МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інженерії

програмного забезпечення

_____ Євген ДАВИДЕНКО

«___» _____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ТОРГІВЛІ ТРАНСПОРТНИМИ ЗАСОБАМИ

Спеціальність 121 Інженерія програмного забезпечення

Освітня програма «Інженерія програмного забезпечення»

Здобувач _____ **Вадим БЕЛЬСЬКИЙ**

«__» _____ 20__ р.

Керівник роботи

Канд. техн. наук, доцент

_____ **Євген ДАВИДЕНКО**

«__» _____ 20__ р.

Миколаїв – 2026

Завдання на виконання кваліфікаційної роботи

Чорноморський національний університет імені Петра Могили

Факультет	Комп'ютерних наук
Кафедра	Інженерії програмного забезпечення
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступінь	Бакалавр
Спеціальність	121 Інженерія програмного забезпечення
Освітня програма	Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри інженерії
програмного забезпечення
_____ Євген ДАВИДЕНКО
«___» _____ 2026 р.

ЗАВДАННЯ

на кваліфікаційну бакалаврську роботу здобувача

Бельського Вадима

1. Тема кваліфікаційної роботи «Вебзастосунок торгівлі транспортними засобами» затверджена наказом ректора ЧНУ ім. Петра Могили № 349 від «26» грудня 2025 р.

2. Строк представлення кваліфікаційної роботи «___» _____ 2026 р.

3. Очікуваний результат роботи та початкові дані якщо такі потрібні.

Очікуваним результатом є розроблений та функціонуючий вебзастосунок для перегляду та продажу транспортних засобів.

4. Перелік питань, що підлягають розробці:

– дослідити предметну область продажу транспортних засобів та провести аналіз існуючих вебплатформ;

– провести порівняльний аналіз сучасних вебсайтів для продажу

автомобілів з метою визначення їх функціональних можливостей;

– сформувати специфікацію функціональних та нефункціональних вимог до програмного забезпечення;

– спроєктувати архітектуру вебзастосунку та структуру бази даних для зберігання інформації про транспортні засоби, користувачів та заявки;

– розробити серверну частину вебзастосунку для обробки запитів користувачів та управління даними системи;

– реалізувати клієнтську частину вебзастосунку з інтерфейсом користувача;

– реалізувати систему реєстрації та авторизації користувачів;

– забезпечити можливість управління каталогом автомобілів через адміністративну панель;

– провести тестування програмного забезпечення для перевірки правильності роботи системи;

– підготувати пояснювальну записку та презентаційні матеріали до захисту.

5. Перелік графічних матеріалів : Презентація

6. Консультанти:

Консультант	**Кафедра (організація)**	**Частина роботи**

Дата видачі завдання «____» _____ 2026 р.

КАЛЕНДАРНИЙ ПЛАН
виконання кваліфікаційної роботи

Тема: Вебзастосунок торгівлі транспортними засобами

№	Найменування роботи	Початок	Закінчення	Примітки
1.	Розробка та затвердження завдання на виконання КБР	26.12.2025	18.02.2026	виконано
2.	Огляд літератури за темою роботи	01.01.2026	05.03.2026	виконано
3.	Складання календарного плану КБР	16.02.2026	18.02.2026	виконано
4.	Аналіз предметної області	20.02.2026	10.03.2026	виконано
5.	Розробка проєктних рішень	05.03.2026	20.03.2026	виконано
6.	Моделювання та конструювання ПЗ	15.02.2026	30.02.2026	виконано
7.	Кодування, тестування та апробація ПЗ, аналіз результатів, розробка керівництва користувача	01.03.2026	20.05.2026	виконано
8.	Відгук керівника КБР	22.05.2026	27.05.2026	виконано
9.	Оформлення КБР та презентації	13.05.2026	28.05.2026	виконано
10.	Попередній захист	01.06.2026	01.06.2026	виконано
11.	Рецензування	02.06.2026	08.06.2026	виконано
12.	Завершення оформлення КБР та презентації	08.06.2026	12.06.2026	виконано
13.	Захист кваліфікаційної роботи	15.06.2026	15.06.2026	

Здобувач _____ **Вадим БЕЛЬСЬКИЙ**

«__» _____ 20__ р.

Керівник роботи

канд. техн. наук,

доцент _____ **Євген ДАВИДЕНКО**

«__» _____ 20__ р.

АНОТАЦІЯ

до кваліфікаційної бакалаврської роботи

Вебзастосунок торгівлі транспортними засобами

Здобувач 408 гр.: Бельський Вадим

Керівник: канд. техн. наук, доцент Давиденко Євген

У сучасних умовах розвитку цифрових технологій та електронної комерції значно зростає попит на онлайн–сервіси для купівлі транспортних засобів. Більшість існуючих платформ мають ряд недоліків, зокрема перевантажений інтерфейс, складність пошуку та відсутність зручної взаємодії користувача із системою. Тому актуальною є задача створення вебзастосунку, який забезпечить ефективний пошук, перегляд та придбання транспортних засобів у зручному та інтуїтивно зрозумілому середовищі.

Об'єктом роботи є процеси організації та управління продажем транспортних засобів за допомогою вебтехнологій.

Предметом роботи є програмні засоби розробки вебзастосунку для представлення каталогу транспортних засобів та взаємодії користувачів із системою.

Метою роботи є розробка вебзастосунку для торгівлі транспортними засобами, який забезпечує зручний перегляд каталогу автомобілів, пошук за параметрами та можливість надсилання заявки на придбання транспортного засобу.

Кваліфікаційна робота складається із вступу, 4 розділів, висновків та переліку джерел посилання.

У вступі обґрунтовано актуальність теми, визначено мету, завдання, об’єкт та предмет дослідження.

У першому розділі проведено аналіз предметної області, розглянуто існуючі аналоги вебзастосунків для продажу автомобілів, визначено їх переваги та недоліки, а також сформовано вимоги до розроблюваної системи.

У другому розділі виконано проєктування системи, визначено функціональні можливості, ролі користувачів та сценарії використання вебзастосунку.

У третьому розділі представлено архітектуру системи, побудовано UML–діаграми (діаграма класів, діаграма розгортання), а також описано структуру бази даних.

У четвертому розділі описано процес програмної реалізації вебзастосунку з використанням ASP.NET MVC, REST API, SQL Server, а також представлено результати тестування функціональності системи.

У висновках узагальнено результати виконаної роботи та визначено напрями подальшого розвитку системи.

Кваліфікаційна робота викладена на 82 сторінках машинописного тексту, складається із вступу, 4 розділів, загальних висновків, переліку джерел посилання з 27 найменувань та 0 додатків. Праця містить 6 таблиць та 31 рисунків.

Ключові слова: вебзастосунок, електронна комерція, інтернет–магазин, ASP.NET MVC, REST API, SQL Server, онлайн–покупки, система управління товарами.

ABSTRACT

to the qualifying bachelor's thesis

Vehicle trading web application

Student of 408 group: Belskyi Vadim

Supervisor: PhD in Technical Sciences, Associate Professor Davydenko Yevhen

In the context of the rapid development of digital technologies and e–commerce, the demand for online services for purchasing vehicles is significantly increasing. Most existing platforms have a number of drawbacks, including overloaded interfaces, complex search functionality, and a lack of convenient user interaction with the system. Therefore, the task of developing a web application that provides efficient search, browsing, and purchasing of vehicles in a user–friendly and intuitive environment is relevant.

The object of the study is the processes of organization and management of vehicle sales using web technologies.

The subject of the study is software tools for developing a web application for presenting a catalog of vehicles and user interaction with the system.

The purpose of the study is to develop a web application for trading vehicles that provides convenient browsing of a vehicle catalog, filtering by parameters, and the ability to submit a purchase request.

The qualification work consists of an introduction, four chapters, conclusions, and a list of references.

The introduction substantiates the relevance of the topic and defines the purpose, objectives, object, and subject of the study.

The first chapter analyzes the subject area, reviews existing web applications for vehicle sales, identifies their advantages and disadvantages, and formulates the requirements for the system being developed.

The second chapter is devoted to system design, including the definition of functional capabilities, user roles, and usage scenarios.

The third chapter presents the system architecture, UML diagrams (class diagram, deployment diagram), and the database structure.

The fourth chapter describes the implementation of the web application using ASP.NET MVC, REST API, and SQL Server, and presents the results of system functionality testing.

The conclusions summarize the results of the work and outline directions for further system development.

The qualification work is presented on 82 pages of typewritten text, consists of an introduction, 4 chapters, general conclusions, a list of references containing 27 sources, and 0 appendices. The work includes 6 tables and 31 figures.

Keywords: web application, e–commerce, vehicle sales, car catalog, vehicle search, ASP.NET MVC, REST API, SQL Server.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

БД	–	база даних
ПЗ	–	програмне забезпечення
СКБД	–	система керування базами даних
API	–	Application Programming Interface
ASP.NET	–	Active Server Pages .NET
CORS	–	Cross–Origin Resource Sharing
CRUD	–	Create, Read, Update, Delete
CSS	–	Cascading Style Sheets
EF Core	–	Entity Framework Core
HTML	–	HyperText Markup Language
HTTPS	–	HyperText Transfer Protocol Secure
JSON	–	JavaScript Object Notation
.NET	–	платформа розробки програмного забезпечення Microsoft
REST	–	Representational State Transfer
SPA	–	Single Page Application
SQL	–	Structured Query Language
SSMS	–	SQL Server Management Studio
UI	–	User Interface
URL	–	Uniform Resource Locator
UX	–	User Experience
XSS	–	Cross–Site Scripting

ВСТУП

Ринок торгівлі транспортними засобами за останнє десятиліття впевнено перемістився в онлайн-простір. Покупці більше не прив'язані до графіку роботи автосалону: вони самостійно порівнюють моделі, перевіряють технічні характеристики й залишають заявки тоді, коли їм зручно. Цей зсув породжує нові вимоги до продавців – автоматизоване управління каталогом, обробка заявок і комунікація з клієнтами без ручного втручання на кожному кроці стають не перевагою, а необхідністю.

Аналіз п'яти діючих платформ – AUTO.RIA, AutoTrader, Cars.com, OLX Авто та RST – виявив спільні системні вади: перевантаженість інтерфейсу рекламними блоками, заплутана навігація через надмірну кількість підкатегорій, незручна фільтрація за технічними параметрами, недостатній адміністративний контроль над каталогом оголошень. Жодна з розглянутих платформ не поєднує зручність для кінцевого користувача з гнучкістю для адміністратора – саме цей розрив і визначає потребу в окремому рішенні.

Актуальність теми. Попит на зручні онлайн-майданчики для продажу автомобілів зростає разом із проникненням інтернету та зміною купівельних звичок. Сучасний покупець не телефонує менеджеру, щоб дізнатись ціну, – він очікує знайти відповідь за кілька кліків. Розроблений вебзастосунок відповідає саме цьому запиту: структурований каталог із пошуком за параметрами й вбудована форма заявки скорочують шлях клієнта від пошуку до звернення. Для продавця це означає менше рутинної роботи і чіткіший контроль над процесом продажу.

Практичне значення роботи полягає у можливості застосування розробленого вебзастосунку в реальних умовах автосалонів або компаній, що займаються продажем автомобілів, а також використання його як основи для подальшого розширення функціональності.

Об'єктом кваліфікаційної роботи є процеси організації та управління продажем транспортних засобів за допомогою вебтехнологій.

Предметом кваліфікаційної роботи є програмні засоби розробки вебзастосунку для представлення каталогу транспортних засобів та взаємодії користувачів із системою.

Метою кваліфікаційної роботи є розробка вебзастосунку для торгівлі транспортними засобами, який забезпечує зручний перегляд каталогу автомобілів, пошук за параметрами та можливість надсилання заявки на придбання транспортного засобу.

Для досягнення поставленої мети у кваліфікаційній роботі необхідно вирішити такі завдання:

1) провести аналіз предметної області електронної комерції у сфері продажу транспортних засобів;

2) дослідити існуючі вебплатформи для продажу автомобілів та визначити їхні переваги й недоліки;

3) сформувати функціональні та нефункціональні вимоги до системи;

4) спроєктувати архітектуру вебзастосунку та структуру бази даних для зберігання інформації про транспортні засоби, користувачів та заявки;

5) розробити серверну частину вебзастосунку для обробки запитів користувачів та управління даними системи;

6) реалізувати клієнтську частину вебзастосунку з інтуїтивно зрозумілим інтерфейсом користувача;

7) реалізувати систему реєстрації та авторизації користувачів;

8) забезпечити можливість управління каталогом автомобілів через адміністративну панель;

9) провести тестування програмного забезпечення для перевірки правильності роботи системи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЕЛЕКТРОННОЇ КОМЕРЦІЇ У СФЕРІ ТОРГІВЛІ ТРАНСПОРТНИМИ ЗАСОБАМИ

1.1 Об'єкт і предмет кваліфікаційної роботи

Об'єктом кваліфікаційної роботи є процеси організації та управління продажем транспортних засобів за допомогою вебтехнологій. Під цими процесами розуміється сукупність взаємопов'язаних операцій, що охоплюють формування каталогу транспортних засобів, пошук і перегляд пропозицій потенційними покупцями, подання та обробку заявок на придбання, а також адміністративне управління системою.

Аналіз об'єкта передбачає розгляд структури автомобільного ринку в онлайн–середовищі, ролей учасників цих процесів (покупець, адміністратор автосалону, менеджер), а також бізнес–логіки взаємодії між ними. Розуміння об'єкта є необхідною передумовою для формулювання коректних вимог до програмної системи та побудови її архітектури.

Предметом кваліфікаційної роботи є програмні засоби розробки вебзастосунку для представлення каталогу транспортних засобів та взаємодії користувачів із системою. Предмет охоплює методи та інструменти проєктування та реалізації серверної частини (бізнес–логіка, REST API, база даних), клієнтської частини (інтерфейс користувача, адаптивний дизайн), а також механізми автентифікації, авторизації та управління даними.

1.2 Тенденції розвитку онлайн–торгівлі транспортними засобами

Автомобільний ринок є одним із найбільших секторів світової економіки. За даними Міжнародної організації виробників автомобілів (OICA), щороку у світі реалізується понад 85–93 мільйони транспортних засобів. Купівля автомобіля – це одне з найважливіших фінансових рішень для більшості людей, яке потребує ретельного вивчення пропозицій, порівняння характеристик та пошуку оптимального співвідношення ціни та якості. Саме тому процес вибору та

придбання транспортного засобу дедалі активніше переміщується в онлайн–середовище.

За даними дослідницької компанії McKinsey & Company, понад 80% покупців автомобілів починають процес вибору з пошуку в Інтернеті. Вони вивчають технічні характеристики, порівнюють ціни, читають відгуки та лише після цього звертаються до автосалону або продавця. Це докорінно змінило модель взаємодії між покупцем і продавцем: тепер покупець приходить до дилера вже підготовленим, маючи чітке уявлення про бажаний автомобіль. Відповідно, якість онлайн–представлення транспортних засобів безпосередньо впливає на обсяги продажів.

В Україні онлайн–торгівля транспортними засобами також демонструє стале зростання. Такі платформи як AUTO.RIA та OLX Авто щомісяця обробляють мільйони запитів від потенційних покупців, а кількість активних оголошень сягає сотень тисяч. Незважаючи на складну економічну ситуацію, попит на придбання автомобілів через онлайн–канали не знижується – повномасштабне вторгнення Росії лише прискорило перехід ринку до цифрового формату, оскільки можливість дистанційного огляду та бронювання автомобіля стала особливо важливою.

Проте більшість існуючих платформ орієнтовані на великі маркетплейси з тисячами оголошень від різних продавців. Для автосалонів та дилерських компаній, які потребують власного керованого онлайн–каталогу з можливістю прийому заявок від покупців, такі рішення є або надмірно складними, або не відповідають специфічним бізнес–потребам. Автосалон, що розміщує свої автомобілі на загальному маркетплейсі, не має повного контролю над поданням інформації, фірмовим стилем та процесом взаємодії з клієнтом.

Таким чином, існує чітка потреба у спеціалізованому вебзастосунку, який надасть автосалону або дилерській компанії власний цифровий майданчик для представлення каталогу транспортних засобів. Такий застосунок повинен вирішувати три ключові завдання: надавати покупцю зручний та інтуїтивно зрозумілий інтерфейс для пошуку і вивчення автомобілів; забезпечувати простий механізм надсилання заявки на придбання або консультацію; надавати

адміністратору повний контроль над каталогом – додавання, редагування та видалення транспортних засобів, а також обробка вхідних заявок від покупців.

1.3 Структурні та функціональні особливості об'єкта дослідження

Онлайн-торгівля транспортними засобами охоплює кілька категорій учасників, кожна з яких по-різному взаємодіє з системою й керується власними очікуваннями щодо її функціональності. Визначення цих ролей і меж відповідальності кожної з них – необхідна умова для проєктування архітектури, яка задовольнятиме потреби всіх сторін одночасно.

Покупець – фізична особа, яка шукає транспортний засіб для придбання. Його взаємодія з платформою, як правило, починається з пошуку: він задає параметри (марка, клас, бюджет), переглядає перелік результатів і обирає конкретну пропозицію для детальнішого вивчення. На цьому етапі покупець очікує побачити вичерпну технічну інформацію, якісні фотографії з різних ракурсів та зручний механізм зворотного зв'язку – так, щоб контакт із продавцем вимагав мінімум зусиль. Доступ до системи покупець може отримати як авторизованим користувачем із особистим кабінетом, так і без реєстрації у режимі гостя.

Адміністратор – представник автосалону або компанії, відповідальний за наповнення й актуальність каталогу. На відміну від покупця, він взаємодіє передусім із внутрішньою частиною системи: додає й редагує оголошення, завантажує фотографії для кожного транспортного засобу, переглядає вхідні заявки від покупців і веде їх поетапне опрацювання. Управління обліковими записами інших користувачів також входить до кола його повноважень. Зручність і швидкість адміністративного інтерфейсу безпосередньо визначають, наскільки оперативно автосалон реагує на зміни ринку й підтримує каталог у актуальному стані.

Менеджер з продажу виконує обмежені адміністративні функції: переглядає вхідні заявки від покупців і змінює їхній статус опрацювання, однак не має прав на редагування або видалення записів у каталозі. Ця роль передбачена для компаній із розподіленим відділом продажу, де менеджер веде комунікацію з клієнтами, а відповідальність за зміст каталогу залишається виключно у веденні адміністратора.

Такий поділ повноважень мінімізує ризик випадкових змін у каталозі з боку персоналу, що не має відповідної кваліфікації.

Взаємодія між учасниками формує основний функціональний потік системи. Покупець подає заявку – система фіксує її в базі даних зі статусом «нова» – адміністратор або менеджер опрацьовує заявку і змінює її статус – покупець отримує зворотний зв'язок. Саме цей ланцюжок визначив архітектурні вимоги до застосунку: форма заявки та особистий кабінет на стороні клієнта, механізм керування статусами й панель управління заявками на стороні адміністратора.

Структура предметної галузі

Предметна галузь вебзастосунку торгівлі транспортними засобами описується через чотири основні інформаційні сутності, між якими встановлено зв'язки на рівні бази даних: транспортний засіб, заявка, користувач і каталог. Кожна з них виконує чітко визначену роль у логіці системи і безпосередньо відповідає одному з функціональних процесів, описаних у наступному підрозділі.

