address	TEXT	Адреса доставки або проживання
Order	Id	INTEGER	Первинний ключ замовлення
Order	user_id	INTEGER	Зовнішній ключ на таблицю користувачів
Order	Date	DATETIME	Дата і час створення замовлення
Order	Total	REAL	Загальна сума замовлення
Order	Status	TEXT	Поточний стан обробки замовлення
OrderItem	Id	INTEGER	Первинний ключ позиції замовлення
OrderItem	order_id	INTEGER	Зовнішній ключ на таблицю замовлень
OrderItem	product_id	INTEGER	Зовнішній ключ на таблицю товарів
OrderItem	Quantity	INTEGER	Кількість одиниць товару в замовленні
OrderItem	Price	REAL	Ціна одиниці товару на момент покупки

Атрибути первинної ідентифікації мають вирішальне значення для цілісності моделі. Поле id в кожній сутності використовується як первинний ключ, що забезпечує однозначне посилання на запис і дозволяє будувати зовнішні зв’язки без неоднозначності. Використання числових ідентифікаторів в межах SQLite є практичним рішенням, оскільки воно спрощує зв’язування таблиць, прискорює індексацію та зменшує обсяг службових даних в порівнянні з довгими текстовими ідентифікаторами.

Окремої уваги потребують атрибути, що описують товар. Поля name, description, price та stock формують мінімально необхідний набір даних для відображення продукції в каталозі та оформлення замовлення. Назва забезпечує ідентифікацію товару в інтерфейсі, опис розкриває його зміст і особливості, ціна бере участь в розрахунку вартості покупки, а залишок на складі використовується під час контролю доступності позиції [22]. Поле category_id зв’язує товар з категорією, що підтримує роботу ієрархічної структури каталогу.

Атрибути сутності користувача повинні забезпечувати не лише авторизацію, а й подальше обслуговування замовлення. Адреса електронної пошти виконує функцію логіна і має бути унікальною в межах таблиці. Телефон та адреса застосовуються під час оформлення та доставки замовлення. Збереження імені у вигляді окремого атрибуту полегшує персоналізацію сервісу та використовується в адміністративній частині під час перегляду даних замовлення.

Атрибути сутності замовлення відображають стан і фінансовий зміст транзакції. Поле date фіксує момент створення заявки, поле total містить підсумкову вартість, а поле status дозволяє відображати етап опрацювання замовлення, наприклад нове, підтверджене, виконане або скасоване. Збереження загальної суми на рівні замовлення є виправданим з практичного погляду, оскільки дозволяє швидко отримувати підсумкові значення без повторного агрегування позицій при кожному зверненні до історії покупок.

Схема зв’язків сутностей бази даних інтернет-магазину автотоварів представлена на рисунку 2.4

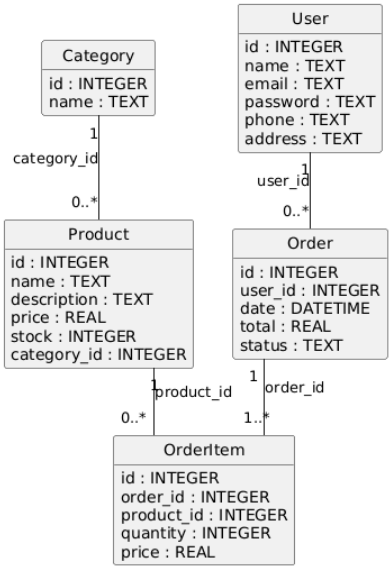


Рисунок 2.4 – Схема зв’язків сутностей бази даних інтернет-магазину автотоварів

Подана схема демонструє, що центр транзакційної частини моделі утворює пара сутностей Order і OrderItem. Саме через них реалізується зв’язок між

користувачем, який ініціює покупку, та товарами, що входять до її складу. Категорія не бере участі безпосередньо в замовленні, проте має важливе значення на етапі навігації та пошуку в каталозі, що відображає поділ моделі на довідковий та транзакційний сегменти.

Характер зв’язків між таблицями повинен підтримувати правила предметної області. Категорія може існувати без жодного товару, але товар без категорії в межах заданої моделі не має сенсу. Користувач може бути зареєстрованим, не маючи жодного оформленого замовлення, однак кожне замовлення обов’язково належить конкретному користувачу. Аналогічно, замовлення не може існувати без хоча б однієї позиції в його складі, тоді як товар може не бути включеним до жодного замовлення на певному етапі роботи системи [22].

З погляду обмежень цілісності, важливо передбачити унікальність адреси електронної пошти, невід’ємність цін і кількості, а також неможливість створення записів з порушеними зовнішніми зв’язками. Для таблиці товарів доцільно контролювати невід’ємне значення поля залишку. Для таблиці позицій замовлення кількість також не може бути нульовою або від’ємною. Подібні перевірки зміцнюють логічну стійкість бази даних та зменшують ризик появи некоректних записів через помилки прикладної логіки.

Значну роль в практичному використанні моделі відіграє індексація ключових полів. Первинні ключі автоматично забезпечують швидкий доступ до записів, але поля зовнішніх ключів category_id, user_id, order_id та product_id також доцільно індексувати. Для електронної пошти користувача доцільним є індекс з обмеженням унікальності. Дане рішення прискорює виконання типових запитів на вибірку історії замовлень користувача, побудову каталогу за категоріями та пошук товарів, що входять до складу конкретного замовлення.

Розкриття змісту сутностей, атрибутів і зв’язків підтверджує, що побудована модель бази даних є узгодженою з функціональними потребами вебзастосунку інтернет-магазину автотоварів. Вона охоплює довідкову частину

каталогу, персоналізовану частину обліку користувачів і транзакційну частину, пов’язану з оформленням та збереженням замовлень.

2.4 Формування алгоритмів бізнес-логіки обробки даних та оформлення замовлень

Функціонування вебзастосунку інтернет-магазину автотоварів неможливо звести лише до зберігання інформації в базі даних або до візуального подання каталогу. Визначальну роль в забезпеченні коректної роботи системи відіграє бізнес-логіка, тобто сукупність правил, процедур і перевірок, що регламентують опрацювання даних під час виконання користувацьких запитів. Саме на рівні бізнес-логіки, формується послідовність дій, за якою система реагує на перегляд каталогу, додавання товару до кошика, авторизацію користувача, створення замовлення, зміну його стану та взаємодію з базою даних. Для інтернет-магазину авто товарів, належне формування алгоритмів має особливу вагу, оскільки помилки в опрацюванні замовлень, в перевірці залишків або у фіксації даних користувача безпосередньо впливають на коректність продажу та надійність сервісу.

Алгоритми бізнес-логіки повинні забезпечувати узгоджену взаємодію між клієнтською частиною, сервером та базою даних. На рівні браузера, користувач ініціює дії через натискання кнопок, заповнення форм і переходи між сторінками. На рівні сервера, запити проходять маршрутизацію, перевірку параметрів, контроль доступу, виконання прикладних правил та звернення до сховища даних. На рівні бази даних, відбувається вибірка, вставка або оновлення записів, після чого сформована відповідь повертається до клієнтської частини. Формування алгоритмів не обмежується окремими фрагментами коду, а охоплює повний життєвий цикл запиту від моменту його ініціювання до фіксації результату в системі [22].

Для торговельного вебзастосунку доцільно виокремити кілька базових напрямів бізнес-логіки. Перший пов’язаний з опрацюванням товарних даних, де ключове значення мають пошук, фільтрація, перевірка наявності та передавання коректної інформації до інтерфейсу. Другий напрям стосується роботи з кошиком, де система повинна додавати позиції, змінювати кількість товарів і підтримувати узгоджений розрахунок сумарної вартості. Третій напрям пов’язаний з процедурою оформлення замовлення, яка охоплює перевірку автентифікації, валідацію контактних даних, перевірку складу покупки, фіксацію замовлення і його позицій в базі даних, а також, оновлення залишків. Четвертий напрям стосується опрацювання платіжної інформації та зміни статусів замовлення, де важливо розмежувати момент створення заявки, момент підтвердження оплати та момент переходу до виконання.

Побудова алгоритмів має спиратися на кілька загальних принципів. По-перше, кожна бізнес-операція повинна починатися з перевірки вхідних даних, оскільки навіть незначні помилки в параметрах запиту можуть призвести до порушення цілісності моделі. По-друге, критичні операції, що змінюють стан системи, повинні супроводжуватися перевіркою прав доступу. По-третє, логіка збереження даних повинна бути відокремлена від логіки інтерфейсу, щоб правила опрацювання замовлень, товарів і користувачів залишалися централізованими на серверному рівні. По-четверте, для будь-якої операції має бути передбачений сценарій обробки помилок, за якого користувач отримує зрозуміле повідомлення, а система не переходить в суперечливий стан.

Однією з основних операцій у вебзастосунку є додавання товару до кошика. Незважаючи на зовнішню простоту, наведена дія потребує послідовного виконання кількох перевірок. Спочатку, система повинна ідентифікувати товар за переданим ідентифікатором, перевірити факт його існування та доступність в каталозі. Далі, виконується перевірка залишку на складі, оскільки кількість, яку користувач намагається додати, не може перевищувати наявний запас. Після успішного проходження перевірок система повинна з’ясувати, чи міститься товар в кошику. Якщо запис вже існує, відбувається оновлення кількості, а якщо

ні, створюється нова позиція. Після завершення операції система перераховує сумарну вартість кошика та повертає оновлений результат до інтерфейсу.

Іншим ключовим процесом виступає оформлення замовлення. Наведена процедура має складнішу структуру, оскільки поєднує перевірку стану сесії користувача, валідацію контактних даних, опрацювання кошика, роботу з таблицями замовлень і позицій замовлення та оновлення залишків товарів. На першому етапі, система визначає, чи має користувач право здійснювати покупку, тобто чи є він авторизованим. Після перевірки автентифікації виконується аналіз наповнення кошика. Якщо кошик порожній, оформлення не може бути продовжене. Якщо в кошику наявні позиції, система повинна повторно перевірити актуальність залишків і цін, оскільки між моментом додавання товару до кошика та моментом підтвердження покупки дані могли змінитися. Лише після наведеної перевірки формується запис замовлення, зберігаються його позиції, зменшуються залишки на складі та фіксується підсумкова сума [22].

Суттєве значення має алгоритм зміни статусів замовлення з боку адміністративної частини. Після надходження нової заявки, адміністратор повинен мати змогу переглянути склад покупки, дані отримувача і поточний стан замовлення. Подальша зміна статусу не повинна бути довільною. Логіка переходів має враховувати послідовність етапів, в межах якої, нове замовлення після підтвердження переходить в стан обробки, далі в стан виконання або завершення, а за потреби, в стан скасування. Завдяки даному алгоритму, система підтримує зрозумілу історію змін і надає користувачеві достовірні відомості про перебіг виконання покупки. Діаграма діяльності процесу оформлення замовлення та опрацювання товарних даних представлена на рисунку 2.5.

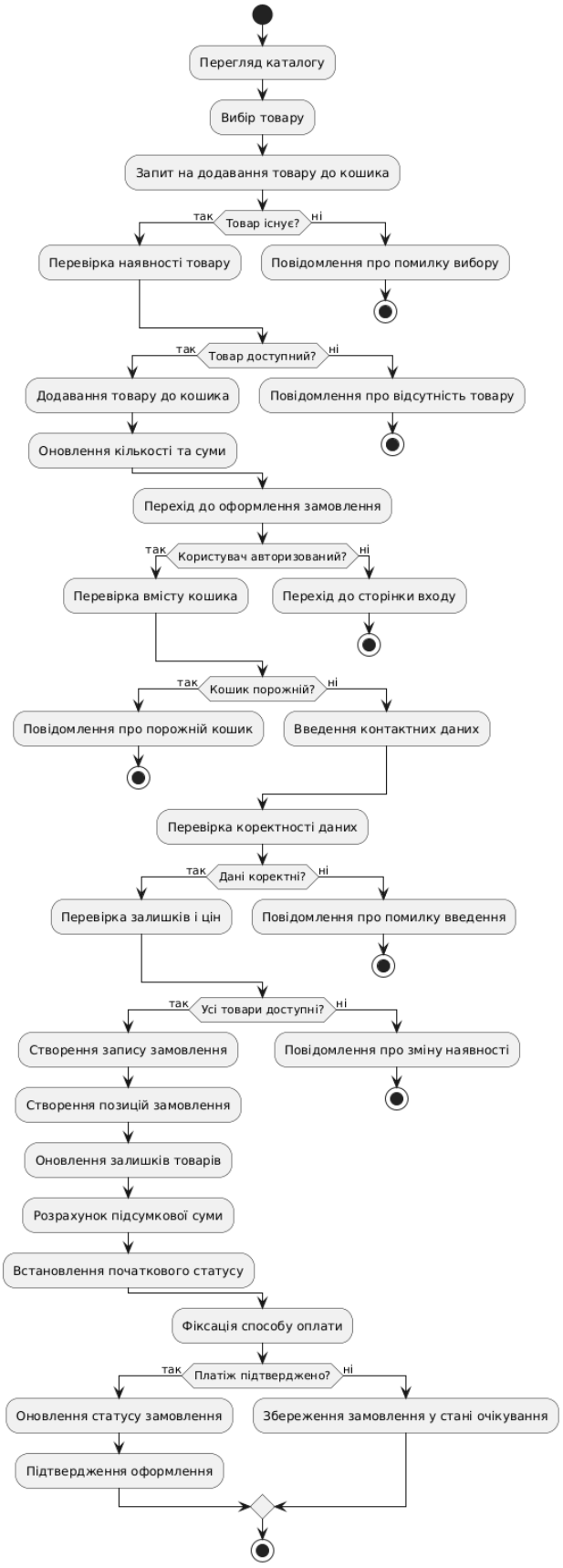


Рисунок 2.5 – Діаграма діяльності процесу оформлення замовлення та опрацювання товарних даних

Діаграма складається з пов’язаних між собою перевірок та дій, де кожний наступний етап залежить від коректного завершення попереднього. На рівні алгоритму, особливо важливою є повторна перевірка товарних залишків перед фіксацією замовлення. Діаграма діяльності відображає роль валідації користувацьких даних. Заповнення контактної форми не повинно автоматично завершуватися записом в базу даних. Перш ніж виконати збереження, система має переконатися в коректності електронної пошти, номера телефону, адреси та інших параметрів, необхідних для супроводу замовлення. Діаграма послідовності оформлення замовлення у вебзастосунку інтернет-магазину автотоварів представлена на рисунку 2.6.