Транспортний засіб – центральна сутність системи, навколо якої будується вся функціональна логіка застосунку. Кожен автомобіль у базі даних описується набором атрибутів: марка, модель, рік випуску, тип кузова, тип пального, об'єм та потужність двигуна, тип коробки передач, пробіг, колір, ціна, опис і статус (доступний, проданий, знятий з продажу). Фотографії виділено в окрему пов'язану сутність: кожне зображення зберігає посилання на файл і прив'язку до конкретного транспортного засобу, що дозволяє підтримувати довільну кількість фото без зміни структури основної таблиці.

Заявка фіксує намір покупця придбати або отримати консультацію щодо конкретного транспортного засобу. Вона містить контактні дані покупця (ім'я, номер телефону, електронна пошта), прив'язку до конкретного транспортного засобу та довільний коментар. Статус заявки відображає етап її опрацювання і змінюється послідовно: «нова» → «в обробці» → «завершена» або «скасована». Цей ланцюжок відстежується як покупцем в особистому кабінеті, так і адміністратором або менеджером у панелі управління.

Користувач – обліковий запис у системі, що ідентифікує особу та визначає її рівень доступу. Система підтримує дві ролі: звичайний користувач (покупець) та адміністратор. У базі даних для кожного запису зберігаються: ім'я, електронна пошта, хешований пароль, призначена роль і дата реєстрації. Роль безпосередньо впливає на доступні операції: покупець переглядає каталог і подає заявки, тоді як адміністратор керує всіма ресурсами системи.

Каталог – представлення транспортних засобів зі статусом «доступний», відкрите для перегляду без авторизації. З технічного боку каталог не є окремою таблицею бази даних: він формується динамічно шляхом запиту до сутності транспортного засобу із застосуванням заданих фільтрів та умови відбору за статусом. Система підтримує фільтрацію за ключовими параметрами й пагінацію результатів – це забезпечує коректну роботу інтерфейсу навіть при значному обсязі записів у базі.

Функціональні процеси системи

Функціональну поведінку вебзастосунку описують чотири основних процеси. Кожен із них охоплює взаємодію між клієнтською частиною, серверною логікою та базою даних, що дозволяє розглядати їх як самостійні сценарії використання системи.

Процес перегляду каталогу. Користувач відкриває каталог і задає параметри фільтрації: марку, модель, рік випуску, тип пального та ціновий діапазон. Сервер повертає відфільтрований і посторінково розбитий список транспортних засобів. Обравши конкретний автомобіль, користувач переходить на його детальну картку – там відображаються повний опис, технічні характеристики та фотогалерея з усіма завантаженими зображеннями.

Процес подання заявки. На сторінці детального перегляду транспортного засобу розміщено форму заявки. Покупець заповнює контактні дані та коментар і підтверджує відправлення – після цього система зберігає заявку зі статусом «нова» й повертає користувачеві підтвердження успішного відправлення. Авторизований

користувач може в будь-який момент переглянути власні заявки та їхній поточний статус опрацювання в особистому кабінеті.

Процес управління каталогом (адміністратор). Адміністратор входить в адміністративну панель і отримує доступ до повного переліку транспортних засобів незалежно від їхнього статусу. Він може додавати нові записи із заповненням усіх атрибутів і завантаженням фотографій, редагувати наявні або видаляти їх. Усі операції виконуються через графічний інтерфейс і не потребують технічних знань чи безпосереднього доступу до бази даних.

Процес реєстрації та автентифікації. Новий користувач заповнює форму реєстрації: ім'я, електронну пошту і пароль. Система перевіряє унікальність електронної пошти, хешує пароль і створює обліковий запис. При вході система звіряє надані облікові дані, формує об'єкт ClaimsPrincipal із іменем користувача та його роллю, після чого встановлює автентифіковану cookie-сесію. Ця сесія долучається до кожного наступного запиту і дає серверу змогу перевірити особу та права доступу без повторної передачі облікових даних.

Тенденції розвитку онлайн–торгівлі транспортними засобами

Цифровізація автомобільного ринку відбувається нерівномірно, але кілька напрямів вже сформували стабільні вимоги до сучасних вебплатформ – і проігнорувати ці вимоги означає свідомо поступитися позицією конкурентам, які їх урахували.

Перехід до мобільного пошуку. За даними Google, понад 60% пошукових запитів щодо купівлі автомобіля надходять із мобільних пристроїв. Це принципово змінює пріоритети розробки: адаптивний дизайн перестав бути додатковою опцією і став базовою вимогою. Зручність на смартфоні має проєктуватися як первинний сценарій, а не як адаптація десктопного інтерфейсу під менший екран.

Зростання ролі фотоконтенту. Сучасний покупець очікує побачити детальні фотографії автомобіля з різних ракурсів ще до першого контакту з продавцем. За даними AutoTrader, оголошення з 20 і більше фотографіями отримують у 4 рази більше переглядів порівняно з оголошеннями без фото. Звідси пряма вимога до

системи: картка транспортного засобу повинна підтримувати завантаження кількох зображень і відображати їх у форматі зручного перегляду – галереї з можливістю переходу між знімками.

Детальна фільтрація як ключовий інструмент. Покупці транспортних засобів орієнтуються на конкретні технічні критерії: марка, модель, рік випуску, тип пального, об'єм двигуна, тип коробки передач, пробіг і ціновий діапазон. Без можливості оперативно звузити результати за цими параметрами користувач змушений вручну переглядати нерелевантні оголошення – і, як правило, залишає платформу на користь конкурента, де фільтрація реалізована зручніше. Якісна система фільтрації скорочує час від першого пошуку до цільового результату.

Онлайн-заявка як замінник першого дзвінка. Значна частина покупців, особливо молодша аудиторія, уникає телефонних розмов і надає перевагу текстовому зверненню у зручний для себе час. Форма заявки, вбудована безпосередньо на сторінці автомобіля, скорочує бар'єр між інтересом і контактом: користувач не шукає номер телефону, не чекає відповіді оператора, а одразу фіксує свій запит. Чим менше кроків між «хочу дізнатись більше» і «заявку відправлено» – тим вища ймовірність того, що покупець не передумає на півдорозі.

Автоматизація управління каталогом. Для дилерів та автосалонів швидке оновлення інформації про наявні автомобілі – питання операційної ефективності, а не лише зручності. Якщо додати чи змінити оголошення можна лише залучивши розробника, каталог неминуче відстає від реального стану запасів. Адміністративна панель із зрозумілим інтерфейсом дозволяє підтримувати актуальність каталогу в режимі реального часу власними силами – без додаткових витрат на технічний супровід рутинних операцій.

Персоналізація та рекомендаційні системи. Провідні платформи ринку впроваджують механізми збереження пошукових запитів і показу схожих пропозицій на основі переглянутих автомобілів. Покупець, який не знайшов підходящий варіант під час першого відвідування, повертається на платформу, коли система пропонує нові оголошення відповідно до його критеріїв. Такий підхід

збільшує частку угод, ініційованих не з першого візиту, і формує звичку повертатися на конкретний майданчик, а не шукати альтернативи заново.

1.4 Аналіз існуючих програмних рішень

Для визначення функціональних вимог та виявлення недоліків існуючих рішень було проаналізовано п'ять провідних платформ для купівлі–продажу транспортних засобів, що працюють на українському та міжнародному ринках.

AUTO.RIA (auto.ria.com)[2]

AUTO.RIA є найбільшою українською платформою для купівлі та продажу автомобілів з мільйонною аудиторією користувачів щомісяця.

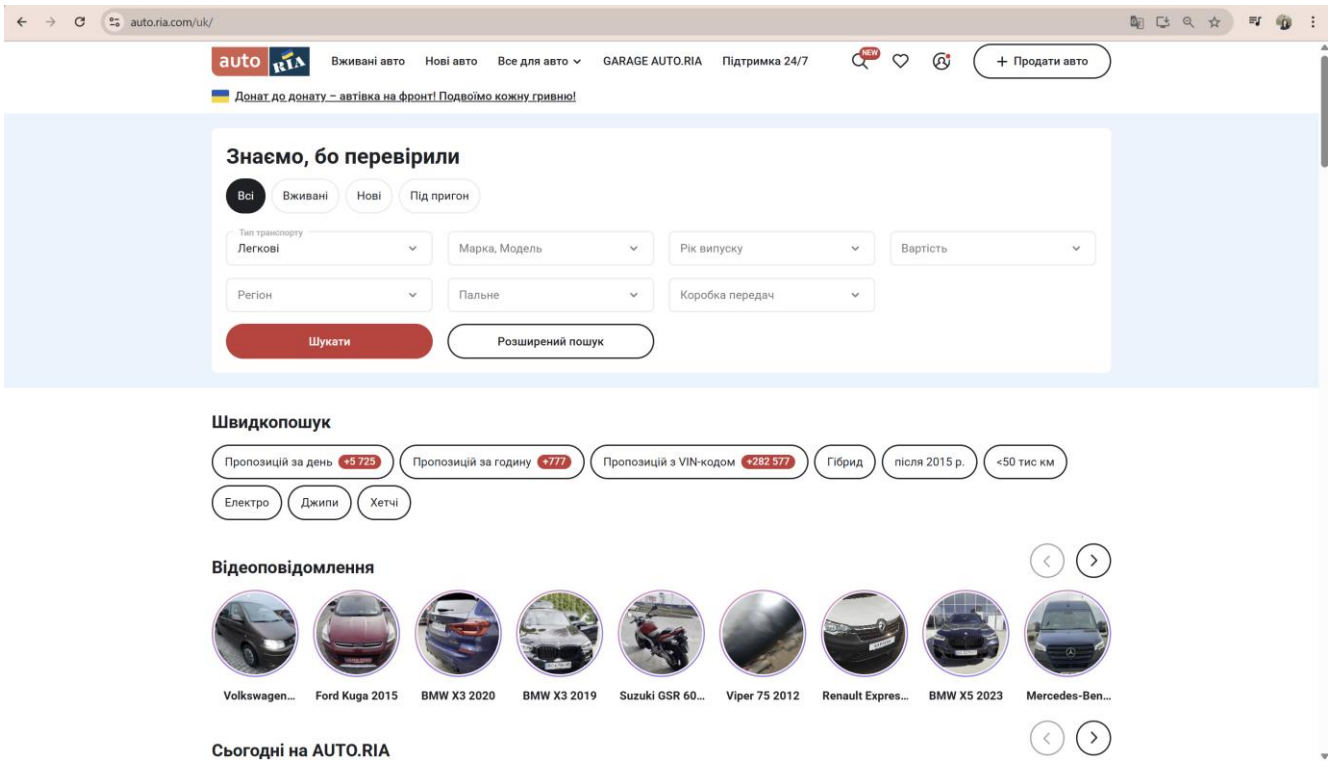

Рисунок 1.1 – Інтерфейс вебзастосунку AUTO.RIA

Основні переваги:

- розвинена система фільтрації за маркою, моделлю, роком, пробігом, ціною та регіоном;
- перевірка автомобіля за VIN–кодом та історією обслуговування;
- інтеграція з TechVIN для перевірки юридичної чистоти;

- наявність мобільного застосунку для iOS та Android;
- система рейтингів продавців та відгуків покупців.

Виявлені недоліки:

- перевантаженість інтерфейсу рекламними банерами та спливаючими вікнами;
- складна навігація через надмірну кількість розділів і підкатегорій;
- повільне завантаження сторінок через велику кількість рекламного контенту;
- платний доступ до частини функцій для продавців;
- недостатня адаптація окремих розділів для мобільних пристроїв.

OLX Авто (olx.ua/uk/transport)[22]

OLX Авто є розділом найбільшого українського маркетплейсу оголошень OLX, де представлені тисячі пропозицій продажу транспортних засобів від приватних осіб та дилерів.

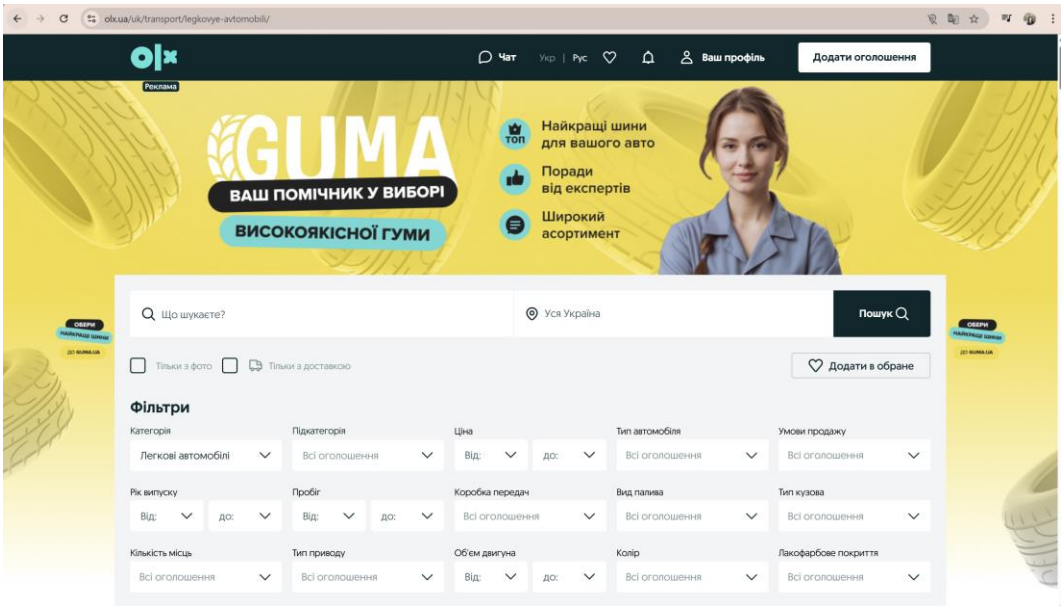

Рисунок 1.2 – Інтерфейс вебзастосунку OLX Авто

Основні переваги:

- широке охоплення аудиторії завдяки загальній популярності платформи OLX;

- простота розміщення оголошення для приватного продавця;
- зручна система повідомлень між покупцем і продавцем;
- безкоштовне базове розміщення оголошень;
- наявність розділу перевірених дилерів.

Виявлені недоліки:

- відсутність спеціалізованих фільтрів для технічних параметрів автомобіля;
- низький рівень верифікації оголошень, що спричиняє появу шахрайських пропозицій;
- змішаний контент із іншими категоріями товарів знижує спеціалізацію платформи;
- обмежені можливості для дилерів щодо управління каталогом;
- відсутність інструментів порівняння автомобілів між собою.

RST (rst.ua)[27]

RST є спеціалізованим українським автомобільним порталом для купівлі та продажу нових і вживаних транспортних засобів.

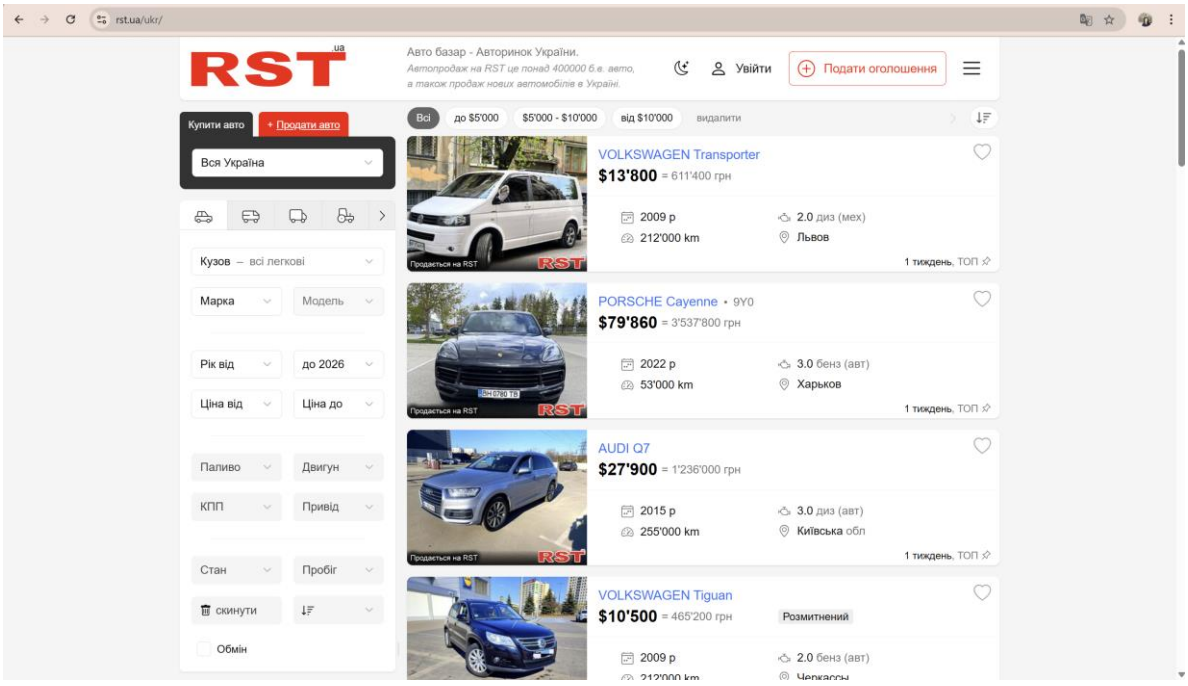

Рисунок 1.3 – Інтерфейс вебзастосунку RST

Основні переваги:

- спеціалізація виключно на ринку транспортних засобів;
- детальні картки оголошень з технічними характеристиками;
- зручна система пошуку з великою кількістю параметрів;
- наявність розділу автосалонів із каталогами дилерів;
- статистика переглядів оголошень для продавців.

Виявлені недоліки:

- застарілий дизайн інтерфейсу, що не відповідає сучасним стандартам UX;
- повільна робота фільтрів при великій кількості результатів пошуку;
- недостатня оптимізація для мобільних пристроїв;
- складний процес реєстрації та верифікації для нових продавців;
- відсутність зручного механізму надсилання заявки безпосередньо через платформу.

AutoTrader (autotrader.com)[3]

AutoTrader є одним із найбільших американських маркетплейсів для купівлі та продажу нових і вживаних автомобілів з багаторічною історією та розвиненою інфраструктурою.

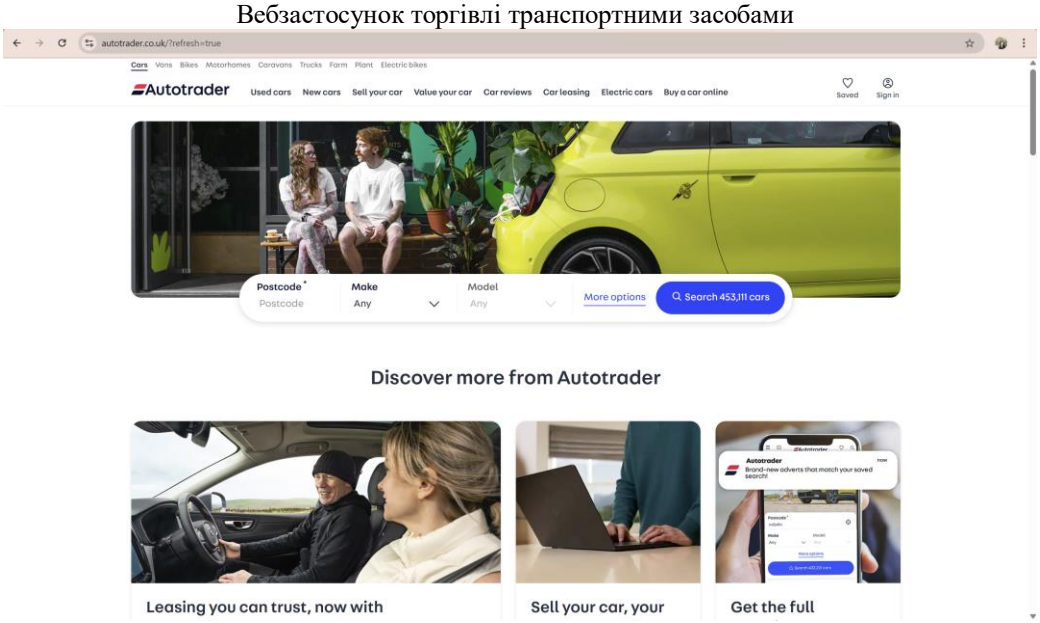

Рисунок 1.4 – Інтерфейс вебзастосунку AutoTrader

Основні переваги:

- потужна система пошуку та фільтрації з десятками параметрів;
- інтеграція з фінансовими сервісами для розрахунку кредиту та лізингу;
- детальні звіти про історію автомобіля через Carfax;
- зручне порівняння кількох автомобілів в одному інтерфейсі;
- висока швидкість завантаження сторінок та зручна мобільна версія.