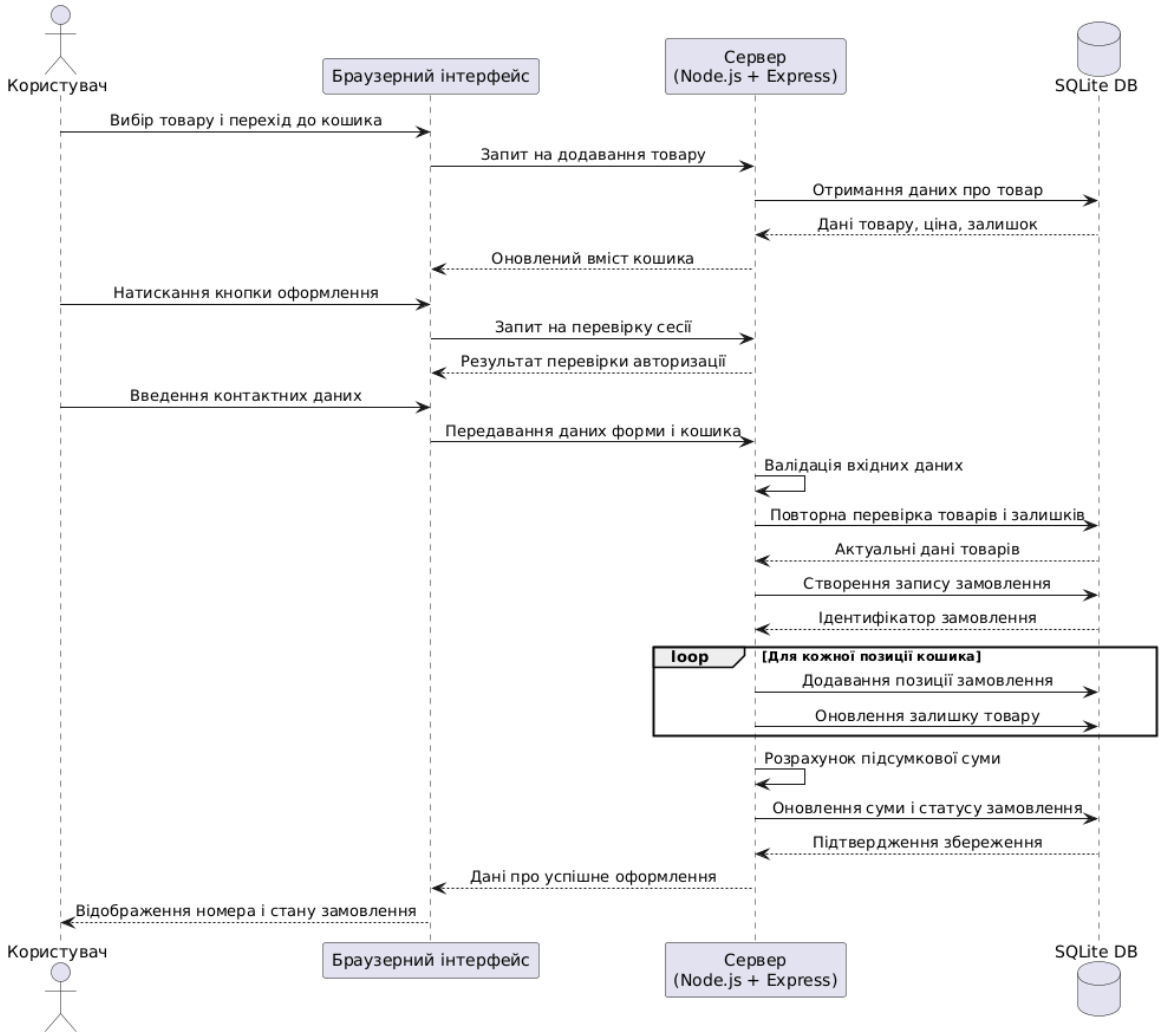


Рисунок 2.6 – Діаграма послідовності оформлення замовлення у вебзастосунку інтернет-магазину автотоварів

Діаграма послідовності наочно демонструє, що оформлення замовлення є результатом координованої взаємодії чотирьох учасників, користувача, клієнтського інтерфейсу, серверної логіки та бази даних. Браузерний інтерфейс в наведеному процесі не виконує складного прикладного опрацювання, а виступає засобом передавання дій користувача та відображення результатів. Основне навантаження припадає на сервер, де зосереджено перевірку авторизації, валідацію введених даних, контроль актуального стану товарів і керування транзакційними змінами в базі даних. Представлений розподіл ролей, відповідає загальним принципам безпечної побудови веб-застосунків, де критичні операції виконуються не на боці клієнта, а в централізованому середовищі обробки запитів [22].

В послідовності дій важливою є наявність двох етапів звернення до товарних даних. Перший відбувається під час додавання позиції до кошика, коли система перевіряє, чи існує товар і чи доступний він для вибору. Другий відбувається вже на етапі оформлення замовлення, коли сервер повторно звертається до бази даних для перевірки актуальності залишків і цін. Дублювання перевірки не є надлишковим, оскільки між моментом додавання товару до кошика та моментом завершення покупки міг змінитися залишок або ціна. Наявність контрольного повторного звернення дозволяє зменшити ризик помилкового продажу та забезпечує коректність фінансових розрахунків.

Цикл додавання позицій замовлення, представлений на діаграмі, відображає практичну реалізацію моделі де одне замовлення може містити декілька товарів. Сервер після створення загального запису замовлення послідовно додає записи в таблицю позицій і водночас, оновлює залишки на складі. Приведений алгоритм, дозволяє синхронізувати структуру замовлення зі станом каталогу, а також, формує основу для подальших перевірок доступності продукції під час нових звернень інших користувачів. Завдяки наведеній логіці, дані в базі залишаються узгодженими не лише в межах одного конкретного замовлення, а й в межах всього торговельного середовища.

Формування алгоритмів бізнес-логіки у вебзастосунку інтернет-магазину автотоварів повинно забезпечувати не тільки виконання окремих операцій, а й передбачувану поведінку системи в цілому. Послідовність дій під час додавання товару, перевірки кошика, створення замовлення та оновлення складських залишків має бути логічно завершеною та стійкою до помилкових сценаріїв. Діаграма діяльності відображає внутрішню структуру процесу через кроки та перевірки, а діаграма послідовності доповнює наведений опис, демонструючи обмін повідомленнями між учасниками системи. В сукупності, обидві моделі підтверджують, що бізнес-логіка вебзастосунку повинна бути побудована на принципах поетапної перевірки, централізованої обробки критичних операцій і узгодженого оновлення даних.

2.5 Формування структури інтерфейсу користувача та навігації вебзастосунку

Структура інтерфейсу користувача вебзастосунку інтернет-магазину автотоварів визначає спосіб подання функціональних можливостей системи у візуальній формі та безпосередньо впливає на зручність взаємодії з каталогом, кошиком, замовленнями та адміністративними даними. Для торговельного веб-ресурсу, інтерфейс не може розглядатися лише як декоративна оболонка, оскільки саме через нього реалізується логіка пошуку товарів, перегляду характеристик, вибору продукції та оформлення покупки. Побудова структури інтерфейсу повинна враховувати послідовність дій користувача, обсяг інформації на сторінках, потребу в швидкому переході між розділами та необхідність зберігати логічну єдність всіх екранів системи.

В контексті інтернет-магазину автотоварів вимоги до інтерфейсу мають підвищену специфіку. На відміну від багатьох інших товарних категорій, автотовари часто потребують детальнішого ознайомлення з технічними характеристиками, сумісністю та призначенням. З наведеної причини, структура

сторінок повинна бути організована так, щоб користувач без зайвих зусиль міг перейти від загального каталогу до конкретної товарної позиції, отримати ключові відомості про неї, зіставити варіанти та виконати покупку. Будь-яке перевантаження другорядними елементами, надлишок декоративних блоків або заплутана система переходів погіршують сприйняття ресурсу та збільшують імовірність відмови від подальшої взаємодії [22].

Формування інтерфейсу доцільно будувати на принципі функціональної ієрархії, за якого всі основні дії користувача розподіляються між кількома базовими екранами. В структурі вебзастосунку інтернет-магазину авто товарів, такими екранами виступають головна сторінка, сторінка каталогу, сторінка окремого товару, сторінка кошика, сторінка оформлення замовлення, сторінка авторизації та реєстрації, особистий кабінет покупця, а також, адміністративна панель. Наведені розділи повинні бути об’єднані єдиною навігаційною системою, що забезпечує передбачувану поведінку інтерфейсу та скорочує час пошуку потрібної функції.

Головна сторінка виконує роль початкової точки взаємодії користувача з системою. Саме вона формує перше уявлення про магазин, знайомить з асортиментом та спрямовує до подальших сценаріїв роботи. На ній доцільно розміщувати верхню навігаційну панель, коротке позиціонування магазину, блок швидкого переходу до основних категорій товарів, поле пошуку, блок популярних або нових товарів, а також, інформаційний футер з контактами та допоміжними посиланнями. Головна сторінка не повинна дублювати повний каталог, оскільки її основна функція полягає у швидкому введенні користувача до структури сервісу й наданні короткого огляду асортименту. Для наочного відображення структури головної сторінки доцільно використати wireframe-прототип, який представлено на рисунку 2.7.

Сторінка каталогу є центральним елементом користувацького інтерфейсу, оскільки саме на ній відбувається основна робота з асортиментом. Її структура повинна поєднувати навігаційний блок категорій, область пошуку, зону фільтрації та область відображення товарних карток. Важливим є баланс між

щільністю подання інформації та зручністю сприйняття. Якщо товарних характеристик на картці надто багато, сторінка виглядає перевантаженою, якщо їх замало, користувач змушений відкривати велику кількість товарів по черзі. З наведеної причини в картці каталогу доцільно відображати назву, зменшене зображення, ціну, коротку інформацію про наявність та кнопку переходу до детального перегляду або додавання до кошика. Пошук і фільтрація повинні розміщуватися у верхній або бічній частині сторінки, залишаючись доступними під час перегляду результатів.

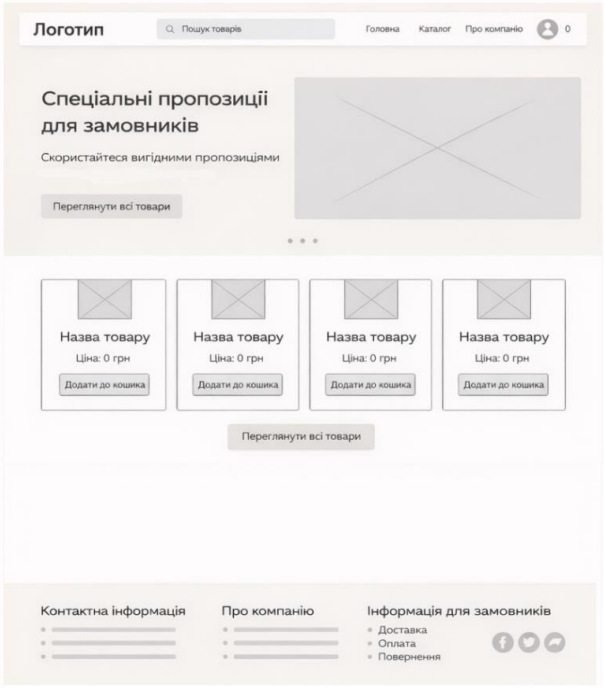


Рисунок 2.7 – Прототип головної сторінки вебзастосунку інтернет-магазину автотоварів

Для магазину автотоварів, особливо важливим є коректне проєктування фільтраційної частини каталогу. Користувач часто здійснює пошук не за загальною назвою товару, а за ознаками сумісності, типом товару, виробником або ціновим діапазоном. Тому навігація в каталозі має підтримувати багаторівневий відбір, який дозволяє звужувати вибір без втрати логічної послідовності. Візуально подібний механізм бажано організувати через окремий бічний блок фільтрів, що не перекриває область результатів і не вимагає

відкриття додаткових модальних вікон. В рамках прототипування сторінки каталогу, важливо зафіксувати саме розташування пошукового поля, фільтраційного блоку та зони карток товарів, оскільки від наведеного компонування залежить загальна ергономіка ресурсу. Прототип сторінки каталогу автотоварів наведено на рисунку 2.8.

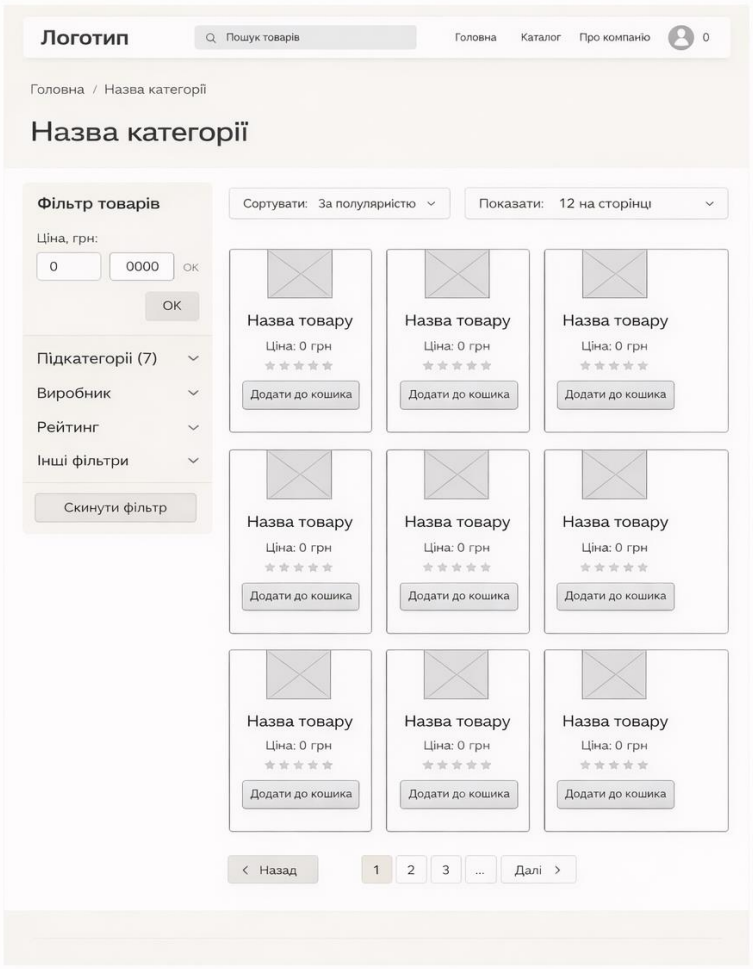


Рисунок 2.8 – Прототип сторінки каталогу автотоварів

Сторінка окремого товару повинна надавати докладнішу інформацію, ніж картка в каталозі. В межах наведеної сторінки, потрібно передбачити велику область для основного зображення, текстовий опис, технічні характеристики, поточну ціну, індикатор наявності, блок вибору кількості та кнопку додавання до кошика. Для предметної області автотоварів особливо важливо виділити область характеристик, де користувач отримує відомості про призначення, можливу сумісність, виробника та інші параметри, що впливають на прийняття

рішення про покупку. Візуальна побудова даної сторінки повинна забезпечувати зручний поділ між комерційними та технічними відомостями. Ціна, наявність і кнопки дії мають розташовуватися в помітній області, тоді як текстові характеристики можуть бути подані нижче або в спеціальному інформаційному блоці.

З погляду навігації, сторінка товару виконує проміжну роль між переглядом каталогу та переходом до кошика. Саме тому на ній необхідно зберігати можливість повернення до каталогу, переходу до суміжних категорій або перегляду подібної продукції. Водночас, не варто перевантажувати сторінку великою кількістю допоміжних блоків, оскільки це відволікає від основної цілі – ознайомлення з характеристиками товару та прийняття рішення про його додавання до кошика. Прототип сторінки окремого товару наведено на рисунку 2.9.

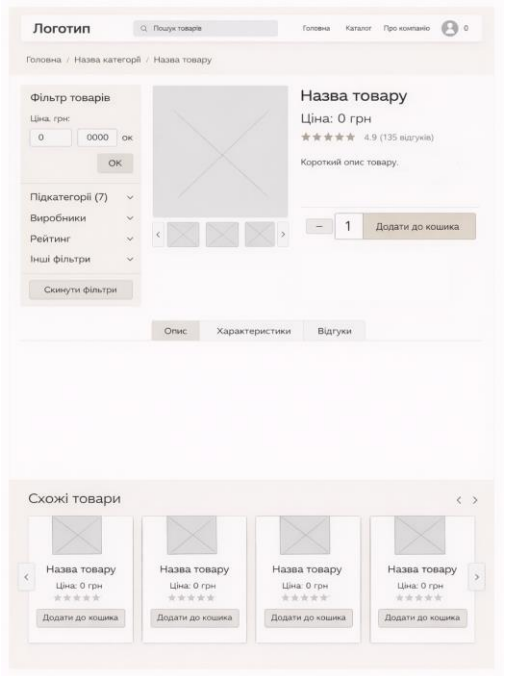


Рисунок 2.9 – Прототип сторінки окремого товару

Сторінка кошика повинна бути побудована як проміжний простір перевірки обраних позицій перед оформленням замовлення. Її структура має відображати перелік товарів, зображення або піктограму кожної позиції, назву,

ціну, кількість, проміжну суму та інструменти зміни кількості або видалення запису. Окремо повинна виводитися загальна сума замовлення, що перераховується після будь-якої зміни вмісту кошика. Навігаційно, сторінка кошика має підтримувати два базові напрями переходу: повернення до каталогу для продовження вибору та перехід до оформлення замовлення. Представлена побудова забезпечує неперервність користувацького сценарію та дозволяє уникнути ситуацій, коли користувач втрачає орієнтацію після зміни кількості або видалення товару.

Під час проєктування кошика необхідно врахувати, що для покупця особливе значення має зрозумілість числових параметрів. Невизначене оновлення суми, затримки у відображенні змін або відсутність помітного підсумкового блоку знижують довіру до сервісу. Прототип сторінки кошика наведено на рисунку 2.10.

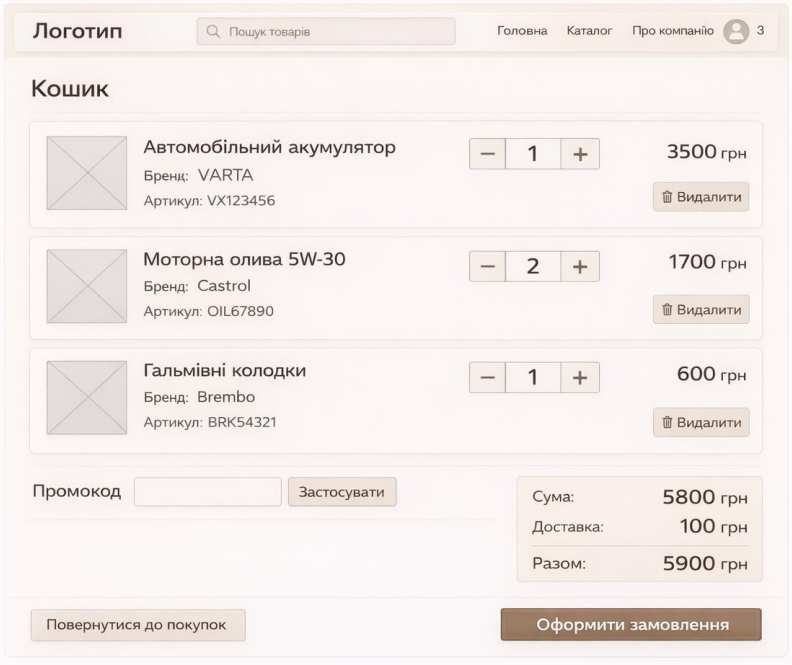


Рисунок 2.10 – Прототип сторінки кошика

Сторінка оформлення замовлення відрізняється від кошика тим, що зосереджується не на складі покупки, а на завершенні транзакції. Її структура повинна забезпечувати введення або перевірку контактних даних, вибір способу

доставки, фіксацію способу оплати та підтвердження замовлення. Водночас, користувач повинен бачити коротке резюме покупки, щоб не втрачати зв’язок між формою введення даних і змістом замовлення.

Для сторінки оформлення замовлення особливо важливою є логіка відображення помилок. Якщо частина полів залишена незаповненою або містить некоректні значення, інтерфейс повинен вказувати на помилку локально, біля відповідного поля, а не обмежуватися загальним повідомленням у верхній частині сторінки. Прототип сторінки оформлення замовлення наведено на рисунку 2.11.

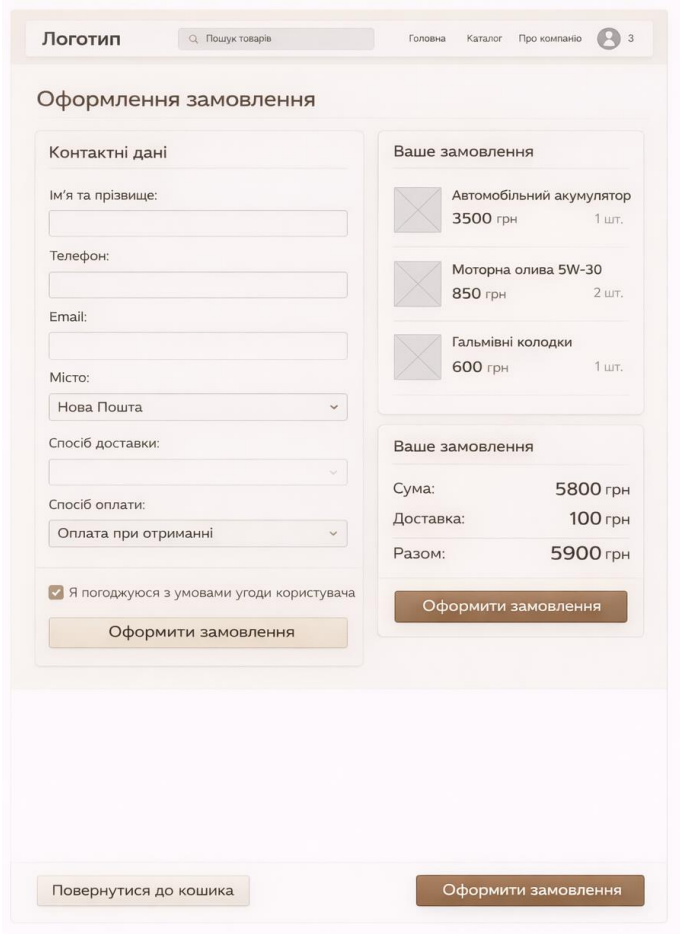


Рисунок 2.11 – Прототип сторінки оформлення замовлення

Авторизація та реєстрація повинні мати спрощену структуру, зосереджену на мінімально необхідних полях, без надлишкового інформаційного наповнення. Основна мета наведених екранів полягає в тому, щоб забезпечити вхід до

системи або створення облікового запису без зайвого когнітивного навантаження. Особистий кабінет, на відміну від форм входу, виконує сервісну функцію та повинен надавати доступ до персональних даних, історії замовлень і поточних станів оформлених покупок. Навігація всередині кабінету повинна бути компактною та однозначною, а історія замовлень – поданою у вигляді впорядкованого переліку з можливістю швидкого перегляду номера, дати, суми та статусу. Прототип сторінки авторизації та особистого кабінету користувача наведено на рисунку 2.12.

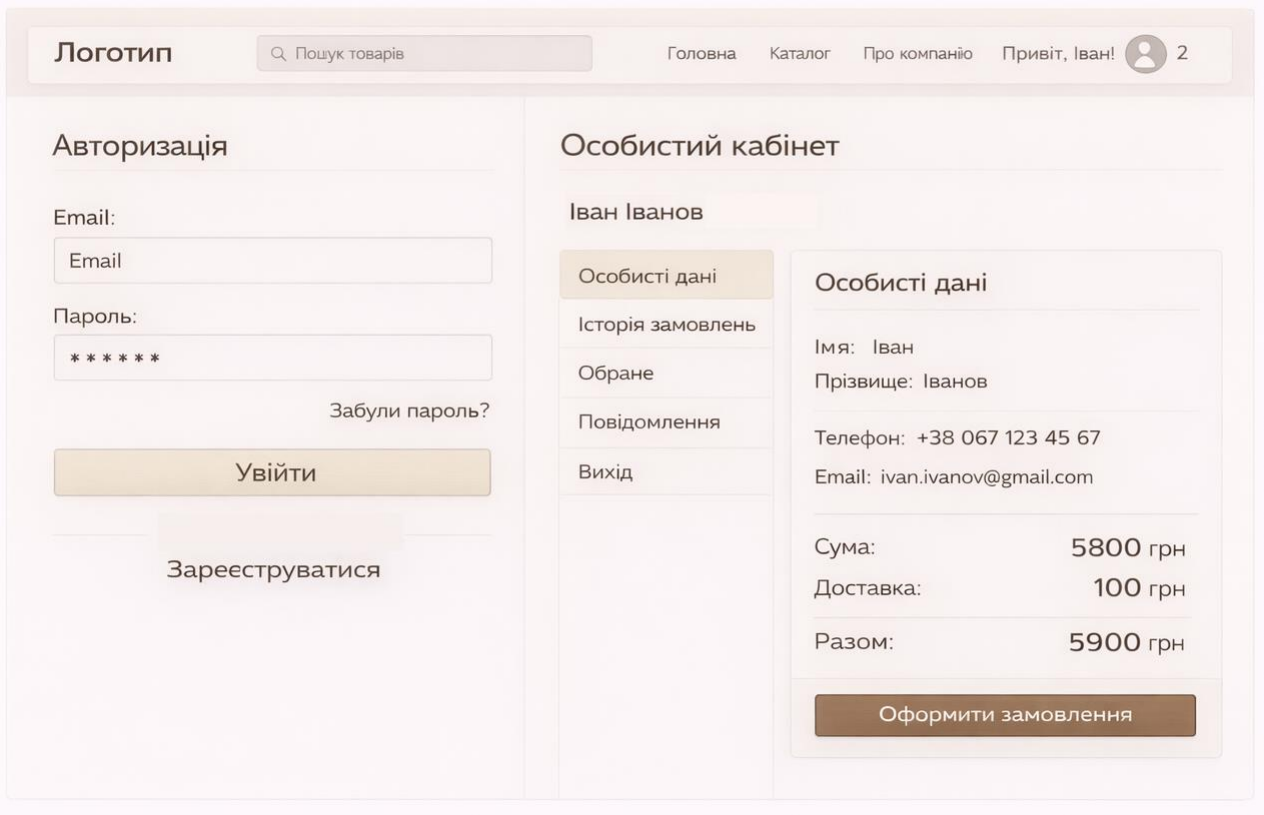


Рисунок 2.12 – Прототип сторінки авторизації та особистого кабінету користувача

Поряд з користувацькою частиною вебзастосунку, необхідно передбачити окрему структуру для адміністративної панелі. Наведений інтерфейс повинен бути орієнтований не на демонстрацію товарів, а на керування ними. Для адміністративної частини, варто передбачити меню переходу до розділів товарів, категорій, замовлень і користувачів, а також робочу область, де відображаються

таблиці даних або форми редагування. На відміну від публічної частини, адміністративна панель може мати суворіший та менш декоративний вигляд, оскільки її головною вимогою є функціональна зручність і швидкість виконання операцій. Прототип адміністративної панелі вебзастосунку наведено на рисунку 2.13.

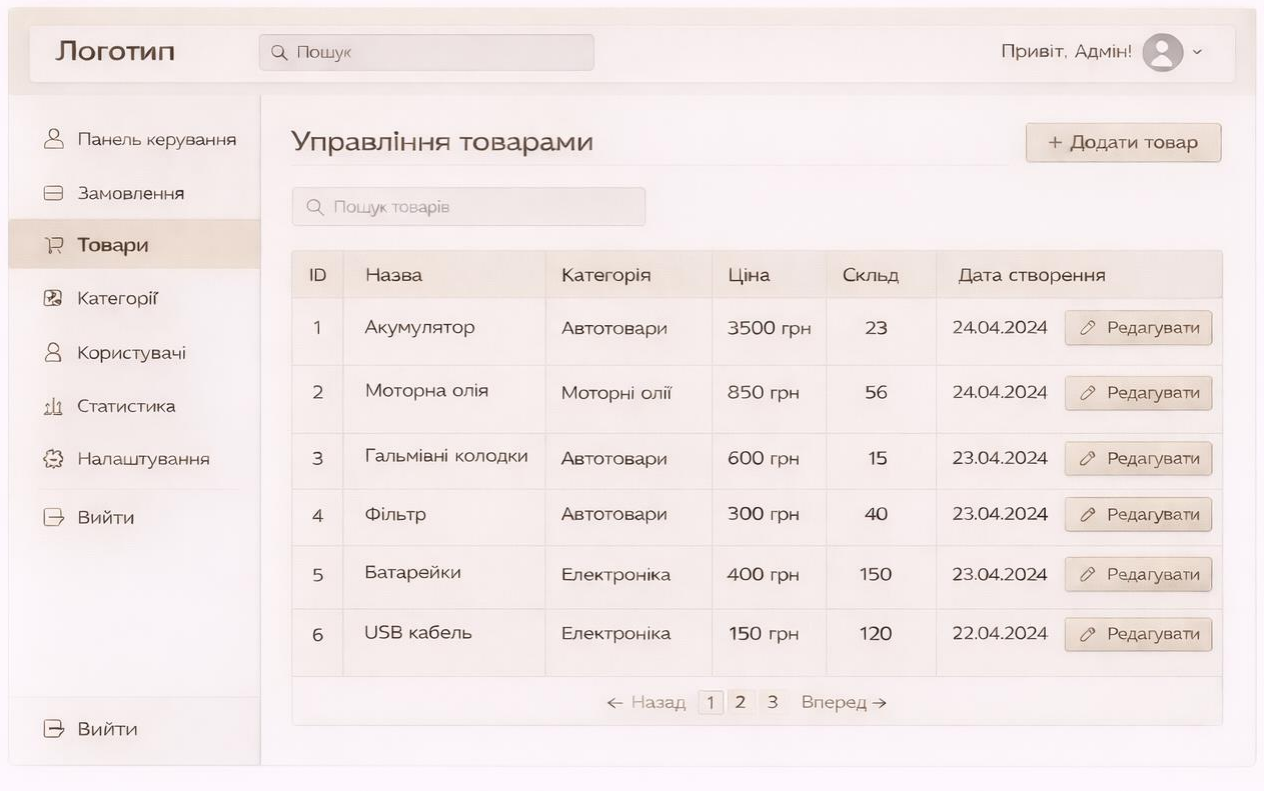


Рисунок 2.13 – Прототип адміністративної панелі вебзастосунку

Формування структури інтерфейсу користувача та навігації вебзастосунку інтернет-магазину автотоварів має розглядатися як самостійний етап проєктування, а не як допоміжне візуальне оформлення вже готової системи. Логіка переходів між екранами, розподіл інформаційних зон, відокремлення користувацької та адміністративної частин, а також попереднє прототипування ключових сторінок безпосередньо впливають на ефективність взаємодії з вебресурсом.

2.6 Висновки до розділу 2

В другому розділі роботи, було розроблено структуру вебзастосунку інтернет-магазину автотоварів та спроєктовано базу даних, що забезпечує зберігання та обробку інформації про товари, користувачів і замовлення. Визначено архітектурну організацію системи, яка передбачає взаємодію клієнтської частини, серверного рівня та бази даних.

Розроблено логічну структуру бази даних з визначенням основних сутностей, їх атрибутів і зв’язків, що забезпечує цілісність та узгодженість даних. Описано алгоритми обробки даних та реалізації бізнес-логіки, пов’язаної з оформленням замовлень. Сформовано структуру інтерфейсу користувача та навігації, що забезпечує зручність взаємодії із системою. Отримані результати створюють основу для практичної реалізації вебзастосунку.

3 СТВОРЕННЯ ТА ВПРОВАДЖЕННЯ ВЕБЗАСТОСУНКУ

3.1 Розробка серверної частини вебзастосунку з використанням Node.js та Express.js

Розробка серверної частини вебзастосунку інтернет-магазину автотоварів визначає логіку функціонування всієї системи, оскільки саме на цьому рівні здійснюється обробка запитів користувачів, перевірка коректності введених даних, керування доступом до функціональності та взаємодія з базою даних. Сервер слугує центральною ланкою, яка забезпечує узгоджене функціонування клієнтської частини та сховища даних, відповідає за реалізацію ключових бізнес-процесів, пов’язаних з формуванням каталогу, обробкою кошика та створенням замовлень.

В процесі реалізації серверної частини, використано середовище Node.js, яке забезпечує виконання JavaScript-коду на стороні сервера, та фреймворк Express.js, що спрощує організацію маршрутизації та обробку HTTP-запитів. Використання зазначених технологій дозволяє побудувати компактну та зрозумілу структуру серверного застосунку, яка легко піддається модифікації та розширенню. Обрана технологічна основа забезпечує високу швидкість обробки запитів, що є важливим для інтернет-магазину, де користувач очікує миттєвого відгуку системи при перегляді товарів або оформленні замовлення.