Виявлені недоліки:

- орієнтованість виключно на американський ринок, що обмежує його застосування в Україні;
- перевантаженість сторінки результатів пошуку рекламними вставками;
- складний інтерфейс для непідготовленого користувача через велику кількість опцій;
- платні преміум–функції для дилерів;
- відсутність україномовного інтерфейсу та локальних методів оплати.

Cars.com (cars.com)[6]

Cars.com є провідним американським агрегатором автомобільних оголошень, що об'єднує пропозиції від дилерів та приватних продавців по всій території США.

Основні переваги:

- сучасний та інтуїтивно зрозумілий інтерфейс користувача;
- розвинена система відгуків та оцінок дилерів;
- зручний інструмент порівняння автомобілів за характеристиками;
- детальна інформація про кожен автомобіль із фотогалереями;
- адаптивний дизайн із повноцінною підтримкою мобільних пристроїв.

Виявлені недоліки:

- орієнтованість виключно на ринок США без підтримки інших регіонів;
- складна система тарифів для дилерів;
- інтерфейс частково перевантажений рекламними блоками;
- обмежені можливості кастомізації сторінки дилера;
- відсутність вбудованого механізму онлайн–оформлення угоди.

Порівняльний аналіз

На основі проведеного аналізу було складено порівняльну таблицю основних характеристик розглянутих платформ (табл. 1.1).

Таблиця 1.1 – Порівняльний аналіз існуючих рішень

Назва	**AUTO.RIA**	**OLX Авто**	**RST**	**AutoTrader**	**Cars.com**
Розробник	AUTO.RIA Group (Україна)	OLX Group (міжнар.)	RST (Україна)	Cox Automotive (США)	Cars.com Inc. (США)
Архітектура	Багаторівнева веб–архітектура, мобільні застосунки	Маркетплейс, мікросервіси, мобільні застосунки	3–tier web application, монолітна структура	Багаторівнева веб–архітектура, API, мобільні клієнти	3–tier web application, REST API, мобільна версія
Спеціалізація	Лише авторинок	Загальний маркетплейс	Лише авторинок	Лише авторинок	Лише авторинок
Фільтрація	Розвинена	Обмежена	Середня	Дуже розвинена	Розвинена

Переваги	VIN–перевірка, велика аудиторія	Широке охоплення, простота	Спеціалізація, детальні картки	Потужний пошук, фінансові сервіси	Сучасний UX, відгуки дилерів
Недоліки	Реклама, перевантажений інтерфейс	Слабка верифікація, немає порівняння	Застарілий дизайн, повільні фільтри	Лише США, складний інтерфейс	Лише США, реклама
Мобільна версія	Є додаток	Є додаток	Часткова	Повна адаптація	Повна адаптація

На основі проведеного аналізу можна виділити такі типові недоліки існуючих рішень:

1. Перевантаженість інтерфейсу. Більшість розглянутих платформ містять надмірну кількість рекламних банерів і спливаючих вікон, що негативно впливає на користувацький досвід.

2. Складна навігація. Багаторівнева структура меню та надмірна кількість розділів ускладнюють пошук потрібного автомобіля, особливо для нових користувачів.

3. Недостатня мобільна оптимізація. Ряд платформ не забезпечують повноцінну адаптацію інтерфейсу для мобільних пристроїв, що суттєво погіршує досвід користувачів на смартфонах.

4. Відсутність зручного механізму заявок. Більшість українських платформ не мають інтегрованого інструменту для надсилання заявки на придбання безпосередньо через інтерфейс картки автомобіля.

5. Обмежений функціонал для дилерів. Існуючі платформи або пропонують лише платні інструменти управління каталогом, або мають недостатній адміністративний функціонал для повноцінної роботи автосалону.

6. Відсутність спеціалізованого рішення для автосалонів. Жодна з розглянутих платформ не є повноцінним інструментом для автосалону з власним брендом, власним каталогом та власним процесом обробки заявок.

1.5 Обґрунтування вибору технологічного стеку

Серверну частину вебзастосунку реалізовано на платформі ASP.NET MVC (.NET 9) компанії Microsoft. Архітектурний патерн MVC (Model–View–Controller) розмежовує три зони відповідальності: модель відповідає за бізнес-логіку та роботу з даними, представлення – за формування відповіді користувачу, контролер – за обробку вхідних запитів. Таке розмежування не є суто структурним рішенням: воно безпосередньо скорочує час супроводу коду, оскільки зміна логіки відображення не зачіпає бізнес-правил, і навпаки. Тестування окремих компонентів у такій архітектурі також відбувається ізольовано, без необхідності підіймати всю систему.

Платформа .NET 9 підтримує кросплатформне розгортання і демонструє стабільно високу продуктивність у серверних сценаріях. За результатами TechEmpower Framework Benchmarks, ASP.NET Core входить до числа найшвидших серверних фреймворків серед усіх популярних технологічних стеків. Екосистема .NET надає зрілі бібліотеки для вирішення типових завдань веброзробки: Entity Framework Core – для взаємодії з базою даних через об'єктно-реляційне відображення, ASP.NET Identity – для управління автентифікацією та авторизацією, вбудований клієнт HttpClient – для виконання HTTP-запитів до зовнішніх REST API.

Під час вибору серверної технології розглядалися дві альтернативи – Node.js (Express) та Java (Spring Boot). Node.js забезпечує високу пропускну здатність для I/O-навантажених систем, однак динамічна типізація JavaScript ускладнює підтримку великих кодових баз і збільшує ймовірність помилок, які виявляються лише у runtime. Spring Boot – зрілий корпоративний стек із широкою екосистемою, проте він вимагає значно складнішої конфігурації та більших ресурсів для розгортання, що надмірно для проєкту такого масштабу. ASP.NET MVC на .NET 9 обрано як збалансоване рішення: строга типізація C# знижує кількість помилок на етапі компіляції, налаштування мінімальне, а продуктивності достатньо для очікуваного навантаження.

Архітектурний підхід: REST API

Взаємодію між клієнтською та серверною частинами організовано за архітектурним стилем REST (Representational State Transfer). REST API базується на стандартних HTTP-методах (GET, POST, PUT, DELETE) і передає дані у форматі JSON. Ця модель розділяє клієнт і сервер на незалежні компоненти: клієнтська частина не знає про внутрішню реалізацію сервера, а сервер не залежить від способу відображення даних на стороні клієнта. Така незалежність спрощує масштабування окремих частин системи та полегшує їх ізольоване тестування. Додатково архітектура REST відкриває можливість підключення мобільного клієнта в майбутньому – без будь-яких змін у серверній логіці.

СКБД: Microsoft SQL Server

Для зберігання даних обрано Microsoft SQL Server – реляційну СКБД корпоративного рівня. Структура даних системи добре відповідає реляційній моделі: транспортні засоби, користувачі, заявки та фотографії утворюють чіткі зв'язки між сутностями, які природно описуються таблицями і зовнішніми ключами. SQL Server підтримує транзакційну цілісність за принципами ACID – це гарантує коректну обробку заявок навіть за умов одночасного доступу кількох користувачів. Щільна інтеграція з екосистемою .NET реалізується через офіційний провайдер Entity Framework Core для SQL Server: він підтримує підхід Code First, де структура бази даних генерується автоматично з C#-класів, що прискорює розробку та знижує ризик розбіжностей між моделлю і схемою.

Клієнтська технологія та безпека

Клієнтську частину реалізовано засобами HTML5, CSS3 та JavaScript у вигляді статичних сторінок, розміщених у директорії wwwroot. Для адаптивного дизайну застосовано фреймворк Bootstrap: він надає готову систему сітки та набір UI-компонентів, що забезпечують коректне відображення інтерфейсу на пристроях із різними розмірами екрану – від смартфона до настільного монітора. Взаємодія з сервером відбувається через REST API за допомогою AJAX/fetch-запитів: сторінки отримують JSON-дані від API і оновлюють DOM точково, без повного

перезавантаження, що підвищує швидкість відгуку інтерфейсу. Для автентифікації користувачів застосовано Cookie Authentication із Claims-based Authorization, а паролі зберігаються у захешованому вигляді через PasswordHasher<Customer> з ASP.NET Core Identity.

Висновки до розділу 1

Перший розділ охоплює аналіз предметної області торгівлі транспортними засобами в умовах цифровізації ринку, структурних та функціональних особливостей об'єкта дослідження, а також обґрунтування вибору технологічного стеку для реалізації системи.

Об'єктом роботи визначено процеси організації та управління продажем транспортних засобів за допомогою вебтехнологій; предметом – програмні засоби розробки відповідного вебзастосунку. Описано три ролі учасників системи – покупець, адміністратор і менеджер, – основні інформаційні сутності (транспортний засіб, заявка, користувач, каталог) та функціональні процеси їх взаємодії.

Дані McKinsey & Company підтверджують, що понад 80% покупців починають пошук автомобіля з інтернет-ресурсів. Аналіз ринку дозволив виокремити п'ять ключових тенденцій: перехід до мобільного пошуку, зростання ролі фотоконтенту, важливість детальної фільтрації, впровадження персоналізації та онлайн-механізмів подання заявок.

Порівняльний аналіз п'яти провідних платформ – AUTO.RIA, OLX Авто, RST, AutoTrader та Cars.com – виявив спільні системні вади: перевантаженість інтерфейсу рекламними блоками, складна навігація, недостатня мобільна оптимізація, відсутність зручного механізму онлайн-заявок та обмежений функціонал для дилерів. Жодна з платформ не поєднує зручність для кінцевого покупця з повноцінним керуванням каталогом на стороні адміністратора.

На підставі проведеного аналізу сформовано технологічний стек проєкту: ASP.NET Core (.NET 9) для серверної частини, REST API для взаємодії між клієнтом і сервером, Microsoft SQL Server з Entity Framework Core для зберігання й

обробки даних, HTML5, CSS3, Bootstrap та JavaScript – для реалізації адаптивного інтерфейсу. Автентифікацію побудовано на Cookie Authentication із Claims-based Authorization, захист паролів забезпечено через PasswordHasher<Customer> з ASP.NET Core Identity.

2 АНАЛІЗ ІНСТРУМЕНТАРІЮ ТА СПЕЦИФІКАЦІЯ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Аналіз сучасного стану засобів розробки вебзастосунків

Розробка сучасних вебзастосунків електронної комерції спирається на усталені архітектурні підходи, зрілі фреймворки та інструменти, перевірені практикою в реальних промислових проєктах. Перед формуванням технологічного стеку для вебзастосунку торгівлі транспортними засобами варто розібратися у поточному стані доступних рішень і критеріях, за якими одні інструменти обираються замість інших, – кожен технологічний вибір несе архітектурні наслідки, які складно переглядати після початку активної реалізації.

Архітектурні підходи до розробки вебзастосунків

Веброзробка виробила кілька усталених підходів до організації серверної частини. Монолітна архітектура об'єднує всі компоненти системи в єдиному застосунку – це спрощує початкове розгортання, однак із ростом кодової бази підтримка монолітів ускладнюється, а можливості незалежного масштабування окремих модулів залишаються обмеженими. Мікросервісна архітектура розбиває систему на незалежні сервіси, кожен із яких відповідає за окрему функціональну область: такий підхід забезпечує гнучкість і горизонтальне масштабування, але потребує значно більших витрат на інфраструктуру та міжсервісну комунікацію. Для вебзастосунків середнього масштабу, до яких належить розроблюваний застосунок торгівлі транспортними засобами, виправданим рішенням виступає патерн MVC у рамках монолітної архітектури з чітким розділенням рівнів – представлення, бізнес-логіки та доступу до даних.

Сучасні підходи до серверної розробки

Серед серверних платформ корпоративного рівня найбільшого практичного поширення набули ASP.NET Core (.NET), Spring Boot (Java), Django та Flask (Python), Express.js (Node.js). Кожна платформа орієнтована на певний тип задач і технічний контекст команди – порівняння між ними не дає однозначної відповіді без урахування конкретних вимог проєкту. ASP.NET Core вирізняється серед

конкурентів строгою типізацією мови C#, розвиненою екосистемою бібліотек і глибокою інтеграцією з реляційними базами даних через Entity Framework Core. За результатами незалежних бенчмарків TechEmpower Framework Benchmarks, ASP.NET Core стабільно потрапляє до числа найшвидших серверних фреймворків за показниками пропускної здатності при обробці HTTP-запитів і роботі з базами даних – і це відтворюваний публічний результат, а не маркетингова заява.

Ключову роль у сучасній серверній розробці відіграє організація взаємодії між клієнтською та серверною частинами. Архітектурний стиль REST (Representational State Transfer) давно став галузевим стандартом для побудови вебAPI. REST спирається на стандартні HTTP-методи (GET, POST, PUT, DELETE), безстановість запитів і передачу даних у форматі JSON. Ця модель розділяє клієнт і сервер на незалежні компоненти: клієнтська частина не знає про внутрішню реалізацію сервера, а сервер не залежить від способу відображення даних. Така незалежність спрощує інтеграцію з іншими системами – новий клієнт підключається до існуючого API без змін серверної логіки.

Об'єктно–реляційне відображення (ORM)

Застосування ORM-інструментів давно стало усталеною практикою у веброзробці: вони абстрагують розробника від деталей SQL-запитів і дозволяють працювати з базою даних через об'єктну модель мови програмування. Entity Framework Core – офіційне ORM-рішення для платформи .NET – підтримує підхід Code First, за якого структура бази даних генерується автоматично на основі класів-моделей через механізм міграцій. Це усуває необхідність ручного написання DDL-скриптів і автоматично синхронізує схему бази з кодом при кожній зміні моделі. Управління еволюцією схеми в процесі розробки стає суттєво простішим: замість ручного накочення SQL-скриптів достатньо виконати команду dotnet ef migrations add і застосувати міграцію.

Реляційні бази даних для вебзастосунків

Для систем із чітко визначеними зв'язками між сутностями та вимогами до транзакційної цілісності реляційні СКБД залишаються найбільш обґрунтованим вибором – і цей висновок зберігає актуальність попри поширення NoSQL-рішень у

певних сценаріях. Провідні реляційні СКБД – Microsoft SQL Server, PostgreSQL, MySQL – повністю відповідають властивостям ACID, підтримують транзакції, зовнішні ключі, індекси та збережені процедури. З-поміж них SQL Server вирізняється найглибшою інтеграцією з екосистемою Microsoft та платформою .NET, що спрощує конфігурацію і налагодження при використанні Entity Framework Core як ORM.

Клієнтська частина та адаптивний дизайн

Коректне відображення інтерфейсу на пристроях із різними розмірами екрана – не опціональна вимога, а базова умова прийнятності сучасного вебзастосунку. Підхід «mobile-first» передбачає проєктування інтерфейсу починаючи зі смартфону, з послідовним масштабуванням для більших екранів. Bootstrap 5 – один із найпоширеніших CSS-фреймворків – надає готову сітку, набір компонентів та утиліти для швидкої реалізації адаптивного дизайну без написання значних обсягів власного CSS. Для забезпечення динамічної поведінки клієнтської частини без переходу до SPA-архітектури (React, Vue.js, Angular) достатньо нативного JavaScript із AJAX-запитами – це виправданий підхід для застосунків із серверним рендерингом на основі Razor Views.

Проведений аналіз підтверджує обґрунтованість сформованого технологічного стеку: ASP.NET Core MVC з мовою C# для серверної частини, Entity Framework Core для взаємодії з базою даних, Microsoft SQL Server як СКБД, Bootstrap 5 і JavaScript – для клієнтської частини. Детальний розгляд кожного інструменту з обґрунтуванням конкретних рішень наведено у підрозділі 2.2.

2.2 Аналіз засобів розробки

Серверна частина вебзастосунку обробляє HTTP-запити, виконує бізнес-логіку, взаємодіє з базою даних і формує відповіді клієнту. Для її реалізації обрано платформу ASP.NET Core MVC із мовою програмування C#.

Мова програмування C#

C# – об'єктно-орієнтована мова програмування загального призначення, розроблена компанією Microsoft. Станом на 2024 рік вона стабільно входить до топ-10 найпопулярніших мов за індексом TIOBE. Мова підтримує кілька парадигм програмування: об'єктно-орієнтовану, функціональну та компонентно-орієнтовану. Серед практично значущих можливостей – строга типізація, автоматичне управління пам'яттю через збирач сміття (Garbage Collector), підтримка асинхронного програмування через конструкції async/await, а також LINQ для зручної роботи з колекціями і базами даних. Строга типізація скорочує кількість помилок часу виконання і прискорює рефакторинг, оскільки компілятор відловлює невідповідності типів ще до запуску. У вебзастосунку торгівлі транспортними засобами C# використовується для реалізації контролерів, сервісів, репозиторіїв та моделей даних.

Фреймворк ASP.NET Core MVC (.NET 9)

ASP.NET Core – кросплатформений відкритий фреймворк від Microsoft для розробки сучасних вебзастосунків та API. Версія .NET 9, випущена у листопаді 2024 року, є актуальною на момент розробки. Порівняно з Node.js (Express), Python (Django) і Java (Spring Boot) ASP.NET Core демонструє одні з найкращих показників продуктивності: за даними TechEmpower Framework Benchmarks, фреймворк стабільно входить до топ-10 найшвидших у категоріях обробки JSON-запитів та роботи з базами даних – цей результат підтверджується дослідженням Patel R. та Williams J. (2023) [2].

Фреймворк реалізує архітектурний патерн MVC (Model–View–Controller), який розмежовує три зони відповідальності: модель відповідає за дані та бізнес-логіку, представлення – за відображення інформації, контролер – за обробку запитів і координацію взаємодії між рештою компонентів. Вбудована підтримка Dependency Injection дозволяє реалізувати принцип інверсії залежностей – кожен клас отримує залежності через конструктор, що знижує зв'язаність компонентів і спрощує ізольоване тестування. У розробленому вебзастосунку через ASP.NET Core реалізовано MVC-контролери для серверного рендерингу сторінок, REST API-

контролери для обробки AJAX-запитів, а також middleware для автентифікації й авторизації.