Архітектурно, серверна частина реалізована за принципом розподілу функціональності між окремими модулями. Основний файл застосунку відповідає за ініціалізацію сервера, підключення до бази даних та реєстрацію маршрутів. Далі, логіка розділяється між окремими компонентами, серед яких виокремлюються модулі маршрутизації, обробки бізнес-логіки та доступу до даних [23].

На етапі практичної реалізації було створено вебзастосунок інтернет-магазину автотоварів під назвою AutoDrive Market, який забезпечує доступ до

каталогу продукції, роботу з кошиком, оформлення замовлень та взаємодію з адміністративною частиною. Загальний вигляд розробленого інтерфейсу представлено на рисунку 3.1.

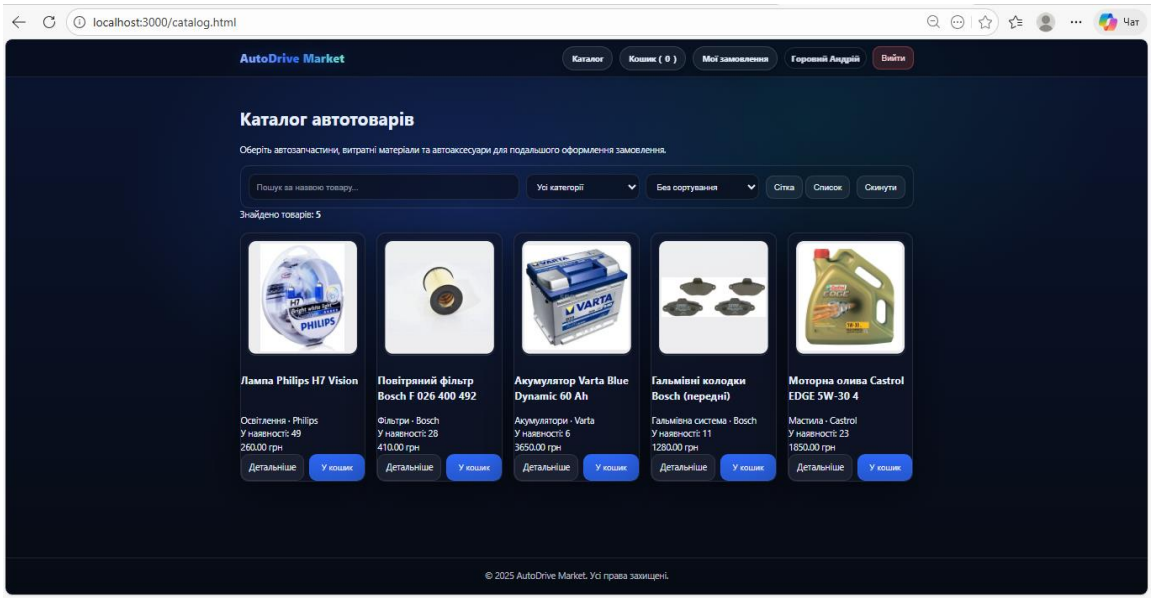


Рисунок 3.1 – Головна сторінка вебзастосунку інтернет-магазину автотоварів

Зображений інтерфейс демонструє реалізацію клієнтської частини, яка взаємодіє з сервером через HTTP-запити. Користувач отримує можливість переглядати каталог товарів, застосовувати пошук та фільтрацію, додавати товари до кошика та переходити до оформлення замовлення. У верхній частині інтерфейсу, розташовано навігаційне меню, яке забезпечує швидкий доступ до основних розділів системи, включно з каталогом, кошиком, історією замовлень.

Функціонування представлених елементів інтерфейсу, безпосередньо пов’язане з серверною частиною застосунку. Під час взаємодії користувача з системою, браузер формує запити до серверу, які обробляються відповідними маршрутами. Наприклад, при відкритті каталогу формується запит на отримання списку товарів, який обробляється сервером з подальшим зверненням до бази даних. Отримані дані повертаються у форматі JSON та відображаються у вигляді карток товарів [24].

Також, важливим є механізм маршрутизації, який визначає обробку запитів залежно від їх призначення. Запити, які стосуються роботи з товарами,

обробляються одним набором маршрутів, тоді як запити, пов’язані з користувачами або замовленнями, спрямовуються до інших модулів, що дозволяє чітко розмежувати функціональність та уникнути перевантаження окремих частин системи.

Бізнес-логіка реалізує перевірку даних, які надходять від користувача, та забезпечує коректне виконання операцій. Під час оформлення замовлення, виконується перевірка наявності товарів на складі, обчислення загальної вартості та формування записів в базі даних. Відповідно, так забезпечується узгодженість інформації та запобігання виникненню помилкових ситуацій, пов’язаних з некоректними даними.

Доступ до адміністративної частини обмежується лише для користувачів з відповідними правами, що досягається за рахунок перевірки токенів автентифікації на серверному рівні. Тобто система захищається від несанкціонованого доступу та забезпечується контроль над операціями керування товарами і замовленнями.

Взаємодія з базою даних реалізована через окремий рівень доступу до даних, який забезпечує виконання SQL-запитів без безпосереднього втручання бізнес-логіки в структуру запитів. Серверна частина відповідає за формування запитів на вибірку товарів, створення користувачів, обробку замовлень та зміну станів записів в базі даних [25].

Реалізована серверна частина вебзастосунку забезпечує стабільну обробку запитів, коректну взаємодію з базою даних та підтримку основних сценаріїв роботи користувачів. Обрана архітектура дозволяє масштабувати систему, доповнювати її новими функціональними можливостями та забезпечувати подальший розвиток вебзастосунку без порушення його базової логіки.

3.2 Створення та налаштування бази даних SQLite

Ефективне функціонування вебзастосунку інтернет-магазину автотоварів значною мірою визначається правильною організацією бази даних, яка забезпечує зберігання, обробку та швидкий доступ до інформації. Для реалізації програмного продукту було обрано реляційну модель даних, яка дозволяє структуровано зберігати інформацію про користувачів, товари та замовлення, підтримувати логічні зв’язки між відповідними сутностями. Як система керування базами даних використовується SQLite, що є компактним рішенням, яке не потребує окремого серверного середовища та добре інтегрується з середовищем Node.js.

Структура бази даних побудована з урахуванням основних функціональних вимог вебзастосунку, зокрема забезпечення роботи каталогу товарів, підтримки облікових записів користувачів, збереження інформації про замовлення та їх склад. При проєктуванні бази даних, враховано необхідність уникнення дублювання даних, підтримання цілісності інформації та можливості розширення структури в майбутньому [26].

Однією з ключових сутностей є таблиця користувачів, яка зберігає інформацію про зареєстрованих осіб, що взаємодіють із системою. Структура даної таблиці представлена в таблиці 3.1.

Таблиця 3.1 – Структура таблиці користувачів

Поле	Тип даних	Опис
id	INTEGER	Унікальний ідентифікатор користувача
name	TEXT	Ім’я користувача
email	TEXT	Електронна адреса
password_hash	TEXT	Хеш пароля
role	TEXT	Роль користувача
phone	TEXT	Контактний номер
address	TEXT	Адреса користувача
created_at	TEXT	Дата створення запису

Зазначена таблиця забезпечує зберігання даних, необхідних для автентифікації користувачів та реалізації механізмів доступу до функціональності системи. Наявність поля ролі дозволяє розмежувати звичайних користувачів та адміністраторів, що є важливим для забезпечення безпеки та контролю доступу.

Наступною важливою складовою бази даних є таблиця товарів, яка забезпечує зберігання інформації про продукцію, представлену в каталозі інтернет-магазину. Структура даної таблиці наведена в таблиці 3.2.

Таблиця 3.2 – Структура таблиці товарів

Поле	Тип даних	Опис
id	INTEGER	Унікальний ідентифікатор товару
name	TEXT	Назва товару
category	TEXT	Категорія
brand	TEXT	Виробник
price	REAL	Ціна товару
price_cents	INTEGER	Ціна у копійках
stock	INTEGER	Кількість на складі
image	TEXT	Шлях до зображення
image_url	TEXT	Альтернативний шлях до зображення
description	TEXT	Опис товару
created_at	TEXT	Дата додавання

Представлена таблиця забезпечує збереження характеристик товарів, необхідних для відображення їх в каталозі та для подальшої обробки під час оформлення замовлення. Наявність окремого поля для зберігання кількості товару дозволяє контролювати залишки на складі та запобігати оформленню замовлень на відсутню продукцію [27].

Для реалізації функціоналу оформлення замовлень використовується таблиця замовлень, яка зберігає загальну інформацію про кожне замовлення. Її структура представлена в таблиці 3.3.

Таблиця 3.3 – Структура таблиці замовлень

Поле	Тип даних	Опис
Id	INTEGER	Ідентифікатор замовлення
user_id	INTEGER	Ідентифікатор користувача
customer_name	TEXT	Ім’я покупця
customer_email	TEXT	Електронна адреса
customer_phone	TEXT	Телефон
customer_address	TEXT	Адреса доставки
Total	REAL	Загальна сума
Status	TEXT	Статус замовлення
created_at	TEXT	Дата створення

Таблиця замовлень забезпечує фіксацію основної інформації про кожну операцію купівлі, включно з контактними даними покупця та загальною сумою. Поле статусу дозволяє відслідковувати стан виконання замовлення, що є важливим для адміністративного контролю.

Оскільки одне замовлення може містити декілька товарів, в структурі бази даних передбачено окрему таблицю для зберігання складу замовлення. Її структура наведена в таблиці 3.4.

Таблиця 3.4 – Структура таблиці елементів замовлення

Поле	Тип даних	Опис
Id	INTEGER	Ідентифікатор запису
order_id	INTEGER	Ідентифікатор замовлення
product_id	INTEGER	Ідентифікатор товару
Qty	INTEGER	Кількість
Price	REAL	Ціна на момент покупки

Наявність окремої таблиці для елементів замовлення дозволяє реалізувати зв’язок між замовленням та товарами за принципом один-до-багатьох, що відповідає вимогам реляційної моделі. Так забезпечується гнучкість структури даних та можливість зберігати детальну інформацію про склад кожного замовлення.

Між розглянутими таблицями встановлено відповідні зв’язки, які забезпечують цілісність даних. Зокрема, кожне замовлення пов’язане з користувачем через поле user_id, а таблиця елементів замовлення пов’язує замовлення з конкретними товарами. Використання зовнішніх ключів дозволяє гарантувати коректність зв’язків та запобігти появі неконсистентних записів в базі даних.

Реалізована структура бази даних дозволяє ефективно виконувати всі необхідні операції, пов’язані з роботою інтернет-магазину, включно з відображенням каталогу, обробкою замовлень та управлінням користувачами.

3.3 Впровадження функціоналу роботи з товарами, користувачами та замовленнями

Реалізація функціоналу роботи з товарами, користувачами та замовленнями забезпечує виконання основних бізнес-процесів вебзастосунку інтернет-магазину автотоварів та формує основу взаємодії між клієнтською nf серверною частинами системи. В процесі впровадження зазначеного функціоналу, було забезпечено обробку запитів користувачів, передачу даних між компонентами системи та їх коректне відображення у веб-інтерфейсі. Узгоджена робота всіх модулів дозволяє досягти стабільного функціонування системи та забезпечити передбачуваність результатів виконання операцій.

Функціонал роботи з товарами передбачає можливість отримання списку продукції, перегляду детальної інформації та додавання товарів до кошика. Клієнтська частина формує запити до серверного API для отримання актуальних даних, після чого відбувається їх обробка та відображення у вигляді карток. Взаємодія з сервером здійснюється асинхронно, що дозволяє оновлювати вміст сторінки без її перезавантаження та забезпечує більш плавний процес роботи користувача з інтерфейсом [28].

Окремим аспектом реалізації функціоналу товарів є адміністративна частина, яка забезпечує можливість керування каталогом. Адміністратор може

створювати нові записи, змінювати характеристики товарів та видаляти їх за необхідності. Всі ці дії супроводжуються передачею відповідних запитів до серверної частини, яка виконує перевірку даних та здійснює зміни в базі даних. Інтерфейс керування товарами в адміністративній панелі наведено на рис. 3.2.

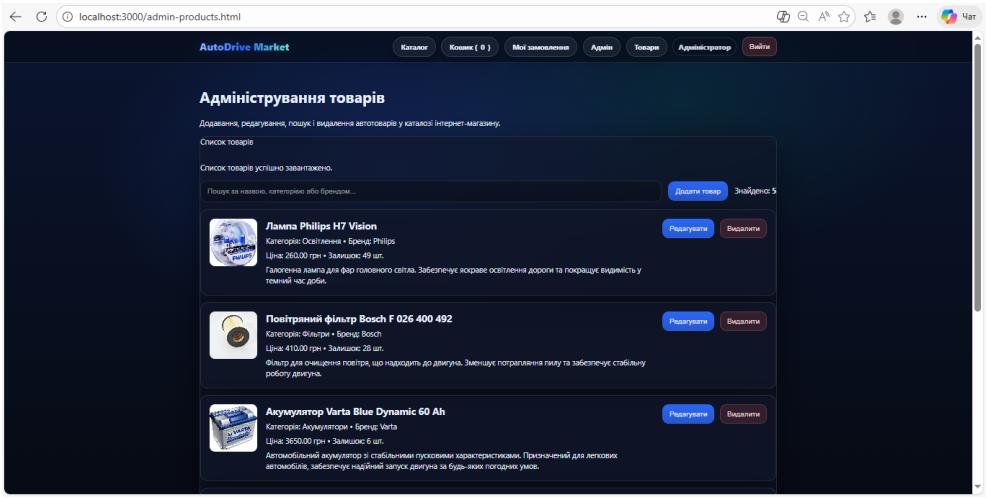


Рисунок 3.2 – Інтерфейс керування товарами в адміністративній панелі

Відображений інтерфейс демонструє реалізацію механізму керування товарами, де всі операції виконуються в межах єдиного середовища. Після виконання дій адміністратором, інформація миттєво оновлюється, що свідчить про коректну взаємодію між клієнтською та серверною частинами, що забезпечує актуальність даних та дозволяє ефективно управляти каталогом продукції.

Функціонал роботи з користувачами реалізує процес створення облікових записів, їх автентифікацію та подальшу взаємодію з системою. Під час реєстрації, користувач вводить персональні дані, які передаються на сервер для обробки та збереження. Для забезпечення захисту інформації використовується механізм хешування паролів, що унеможливлює їх збереження у відкритому вигляді. Інтерфейс авторизації користувача представлено на рисунку 3.3.

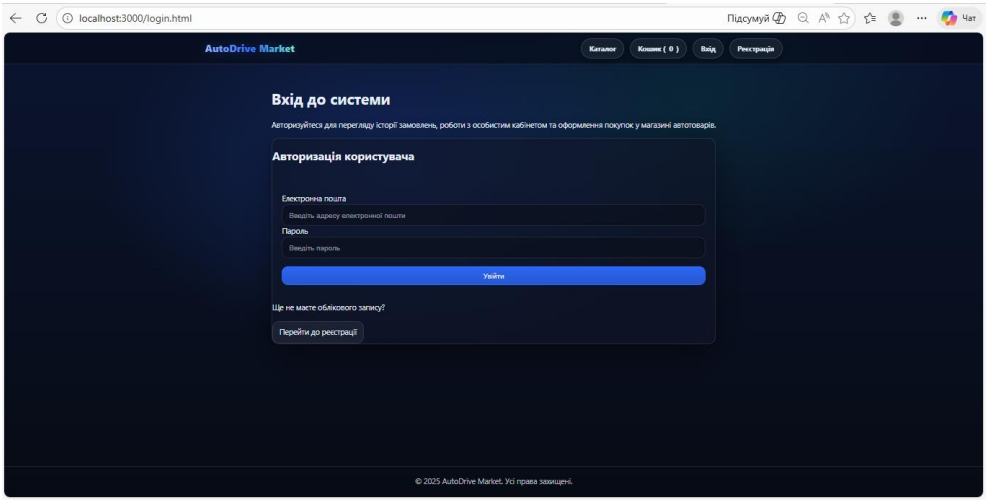


Рисунок 3.3 – Інтерфейс авторизації користувача

Після проходження процедури автентифікації, користувач отримує доступ до розширеного функціоналу системи, включно з можливістю перегляду історії замовлень. Реалізація механізму авторизації дозволяє обмежити доступ до окремих функцій, що є важливим для забезпечення безпеки та цілісності даних.