Entity Framework Core

Entity Framework Core – сучасний об'єктно-реляційний маппер (ORM) для платформи .NET, що організовує взаємодію між об'єктною моделлю застосунку та реляційною базою даних. Як зазначають Nguyen T. та співавтори (2022) [4], EF Core забезпечує оптимальний баланс між продуктивністю й зручністю розробки для проєктів середнього масштабу. Інструмент дозволяє розробнику працювати з даними через C#-об'єкти та LINQ-запити без написання SQL вручну – це прискорює розробку і знижує ризик синтаксичних помилок у запитах. EF Core підтримує підхід Code First: структура бази даних визначається через класи-сутності в коді, а система автоматично генерує відповідні таблиці через механізм міграцій. У вебзастосунку Entity Framework Core застосовується для визначення моделей даних і виконання CRUD-операцій.

Автентифікація та авторизація

Безпеку застосунку забезпечено через Cookie Authentication із Claims-based Authorization. Після успішного входу AuthService формує об'єкт ClaimsPrincipal із даними користувача, його ідентифікатором і роллю. ASP.NET Core встановлює автентифіковану cookie-сесію, яка долучається до кожного наступного запиту до захищених ресурсів. Роль користувача зберігається у claims і дозволяє реалізувати гнучку рольову модель доступу без звернення до бази даних при кожній перевірці прав. Паролі зберігаються у захешованому вигляді через PasswordHasher<Customer> з ASP.NET Core Identity.

Засоби розробки клієнтської частини

Клієнтська частина відображає інформацію для користувача й забезпечує інтерактивність інтерфейсу. Для її реалізації обрано стек: HTML5, CSS3, Bootstrap 5 і JavaScript.

HTML5, CSS3 та JavaScript

HTML5 формує структуру сторінок вебзастосунку, CSS3 і Bootstrap відповідають за стилізацію та адаптивну верстку, JavaScript – за інтерактивність і обмін даними із сервером. Основні сторінки розміщено у директорії wwwroot як статичні HTML-файли: головна сторінка, каталог, картка транспортного засобу, сторінки входу та реєстрації, профіль, обране, порівняння, сторінки продавця та інші. Дані завантажуються з REST API у форматі JSON і динамічно відображаються на сторінці без повного перезавантаження.

CSS3 та Bootstrap 5

CSS3 – мова стилів, що керує зовнішнім виглядом HTML-елементів і підтримує анімації, медіа-запити для адаптивного дизайну, Flexbox і Grid для гнучкого компонування. Bootstrap 5 – один із найпоширеніших CSS-фреймворків, що надає готову компонентну бібліотеку та адаптивну сіткову систему на основі 12 колонок. За оцінкою Silva M. та Costa R. (2022) [6], Bootstrap 5 – найбільш виправданий вибір для enterprise-застосунків завдяки розвиненій компонентній базі. На основі Bootstrap 5 у вебзастосунку реалізовано адаптивну навігацію, картки транспортних засобів, форми заявок, модальні вікна й таблиці адміністративної панелі. Іконографія забезпечується бібліотекою Bootstrap Icons.

JavaScript

JavaScript – інтерпретована мова програмування, що виконується у браузері й забезпечує динамічну поведінку вебсторінок. У вебзастосунку JavaScript відповідає за валідацію форм на стороні клієнта, обробку подій і динамічне оновлення вмісту без перезавантаження сторінки, а також за AJAX-запити до REST API. AJAX надсилає HTTP-запити у фоні – завдяки цьому оновлюється лише потрібна частина DOM, а не вся сторінка. Саме такий підхід забезпечує плавну взаємодію під час фільтрації каталогу та надсилання заявок.

Засоби управління базою даних

Microsoft SQL Server

Microsoft SQL Server – реляційна СКБД корпоративного рівня, розроблена компанією Microsoft. Як обґрунтовано у дослідженні Ivanova O. та Kovalenko V.

(2023) [7], реляційні СКБД залишаються оптимальним вибором для систем із чітко визначеними зв'язками між сутностями та вимогами до транзакційної цілісності. SQL Server забезпечує повне дотримання властивостей ACID, підтримує транзакції, зовнішні ключі та індекси. Інтеграція з Entity Framework Core реалізована через офіційний пакет Microsoft.EntityFrameworkCore.SqlServer, що гарантує сумісність із платформою і доступ до всіх специфічних можливостей СКБД.

SQL Server Management Studio (SSMS)

SQL Server Management Studio – безкоштовний графічний інструмент від Microsoft для адміністрування баз даних SQL Server. SSMS дозволяє виконувати SQL-запити, переглядати й редагувати дані безпосередньо в таблицях, аналізувати плани виконання запитів для виявлення вузьких місць, а також управляти правами доступу на рівні бази. У процесі розробки SSMS слугував інструментом для первинного наповнення бази тестовими даними, перевірки коректності структури таблиць після застосування міграцій EF Core і налагодження складних вибірок.

Visual Studio 2026

Visual Studio 2026 – інтегроване середовище розробки від Microsoft, що охоплює повний цикл роботи над проєктом. IDE надає інтелектуальне автодоповнення коду (IntelliSense), вбудований відладчик з підтримкою точок зупинки та покрокового виконання, інтеграцію з системою контролю версій Git, шаблони проєктів ASP.NET Core, менеджер пакетів NuGet та аналізатори коду в реальному часі. Visual Studio 2026 слугував основним середовищем розробки вебзастосунку впродовж усього проєкту.

2.3 Специфікація вимог до програмного забезпечення

Призначення та межі проєкту

Призначення системи

Розроблений вебзастосунок автоматизує представлення автомобільного каталогу автосалону в мережі Інтернет і організовує прийом заявок на придбання транспортних засобів від потенційних покупців. Покупець отримує доступ до структурованого каталогу з детальними технічними характеристиками та

фотогалереями і може надіслати заявку безпосередньо зі сторінки обраного автомобіля, не звертаючись до менеджера телефоном. Адміністратор, зі свого боку, керує каталогом через вбудовану панель управління, опрацьовує вхідні заявки та адмініструє облікові записи зареєстрованих користувачів.

Цільова аудиторія охоплює три групи учасників із якісно різними сценаріями взаємодії з системою:

– потенційних покупців транспортних засобів, що шукають автомобіль онлайн;

– адміністраторів автосалону, відповідальних за управління каталогом та обробку заявок;

– власників компанії, яка займається продажем автомобілів.

Саме ця відмінність у ролях і сценаріях лягла в основу розмежування функціональності та проєктування системи прав доступу.

Погодження програмної документації

При розробці дотримано таких нормативних документів:

– ISO/IEC 25010:2011 – основа для формування нефункціональних вимог та оцінки характеристик якості ПЗ;

– IEEE 830–1998, методологічний зразок якого покладено в основу структури цієї специфікації;

– рекомендації OWASP щодо безпеки вебзастосунків;

– стандарт REST, архітектурні принципи якого визначили проєктування API.

Межі проєкту

Система включає:

– вебінтерфейс для перегляду каталогу та надсилання заявок (клієнтська частина);

– RESTful API для взаємодії клієнта з сервером;

– серверну логіку обробки запитів та управління даними;

– базу даних для зберігання інформації про транспортні засоби, користувачів та заявки;

– адміністративну панель для управління каталогом, заявками та користувачами.

Система не включає:

– інтеграцію з платіжними системами;

– інтеграцію із зовнішніми сервісами перевірки автомобілів за VIN–кодом;

– мобільний застосунок для iOS та Android;

– систему онлайн–чату між покупцем та менеджером.

Перелічені виключення зумовлені цільовим призначенням системи як інструменту генерації первинних звернень, а не повноцінної торгової платформи з онлайн-оплатою та миттєвою комунікацією.

Сфера застосування

Вебзастосунок розрахований на автосалони та дилерські компанії з продажу нових або вживаних транспортних засобів, яким потрібен власний онлайн–майданчик для представлення каталогу. Залежно від бізнес-моделі підприємства, система виконує різні функції: або слугує єдиною цифровою точкою контакту з потенційним покупцем, або доповнює фізичний автосалон – в останньому випадку онлайн–заявка перебирає функцію первинного звернення замість традиційного телефонного дзвінка. Таке рішення дозволяє фіксувати інтерес покупця в будь-який момент прийняття рішення: у позаробочий час, у вихідні або при першому знайомстві з пропозицією через мережу.

Впровадження системи надає автосалону такі операційні переваги порівняно з традиційним форматом обробки звернень:

– цілодобова доступність каталогу транспортних засобів для потенційних покупців;

– автоматизація прийому та відстеження заявок на придбання;

– зменшення навантаження на менеджерів завдяки централізованій обробці заявок через адміністративну панель;

– зручність вибору автомобіля для покупця без необхідності телефонного дзвінка;

– повний контроль адміністратора над актуальністю каталогу в режимі реального часу.

Сукупний ефект від упровадження скорочує час між першим інтересом покупця та фіксацією його звернення, що безпосередньо впливає на конверсію відвідувачів сайту в реальних клієнтів.

Характеристики користувачів

Таблиця 2.1 – Характеристики користувачів системи

Тип користувача	Роль	Рівень технічної підготовки	Основні завдання
Гість	Відвідувач	Базовий (користувач інтернету)	Перегляд каталогу, пошук та фільтрація автомобілів, реєстрація
Зареєстрований користувач	Покуець	Базовий (користувач інтернету)	Надсилання заявок на придбання, відстеження їх статусу в особистому кабінеті
Адміністратор	Менеджер магазину	Середній (досвід роботи з CMS)	Управління каталогом транспортних засобів, обробка заявок, управління користувачами

Загальна структура і склад системи

Система побудована за трирівневою клієнт–серверною архітектурою:

1) Рівень представлення (Presentation Layer):

- вебінтерфейс користувача на основі HTML5, CSS3, Bootstrap 5;
- адаптивний дизайн для коректного відображення на різних пристроях;
- клієнтська валідація форм засобами JavaScript.

2) Рівень бізнес–логіки (Business Logic Layer):

- MVC–контролери ASP.NET Core для обробки запитів;
- REST API контролери для AJAX–взаємодії;
- сервіси для реалізації бізнес–правил системи;
- middleware для автентифікації, авторизації та обробки помилок.

3) Рівень даних (Data Access Layer):

- Entity Framework Core як ORM;
- репозиторії для інкапсуляції логіки доступу до даних;
- міграції для управління схемою бази даних;
- Microsoft SQL Server для зберігання всіх даних системи.

Загальні обмеження

Технологічні обмеження:

- система розроблена для роботи на платформі ASP.NET Core (.NET 9);
- мінімальна версія Microsoft SQL Server – 2019;
- підтримувані браузери: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+.

Організаційні обмеження:

- розробка виконується одним розробником;
- відсутність бюджету для придбання комерційних бібліотек;
- термін розробки обмежений академічним роком.

Юридичні обмеження:

- система має відповідати вимогам захисту персональних даних (GDPR);
- обов'язкове використання HTTPS для передачі персональних даних користувачів.

Функції системи

Функція "Перегляд та пошук транспортних засобів"

Реалізує основний сценарій взаємодії покупця з каталогом: перегляд наявних транспортних засобів із фільтрацією за технічними параметрами, текстовим

пошуком та впорядкуванням результатів. Виклик функції не потребує автентифікації – каталог відкритий для будь-якого відвідувача сайту.

Вхідна інформація:

- параметри фільтрації: марка, модель, рік випуску від/до, тип пального, тип коробки передач, мінімальна та максимальна ціна;
- текстовий пошуковий запит за маркою або моделлю;
- параметр сортування: за ціною, за роком випуску, за датою додавання;
- номер сторінки для пагінації.

Вихідна інформація:

- список транспортних засобів з основною інформацією (марка, модель, рік, ціна, головне фото);
- загальна кількість знайдених автомобілів;
- деталізована картка конкретного транспортного засобу з повним описом, технічними характеристиками та фотогалереєю.

Функціональні вимоги:

- FR–1.1: система повинна відображати транспортні засоби у вигляді сітки карток;
- FR–1.2: кожна картка повинна містити головне фото, марку, модель, рік випуску, ціну та кнопку переходу до детальної картки;
- FR–1.3: система повинна підтримувати пагінацію з відображенням 9 автомобілів на сторінці;
- FR–1.4: фільтри повинні застосовуватися без повного перезавантаження сторінки;
- FR–1.5: детальна картка повинна містити всі технічні характеристики, опис, фотогалерею та форму заявки;
- FR–1.6: за відсутності транспортних засобів за заданими критеріями система повинна відображати відповідне повідомлення.

Функція "Реєстрація та автентифікація користувачів"

Відкриває доступ до особистого кабінету та функціоналу надсилання заявок: новий відвідувач проходить реєстрацію, після якої отримує обліковий запис із роллю customer і може увійти в систему для роботи із захищеними розділами.

Вхідна інформація:

- при реєстрації: ім'я, електронна пошта, пароль, підтвердження пароля;
- при вході: електронна пошта, пароль.

Вихідна інформація:

- підтвердження успішної реєстрації або повідомлення про помилку;
- автентифікований сеанс роботи з системою після успішного входу;
- автентифікована користувацька сесія на основі cookie та claims для подальшої авторизації захищених запитів.

Функціональні вимоги:

- FR–2.1: система повинна перевіряти унікальність електронної пошти при реєстрації;
- FR–2.2: пароль повинен містити не менше 8 символів;
- FR–2.3: пароль зберігається виключно у хешованому вигляді з використанням PasswordHasher<Customer> із ASP.NET Core Identity;
- FR–2.4: після успішної реєстрації користувач автоматично отримує роль customer;
- FR–2.5: система повинна відображати зрозумілі повідомлення про помилки при некоректному введенні даних.

Вимоги до інформаційного забезпечення

Джерела вхідної інформації:

1. дані транспортних засобів – вводяться адміністратором через адміністративну панель при створенні або редагуванні записів каталогу;

2. дані користувачів – вводяться при реєстрації нового облікового запису та при редагуванні профілю;

3. дані заявок – формуються зареєстрованим користувачем при заповненні форми заявки на сторінці конкретного транспортного засобу;

4. фотографії транспортних засобів – завантажуються адміністратором через форму редагування картки автомобіля.

Нормативно–довідкова інформація:

Система використовує наступні довідники:

1. дані про виробника та модель транспортного засобу зберігаються безпосередньо в таблиці Vehicles у полях Manufacturer та Model;

2. довідник типів пального – бензин, дизель, газ, електро, гібрид;

3. довідник типів коробки передач – механічна, автоматична, варіатор, роботизована;

4. довідник типів кузова – седан, хетчбек, універсал, кросовер, позашляховик, купе, кабріолет;

5. статуси транспортних засобів використовуються для керування доступністю оголошень у каталозі;

6. довідник статусів транспортного засобу – Доступний, Проданий;

7. довідник ролей користувачів реалізовано через поле Role у таблиці Customers, основними ролями є customer та admin.

Вимоги до організації та ведення інформації:

1. усі дані зберігаються в реляційній базі даних Microsoft SQL Server;

2. цілісність даних забезпечується через зовнішні ключі та обмеження на рівні бази даних;

3. паролі користувачів зберігаються виключно у хешованому вигляді за допомогою PasswordHasher<Customer> із ASP.NET Core Identity;

4. логування критичних операцій: зміна статусу заявки, видалення транспортного засобу, зміна ролі користувача;

5. усі записи містять часові мітки створення та останнього оновлення.

Вимоги до технічного забезпечення

Вимоги до серверного обладнання:

1. Процесор: мінімум 2 ядра, рекомендовано 4 ядра.
2. Оперативна пам'ять: мінімум 4 ГБ, рекомендовано 8 ГБ.
3. Дисковий простір: мінімум 20 ГБ (з урахуванням зберігання фотографій транспортних засобів).
4. Мережеве з'єднання: широкосмугове підключення до Інтернету.

Вимоги до клієнтського обладнання:

1. Будь–який пристрій з сучасним веббраузером (ПК, ноутбук, планшет, смартфон).
2. Мінімальна роздільна здатність екрану: 320×568 пікселів.
3. Підключення до Інтернету зі швидкістю не менше 1 Мбіт/с.

Вимоги до програмного забезпечення

Архітектура програмної системи

Система реалізована за патерном MVC з розділенням на три компоненти:

1. Model: Entity–класи Vehicle, Customer, Category, Inventory, Transaction, Maintenance, VehicleInquiry, VehicleFavorite, VehicleView, VehicleHistory та інші моделі предметної області; DTO–класи для передачі даних між рівнями;
2. View: статичні HTML-сторінки, CSS-стилі та JavaScript-модулі, розміщені у директорії wwwroot. Клієнтська частина взаємодіє із сервером через REST API та оновлює DOM без повного перезавантаження сторінки.
3. Controller: MVC–контролери для обробки запитів від представлень; API–контролери для AJAX–запитів; Middleware для автентифікації та обробки помилок.

Системне програмне забезпечення:

- ОС: Windows Server 2019+ або Linux Ubuntu 20.04+;
- платформа виконання: ASP.NET Core (.NET 9);
- СКБД: Microsoft SQL Server 2019+;

- ORM: Entity Framework Core з підходом Code First та підтримкою міграцій.

Мережеве програмне забезпечення:

- протокол взаємодії: HTTPS;
- формат обміну даними між клієнтом та API: JSON.

Мова і технології розробки

Backend: C#, ASP.NET Core (.NET 9), REST API, Entity Framework Core, Cookie Authentication, Claims-based Authorization, PasswordHasher<Customer>.

HTML5, CSS3 (custom design system), JavaScript (ES6+), jQuery 3.72.3.7 Вимоги до зовнішніх інтерфейсів.

Інтерфейс користувача:

- інтуїтивно зрозумілий інтерфейс без необхідності спеціального навчання;
- уніфікований дизайн на всіх сторінках системи;
- адаптивна верстка для коректного відображення на різних розмірах екрану;
- підтримка браузерів: Chrome, Firefox, Safari, Edge.

Основні елементи інтерфейсу: навігаційне меню з посиланнями та інформацією про поточного користувача; панель фільтрів для каталогу; картки транспортних засобів; детальна картка автомобіля з фотогалереєю та формою заявки; особистий кабінет із списком заявок; адміністративна панель управління.

Апаратний інтерфейс:

- стандартні пристрої введення: клавіатура, миша, сенсорний екран;
- дисплей з роздільною здатністю від 320×568 пікселів.

Властивості програмного забезпечення

Нефункціональні характеристики системи фіксуються у вигляді атрибутів якості з вимірюваними критеріями – саме вони визначають, чи відповідає готовий

продукт поставленим вимогам, а не лише реалізує задану функціональність. Для вебзастосунку торгівлі транспортними засобами визначено п'ять груп таких атрибутів.

Доступність. Коефіцієнт готовності системи встановлено на рівні не нижче 99%, що математично допускає простій не більше 87,6 годин на рік, або приблизно 7,3 години на місяць. Цей поріг обрано як мінімально прийнятний для комерційного застосунку з постійним потоком запитів – нижче нього втрати від недоступності сервісу перевищують витрати на забезпечення надійності. Система забезпечує цілодобовий доступ без планових вікон обслуговування у години пік. Архітектура застосункового шару уникає єдиних точок відмови, а конфігурація бази даних передбачає регулярне резервне копіювання з перевіркою відновлення.