Під час оформлення замовлення, клієнтська частина передає інформацію про обрані товари, після чого сервер виконує обчислення загальної вартості та зберігає отримані дані [29]. Сторінка оформлення замовлення наведена на рис. 3.4.

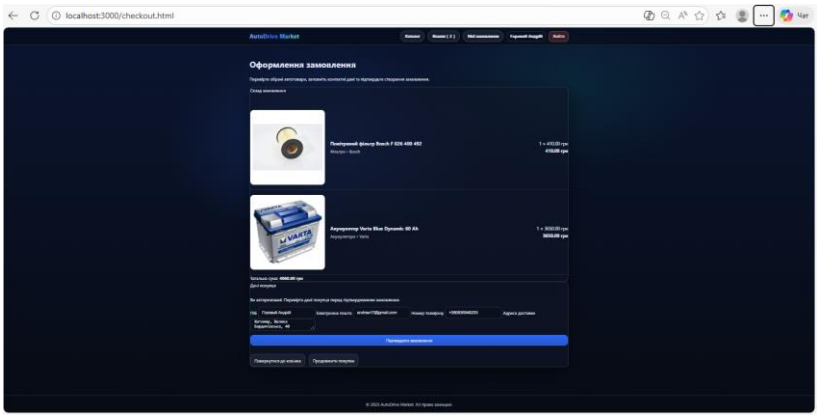


Рисунок 3.4 – Сторінка оформлення замовлення

Зазначена сторінка, забезпечує введення необхідних даних для оформлення замовлення та відображає обрані товари разом з їх вартістю. Перед

передачею даних на сервер, виконується перевірка їх коректності, що дозволяє запобігти виникненню помилок під час обробки запиту.

Після створення замовлення, користувач отримує можливість перегляду власної історії покупок, що реалізовано на окремій сторінці. Відображення замовлень виконується на основі даних, отриманих із серверної частини, що забезпечує актуальність інформації. Сторінка історії замовлень користувача наведена на рисунку 3.5.

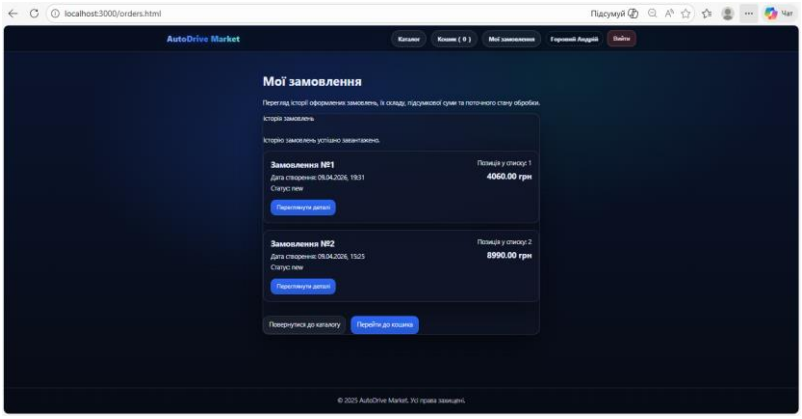


Рисунок 3.5 – Сторінка історії замовлень користувача

Реалізований інтерфейс дозволяє користувачу відстежувати стан замовлень та отримувати інформацію про їх обробку, що підвищує прозорість роботи системи та покращує взаємодію користувача з вебзастосунком.

В адміністративній частині реалізовано можливість керування замовленнями, що передбачає зміну їх статусів відповідно до етапів виконання. Це дозволяє контролювати процес обробки замовлень та оперативно реагувати на зміни. Інтерфейс керування замовленнями в адміністративній панелі наведено на рисунку 3.6.

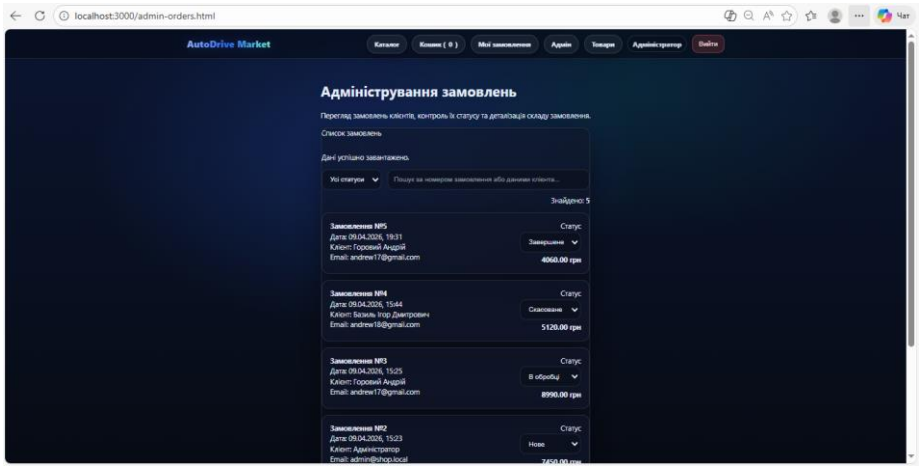


Рисунок 3.6 – Інтерфейс керування замовленнями в адміністративній панелі

Впроваджений функціонал забезпечує повну інтеграцію всіх компонентів вебзастосунку та дозволяє реалізувати основні сценарії роботи інтернет-магазину. Узгоджена взаємодія між клієнтською та серверною частинами створює цілісну систему, яка відповідає вимогам до сучасних вебзастосунків електронної комерції та забезпечує ефективну обробку даних і зручність використання.

3.4 Розробка клієнтської частини вебзастосунку засобами HTML, CSS та JavaScript

Клієнтська частина вебзастосунку інтернет-магазину автотоварів забезпечує взаємодію користувача з системою та відповідає за відображення даних, отриманих з серверної частини та за формування запитів до API. Реалізація інтерфейсу здійснюється засобами HTML для структурування вмісту, CSS для оформлення зовнішнього вигляду та JavaScript для реалізації динамічної поведінки сторінок та обробки подій користувача.

На етапі розробки клієнтської частини, особливу увагу приділено формуванню зрозумілої структури сторінок, яка дозволяє користувачу швидко орієнтуватися у функціональних можливостях системи. Основні сторінки вебзастосунку включають каталог товарів, сторінку детального перегляду,

кошик, оформлення замовлення, історію замовлень, сторінки авторизації та адміністративного керування. Кожна зі сторінок, реалізована як окремий HTML-документ, що містить відповідні елементи інтерфейсу та підключення до скриптів.

HTML-структура сторінок побудована з використанням семантичних елементів, що дозволяє забезпечити логічне групування інформації та підвищити зрозумілість розмітки. Ключові блоки сторінки, зокрема навігаційна панель, основна область контенту та допоміжні елементи, мають чітке розмежування, що сприяє зручності подальшої стилізації та модифікації.

Оформлення інтерфейсу реалізовано засобами CSS, що дозволяє забезпечити єдиний стиль для всіх сторінок вебзастосунку. Використання сучасних можливостей CSS, зокрема гнучких контейнерів та адаптивних відступів, дозволяє досягти коректного відображення елементів на різних розмірах екранів. В рамках проєкту, досягнуто узгодженості кольорової гами, розмірам елементів та розташуванню кнопок, що формує цілісне візуальне сприйняття системи. Візуальне оформлення інтерфейсу вебзастосунку наведено на рисунку 3.7.

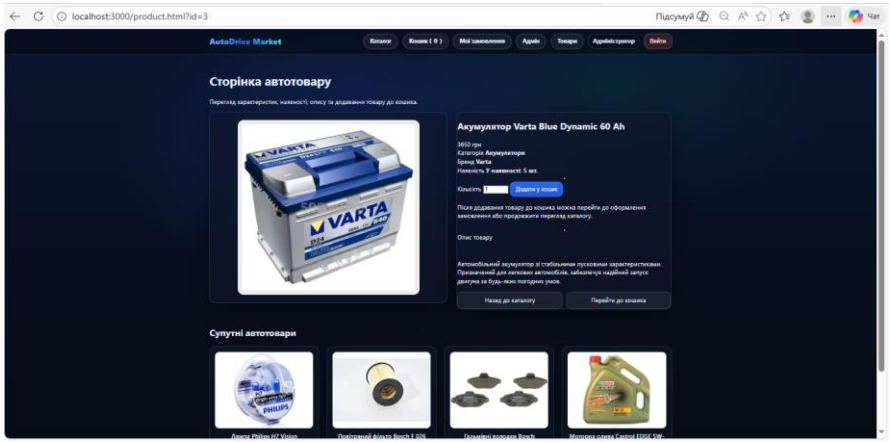


Рисунок 3.7 – Візуальне оформлення інтерфейсу вебзастосунку

Представлений інтерфейс демонструє результати застосування стилів, що забезпечують читабельність тексту, зручність взаємодії та логічну ієрархію елементів. Розташування компонентів інтерфейсу сприяє швидкому виконанню основних дій, пов’язаних з переглядом товарів та оформленням замовлення.

Функціональність клієнтської частини, реалізована за допомогою мови JavaScript, яка забезпечує обробку подій користувача, формування запитів до серверної частини та оновлення вмісту сторінок. Застосування асинхронних запитів дозволяє отримувати дані з сервера без перезавантаження сторінки, що значно покращує швидкість роботи застосунку.

Значна частина логіки клієнтської частини пов’язана з роботою з кошиком, який реалізовано з використанням локального сховища браузера. При відкритті сторінки кошика дані з локального сховища зчитуються та відображаються у вигляді списку позицій з можливістю зміни кількості та видалення товарів. Відображення кошика з використанням локального сховища браузера. Відображення кошика з використанням локального сховища браузера представлено на рисунку 3.8.

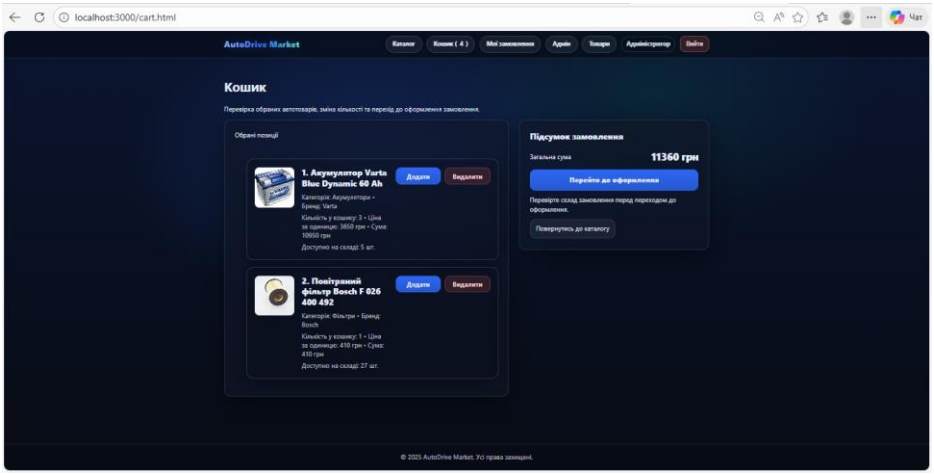


Рисунок 3.8 – Відображення кошика з використанням локального сховища браузера

Реалізація сторінки оформлення замовлення передбачає взаємодію між клієнтською та серверною частинами. Після введення даних користувача, формується запит до серверу, який містить інформацію про товари та контактні дані. Перед відправленням запиту виконується перевірка введених значень, що дозволяє уникнути помилок і підвищити якість обробки даних.

В процесі розробки клієнтської частини також реалізовано механізми відображення повідомлень про результати виконання операцій, що дозволяє користувачу отримувати зворотний зв’язок від системи. Наприклад, після успішного оформлення замовлення відображається повідомлення про створення запису, а у випадку помилки – відповідне попередження.

Важливим елементом клієнтської частини є взаємодія з API серверної частини, яка реалізується через HTTP-запити. Кожна дія користувача, що потребує доступу до даних, супроводжується формуванням запиту до відповідного маршруту сервера. Отримані відповіді обробляються JavaScript-кодом та відображаються у вигляді змін в інтерфейсі.

3.5 Забезпечення захисту даних та стабільності роботи вебзастосунку

Забезпечення захисту даних та стабільності функціонування вебзастосунку інтернет-магазину автотоварів є необхідною умовою його ефективного використання, оскільки обробка персональної інформації користувачів і даних про замовлення, потребує реалізації відповідних механізмів безпеки. В процесі розробки, було враховано основні принципи захисту інформації, які дозволяють мінімізувати ризики несанкціонованого доступу, втрати даних або порушення цілісності системи [30].

Одним з ключових аспектів захисту є забезпечення безпечного зберігання облікових даних користувачів. В системі реалізовано механізм хешування паролів перед їх збереженням в базі даних, що унеможливлює відновлення початкового значення пароля навіть при отриманні доступу до бази даних. Використання криптографічних алгоритмів хешування дає можливість суттєво підвищити рівень захищеності персональної інформації користувачів.

Для контролю доступу до функціональності вебзастосунку, реалізовано механізм автентифікації та авторизації, який базується на використанні токенів. Після успішного входу в систему, користувач отримує унікальний токен, який

використовується для підтвердження його прав доступу при виконанні подальших запитів. Серверна частина перевіряє коректність токена перед обробкою кожного запиту, що дозволяє обмежити доступ до захищених ресурсів та запобігти виконанню операцій неавторизованими особами [31].

В системі передбачено ролі, які визначають доступ до певних функцій, зокрема до адміністративної частини вебзастосунку. Це дозволяє обмежити можливість зміни даних лише для користувачів з відповідними правами та запобігти несанкціонованому втручанню в роботу системи.

З метою підвищення стабільності роботи вебзастосунку реалізовано механізми перевірки вхідних даних, що надходять від клієнтської частини. Перед обробкою запитів, виконується контроль правильності введених значень, що дозволяє уникнути помилок під час виконання операцій та запобігти некоректному внесенню інформації до бази даних [32].

Стабільність роботи системи забезпечується за рахунок використання транзакцій при виконанні критичних операцій, пов’язаних зі створенням замовлень. В процесі оформлення замовлення всі дії, включно зі збереженням інформації про замовлення та його склад, виконуються як єдина операція. У разі виникнення помилки, виконання змін не фіксується, що дозволяє уникнути часткового збереження даних і забезпечити їх узгодженість. У разі виникнення помилки система формує повідомлення про помилку, що дозволяє користувачу скоригувати введені дані та повторити операцію.

Окрім цього, стабільність роботи вебзастосунку забезпечується за рахунок обробки помилок на серверному рівні. Всі запити, які надходять до сервера, супроводжуються перевіркою коректності параметрів, а у випадку виникнення помилки формується відповідна відповідь, яка передається клієнтській частині.

Безпосередній доступ клієнта до бази даних відсутній, що унеможливлює несанкціоноване втручання в структуру або зміст даних. Всі операції виконуються через сервер, який контролює їх виконання та перевіряє права доступу.

Поєднання контролю доступу, перевірки даних, використання транзакцій та обробки помилок забезпечує цілісність інформації, захист персональних даних користувачів і безперебійну роботу системи навіть у випадку виникнення нестандартних ситуацій.

3.6 Висновки до розділу 3

В третьому розділі роботи, здійснено розробку та впровадження вебзастосунку інтернет-магазину автотоварів з використанням сучасних технологій. Реалізовано серверну частину на базі Node.js та Express.js, що забезпечує обробку запитів та взаємодію з базою даних SQLite.