Супроводжуваність. Кодова база організована за модульним принципом: кожен компонент системи відповідає за єдину функціональну область, завдяки чому модифікація однієї підсистеми не тягне каскадних змін в інших. Публічні методи та класи документуються засобами XML-коментарів у форматі, сумісному з інструментом DocFX, – це дозволяє автоматично генерувати актуальну технічну документацію безпосередньо з коду. Підсистема структурованого логування на базі вбудованого ILogger<T> фіксує ключові події виконання, помилки та попередження з контекстними метаданими, що скорочує час локалізації виробничих інцидентів. Єдиний стиль коду закріплено у файлі .editorconfig і перевіряється інструментами статичного аналізу на етапі збірки.

Переносимість. Цільова платформа .NET 9 надає єдине виконавче середовище для Windows і Linux, що усуває необхідність у платформозалежних гілках коду або окремих збірках для різних ОС. Конфігурація середовища повністю винесена в зовнішні файли appsettings.json і змінні оточення – та сама бінарна збірка розгортається в локальному середовищі розробки, тестовому стенді та виробничому сервері без модифікацій. Мінімальні апаратні вимоги обмежуються наявністю сумісної ОС і достатнього обсягу оперативної пам'яті для виконання .NET-процесу; прив'язки до конкретних постачальників заліза або хмарних провайдерів немає.

Продуктивність. Порогові значення швидкодії визначено з урахуванням галузевих орієнтирів для вебзастосунків електронної комерції: час відгуку сервера на запит не перевищує 2 секунди, завантаження головної сторінки разом зі статичними ресурсами – не більше 3 секунд. Система витримує одночасну роботу до 50 користувачів без деградації цих показників, що покриває прогнозоване пікове навантаження для застосунку регіонального масштабу. Для досягнення цих характеристик застосовано два технічних рішення: кешування результатів повторюваних запитів на рівні ORM і явне керування стратегією завантаження пов'язаних сутностей EF Core через Include/ThenInclude – це унеможливлює проблему N+1 запитів, типову для ORM-based застосунків.

Безпека. Модель безпеки застосунку охоплює чотири незалежні рівні захисту, кожен з яких адресує окремий клас загроз. Паролі зберігаються у хешованому вигляді через PasswordHasher<Customer> з бібліотеки ASP.NET Core Identity: алгоритм PBKDF2 з унікальною сіллю та адаптивним числом ітерацій унеможливлює відновлення пароля навіть при компрометації бази даних. Entity Framework Core формує параметризовані SQL-запити на рівні ORM, що структурно виключає SQL-ін'єкції – вхідні дані ніколи не потрапляють у тіло запиту як рядок. Сесія автентифікованого користувача підтримується Cookie Authentication з прапором HttpOnly: браузер надсилає cookie з кожним запитом, але JavaScript-код на сторінці не має до неї доступу, що нейтралізує атаки типу XSS із крадіжкою сесії. Розмежування прав між ролями customer та admin реалізоване через Claims-based Authorization: атрибут [Authorize(Roles = "admin")] на рівні контролерів гарантує, що адміністративні дії недоступні звичайним користувачам незалежно від маніпуляцій на боці клієнта.

Висновки до розділу 2

Другий розділ вирішував два взаємопов'язані завдання: обґрунтування технологічного стеку майбутньої системи та формалізацію вимог до неї. Результати обох складових підпорядковані єдиній логіці – забезпечити відповідність архітектурних рішень задокументованим вимогам.

Монолітна архітектура з патерном MVC обрана як найбільш обґрунтований підхід для системи даного масштабу: вона гарантує чітке розділення відповідальностей між компонентами й водночас уникає надмірного ускладнення, характерного для мікросервісних рішень. ASP.NET Core MVC та C# визначені як серверна основа з урахуванням зрілості платформи, якості інструментарію та активної підтримки з боку спільноти. Entity Framework Core разом із Microsoft SQL Server утворюють рівень даних, де типобезпечні запити EF Core унеможливлюють цілий клас вразливостей, а реляційна модель SQL Server забезпечує транзакційну цілісність.

Клієнтська частина реалізується засобами HTML5, CSS3, Bootstrap 5 та JavaScript – набором технологій, що відповідає поточним стандартам адаптивного вебдизайну. Bootstrap 5 обрано з практичних міркувань: він скорочує обсяг власного CSS і забезпечує передбачувану поведінку макету на різних роздільних здатностях без залучення додаткових залежностей. Вибраний комплекс засобів безпеки – PasswordHasher, параметризовані запити EF Core, Cookie Authentication та Claims-based Authorization – покриває основні вектори атак, актуальні для вебзастосунків категорії електронної комерції.

Специфікація вимог охопила функціональний і нефункціональний виміри системи. Функціональні вимоги структуровані навколо трьох основних підсистем: каталогу транспортних засобів із пошуком і фільтрацією, управління обліковими записами користувачів та подання заявок на придбання. Нефункціональні вимоги встановлюють вимірювані показники якості – продуктивність, надійність, безпеку, супроводжуваність і переносимість – що утворюють базис для верифікації системи на етапі тестування.

3 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Третій розділ присвячено проєктним рішенням вебзастосунку торгівлі транспортними засобами. Спочатку визначається фізична топологія розгортання і шарова архітектура системи, далі – комплекс UML-діаграм, що формалізують поведінку та структуру застосунку, і нарешті реляційна схема бази даних. Послідовність викладу відповідає ієрархії рівнів: від інфраструктурного до логічного й інформаційного.

3.1 Архітектура програмного забезпечення

Фізична топологія системи

Розгортання системи охоплює три функціонально ізольовані вузли, між якими немає прямих залежностей на рівні коду: клієнтський браузер, серверний процес ASP.NET Core і екземпляр СКБД.

Клієнтська нода – браузер. Статичні HTML-сторінки завантажуються з директорії wwwroot сервера. Динамічна взаємодія реалізується двома механізмами: стандартні HTTP-переходи між сторінками для навігаційних операцій і AJAX-запити до REST API для дій без перезавантаження – фільтрація каталогу та надсилання заявок. Такий поділ дозволяє мінімізувати обсяг трафіку при інтерактивних операціях, не ускладнюючи серверний рендеринг.

Серверна нода – процес ASP.NET Core (.NET 9) під управлінням IIS або Kestrel. На цьому рівні зосереджено всю бізнес-логіку: маршрутизація запитів, cookie-автентифікація, claims-based авторизація, валідація вхідних даних, сервісний шар і доступ до даних через Entity Framework Core.

Нода СКБД – ізольований екземпляр Microsoft SQL Server. Транзакційна цілісність забезпечується на рівні ACID незалежно від прикладного коду – це принципово важливо для операцій, що зачіпають кілька таблиць одночасно.

Клієнт і сервер взаємодіють виключно через HTTPS. API-контролери повертають JSON, який JavaScript-модулі на клієнті отримують і використовують для точкового оновлення інтерфейсу без повного перезавантаження сторінки.

Патерн MVC із сервісним шаром

Прикладний рівень побудовано на патерні MVC, розширеному сервісним шаром між контролерами і репозиторіями. Це рішення продиктоване практичною необхідністю: без сервісного шару бізнес-логіка або осідає в контролерах, перевантажуючи їх, або дублюється між ними.

Model охоплює два типи класів із різними зонами відповідальності. Entity-класи – Vehicles, Customers, Transactions, VehicleInquiries та інші – відображають таблиці бази даних і конфігуруються через Fluent API у DbContext. DTO-класи утворюють контракт між клієнтською частиною та API: вони визначають структуру JSON-відповідей і JSON-запитів, не виводячи Entity-об'єкти за межі сервісного рівня. Цей бар'єр блокує дві ключові вразливості – over-posting, коли клієнт надсилає поля, яких не повинен чіпати, і витік внутрішньої схеми даних у відповідях.

View-рівень реалізовано статичними HTML-сторінками у директорії wwwroot: каталог, картка транспортного засобу, особистий кабінет, сторінки входу та реєстрації, сторінки продавця, порівняння й обраного. Інтерактивність забезпечується JavaScript-модулями, що звертаються до REST API. Адаптивність верстки реалізовано через Bootstrap 5 з mobile-first підходом.

Controller розбито на два незалежні типи. VehiclesController обслуговує каталог, фільтрацію, перегляд, додавання, редагування і приховування автомобілів, а також фіксацію переглядів, обраного та заявок. CustomersController відповідає за реєстрацію, вхід, профіль і вихід користувача. Обидва типи повертають JSON-відповіді для JavaScript-коду на сторінках wwwroot.

Service Layer представлено класами VehicleService, AuthService, CustomerService, FavoritesService, SellerDashboardService та TransactionService. Вони реалізують бізнес-логіку каталогу, автентифікації, профілю користувача, обраного, статистики продавця та створення угод. Контролери не звертаються до DbContext напряму – весь доступ до даних проходить через сервісний шар, що дозволяє тестувати бізнес-правила без підняття реальної бази даних.

Middleware–конвеєр

Middleware-конвеєр обробляє кожен HTTP-запит послідовно, причому порядок компонентів має практичне значення: зміна черговості автентифікації та авторизації призведе до некоректної роботи захисту.

Автентифікаційний middleware першим перевіряє cookie автентифікованого користувача і формує об'єкт ClaimsPrincipal із набором тверджень про ідентичність і роль. Authorization Middleware, що стоїть наступним у конвеєрі, використовує цей об'єкт для прийняття рішень про доступ до захищених ресурсів на основі claims і ролі користувача. Операції, доступні лише адміністратору – наприклад, приховування автомобілів або перегляд панелі продавця, – захищені атрибутом [Authorize(Roles = "admin")] безпосередньо на рівні контролера, тоді як дії автентифікованого покупця, такі як подання заявки або управління обраним, контролюються атрибутом [Authorize] без уточнення ролі.

3.2 UML – моделювання системи

До складу UML–моделі входять діаграми варіантів використання, послідовностей, діяльності, класів і розгортання.

Діаграма варіантів використання

З системою взаємодіють чотири категорії учасників із різними рівнями доступу. Межі між ролями визначаються не лише набором дозволених операцій, а й механізмом Claims-based Authorization: кожен авторизований запит перевіряє відповідні claims перед виконанням будь-якої захищеної дії.

Гість – неавтентифікований відвідувач. Він переглядає каталог, застосовує фільтри, відкриває детальні картки автомобілів з характеристиками, фотогалереєю та даними VehicleHistory. Якщо гість намагається надіслати заявку – система перенаправляє його до форми реєстрації, а не просто блокує дію.

Покупець – зареєстрований користувач. Він отримує всі можливості гостя, а також надсилає заявки через VehicleInquiries, веде персональний список обраних

автомобілів (VehicleFavorites), а перегляди карток фіксуються автоматично в VehicleViews – це дані, які може використовувати продавець для аналітики.

Продавець – зареєстрований користувач з розширеними правами на публікацію. Він розміщує власні оголошення (поле SellerCustomerCode у таблиці Vehicles), керує їхнім статусом і видимістю, а також переглядає заявки від покупців, надіслані щодо його автомобілів.

Адміністратор – роль із повним доступом до back-office. Він виконує CRUD-операції по каталогу Vehicles (зокрема змінює IsVisible та Status), опрацьовує всі VehicleInquiries, управляє обліковими записами Customers, переглядає Transactions, керує Categories та Inventory, а також веде журнал Maintenance.

Діаграма варіантів використання

Система охоплює три рівні доступу: гість, зареєстрований користувач та адміністратор. Гість бачить каталог, може шукати автомобілі й відкривати їхні картки – на цьому його взаємодія із системою завершується. Зареєстрований користувач отримує доступ до заявок на придбання, списку обраного, історії звернень і налаштувань профілю; більшість цих операцій перевіряє автентифіковану cookie-сесію і відповідні claims перед виконанням. Адміністратор керує каталогом транспортних засобів, категоріями, складськими записами та іншими довідковими сутностями. Заявки покупців фіксуються в VehicleInquiries і використовуються продавцем для аналізу попиту. Частина сценаріїв пов'язана залежностями include і extend: зокрема, «Надсилання заявки» завжди запускає перевірку автентифікації, а перегляд картки автомобіля може розширитися додаванням позиції до обраного.

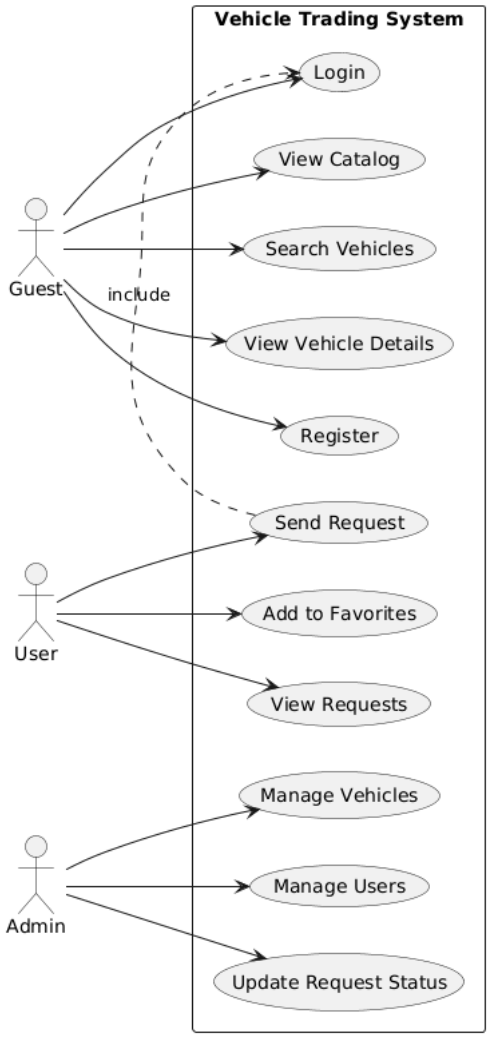

Рисунок 3.1 – Діаграма варіантів використання

Діаграма варіантів використання показує, які операції доступні кожній ролі і де проходять межі між рівнями доступу. На її основі деталізовано конкретні сценарії взаємодії у вигляді діаграм послідовностей.

Діаграма послідовностей автентифікації користувача

Користувач вводить логін і пароль та підтверджує відправлення. Браузер надсилає POST-запит до CustomersController за маршрутом /api/Customers/login. Контролер передає отримані дані до AuthService, який через CursWorkContext шукає відповідний запис у таблиці Customers за електронною поштою або іменем користувача.

Якщо запис знайдено, AuthService звіряє введений пароль з хешем через PasswordHasher<Customer>. За успішної перевірки формується об'єкт

ClaimsPrincipal із ідентифікатором, іменем і роллю користувача, після чого ASP.NET Core встановлює автентифіковану cookie-сесію. Якщо перевірка не пройдена – система повідомляє про невірні облікові дані, але не уточнює, яке саме поле хибне. Це стандартна практика захисту проти перебирання паролів методом порівняння відповідей.

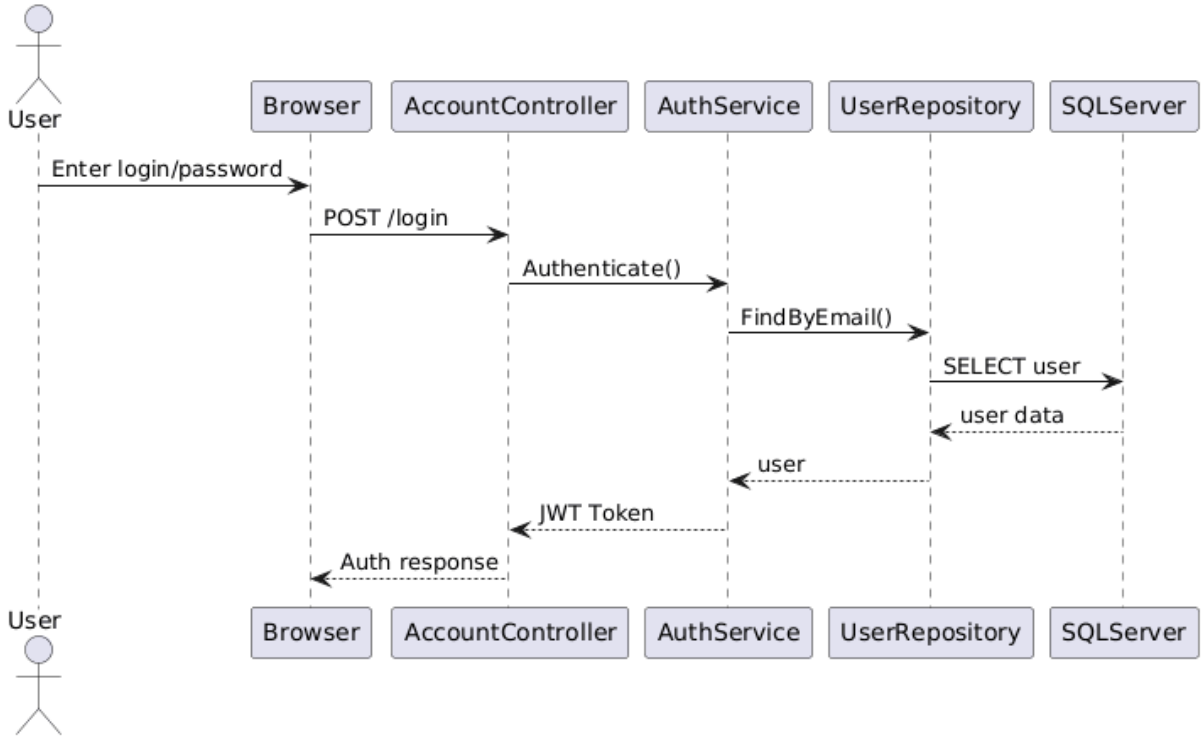

Рисунок 3.2 – Діаграма послідовностей автентифікації користувача

Діаграма послідовностей розкриває покроковий механізм перевірки облікових даних і формування автентифікованої сесії. Пройшовши цей процес, користувач отримує доступ до захищених операцій – зокрема до надсилання заявок на придбання транспортних засобів.

Діаграма послідовностей надсилання заявки

Коли покупець натискає «Надіслати заявку», браузер виконує AJAX POST-запит до маршруту /api/Vehicles/{id}/inquiry. VehiclesController передає дані до SellerDashboardService, який спочатку перевіряє наявність транспортного засобу з указаним ідентифікатором і лише після цього створює новий запис у таблиці VehicleInquiries. Тіло запиту містить ідентифікатор автомобіля, ідентифікатор

користувача (за наявності автентифікованої сесії) і текст повідомлення. Після збереження клієнт отримує підтвердження – без перезавантаження сторінки і без переривання перегляду.

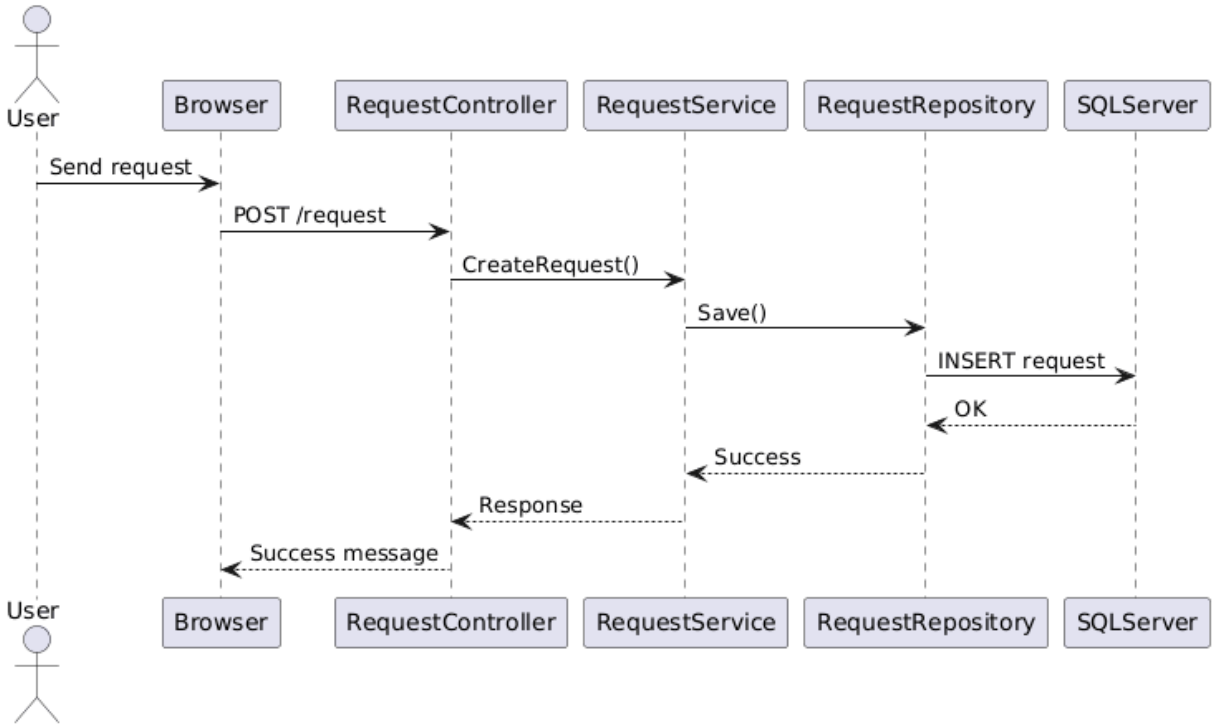

Рисунок 3.3 – Діаграма послідовностей надсилання заявки

Діаграма показує, як система перевіряє доступність транспортного засобу перед створенням заявки і яким чином відповідь повертається клієнту без переривання взаємодії. Наступний сценарій розглядає операції з боку адміністратора – додавання і редагування записів у каталозі.

Діаграма послідовностей управління транспортними засобами

Після заповнення форми редагування або додавання автомобіля браузер надсилає запит до VehicleController, який передає дані до VehicleService. Сервіс виконує валідацію: перевіряє обов'язкові поля, рік випуску, формат і допустимий розмір фотографій, коректність числових параметрів – і лише після успішної перевірки передає дані до VehicleRepository для збереження. Зображення транспортних засобів зберігаються в моделі Vehicle як посилання або шляхи до файлів у полях Image, Image2, Image3 та Image4. Окрім основних параметрів

автомобіля, VehicleService перевіряє VIN-код, категорію та інші характеристики, що забезпечує цілісність записів у каталозі.

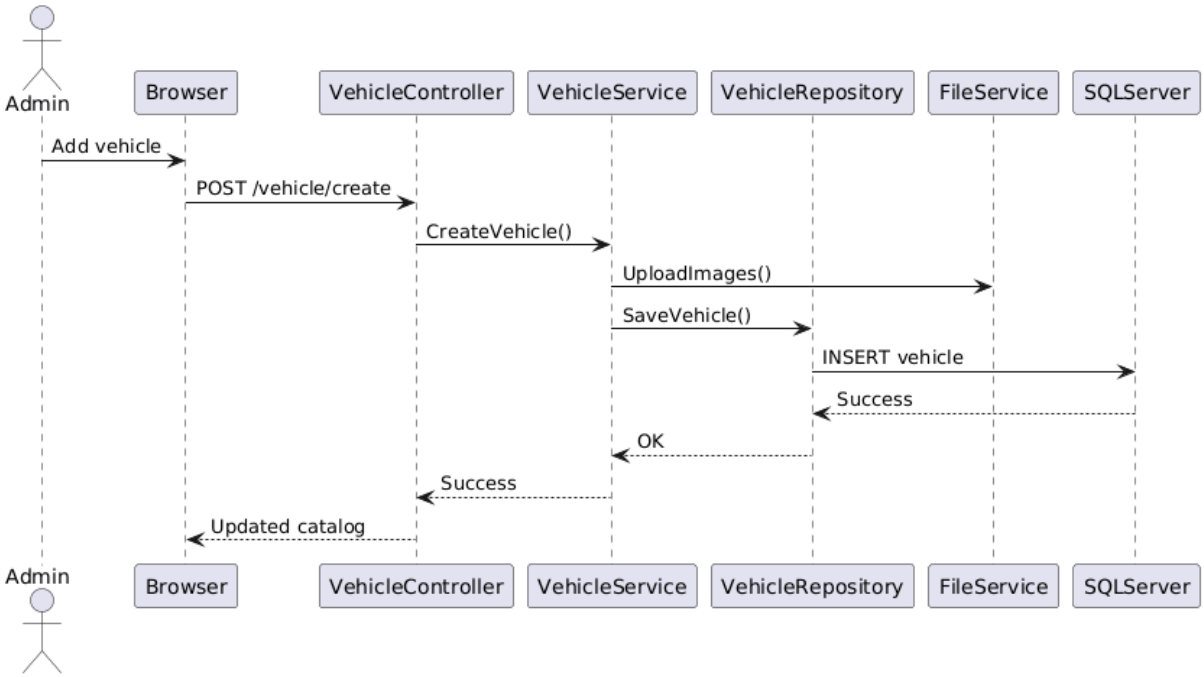

Рисунок 3.4 – Діаграма послідовностей управління транспортними засобами

Діаграма фіксує послідовність операцій при додаванні та редагуванні транспортного засобу – від заповнення форми до збереження запису через сервісний шар і механізм завантаження файлів. Детальнішу логіку виконання цих сценаріїв розкривають діаграми діяльності, наведені нижче.

Діаграма діяльності процесу автентифікації

Після введення облікових даних і натискання «Увійти» система перевіряє їхню коректність. Якщо перевірка не пройдена – форма повертається з повідомленням про помилку без уточнення, яке поле хибне. При успішному вході система зчитує роль користувача і відповідно маршрутизує його: адміністратор потрапляє до панелі керування, звичайний користувач – до особистого кабінету. Паралельно сервер встановлює cookie автентифікованої сесії, а клієнтська частина зберігає базові дані користувача для налаштування відображення інтерфейсу.

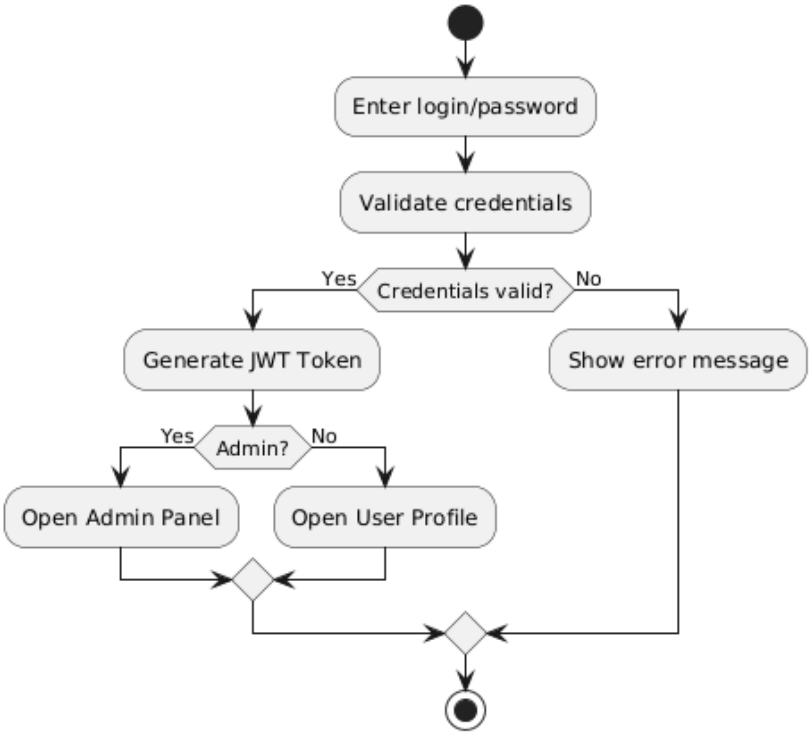

Рисунок 3.5 – Діаграма діяльності автентифікації

Діаграма діяльності розкриває алгоритм автентифікації та логіку маршрутизації користувачів залежно від їхніх ролей. Схожим чином побудовано опис процесу фільтрації транспортних засобів у каталозі.

Діаграма діяльності фільтрації каталогу

При відкритті каталогу завантажується повний список доступних автомобілів разом із параметрами фільтрації: виробник, модель, рік, тип пального, коробка передач, ціна. Зміна будь-якого з цих параметрів автоматично ініціює GET-запит до сервера без дій з боку користувача. На сервері Entity Framework Core формує LINQ-запит із динамічними умовами фільтрації: відповідні позиції відбираються на рівні SQL, а не у пам'яті застосунку. Якщо за заданими критеріями автомобілів не знайдено – інтерфейс відображає відповідне повідомлення; якщо результати є – сітка карток оновлюється точково, без перезавантаження сторінки.

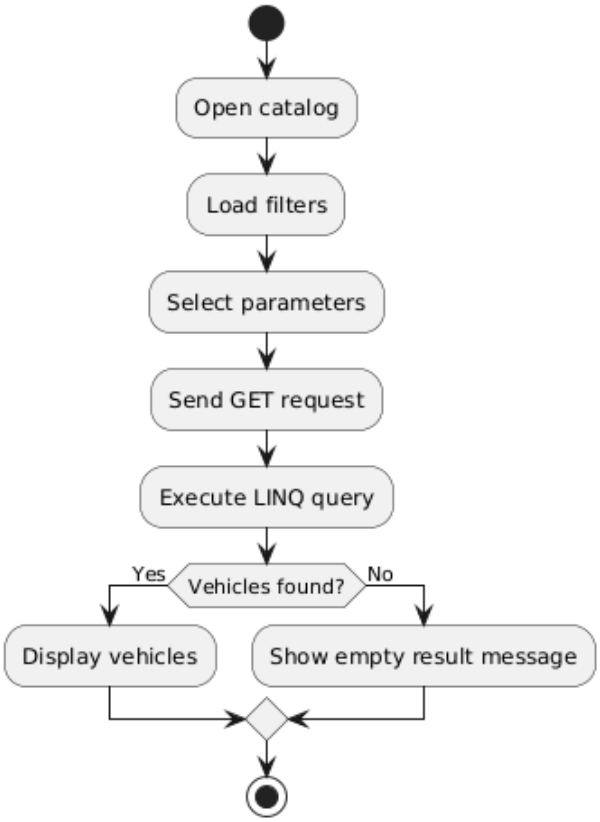

Рисунок 3.6 – Діаграма діяльності фільтрації каталогу

Діаграма діяльності відображає алгоритм формування і виконання запиту на пошук транспортних засобів за заданими критеріями – від введення параметрів до оновлення сітки результатів. Окремим сценарієм розглянуто процес додавання нових записів до каталогу.

Діаграма діяльності додавання транспортного засобу

Адміністратор заповнює форму додавання: вносить технічні характеристики автомобіля, текст опису та завантажує фотографії. До відправки форми клієнтський JavaScript перевіряє очевидні помилки – порожні обов'язкові поля, невідповідний формат значень – і одразу сигналізує про них без звернення до сервера. На серверному рівні підключаються DataAnnotations і FluentValidation, які виконують глибшу валідацію: перевіряють діапазони числових значень, унікальність VIN-коду, коректність категорії. Якщо валідація виявила помилки – форма повертається з конкретними підказками поруч із відповідними полями. Якщо всі перевірки

пройдено успішно – транспортний засіб зберігається в базі та одразу з'являється в каталозі.

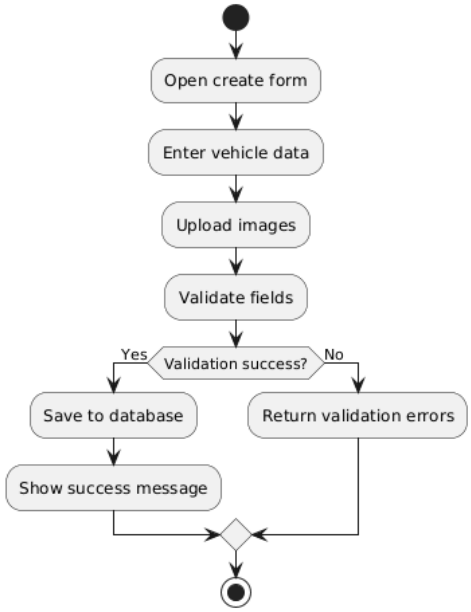

Рисунок 3.7 – Діаграма діяльності додавання транспортного засобу

ER-діаграма відображає структуру даних системи та взаємозв'язки між основними сутностями предметної області. На її підставі визначено правила підтримки референційної цілісності між таблицями бази даних.

Специфікації варіантів використання

UC–01: Перегляд та пошук транспортних засобів

Атрибут	Опис
Актор	Гість, Покупець
Передумова	Відсутня
Основний сценарій	1) Актор відкриває каталог. 2) Система відображає ТЗ з IsVisible = true, Status = Available. 3) Актор застосовує фільтри (категорія, марка, рік, тип пального, КПП, ціна). 4) Актор переходить до детальної картки.

Кінець UC–01: Перегляд та пошук транспортних засобів

Альтернативний сценарій	Немає результатів – система пропонує скинути фільтри.
Постумова	Для авторизованого користувача перегляд зафіксовано в VehicleViews.

UC–02: Надсилання заявки на придбання

Атрибут	Опис
Актор	Покупець
Передумова	Користувач автентифікований; ТЗ має статус Available.
Основний сценарій	1) Актор відкриває картку автомобіля. 2) Заповнює форму (Message – необов'язково; контактні дані підставляються з профілю). 3) Підтверджує відправлення. 4) Система зберігає запис у VehicleInquiries. 5) Підтвердження відображається без перезавантаження сторінки.
Альтернативний сценарій	ТЗ отримав статус Sold до моменту збереження – система повертає помилку, інтерфейс оновлює статус.
Постумова	Запис видно адміністратору і в особистому кабінеті покупця.

UC–03: Управління каталогом

Атрибут	Опис
Актор	Адміністратор
Передумова	Автентифікований з роллю Admin.

Кінець UC–03: Управління каталогом

Основний сценарій	1) Відкриває розділ «Транспортні засоби» в панелі управління. 2) Переглядає повний список (включно з прихованими). 3) Виконує операцію: додати, редагувати, змінити IsVisible / Status, видалити. 4) Зміни зберігаються через сервісний шар.
Постумова	Публічний каталог оновлено відповідно до нових значень IsVisible та Status.

3.3 Проєктування структури бази даних

Базу даних побудовано на Microsoft SQL Server. Структура нормалізована: кожна сутність розміщується в окремій таблиці, зв'язки між ними підтримуються через зовнішні ключі. Така організація унеможливлює появу «сирітських» записів без батьківського контексту і спрощує підтримку референційної цілісності на рівні СКБД.

Центральна таблиця – Vehicles. Вона містить повний набір характеристик автомобіля: назву, ціну, рік випуску, пробіг, виробника, модель, тип пального, об'єм двигуна, коробку передач, VIN-код, опис комплектації, статус і прапор видимості. З нею пов'язані чотири поля для зображень (Image, Image2, Image3, Image4) та зовнішній ключ на продавця через SellerCustomerCode. Решта таблиць будується навколо Vehicles – всі аналітичні, транзакційні та поведінкові сутності посилаються на неї напряму або через проміжні зв'язки.

Customers зберігає дані і покупців, і продавців: ім'я, телефон, електронну пошту, хеш пароля, роль і дату реєстрації. Хеш пароля формується засобами PasswordHasher<Customer> з ASP.NET Core Identity. Об'єднання обох ролей в одній таблиці – свідоме рішення: воно уникає дублювання ідентичних атрибутів у двох окремих таблицях і спрощує управління обліковими записами через єдиний механізм автентифікації.

VehicleViews і VehicleFavorites забезпечують поведінкову аналітику: перша таблиця фіксує кожен перегляд картки автомобіля, друга – додавання до списку обраного. Обидві прив'язані до клієнта і автомобіля через зовнішні ключі. Ці дані дають продавцеві можливість бачити реальний інтерес покупців до конкретних позицій каталогу і приймати рішення про ціноутворення чи акції на підставі фактичної поведінки аудиторії.

Для прискорення частих запитів на таблицях створено індекси за полями, що використовуються у фільтрації та сортуванні: Manufacturer, Model, Price, Year, Status, CreatedAt, Customer_Code. Без цих індексів кожна фільтрація каталогу потребувала б повного перебору таблиці – при значному обсязі даних це безпосередньо впливало б на час відгуку інтерфейсу.

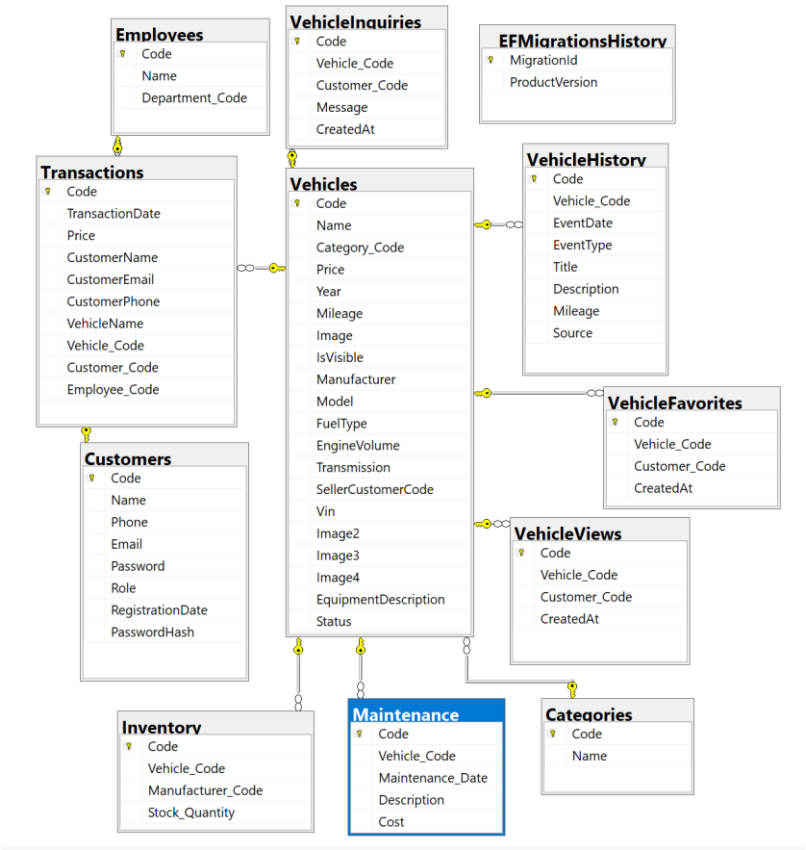

Рисунок 3.8 – ER–діаграма бази даних

ER-діаграма відображає структуру даних системи та взаємозв'язки між основними сутностями предметної області. На її підставі визначено правила підтримки референційної цілісності між таблицями бази даних.

Стратегія референційної цілісності

Вибір між CASCADE і RESTRICT для кожного зовнішнього ключа продиктований бізнес–логікою. Правило просте: якщо дочірній запис без батьківського не має сенсу – CASCADE. Якщо дочірній запис є юридичним або комерційним фактом – RESTRICT.