Впроваджено функціонал роботи з товарами, користувачами та замовленнями, що дозволяє виконувати основні операції електронної комерції. Розроблено клієнтську частину засобами HTML, CSS та JavaScript, яка забезпечує зручний інтерфейс та динамічну взаємодію користувача з системою. Реалізовано механізми захисту даних і забезпечення стабільності роботи застосунку. Отримані результати підтверджують можливість створення функціонального вебзастосунку, що відповідає поставленим вимогам.

ВИСНОВКИ

В процесі виконання кваліфікаційної роботи, було здійснено комплексне дослідження особливостей функціонування інтернет-магазинів автотоварів та реалізовано вебзастосунок, який забезпечує повний цикл взаємодії користувача з системою в рамках електронної комерції. Проведений аналіз дозволив встановити, що сучасні інтернет-магазини в даній сфері, повинні поєднувати широкий асортимент продукції, зручну навігацію та високий рівень інформативності, оскільки процес вибору автотоварів безпосередньо пов’язаний з технічними характеристиками та сумісністю продукції.

В ході дослідження предметної області, було визначено ключові особливості функціонування систем такого типу, серед яких значний обсяг даних, необхідність забезпечення швидкого доступу до інформації та важливість коректного представлення характеристик товарів. Аналіз існуючих вебзастосунків дозволив виявити як ефективні рішення, пов’язані з організацією каталогів і пошуку, так і недоліки, що проявляються в складності інтерфейсу або недостатній зручності користування.

В результаті проєктування, було визначено структуру веб-застосунку, яка забезпечує взаємодію клієнтської частини, серверного рівня та бази даних. Розроблена модель даних дозволяє ефективно зберігати інформацію про товари, користувачів та замовлення, підтримуючи цілісність та узгодженість даних. Запропонована архітектура системи забезпечує можливість її подальшого розширення та адаптації до нових функціональних вимог.

Практична реалізація вебзастосунку виконана з використанням сучасних технологій, що дозволило створити стабільну та функціональну систему. Серверна частина забезпечує обробку запитів, виконання бізнес-логіки та взаємодію з базою даних, тоді як клієнтська частина відповідає за відображення інформації та взаємодію користувача з інтерфейсом. Узгоджена робота даних

компонентів, дозволяє забезпечити швидкий доступ до даних та зручність використання системи.

Реалізований функціонал охоплює основні процеси, пов’язані з роботою інтернет-магазину, включно з переглядом каталогу товарів, додаванням продукції до кошика, оформленням замовлень та переглядом історії покупок. Додатково, впроваджено адміністративну частину, що дозволяє здійснювати керування товарами та замовленнями, забезпечуючи контроль за станом системи та актуальністю даних.

Особливу увагу приділено забезпеченню захисту даних та стабільності роботи вебзастосунку. Використання механізмів автентифікації, контролю доступу та перевірки вхідних даних дозволяє мінімізувати ризики несанкціонованого доступу та забезпечити коректність обробки інформації. Реалізовані підходи сприяють підвищенню надійності системи та її стійкості до можливих помилок під час експлуатації.

Отриманий результат демонструє можливість створення повноцінного веб-застосунку інтернет-магазину автотоварів, який відповідає сучасним вимогам до електронної комерції та забезпечує ефективну взаємодію користувача з інформаційною системою. Розроблений програмний продукт може бути використаний як основа для подальшого розвитку, вдосконалення функціональності та впровадження в практичну діяльність.

Перспективи подальшого розвитку системи, пов’язані з розширенням функціональних можливостей, зокрема впровадженням більш складних механізмів пошуку та фільтрації, інтеграцією платіжних систем, а також підвищенням рівня адаптивності інтерфейсу для різних типів пристроїв. Це дозволить підвищити ефективність використання вебзастосунку та забезпечити його відповідність сучасним тенденціям розвитку інформаційних технологій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Швиденко М. З., Касаткіна О. М., Швиденко О. М. Електронна комерція : підручник. Київ : ФОП Ямчинський О. В., 2020. 478 с.

2. Краус Н. М., Краус К. М., Манжура О. В. Електронна комерція та Інтернет-торгівля : навч. посіб. Київ : Аграр Медіа Груп, 2021. 454 с.

3. Laudon K. C., Traver C. G. E-Commerce: Business, Technology, Society. 17th ed. Harlow : Pearson, 2021. 912 p.

4. Autodoc. URL : https://www.autodoc.eu (дата звернення : 10.02.2026).

5. OnlineCarParts. URL : https://www.onlinecarparts.co.uk (дата звернення : 10.02.2026).

6. AvtoTO. URL : https://avtoto.com.ua (дата звернення : 13.02.2026).

7. Signeda. URL : https://www.signeda.com (дата звернення : 14.02.2026).

8. Баран С. В. Основи web-програмування : навч. посіб. Кривий Ріг : Державний університет економіки і технологій, 2021. 316 с.

9. Richards M. Software Architecture Patterns. Sebastopol : O'Reilly Media, 2015. 54 p.

10. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack. 2nd ed. Sebastopol : O'Reilly Media, 2019. 348 p.

11. Casciaro M., Mammino L. Node.js Design Patterns. 3rd ed. Birmingham : Packt Publishing, 2020. 660 p.

12. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань. Кн. 2. Системи управління базами даних та знань : підручник. 3-тє вид., стер. Львів : Магнолія 2006, 2024. 582 с.

13. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. New York : McGraw-Hill Education, 2020. 1376 p.

14. SQLite Documentation. URL : https://www.sqlite.org/docs.html (дата звернення : 02.03.2026).

15. Полєтаєв М. І., Коновалов С. М. Методи, моделі та технології проектування інформаційних управляючих систем : метод. матеріали до лаб. занять. Одеса : ОНПУ, 2019.

16. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd ed. Boston : Addison-Wesley, 2004. 208 p.

17. Flanagan D. JavaScript : The Definitive Guide. 7th ed. Sebastopol : O'Reilly Media, 2020. 706 p.

18. Mozilla Developer Network. JavaScript documentation. URL : https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення : 05.03.2026).

19. Mozilla Developer Network. Fetch API. URL : https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення : 06.03.2026).

20. Express.js. URL : https://expressjs.com (дата звернення : 08.03.2026).

21. Node.js Documentation. URL : https://nodejs.org/en/docs (дата звернення : 17.03.2026).

22. Grigorik I. High Performance Browser Networking. Sebastopol : O'Reilly Media, 2013. 400 p.

23. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 616 p.

24. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : doctoral dissertation. Irvine : University of California, 2000. 180 p.

25. Козюра В. Д., Хорошко В. О., Шелест М. Є., Ткач Ю. М., Балюнов О. О. Захист інформації в комп'ютерних системах : підручник. Ніжин : ФОП Лук'яненко В. В., ТПК «Орхідея», 2020. 236 с.

26. OWASP Top 10:2021. URL : https://owasp.org/Top10 (дата звернення : 19.03.2026).

27. OWASP Web Security Testing Guide. URL : https://owasp.org/www-project-web-security-testing-guide (дата звернення : 22.03.2026).

28. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.

29. ISO/IEC 25010:2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Geneva : ISO, 2011. 34 p.

30. Martin R. C. Clean Architecture: A Craftsman’s Guide to Software Structure and Design. Boston : Pearson, 2018. 432 p.

31. Google Developers. Web Performance. URL : https://web.dev/learn/performance (дата звернення : 01.04.2026).

32. HTML Living Standard. URL : https://html.spec.whatwg.org (дата звернення : 02.04.2026).

ДОДАТОК А

Серверна частина вебзастосунку

```
"use strict";

const path = require("path");
const fs = require("fs");
const express = require("express");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");
const Database = require("better-sqlite3");

const app = express();
const PORT = process.env.PORT || 3000;

const CLIENT_DIR = path.join(__dirname, "..", "..", "client");
const DB_PATH =
  process.env.DB_PATH ||
  path.join(__dirname, "shop.db");

const JWT_SECRET = process.env.JWT_SECRET || "dev_secret_change_me";
const JWT_EXPIRES_IN = "7d";

const ADMIN_EMAIL = String(process.env.ADMIN_EMAIL ||
"admin@shop.local").trim().toLowerCase();
const ADMIN_PASSWORD = String(process.env.ADMIN_PASSWORD || "admin12345");
const ADMIN_NAME = String(process.env.ADMIN_NAME || "Адміністратор").trim();

const GUEST_EMAIL = String(process.env.GUEST_EMAIL ||
"guest@shop.local").trim().toLowerCase();
const GUEST_NAME = String(process.env.GUEST_NAME || "Гість").trim();

const ALLOWED_STATUSES = ["new", "processing", "completed", "cancelled"];

/* ================ DB ================ */

const db = new Database(DB_PATH);
db.pragma("foreign_keys = ON");

function hasColumn(table, column) {
  const columns = db.prepare(`PRAGMA table_info(${table})`).all();
  return columns.some((item) => item.name === column);
}

function addColumnIfMissing(table, column, sqlTypeAndDefault) {
  if (!hasColumn(table, column)) {
    db.exec(`ALTER TABLE ${table} ADD COLUMN ${column} ${sqlTypeAndDefault}`);
  }
}

function normalizeImagePath(value) {
  let image = String(value || "").trim();

  if (!image) {
    return "/assets/img/charger.png";
  }
```

```
  image = image.replace(/^\/assets\/images\//, "/assets/img/");

  if (!image.startsWith("/")) {
    image = "/" + image;
  }

  if (image.startsWith("/assets/img/")) {
    image = image.replace(/\.(jpg|jpeg)$/i, ".png");
  }

  return image;
}

function productToClient(product) {
  const priceCents = Number(product.price_cents || 0);
  const price =
    Number.isFinite(Number(product.price))
      ? Number(product.price)
      : (priceCents ? priceCents / 100 : 0);

  return {
    id: product.id,
    name: product.name,
    category: product.category || "Автотовари",
    brand: product.brand || "",
    description: product.description || "",
    stock: Number(product.stock || 0),
    price,
    price_cents: Math.round(price * 100),
    image: normalizeImagePath(product.image || product.image_url),
    image_url: normalizeImagePath(product.image || product.image_url)
  };
}


db.exec(`
CREATE TABLE IF NOT EXISTS users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  email TEXT UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  role TEXT DEFAULT 'user',
  phone TEXT DEFAULT '',
  address TEXT DEFAULT '',
  created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
`);

db.exec(`
CREATE TABLE IF NOT EXISTS products (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  category TEXT NOT NULL,
  brand TEXT DEFAULT '',
  price REAL DEFAULT 0,
  price_cents INTEGER DEFAULT 0,
  stock INTEGER NOT NULL DEFAULT 0,
  image TEXT DEFAULT '',
  image_url TEXT DEFAULT '',
  description TEXT DEFAULT '',
  created_at TEXT DEFAULT CURRENT_TIMESTAMP
);
`);
```

```
db.exec(`
CREATE TABLE IF NOT EXISTS orders (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  user_id INTEGER NOT NULL,
  customer_name TEXT NOT NULL,
  customer_email TEXT NOT NULL,
  customer_phone TEXT DEFAULT '',
  customer_address TEXT DEFAULT '',
  total REAL NOT NULL DEFAULT 0,
  status TEXT DEFAULT 'new',
  created_at TEXT DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (user_id) REFERENCES users(id)
);
`);

db.exec(`
CREATE TABLE IF NOT EXISTS order_items (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  order_id INTEGER NOT NULL,
  product_id INTEGER NOT NULL,
  qty INTEGER NOT NULL,
  price REAL NOT NULL,
  FOREIGN KEY (order_id) REFERENCES orders(id) ON DELETE CASCADE,
  FOREIGN KEY (product_id) REFERENCES products(id)
);
`);


addColumnIfMissing("users", "role", "TEXT DEFAULT 'user'");
addColumnIfMissing("users", "phone", "TEXT DEFAULT ''");
addColumnIfMissing("users", "address", "TEXT DEFAULT ''");
addColumnIfMissing("users", "created_at", "TEXT");

addColumnIfMissing("products", "brand", "TEXT DEFAULT ''");
addColumnIfMissing("products", "price", "REAL DEFAULT 0");
addColumnIfMissing("products", "price_cents", "INTEGER DEFAULT 0");
addColumnIfMissing("products", "stock", "INTEGER DEFAULT 0");
addColumnIfMissing("products", "image", "TEXT DEFAULT ''");
addColumnIfMissing("products", "image_url", "TEXT DEFAULT ''");
addColumnIfMissing("products", "description", "TEXT DEFAULT ''");
addColumnIfMissing("products", "created_at", "TEXT");

addColumnIfMissing("orders", "status", "TEXT DEFAULT 'new'");
addColumnIfMissing("orders", "customer_phone", "TEXT DEFAULT ''");
addColumnIfMissing("orders", "customer_address", "TEXT DEFAULT ''");
addColumnIfMissing("orders", "created_at", "TEXT");


const adminExists = db.prepare("SELECT id FROM users WHERE email =
?").get(ADMIN_EMAIL);
if (!adminExists) {
  const passwordHash = bcrypt.hashSync(ADMIN_PASSWORD, 10);
  db.prepare(`
    INSERT INTO users (name, email, password_hash, role, phone, address)
    VALUES (?, ?, ?, 'admin', '', '')
  `).run(ADMIN_NAME, ADMIN_EMAIL, passwordHash);

  console.log(`Admin created: ${ADMIN_EMAIL}`);
}

let guestUser = db.prepare("SELECT id, name, email, role FROM users WHERE email
= ?").get(GUEST_EMAIL);
if (!guestUser) {
```

```
  const randomPassword = `${Date.now()}_${Math.random()}`;
  const passwordHash = bcrypt.hashSync(randomPassword, 10);

  const info = db.prepare(`
    INSERT INTO users (name, email, password_hash, role, phone, address)
    VALUES (?, ?, ?, 'guest', '', '')
  `).run(GUEST_NAME, GUEST_EMAIL, passwordHash);

  guestUser = {
    id: Number(info.lastInsertRowid),
    name: GUEST_NAME,
    email: GUEST_EMAIL,
    role: "guest"
  };

  console.log(`Guest created: ${GUEST_EMAIL}`);
}


app.use(express.json());


app.get("/", (_req, res) => {
  return res.redirect(302, "/catalog.html");
});

app.get("/assets/images/:file", (req, res) => {
  const baseName = String(req.params.file || "").replace(/\.(jpg|jpeg|png)$/i,
"");
  const pngPath = path.join(CLIENT_DIR, "assets", "img", `${baseName}.png`);

  if (fs.existsSync(pngPath)) {
    return res.sendFile(pngPath);
  }

  return res.status(404).end();
});

app.use(express.static(CLIENT_DIR));
app.use("/assets", express.static(path.join(CLIENT_DIR, "assets")));



function signToken(user) {
  return jwt.sign(user, JWT_SECRET, { expiresIn: JWT_EXPIRES_IN });
}

function auth(req, res, next) {
  const header = req.headers.authorization || "";
  const match = header.match(/^Bearer\s+(.+)$/);

  if (!match) {
    return res.status(401).json({ error: "Unauthorized" });
  }

  try {
    req.user = jwt.verify(match[1], JWT_SECRET);
    next();
  } catch {
    return res.status(401).json({ error: "Unauthorized" });
  }
}

function optionalAuth(req, _res, next) {
```

```
  const header = req.headers.authorization || "";
  const match = header.match(/^Bearer\s+(.+)$/);

  if (!match) {
    req.user = null;
    return next();
  }

  try {
    req.user = jwt.verify(match[1], JWT_SECRET);
  } catch {
    req.user = null;
  }

  next();
}

function adminOnly(req, res, next) {
  if (req.user?.role !== "admin") {
    return res.status(403).json({ error: "Forbidden" });
  }

  next();
}


app.post("/api/auth/register", (req, res) => {
  const body = req.body || {};

  const name = String(body.name || "").trim();
  const email = String(body.email || "").trim().toLowerCase();
  const phone = String(body.phone || "").trim();
  const address = String(body.address || "").trim();
  const password = String(body.password || "");

  if (name.length < 3) {
    return res.status(400).json({ error: "Name too short" });
  }

  if (!email || !email.includes("@")) {
    return res.status(400).json({ error: "Invalid email" });
  }

  if (password.length < 6) {
    return res.status(400).json({ error: "Password too short" });
  }

  const exists = db.prepare("SELECT id FROM users WHERE email = ?").get(email);
  if (exists) {
    return res.status(409).json({ error: "Email already exists" });
  }

  const passwordHash = bcrypt.hashSync(password, 10);

  const info = db.prepare(`
    INSERT INTO users (name, email, password_hash, role, phone, address)
    VALUES (?, ?, ?, 'user', ?, ?)
  `).run(name, email, passwordHash, phone, address);

  const user = {
    id: Number(info.lastInsertRowid),
    name,
    email,
    role: "user",
```

```
    phone,
    address
  };

  return res.json({
    user,
    token: signToken(user)
  });
});

app.post("/api/auth/login", (req, res) => {
  const body = req.body || {};

  const email = String(body.email || "").trim().toLowerCase();
  const password = String(body.password || "");

  const userRow = db.prepare("SELECT * FROM users WHERE email = ?").get(email);

  if (!userRow || !bcrypt.compareSync(password, userRow.password_hash)) {
    return res.status(401).json({ error: "Invalid credentials" });
  }

  const user = {
    id: userRow.id,
    name: userRow.name,
    email: userRow.email,
    role: userRow.role || "user",
    phone: userRow.phone || "",
    address: userRow.address || ""
  };

  return res.json({
    user,
    token: signToken(user)
  });
});


app.get("/api/products", (_req, res) => {
  const rows = db.prepare(`
    SELECT *
    FROM products
    ORDER BY id DESC
  `).all();

  return res.json(rows.map(productToClient));
});

app.get("/api/products/:id", (req, res) => {
  const id = Number(req.params.id);
  const row = db.prepare("SELECT * FROM products WHERE id = ?").get(id);

  if (!row) {
    return res.status(404).json({ error: "Not found" });
  }

  return res.json(productToClient(row));
});


app.post("/api/orders", optionalAuth, (req, res) => {
  const body = req.body || {};
  const items = Array.isArray(body.items) ? body.items : [];
  const customer = body.customer || {};
```

```
  if (!items.length) {
    return res.status(400).json({ error: "Empty cart" });
  }

  const customerName = String(customer.name || "").trim();
  const customerEmail = String(customer.email || "").trim();
  const customerPhone = String(customer.phone || "").trim();
  const customerAddress = String(customer.address || "").trim();

  const isAuthorized = Boolean(req.user?.id);

  if (!customerName) {
    return res.status(400).json({ error: "customer.name required" });
  }

  if (isAuthorized) {
    if (!customerEmail) {
      return res.status(400).json({ error: "customer.email required" });
    }
  } else if (!customerEmail && !customerPhone) {
    return res.status(400).json({ error: "customer.email or customer.phone
required" });
  }

  let total = 0;
  const orderRows = [];

  for (const item of items) {
    const productId = Number(item.product_id ?? item.productId);
    const qty = Number(item.qty ?? item.quantity);

    if (!Number.isFinite(productId) || productId <= 0) {
      return res.status(400).json({ error: "Invalid product_id" });
    }

    if (!Number.isFinite(qty) || qty <= 0) {
      return res.status(400).json({ error: "Invalid qty" });
    }

    const product = db.prepare("SELECT * FROM products WHERE id =
?").get(productId);
    if (!product) {
      return res.status(400).json({ error: `Product not found: ${productId}` });
    }

    if (qty > Number(product.stock || 0)) {
      return res.status(400).json({ error: `Not enough stock for:
${product.name}` });
    }

    const unitPrice =
      Number.isFinite(Number(product.price))
        ? Number(product.price)
        : Number(product.price_cents || 0) / 100;

    total += unitPrice * qty;
    orderRows.push({
      productId,
      qty,
      price: unitPrice
    });
  }
```

```
  const userId = isAuthorized ? req.user.id : guestUser.id;
  const storedEmail =
    customerEmail || `phone:${customerPhone || "unknown"}`;

  const insertOrder = db.prepare(`
    INSERT INTO orders (
      user_id,
      customer_name,
      customer_email,
      customer_phone,
      customer_address,
      total,
      status
    )
    VALUES (?, ?, ?, ?, ?, ?, 'new')
  `);

  const insertItem = db.prepare(`
    INSERT INTO order_items (order_id, product_id, qty, price)
    VALUES (?, ?, ?, ?)
  `);

  const updateStock = db.prepare(`
    UPDATE products
    SET stock = stock - ?
    WHERE id = ?
  `);

  const transaction = db.transaction(() => {
    const orderInfo = insertOrder.run(
      userId,
      customerName,
      storedEmail,
      customerPhone,
      customerAddress,
      total
    );

    const orderId = Number(orderInfo.lastInsertRowid);

    for (const row of orderRows) {
      insertItem.run(orderId, row.productId, row.qty, row.price);
      updateStock.run(row.qty, row.productId);
    }

    return orderId;
  });

  const orderId = transaction();

  return res.json({
    id: orderId,
    order: {
      id: orderId,
      total,
      status: "new"
    },
    guest: !isAuthorized
  });
});

app.get("/api/orders/my", auth, (req, res) => {
  const orders = db.prepare(`
    SELECT
```

```
      id,
      total,
      status,
      created_at,
      customer_name,
      customer_email,
      customer_phone,
      customer_address
    FROM orders
    WHERE user_id = ?
    ORDER BY id DESC
  `).all(req.user.id);

  return res.json({ orders });
});

app.get("/api/orders/:id", auth, (req, res) => {
  const orderId = Number(req.params.id);

  const order = db.prepare(`
    SELECT
      id,
      user_id,
      total,
      status,
      created_at,
      customer_name,
      customer_email,
      customer_phone,
      customer_address
    FROM orders
    WHERE id = ?
  `).get(orderId);

  if (!order) {
    return res.status(404).json({ error: "Not found" });
  }

  if (req.user.role !== "admin" && Number(order.user_id) !==
Number(req.user.id)) {
    return res.status(403).json({ error: "Forbidden" });
  }

  const items = db.prepare(`
    SELECT
      oi.product_id,
      oi.qty,
      oi.price,
      p.name AS product_name
    FROM order_items oi
    LEFT JOIN products p ON p.id = oi.product_id
    WHERE oi.order_id = ?
  `).all(orderId);

  return res.json({ order, items });
});


app.get("/api/admin/products", auth, adminOnly, (_req, res) => {
  const rows = db.prepare(`
    SELECT *
    FROM products
    ORDER BY id DESC
  `).all();
```

```
  return res.json({
    products: rows.map(productToClient)
  });
});

app.post("/api/admin/products", auth, adminOnly, (req, res) => {
  const body = req.body || {};

  const name = String(body.name || "").trim();
  const category = String(body.category || "").trim();
  const brand = String(body.brand || "").trim();
  const price = Number(body.price);
  const stock = Number(body.stock);
  const description = String(body.description || "").trim();
  const imageValue = body.image_url ?? body.image;

  if (!name) {
    return res.status(400).json({ error: "name required" });
  }

  if (!category) {
    return res.status(400).json({ error: "category required" });
  }

  if (!Number.isFinite(price) || price < 0) {
    return res.status(400).json({ error: "Invalid price" });
  }

  if (!Number.isInteger(stock) || stock < 0) {
    return res.status(400).json({ error: "Invalid stock" });
  }

  const imagePath = normalizeImagePath(imageValue);

  const info = db.prepare(`
    INSERT INTO products (
      name,
      category,
      brand,
      price,
      price_cents,
      stock,
      image,
      image_url,
      description
    )
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)
  `).run(
    name,
    category,
    brand,
    price,
    Math.round(price * 100),
    stock,
    imagePath,
    imagePath,
    description
  );

  const product = db.prepare("SELECT * FROM products WHERE id =
?").get(Number(info.lastInsertRowid));

  return res.json({
```

```
    product: productToClient(product)
  });
});

app.put("/api/admin/products/:id", auth, adminOnly, (req, res) => {
  const id = Number(req.params.id);
  const existing = db.prepare("SELECT * FROM products WHERE id = ?").get(id);

  if (!existing) {
    return res.status(404).json({ error: "Not found" });
  }

  const body = req.body || {};

  const name = String(body.name ?? existing.name).trim();
  const category = String(body.category ?? existing.category).trim();
  const brand = String(body.brand ?? existing.brand).trim();
  const price = Number(body.price ?? existing.price);
  const stock = Number(body.stock ?? existing.stock);
  const description = String(body.description ?? existing.description ??
"").trim();
  const imageValue = body.image_url ?? body.image ?? existing.image_url ??
existing.image;

  if (!name) {
    return res.status(400).json({ error: "name required" });
  }

  if (!category) {
    return res.status(400).json({ error: "category required" });
  }

  if (!Number.isFinite(price) || price < 0) {
    return res.status(400).json({ error: "Invalid price" });
  }

  if (!Number.isInteger(stock) || stock < 0) {
    return res.status(400).json({ error: "Invalid stock" });
  }

  const imagePath = normalizeImagePath(imageValue);

  db.prepare(`
    UPDATE products
    SET
      name = ?,
      category = ?,
      brand = ?,
      price = ?,
      price_cents = ?,
      stock = ?,
      image = ?,
      image_url = ?,
      description = ?
    WHERE id = ?
  `).run(
    name,
    category,
    brand,
    price,
    Math.round(price * 100),
    stock,
    imagePath,
    imagePath,
```

```
    description,
    id
  );

  const product = db.prepare("SELECT * FROM products WHERE id = ?").get(id);

  return res.json({
    product: productToClient(product)
  });
});

app.delete("/api/admin/products/:id", auth, adminOnly, (req, res) => {
  const id = Number(req.params.id);

  const exists = db.prepare("SELECT id FROM products WHERE id = ?").get(id);
  if (!exists) {
    return res.status(404).json({ error: "Not found" });
  }

  const used = db.prepare("SELECT COUNT(*) AS c FROM order_items WHERE
product_id = ?").get(id);
  if (Number(used?.c || 0) > 0) {
    return res.status(409).json({ error: "Cannot delete: product is used in
orders" });
  }

  db.prepare("DELETE FROM products WHERE id = ?").run(id);

  return res.json({ ok: true });
});


app.get("/api/admin/orders", auth, adminOnly, (_req, res) => {
  const orders = db.prepare(`
    SELECT
      o.id,
      o.total,
      o.status,
      o.created_at,
      o.customer_name,
      o.customer_email,
      o.customer_phone,
      o.customer_address,
      u.name AS user_name,
      u.email AS user_email
    FROM orders o
    LEFT JOIN users u ON u.id = o.user_id
    ORDER BY o.id DESC
  `).all();

  return res.json({
    orders,
    statuses: ALLOWED_STATUSES
  });
});

app.get("/api/admin/orders/:id", auth, adminOnly, (req, res) => {
  const orderId = Number(req.params.id);

  const order = db.prepare(`
    SELECT
      o.id,
      o.total,
      o.status,
```

```
      o.created_at,
      o.customer_name,
      o.customer_email,
      o.customer_phone,
      o.customer_address,
      u.name AS user_name,
      u.email AS user_email
    FROM orders o
    LEFT JOIN users u ON u.id = o.user_id
    WHERE o.id = ?
  `).get(orderId);

  if (!order) {
    return res.status(404).json({ error: "Not found" });
  }

  const items = db.prepare(`
    SELECT
      oi.product_id,
      oi.qty,
      oi.price,
      p.name AS product_name
    FROM order_items oi
    LEFT JOIN products p ON p.id = oi.product_id
    WHERE oi.order_id = ?
  `).all(orderId);

  return res.json({
    order,
    items,
    statuses: ALLOWED_STATUSES
  });
});

app.put("/api/admin/orders/:id", auth, adminOnly, (req, res) => {
  const orderId = Number(req.params.id);
  const status = String(req.body?.status || "").trim();

  if (!ALLOWED_STATUSES.includes(status)) {
    return res.status(400).json({ error: "Invalid status" });
  }

  const exists = db.prepare("SELECT id FROM orders WHERE id = ?").get(orderId);
  if (!exists) {
    return res.status(404).json({ error: "Not found" });
  }

  db.prepare("UPDATE orders SET status = ? WHERE id = ?").run(status, orderId);

  return res.json({ ok: true });
});

/* optional compatibility route */
app.patch("/api/admin/orders/:id/status", auth, adminOnly, (req, res) => {
  const orderId = Number(req.params.id);
  const status = String(req.body?.status || "").trim();

  if (!ALLOWED_STATUSES.includes(status)) {
    return res.status(400).json({ error: "Invalid status" });
  }

  const exists = db.prepare("SELECT id FROM orders WHERE id = ?").get(orderId);
  if (!exists) {
    return res.status(404).json({ error: "Not found" });
```

```
  }

  db.prepare("UPDATE orders SET status = ? WHERE id = ?").run(status, orderId);

  return res.json({ ok: true });
});


app.listen(PORT, () => {
  console.log(`Server running: http://localhost:${PORT}`);
  console.log(`CLIENT_DIR = ${CLIENT_DIR}`);
  console.log(`DB_PATH = ${DB_PATH}`);
});
```