Для сутностей, що мають комерційне або облікове значення, використовується обмеження видалення. Зокрема, зв’язки Transactions, Inventory та Maintenance із Vehicles налаштовані з поведінкою Restrict, щоб уникнути втрати важливих даних. Для допоміжних поведінкових сутностей, таких як VehicleViews, VehicleFavorites, VehicleInquiries та VehicleHistory, використовується каскадне видалення разом із відповідним транспортним засобом.VehicleViews і VehicleFavorites, навпаки, видаляються каскадно – втрата цих даних не порушує жодного бізнес–процесу.

Code First і міграції EF Core

Схему бази даних не описано вручну через DDL-скрипти – Entity Framework Core генерує структуру таблиць на основі C#-класів і конфігурацій Fluent API у DbContext. Кожна зміна схеми оформлюється як окремий пронумерований файл міграції, що зберігається разом із кодом у системі контролю версій. При розгортанні команда dotnet ef database update ідемпотентно застосовує всі ще невиконані міграції. Службова таблиця __EFMigrationsHistory фіксує, які міграції вже виконані на конкретному екземплярі бази – це дозволяє відтворювати схему на будь-якому середовищі без ручних скриптів і без ризику повторного застосування вже виконаних змін.

3.4 Діаграма класів системи

Центральний клас – Vehicle. Він зберігає повний набір характеристик автомобіля, прапор видимості, статус і навігаційні властивості до пов'язаних сутностей: історії, переглядів, обраного, запитів і записів обслуговування.

Customer описує і покупців, і продавців – роль конкретного облікового запису визначається полем Role. Клас зберігає контактні дані та хеш пароля, необхідний для автентифікації через PasswordHasher<Customer>.

Transaction фіксує завершений продаж і посилається на Vehicle, Customer та Employee. VehicleInquiry зберігає звернення покупця щодо конкретного автомобіля – як правило, це запит перед придбанням або прохання про консультацію. Журнал подій по кожному автомобілю – і сервісних, і комерційних – веде VehicleHistory. Довідкові класи Maintenance, Inventory і Category доповнюють систему інформацією про технічне обслуговування, складський облік і класифікацію транспортних засобів відповідно.

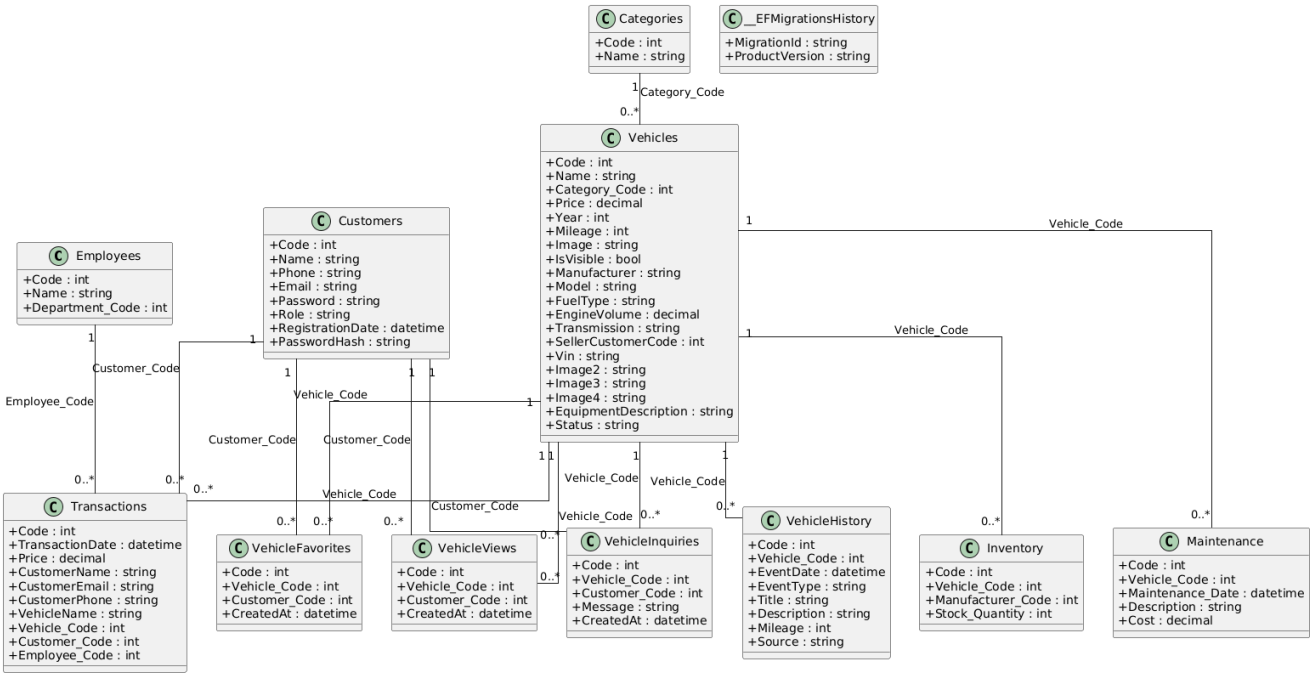

Рисунок 3.9 – Діаграма класів системи

Діаграма класів відображає статичну структуру програмної системи та зв'язки між її основними сутностями. Наступний етап – опис фізичного розгортання компонентів системи на вузлах інфраструктури.

3.5 Діаграма розгортання системи

Клієнт працює у браузері й з'єднується з сервером через HTTPS. Усі запити – і стандартні HTTP, і асинхронні AJAX – проходять через єдину точку входу. На

сервері застосунку розгорнуто ASP.NET MVC з REST API і повним стеком бізнес-логіки: автентифікація побудована на Cookie Authentication і Claims-based Authorization, взаємодія з базою – через Entity Framework Core. Окремо розміщено сервер Microsoft SQL Server, який зберігає всі реляційні дані – автомобілі, клієнтів, співробітників, угоди, заявки, історію та аналітику переглядів.

У базі даних для кожного транспортного засобу зберігаються не самі зображення, а посилання або шляхи до файлів у полях Image, Image2, Image3 та Image4. Реляційні СКБД погано підходять для зберігання бінарних даних – великі BLOB-об'єкти уповільнюють вибірки і ускладнюють резервне копіювання. Винесення медіафайлів окремо від таблиці вирішує обидві проблеми й відповідає усталеній практиці у веброзробці.

Версію схеми бази даних відстежує таблиця EFMigrationsHistory, яку Entity Framework Core веде автоматично при кожному застосуванні міграцій. Завдяки цьому система точно знає, які зміни схеми вже виконано на конкретному середовищі, і відтворює її на будь-якому новому екземплярі без ручних скриптів і без ризику повторного застосування вже виконаних змін.

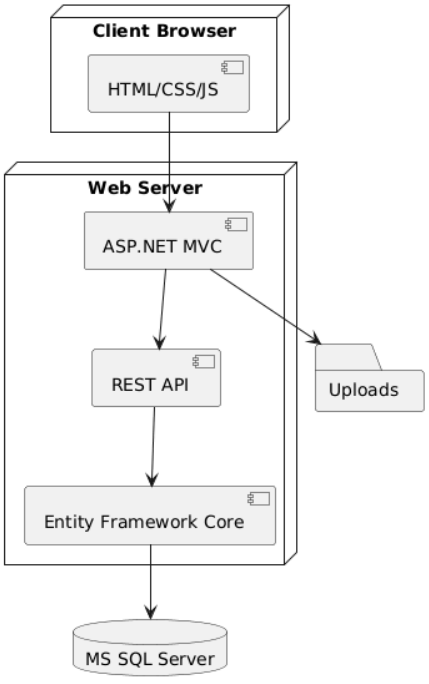

Рисунок 3.10 – Діаграма класів системи

Діаграма розгортання відображає фізичну структуру системи та взаємодію її основних компонентів у режимі реальної експлуатації. Спроєктована архітектура визначає технічну основу для практичної реалізації вебзастосунку, описаної в наступному розділі.

Висновки до розділу 3

Третій розділ присвячено проєктуванню програмної системи вебзастосунку торгівлі транспортними засобами. Під час проєктування сформовано фізичну топологію системи, яка визначає взаємодію клієнтської частини, сервера застосунку та сервера бази даних. Обмін даними між цими компонентами здійснюється через захищений протокол HTTPS, що дає змогу забезпечити належний рівень безпеки під час передавання інформації та відповідає сучасним вимогам до вебсистем такого типу.

Основу архітектури програмного забезпечення становить патерн MVC у поєднанні із сервісним шаром. Такий підхід дозволив розділити логіку роботи системи між окремими компонентами відповідно до їхнього призначення. У результаті зміни в одному шарі не потребують суттєвого втручання в роботу інших частин застосунку. Це спрощує супроводження програмного забезпечення, полегшує тестування окремих модулів і створює умови для подальшого розширення функціональності системи.

Для опису структури та поведінки системи побудовано набір UML-діаграм, до якого увійшли діаграми варіантів використання, послідовностей, діяльності, класів і розгортання. Кожна з них відображає окремий аспект функціонування системи та доповнює загальну картину її роботи. Розроблені моделі охоплюють основні бізнес-процеси предметної області, взаємодію користувачів із функціональними підсистемами, а також порядок виконання операцій автентифікації, обробки заявок і керування каталогом транспортних засобів. Використання UML-моделювання ще на етапі проєктування дало можливість перевірити логіку взаємодії компонентів і заздалегідь виявити потенційні проблеми, які могли б виникнути під час реалізації системи.

Окрему увагу приділено проєктуванню бази даних. Для цього розроблено ER-діаграму та визначено склад основних сутностей предметної області, їхні атрибути і взаємозв'язки. Сформована модель даних охоплює інформацію про транспортні засоби, користувачів, заявки, транзакції, історію автомобілів та інші допоміжні сутності, необхідні для забезпечення роботи вебзастосунку. Цілісність даних підтримується за допомогою правил CASCADE і RESTRICT, які застосовуються залежно від характеру зв'язків між сутностями. Для керування структурою бази даних використано підхід Code First та механізм міграцій Entity Framework Core, що спрощує внесення змін до схеми бази даних протягом розвитку проєкту.

У результаті виконаного проєктування сформовано цілісну модель програмної системи, яка охоплює її архітектурний, функціональний та інформаційний аспекти. Розроблені моделі та прийняті архітектурні рішення визначили структуру майбутнього вебзастосунку і принципи взаємодії його основних компонентів. Саме ці напрацювання стали основою для подальшої реалізації системи. У наступному розділі розглянуто структуру програмних модулів вебзастосунку та особливості реалізації його основних функціональних можливостей.

4.ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

Четвертий розділ присвячено практичній реалізації вебзастосунку AUTOPULSE. У ньому наведено структуру проєкту та принципи розподілу відповідальності між окремими шарами системи, розглянуто реалізацію ключових серверних компонентів із відповідними фрагментами програмного коду, а також подано опис інтерфейсу застосунку у форматі керівництва користувача. Окремо представлено результати функціонального тестування, виконаного шляхом надсилання ручних HTTP-запитів до REST API. Така послідовність викладу дозволяє простежити зв'язок між проєктними рішеннями та їх практичним втіленням у програмному продукті.

4.1 Структура проєкту

Вебзастосунок реалізовано у вигляді монолітного ASP.NET Core проєкту CursWork (.NET 9), структура якого охоплює чотири логічні шари: статичну клієнтську частину (wwwroot), контролери API (Controllers), бізнес-логіку (Services) та компоненти передавання даних між шарами (DTOs). Вибір єдиного проєкту був свідомим архітектурним рішенням. За таких умов відсутня потреба підтримувати додаткові міжпроєктні залежності, які зазвичай виникають у мікросервісних системах або під час розробки великими командами. Це дозволило зосередитися на реалізації функціональності без ускладнення структури програмного рішення.

Шар wwwroot містить набір HTML-сторінок разом із відповідними CSS- та JS-модулями. Кожна сторінка функціонує автономно, підключаючи власний сценарій із директорії js/ та спільні стилі з каталогу css/. Бібліотеки Bootstrap 5 і Bootstrap Icons підключено з директорії lib/. На стороні клієнта не використовується серверний рендеринг, тому сторінки взаємодіють із сервером через Fetch-запити та самостійно оновлюють DOM після отримання відповіді. Такий підхід спрощує налагодження клієнтської частини та дає можливість розробляти її незалежно від серверних компонентів.

До складу шару Controllers/ входять вісім контролерів: AdminController, CategoriesController, CustomersController, HomeController, InventoryController, MaintenanceController, TransactionsController та VehiclesController. Для кожного з них використано атрибути [ApiController] і [Route("api/[controller]")], що забезпечують стандартизовану REST-маршрутизацію запитів. Контролери виконують лише обробку вхідних даних та передають керування відповідним сервісам. Бізнес-логіка винесена за межі цього шару, завдяки чому контролери залишаються компактними та виконують виключно функції взаємодії між клієнтом і серверною частиною застосунку.

У шарі Services/ реалізовано основні бізнес-правила системи. Зокрема, AuthService відповідає за реєстрацію користувачів, авторизацію, формування ClaimsPrincipal та підтримку cookie-автентифікації, VehicleService забезпечує операції CRUD для каталогу транспортних засобів, фільтрацію та пагінацію даних, CustomerService керує профілями користувачів, FavoritesService працює з обраними транспортними засобами, SellerDashboardService формує статистичні показники для продавця, а TransactionService фіксує виконані угоди. Клас CursWorkContext виконує роль ApplicationDbContext та містить DbSet для кожної таблиці бази даних. Усі міграції зберігаються в директорії Migrations/. Така організація сервісного шару концентрує бізнес-логіку в одному місці та спрощує її подальший супровід.

Шар DTOs/ визначає структуру обміну даними між клієнтською та серверною частинами застосунку. До нього належать AddVehicleDTO, LoginDTO, RegisterDTO, VehicleDTO, VehicleHistoryDto та SellerDashboardDto. Використання DTO дозволяє відокремити внутрішні Entity-моделі від зовнішнього API та контролювати склад даних, які передаються клієнту. Завдяки цьому зміни у внутрішній структурі моделей не порушують стабільність API-контракту та не впливають на роботу клієнтської частини.

Обробка вхідних HTTPS-запитів відбувається через конвеєр Middleware. Спочатку запит проходить через CORS Middleware, який контролює доступ із дозволених клієнтських origin, після чого Authentication Middleware перевіряє

автентифікаційну cookie користувача, а Authorization Middleware визначає його права доступу. Після завершення цих перевірок маршрутизатор передає запит відповідному контролеру. Водночас статичні ресурси з директорії wwwroot обслуговуються через UseStaticFiles() і не потребують проходження через контролери. Така схема дозволяє розділити обробку динамічного та статичного вмісту, не створюючи зайвого навантаження на серверну частину.

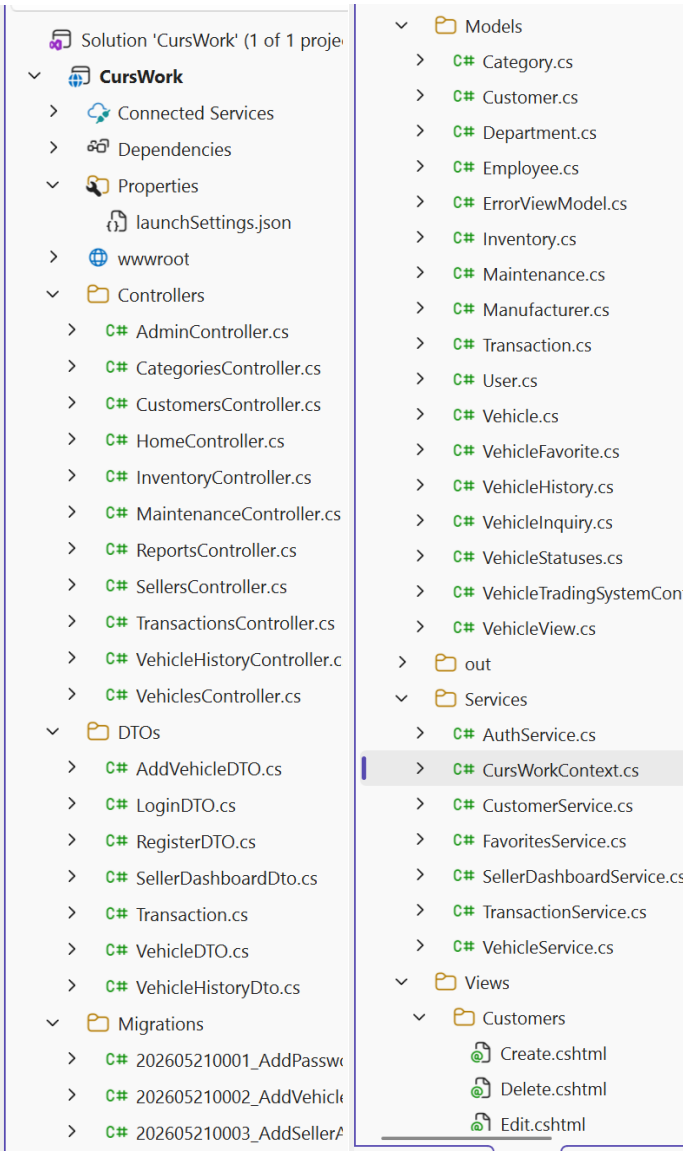

Рисунок 4.1 – Структура проєкту CursWork

Наведена структура проєкту демонструє розподіл компонентів між основними шарами системи та забезпечує чітке розмежування відповідальності. Далі розглянуто реалізацію ключових серверних компонентів вебзастосунку.

4.2 Реалізація серверної частини

Серверну частину реалізовано на платформі .NET 9 з ASP.NET Core Web API. Нижче описано чотири ключових компоненти: AuthService (реєстрація, перевірка облікових даних і формування автентифікованої сесії), VehicleService (фільтрація каталогу), CustomersController (вхідна точка API) і CursWorkContext (доступ до БД).

Автентифікація користувачів та формування ClaimsPrincipal (AuthService)

AuthService є центральним компонентом системи автентифікації та авторизації користувачів. Сервіс реалізує повний цикл роботи з обліковими записами та відповідає за:

– реєстрацію користувачів;
– перевірку логіна і пароля;
– створення ClaimsPrincipal;
– генерацію авторизаційної сесії;
– роботу з ролями;
– захищене зберігання паролів;
– підтримку міграції старих паролів.

Для реалізації сервісу використано:

– ASP.NET Core Identity;
– Cookie Authentication;
– Entity Framework Core;
– Claims–based Authorization.

Хешування паролів

Для захисту паролів використовується PasswordHasher<Customer> із ASP.NET Core Identity.

```
customer.PasswordHash = _passwordHasher.HashPassword(customer, password);
_context.Entry(customer).Property("LegacyPassword").CurrentValue = string.Empty;
await _context.SaveChangesAsync();
return true;
```

Рисунок 4.2 – Хешування паролів

PasswordHasher автоматично:

– генерує криптографічну сіль;
– використовує сучасний алгоритм PBKDF2;
– створює захищений PasswordHash.

У базі даних пароль у відкритому вигляді не зберігається. Це дозволяє захистити систему навіть у випадку компрометації бази даних.