ДОДАТОК Б

Реалізація обробки замовлень у веб-застосунку

```
"use strict";

(function () {
  const statusEl = document.getElementById("ordersStatus");
  const listEl = document.getElementById("ordersList");

  if (!statusEl || !listEl) return;

  function setStatus(message, isError = false) {
    statusEl.textContent = message || "";
    statusEl.className = isError ? "alert alert-error" : "muted small";
  }

  function formatMoney(value) {
    const number = Number(value || 0);
    if (!Number.isFinite(number)) return "0.00 грн";
    return `${number.toFixed(2)} грн`;
  }

  function escapeHtml(value) {
    return String(value ?? "")
      .replaceAll("&", "&amp;")
      .replaceAll("<", "&lt;")
      .replaceAll(">", "&gt;")
      .replaceAll('"', "&quot;")
      .replaceAll("'", "&#039;");
  }

  function normalizeDate(dateValue) {
    if (!dateValue) return "—";

    const raw = String(dateValue).trim();
    if (!raw) return "—";

    const isoCandidate = raw.includes("T") ? raw : raw.replace(" ", "T");
    const date = new Date(isoCandidate);

    if (Number.isNaN(date.getTime())) {
      return raw;
    }

    return new Intl.DateTimeFormat("uk-UA", {
      year: "numeric",
      month: "2-digit",
      day: "2-digit",
      hour: "2-digit",
      minute: "2-digit"
    }).format(date);
  }

  function isUnauthorizedError(error) {
    const message = String(error?.message || "").toLowerCase();
    return message.includes("unauthorized") || message.includes("401");
  }
```

```
  function clearAuthStorage() {
    try {
      localStorage.removeItem("auth");
      localStorage.removeItem("auth_token");
      localStorage.removeItem("token");
      localStorage.removeItem("auth_user");
      localStorage.removeItem("user");
    } catch {
      /* ignore */
    }

    if (typeof clearAuth === "function") {
      clearAuth();
    }
  }

  async function apiAuth(url, options = {}) {
    if (typeof apiAuthJson === "function") {
      return apiAuthJson(url, options);
    }

    const authRaw = localStorage.getItem("auth");
    const auth = authRaw ? JSON.parse(authRaw) : null;
    const token =
      auth?.token ||
      localStorage.getItem("auth_token") ||
      localStorage.getItem("token");

    const response = await fetch(url, {
      ...options,
      headers: {
        "Content-Type": "application/json",
        ...(token ? { Authorization: `Bearer ${token}` } : {}),
        ...(options.headers || {})
      }
    });

    const data = await response.json().catch(() => null);

    if (!response.ok) {
      throw new Error(data?.error || data?.message || `HTTP
${response.status}`);
    }

    return data;
  }

  function renderEmptyState() {
    listEl.innerHTML = `
      <div class="card" style="padding:16px;">
        У вас поки що немає оформлених замовлень.
      </div>
    `;
  }

  function renderOrderDetails(container, order, items) {
    if (!items || !items.length) {
      container.innerHTML = `
        <div class="muted small" style="margin-top:12px;">
          Для цього замовлення позиції не знайдено.
        </div>
      `;
      return;
```

```
    }

    const itemsHtml = items
      .map((item) => {
        const quantity = Number(item.qty ?? item.quantity ?? 0);
        const price = Number(item.price ?? 0);
        const rowTotal = quantity * price;
        const productName =
          item.product_name ||
          item.name ||
          `Товар #${item.product_id ?? item.id ?? "—"}`;

        return `
          <div style="padding:12px 0;border-top:1px solid
rgba(255,255,255,0.08);">
            <div style="font-weight:600;">${escapeHtml(productName)}</div>
            <div class="muted small" style="margin-top:6px;">
              Кількість: ${quantity} · Ціна: ${formatMoney(price)} · Сума:
${formatMoney(rowTotal)}
            </div>
          </div>
        `;
      })
      .join("");

    container.innerHTML = `
      <div style="margin-top:12px;padding:14px;border:1px solid
rgba(255,255,255,0.08);border-radius:12px;background:rgba(255,255,255,0.02);">
        <div style="font-weight:600;margin-bottom:8px;">Склад замовлення</div>
        ${itemsHtml}
      </div>
    `;
  }

  async function toggleOrderDetails(button, orderId) {
    const card = button.closest("[data-order-card]");
    if (!card) return;

    const detailsBox = card.querySelector("[data-order-details]");
    if (!detailsBox) return;

    const expanded = button.getAttribute("data-expanded") === "true";

    if (expanded) {
      detailsBox.innerHTML = "";
      detailsBox.style.display = "none";
      button.textContent = "Переглянути деталі";
      button.setAttribute("data-expanded", "false");
      return;
    }

    try {
      button.disabled = true;
      button.textContent = "Завантаження...";

      const data = await apiAuth(`/api/orders/${encodeURIComponent(orderId)}`);
      const order = data?.order || null;
      const items = Array.isArray(data?.items) ? data.items : [];

      renderOrderDetails(detailsBox, order, items);
      detailsBox.style.display = "block";
      button.textContent = "Сховати деталі";
      button.setAttribute("data-expanded", "true");
    } catch (error) {
```

```
      console.error(error);

      if (isUnauthorizedError(error)) {
        clearAuthStorage();
        window.location.href = "/login.html";
        return;
      }

      detailsBox.innerHTML = `
        <div class="alert alert-error" style="margin-top:12px;">
          Не вдалося завантажити деталі замовлення.
        </div>
      `;
      detailsBox.style.display = "block";
      button.textContent = "Переглянути деталі";
      button.setAttribute("data-expanded", "false");
    } finally {
      button.disabled = false;
    }
  }

  function renderOrders(orders) {
    if (!orders.length) {
      renderEmptyState();
      return;
    }

    listEl.innerHTML = orders
      .map((order, index) => {
        const orderId = order.id ?? "—";
        const displayNumber = index + 1;
        const createdAt = normalizeDate(order.created_at || order.date ||
order.createdAt);
        const status = order.status || "new";
        const total = Number(order.total ?? order.total_price ?? 0);

        return `
          <div class="card" data-order-card style="padding:16px;margin-
bottom:14px;">
            <div style="display:flex;justify-content:space-
between;gap:16px;align-items:flex-start;flex-wrap:wrap;">
              <div>
                <div style="font-weight:700;font-size:18px;">Замовлення
№${displayNumber}</div>
                <div class="muted small" style="margin-top:6px;">Дата створення:
${escapeHtml(createdAt)}</div>
                <div class="muted small" style="margin-top:4px;">Статус:
${escapeHtml(status)}</div>
              </div>

              <div style="text-align:right;">
                <div class="muted small">Позиція у списку: ${index + 1}</div>
                <div style="font-weight:700;font-size:18px;margin-
top:6px;">${formatMoney(total)}</div>
              </div>
            </div>

            <div style="margin-top:14px;display:flex;gap:12px;flex-wrap:wrap;">
              <button
                type="button"
                class="btn btn-primary"
                data-order-id="${escapeHtml(orderId)}"
                data-expanded="false"
              >
```

```
              Переглянути деталі
            </button>
          </div>

          <div data-order-details style="display:none;"></div>
        </div>
      `;
    })
    .join("");

  listEl.querySelectorAll("[data-order-id]").forEach((button) => {
    button.addEventListener("click", () => {
      const orderId = button.getAttribute("data-order-id");
      toggleOrderDetails(button, orderId);
    });
  });
}

async function initOrders() {
  try {
    if (typeof updateCartBadge === "function") {
      updateCartBadge();
    }

    setStatus("Завантаження історії замовлень...");

    const data = await apiAuth("/api/orders/my");
    const orders = Array.isArray(data?.orders) ? data.orders : [];

    renderOrders(orders);

    if (orders.length) {
      setStatus("Історію замовлень успішно завантажено.");
    } else {
      setStatus("Історія замовлень порожня.");
    }
  } catch (error) {
    console.error(error);

    if (isUnauthorizedError(error)) {
      clearAuthStorage();
      window.location.href = "/login.html";
      return;
    }

    setStatus(
      error.message || "Не вдалося завантажити історію замовлень.",
      true
    );
    renderEmptyState();
  }
}

document.addEventListener("DOMContentLoaded", initOrders);
                                    })();
```

ДОДАТОК В

Програмна реалізація клієнтської частини веб-застосунку

```
"use strict";

function safeJsonParse(value, fallback) {
  try {
    return JSON.parse(value);
  } catch {
    return fallback;
  }
}

function getToken() {
  const authRaw = localStorage.getItem("auth");
  if (authRaw) {
    const auth = safeJsonParse(authRaw, null);
    if (auth?.token) return auth.token;
  }

  return localStorage.getItem("auth_token") || localStorage.getItem("token") ||
null;
}

function getAuthUser() {
  const authRaw = localStorage.getItem("auth");
  if (authRaw) {
    const auth = safeJsonParse(authRaw, null);
    if (auth?.user) return auth.user;
  }

  const userRaw = localStorage.getItem("auth_user") ||
localStorage.getItem("user");
  return userRaw ? safeJsonParse(userRaw, null) : null;
}

function formatUAH(value) {
  const number = Number(value);
  if (!Number.isFinite(number)) return "0.00 грн";
  return `${number.toFixed(2)} грн`;
}

function escapeHtml(value) {
  return String(value ?? "")
    .replaceAll("&", "&amp;")
    .replaceAll("<", "&lt;")
    .replaceAll(">", "&gt;")
    .replaceAll('"', "&quot;")
    .replaceAll("'", "&#039;");
}

function getCartNormalized() {
  const keys = ["cart", "mas_cart", "shop_cart", "basket"];
  const merged = new Map();

  for (const key of keys) {
    const raw = localStorage.getItem(key);
```

```
    if (!raw) continue;

    const items = safeJsonParse(raw, []);
    if (!Array.isArray(items)) continue;

    for (const row of items) {
      const productId = Number(row?.product_id ?? row?.productId ?? row?.id ??
0);
      const qty = Number(row?.qty ?? row?.quantity ?? row?.count ?? 0);

      if (!Number.isFinite(productId) || productId <= 0) continue;
      if (!Number.isFinite(qty) || qty <= 0) continue;

      merged.set(productId, {
        product_id: productId,
        qty: (merged.get(productId)?.qty || 0) + qty
      });
    }
  }

  const normalized = Array.from(merged.values());
  localStorage.setItem("cart", JSON.stringify(normalized));
  return normalized;
}

async function apiJson(url, options = {}) {
  const headers = { ...(options.headers || {}) };

  if (options.body !== undefined && options.body !== null) {
    headers["Content-Type"] = "application/json";
  }

  const response = await fetch(url, { ...options, headers });
  const data = await response.json().catch(() => null);

  if (!response.ok) {
    const message = data?.error || data?.message || `HTTP ${response.status}`;
    throw new Error(message);
  }

  return data;
}

async function apiJsonAuth(url, options = {}) {
  const headers = { ...(options.headers || {}) };
  const token = getToken();

  if (token) {
    headers.Authorization = `Bearer ${token}`;
  }

  return apiJson(url, { ...options, headers });
}

function clearCart() {
  localStorage.setItem("cart", "[]");
  window.dispatchEvent(new Event("cart:changed"));
}

function setText(element, text) {
  if (element) {
    element.textContent = text;
  }
}
```

```
function setMessage(element, text, isError = false) {
  if (!element) return;

  element.textContent = text;
  element.className = isError ? "alert alert-error" : "muted small mt-2";
}

function isValidEmail(email) {
  const value = String(email || "").trim();
  if (!value) return false;
  return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(value);
}

function normalizeImage(product) {
  let imagePath =
    (product && (product.image || product.image_url || product.img ||
product.photo)) || "";

  imagePath = String(imagePath || "").trim();

  if (!imagePath) {
    return "/assets/img/charger.png";
  }

  imagePath = imagePath.split("?")[0];

  if (!imagePath.startsWith("/")) {
    imagePath = "/" + imagePath;
  }

  imagePath = imagePath.replace(/^\/assets\/images\//, "/assets/img/");

  if (imagePath.startsWith("/assets/img/")) {
    imagePath = imagePath.replace(/\.(jpg|jpeg)$/i, ".png");
  }

  return imagePath;
}

async function initCheckout() {
  const itemsBox = document.getElementById("orderItems");
  const totalEl = document.getElementById("orderTotal");
  const statusEl = document.getElementById("orderStatus");
  const msgEl = document.getElementById("orderMsg");
  const hintEl = document.getElementById("authHint");

  const form = document.getElementById("checkoutForm");
  const inputName = document.getElementById("custName");
  const inputEmail = document.getElementById("custEmail");
  const inputPhone = document.getElementById("custPhone");
  const inputAddress = document.getElementById("custAddress");
  const submitBtn = document.getElementById("submitOrderBtn");

  if (!itemsBox || !totalEl || !statusEl || !msgEl || !form) return;

  const token = getToken();
  const isAuthorized = Boolean(token);

  if (hintEl) {
    hintEl.textContent = isAuthorized
      ? "Ви авторизовані. Перевірте дані покупця перед підтвердженням
замовлення."
```

```
      : "Замовлення можна оформити без авторизації. Необхідно вказати ПІБ та
щонайменше один контактний спосіб зв’язку.";
  }

  const authUser = getAuthUser();
  if (isAuthorized && authUser) {
    if (authUser.name && inputName) inputName.value = authUser.name;
    if (authUser.email && inputEmail) inputEmail.value = authUser.email;
    if (authUser.phone && inputPhone) inputPhone.value = authUser.phone;
    if (authUser.address && inputAddress) inputAddress.value = authUser.address;
  }

  const cart = getCartNormalized();

  if (!cart.length) {
    setText(statusEl, "Кошик порожній. Додайте автотовари перед оформленням
замовлення.");
    itemsBox.innerHTML = "";
    setText(totalEl, formatUAH(0));
    return;
  }

  let products = [];

  try {
    products = await apiJson("/api/products");
  } catch (error) {
    console.error(error);
    setText(statusEl, "Не вдалося завантажити дані каталогу.");
    setMessage(msgEl, error.message || "Сталася помилка під час завантаження
товарів.", true);
    return;
  }

  const productMap = new Map(
    (Array.isArray(products) ? products : []).map((product) =>
[Number(product.id), product])
  );

  let total = 0;

  itemsBox.innerHTML = cart
    .map((row) => {
      const productId = Number(row.product_id);
      const qty = Number(row.qty);
      const product = productMap.get(productId);

      if (!product) return "";

      const price = Number(product.price);
      const sum = (Number.isFinite(price) ? price : 0) * qty;
      total += sum;

      return `
        <div
          class="checkout-row"
          style="
            display:flex;
            gap:18px;
            align-items:center;
            padding:16px 0;
            border-bottom:1px solid rgba(255,255,255,.08);
          "
        >
```

```
          <div
            class="checkout-thumb"
            style="
              width:250px;
              height:250px;
              min-width:250px;
              max-width:250px;
              border-radius:12px;
              overflow:hidden;
              background:#ffffff;
              display:flex;
              align-items:center;
              justify-content:center;
              padding:8px;
            "
          >
            <img
              src="${escapeHtml(normalizeImage(product))}"
              alt="${escapeHtml(product.name || "Автотовар")}"
              onerror="this.onerror=null; this.src='/assets/img/charger.png';"
              style="
                width:100%;
                height:100%;
                object-fit:contain;
                display:block;
              "
            />
          </div>

          <div style="flex:1 1 auto; min-width:0;">
            <div class="checkout-name" style="font-weight:700; margin-
bottom:6px; font-size:16px;">
              ${escapeHtml(product.name || "Без назви")}
            </div>
            <div class="checkout-meta" style="color:rgba(232,238,252,.75); font-
size:14px;">
              ${escapeHtml(product.category || "Автотовари")}
              ${product.brand ? ` • ${escapeHtml(product.brand)}` : ""}
            </div>
          </div>

          <div
            class="checkout-sum"
            style="
              min-width:170px;
              text-align:right;
              font-size:15px;
              line-height:1.6;
            "
          >
            ${qty} × ${formatUAH(price)}<br />
            <b>${formatUAH(sum)}</b>
          </div>
        </div>
      `;
    })
    .join("");

  setText(totalEl, formatUAH(total));
  setText(statusEl, "");
  setMessage(msgEl, "", false);

  form.addEventListener("submit", async (event) => {
    event.preventDefault();
```

```
    const customer = {
      name: String(inputName?.value || "").trim(),
      email: String(inputEmail?.value || "").trim(),
      phone: String(inputPhone?.value || "").trim(),
      address: String(inputAddress?.value || "").trim()
    };

    if (!customer.name) {
      setMessage(msgEl, "Вкажіть ПІБ покупця.", true);
      return;
    }

    if (isAuthorized) {
      if (!customer.email) {
        setMessage(msgEl, "Для авторизованого користувача поле Email є
обов’язковим.", true);
        return;
      }

      if (!isValidEmail(customer.email)) {
        setMessage(msgEl, "Email введено у некоректному форматі.", true);
        return;
      }
    } else {
      const hasEmail = Boolean(customer.email);
      const hasPhone = Boolean(customer.phone);

      if (!hasEmail && !hasPhone) {
        setMessage(msgEl, "Для оформлення замовлення вкажіть Email або номер
телефону.", true);
        return;
      }

      if (hasEmail && !isValidEmail(customer.email)) {
        setMessage(msgEl, "Email введено у некоректному форматі.", true);
        return;
      }
    }

    const payload = {
      customer,
      items: cart.map((item) => ({
        product_id: Number(item.product_id),
        qty: Number(item.qty)
      }))
    };

    try {
      if (submitBtn) submitBtn.disabled = true;

      setMessage(msgEl, "Замовлення обробляється…", false);

      const result = await apiJsonAuth("/api/orders", {
        method: "POST",
        body: JSON.stringify(payload)
      });

      clearCart();
      setText(
        statusEl,
        `Замовлення успішно створено. Номер замовлення: ${result?.id ??
"невідомий"}.`
      );
```

```
      setText(totalEl, formatUAH(0));
      itemsBox.innerHTML = "";
      setMessage(
        msgEl,
        "Замовлення підтверджено. Ви можете перейти до перегляду історії
замовлень.",
        false
      );

      setTimeout(() => {
        window.location.href = "/orders.html";
      }, 1200);
    } catch (error) {
      console.error(error);
      setMessage(
        msgEl,
        error.message || "Не вдалося оформити замовлення. Повторіть спробу
пізніше.",
        true
      );
    } finally {
      if (submitBtn) submitBtn.disabled = false;
    }
  });
}

document.addEventListener("DOMContentLoaded", initCheckout);
```