Перевірка пароля

Після отримання користувача виконується перевірка PasswordHash:

```
1 reference
private async Task<bool> VerifyPasswordAsync(Customer customer, string password)
{
    if (!string.IsNullOrWhiteSpace(customer.PasswordHash))
    {
        var result = _passwordHasher.VerifyHashedPassword(customer, customer.PasswordHash, password);
        if (result == PasswordVerificationResult.SuccessRehashNeeded)
        {
            customer.PasswordHash = _passwordHasher.HashPassword(customer, password);
            await _context.SaveChangesAsync();
        }

        return result != PasswordVerificationResult.Failed;
    }
```

Рисунок 4.3 – Перевірка пароля

ASP.NET Core Identity автоматично:

– порівнює хеші;
– визначає коректність пароля;
– перевіряє актуальність алгоритму.

Якщо алгоритм застарів, система автоматично виконує повторне хешування.

Фільтрація і пагінація каталогу (VehicleService)

VehicleService реалізує бізнес–логіку роботи з каталогом транспортних засобів. Сервіс відповідає за:

– динамічну фільтрацію автомобілів;
– роботу з VIN–кодами;
– створення та редагування автомобілів;
– отримання контактів продавця;
– формування DTO;

– роботу з Entity Framework Core.

Динамічна фільтрація каталогу

Основний метод сервісу – GetVehiclesAsync(). Метод формує LINQ–запит динамічно залежно від переданих фільтрів.

```
1 reference
public async Task<List<VehicleDTO>> GetVehiclesAsync(VehicleFilters filters)
{
    var query = _context.Vehicles
        .AsNoTracking()
        .Include(v => v.Category)
        .Where(v => v.IsVisible && v.Status == VehicleStatuses.Active);
```

Рисунок 4.4 – Динамічна фільтрація каталогу

AsNoTracking() вимикає відстеження Entity Framework та підвищує продуктивність. Include() виконує eager loading категорій автомобілів.

LINQ–фільтрація

Система підтримує фільтрацію за:

– категорією;

– маркою;

– моделлю;

– роком;

– типом пального;

Приклад динамічної фільтрації:

```
if (filters.CategoryCode.HasValue)
    query = query.Where(v => v.Category_Code == filters.CategoryCode.Value);
if (!string.IsNullOrWhiteSpace(filters.Manufacturer))
    query = query.Where(v => v.Manufacturer != null && v.Manufacturer == filters.Manufacturer);
if (!string.IsNullOrWhiteSpace(filters.Model))
    query = query.Where(v => v.Model != null && v.Model == filters.Model);
if (filters.YearFrom.HasValue)
    query = query.Where(v => v.Year >= filters.YearFrom.Value);
if (filters.YearTo.HasValue)
    query = query.Where(v => v.Year <= filters.YearTo.Value);
if (!string.IsNullOrWhiteSpace(filters.FuelType))
    query = query.Where(v => v.FuelType != null && v.FuelType == filters.FuelType);
if (filters.EngineFrom.HasValue)
    query = query.Where(v => v.EngineVolume.HasValue && v.EngineVolume.Value >= filters.EngineFrom.Value);
if (filters.EngineTo.HasValue)
    query = query.Where(v => v.EngineVolume.HasValue && v.EngineVolume.Value <= filters.EngineTo.Value);
if (filters.PriceFrom.HasValue)
    query = query.Where(v => v.Price >= filters.PriceFrom.Value);
if (filters.PriceTo.HasValue)
    query = query.Where(v => v.Price <= filters.PriceTo.Value);
if (!string.IsNullOrWhiteSpace(filters.Transmission))
    query = query.Where(v => v.Transmission != null && v.Transmission == filters.Transmission);
```

Рисунок 4.5 – LINQ–фільтрація

Наведений підхід дозволяє формувати ефективні SQL-запити без написання SQL-коду вручну. Наступний підрозділ присвячено реалізації контролера API.

Контролер API (CustomersController)

CustomersController є HTTP API–контролером системи та забезпечує взаємодію клієнтської частини з сервером.

Контролер реалізує:

– REST API;
– авторизацію;
– CRUD–операції;
– роботу з профілем користувача;
– login/logout;
– інтеграцію із сервісами.

REST API

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class CustomersController : ControllerBase
{
    private readonly CustomerService _customerService;
    private readonly AuthService _authService;

    0 references
    public CustomersController(CustomerService customerService, AuthService authService)
    {
        _customerService = customerService;
        _authService = authService;
    }
```

Рисунок 4.6 – REST API

ASP.NET Core автоматично:

– виконує model binding;
– серіалізує JSON;
– обробляє HTTP–запити.

Login Endpoint

Метод Login() виконує:

– перевірку логіна;

– створення cookie;

– повернення інформації про користувача.

```
await HttpContext.SignInAsync(
    CookieAuthenticationDefaults.AuthenticationScheme,
    result.Principal!,
    authProperties);
```

Рисунок 4.7 – Login Endpoint

Контекст бази даних (CursWorkContext)

CursWorkContext є центральним компонентом доступу до бази даних у Entity Framework Core.

Контекст відповідає за:

– DbSet–моделі;

– конфігурацію таблиць;

– зв’язки між сутностями;

– індекси;

– precision;

– Fluent API.

```
29 references
public class CursWorkContext : DbContext
{
    0 references
    public CursWorkContext(DbContextOptions options) : base(options) { }

    7 references
    public DbSet<Category> Categories { get; set; }
    27 references
    public DbSet<Vehicle> Vehicles { get; set; }
    14 references
    public DbSet<Inventory> Inventory { get; set; }
    1 reference
    public DbSet<Employee> Employees { get; set; }
    10 references
    public DbSet<Transaction> Transactions { get; set; }
    6 references
    public DbSet<Maintenance> Maintenance { get; set; }
    20 references
    public DbSet<Customer> Customers { get; set; }
    2 references
    public DbSet<VehicleView> VehicleViews { get; set; }
    2 references
    public DbSet<VehicleFavorite> VehicleFavorites { get; set; }
    2 references
    public DbSet<VehicleInquiry> VehicleInquiries { get; set; }
    4 references
    public DbSet<VehicleHistory> VehicleHistory { get; set; }
```

Рисунок 4.8 – DbSet–моделі

Сукупність DbSet-властивостей визначає набір сутностей, з якими працює Entity Framework Core. Важливим аспектом також є конфігурація зв'язків між таблицями.

```
0 references
protected override void OnModelCreating(ModelBuilder modelBuilder)
{

    modelBuilder.Entity<Vehicle>()
        .HasOne(v => v.Category)
        .WithMany(c => c.Vehicles)
        .HasForeignKey(v => v.Category_Code)
        .OnDelete(DeleteBehavior.Restrict);

    modelBuilder.Entity<Vehicle>()
        .Property(v => v.Price)
        .HasPrecision(18, 2);

    modelBuilder.Entity<Vehicle>()
        .Property(v => v.EngineVolume)
        .HasPrecision(4, 1);

    modelBuilder.Entity<Vehicle>()
        .Property(v => v.Status)
        .HasMaxLength(20)
        .HasDefaultValue(VehicleStatuses.Active);

    modelBuilder.Entity<Vehicle>()
        .HasIndex(v => v.Status)
        .HasDatabaseName("IX_Vehicles_Status");
```

Рисунок 4.9 – Зв’язки між таблицями

Налаштовані зв'язки забезпечують підтримання цілісності даних та коректну роботу навігаційних властивостей. Далі наведено керівництво користувача вебзастосунку.

4.3 Керівництво користувача

Вебзастосунок AUTOPULSE працює у браузері без встановлення стороннього програмного забезпечення. Підтримувані браузери: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+. Мінімальна ширина екрана: 320 px. Інтерфейс виконано в темній стилістиці з золотими акцентами, яка відповідає преміальному позиціонуванню платформи.

Головна сторінка

Після відкриття сторінки index.html користувач бачить повноекранний блок із фотографією та назвою актуального автомобіля з каталогу (рис. 4.10). На сторінці відображаються марка, модель, категорія, рік випуску та основні характеристики транспортного засобу. Праворуч розміщено перелік інших featured-автомобілів для швидкого переходу між пропозиціями. Нижня панель містить статистику каталогу: кількість активних оголошень, середню ціну та найновіший рік випуску. Кнопка BROWSE → відкриває сторінку catalog.html.

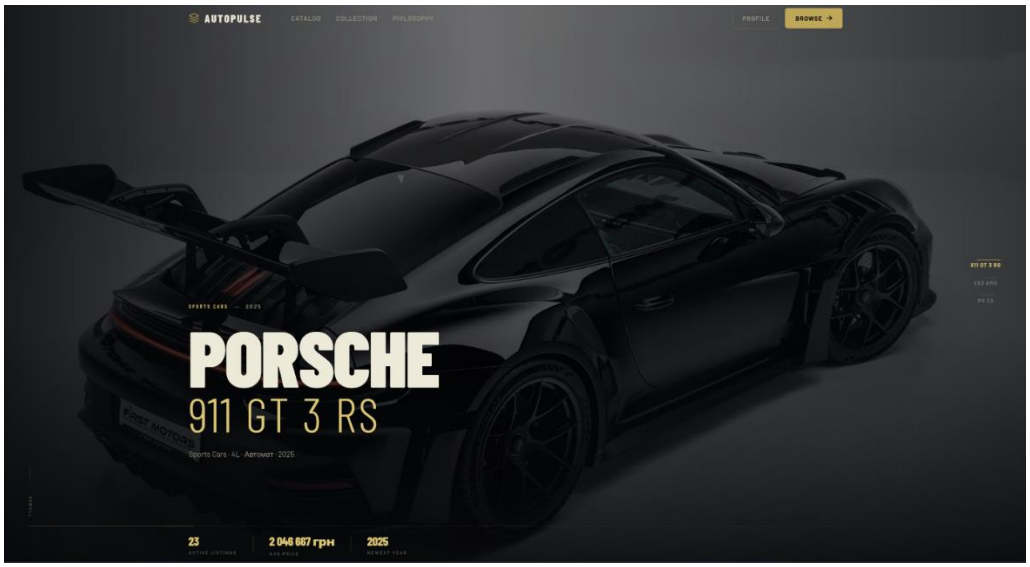

Рисунок 4.10 – Головна сторінка застосунку AUTOPULSE

Головна сторінка виконує роль точки входу до системи та забезпечує швидкий доступ до основних функцій платформи. Наступним елементом інтерфейсу є сторінка каталогу транспортних засобів.

Сторінка каталогу

Сторінка catalog.html (рис. 4.11) складається з навігаційної панелі та області відображення транспортних засобів. Бічна панель містить статистику каталогу та посилання на розділи CATALOG, FAVORITES, COMPARE, PROFILE і SERVICE BOOKING. Для неавторизованих користувачів доступні кнопки SIGN IN та REGISTER.

Основна частина сторінки містить панель пошуку ASPHALT GOLD SEARCH із фільтрами за категорією, маркою, моделлю, роком випуску, типом пального,

об'ємом двигуна та ціною. Нижче відображається кількість знайдених результатів, елементи сортування та список транспортних засобів.

Пошук виконується без перезавантаження сторінки. JavaScript формує параметри запиту та надсилає їх до API, після чого отримані дані автоматично оновлюють вміст сторінки.

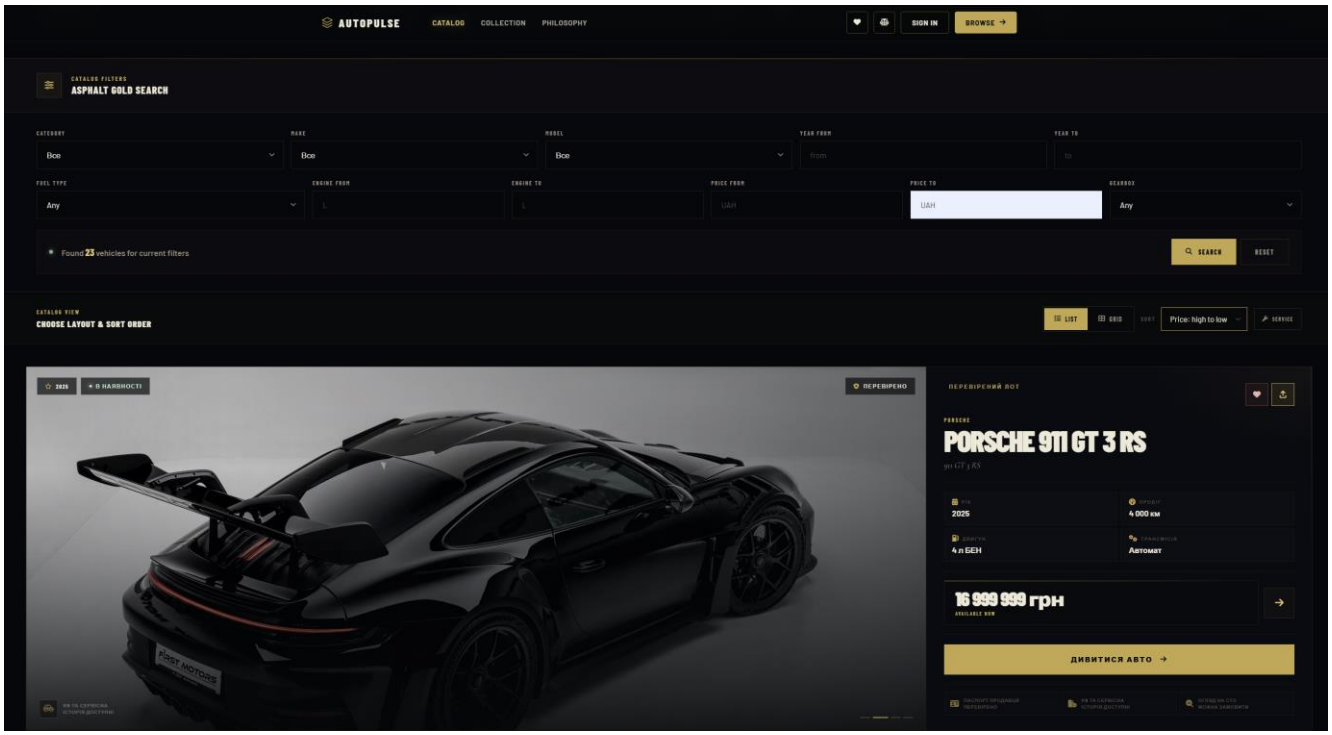

Рисунок 4.11 – Сторінка каталогу з панеллю фільтрів

Каталог є основним інструментом пошуку та перегляду транспортних засобів. Для виконання операцій, пов'язаних із придбанням автомобілів, користувачеві необхідно пройти реєстрацію та авторизацію.

Реєстрація та авторизація

Для подання заявки користувач повинен створити обліковий запис. Форма реєстрації register.html (рис. 4.13) містить поля для введення імені, електронної пошти та пароля. Перед надсиланням виконується перевірка коректності email і підтвердження пароля. Після успішної реєстрації система автоматично авторизує користувача та перенаправляє його до каталогу.

Сторінка login.html (рис. 4.12) містить поля Email і Пароль. У разі помилки відображається узагальнене повідомлення без уточнення причини. Після успішного входу сервер створює cookie-сесію, яка використовується для подальших захищених запитів.

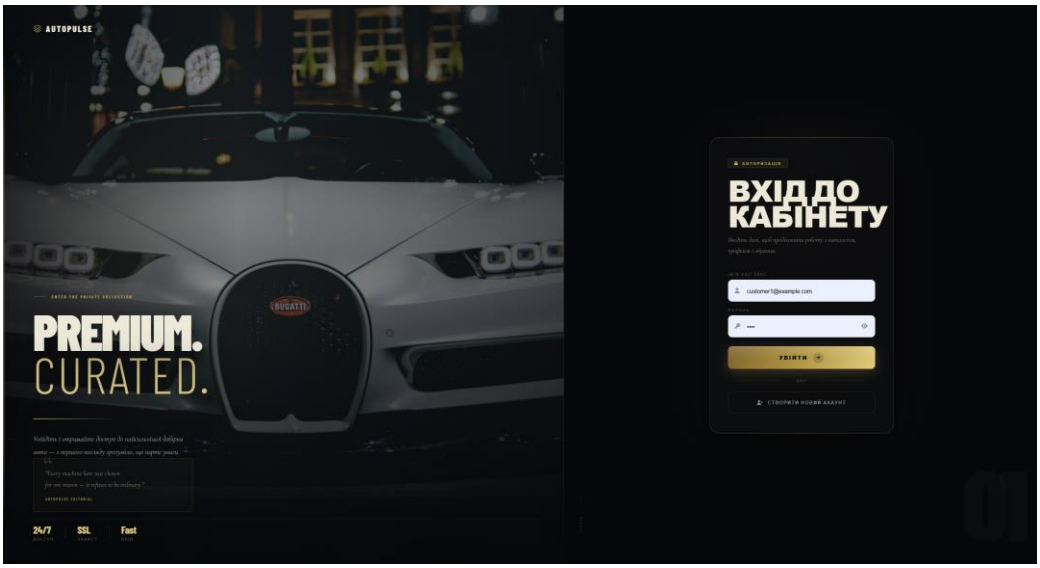

Рисунок 4.12 – Сторінка login.html

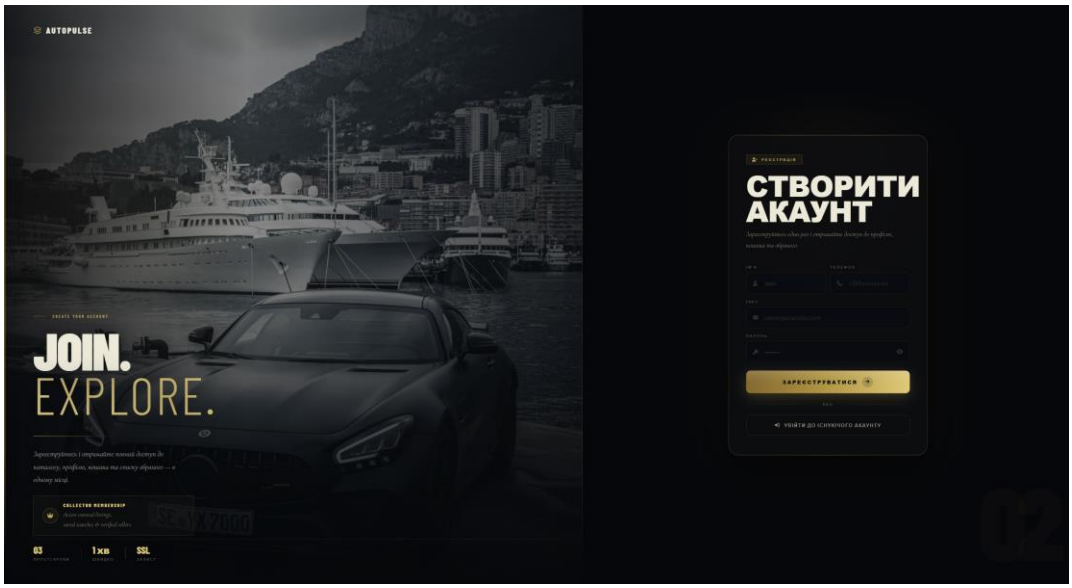

Рисунок 4.13 – Сторінка register.html

Реєстрація та автентифікація забезпечують доступ до персоналізованих можливостей системи. Далі розглянуто сторінку детального перегляду транспортного засобу.

Картка транспортного засобу та подання заявки

Сторінка details.html відображає повну інформацію про транспортний засіб: фотогалерею, технічні характеристики, опис комплектації та історію обслуговування (рис. 4.14). Авторизований користувач може подати заявку через спеціальну форму, вказавши коментар до запиту. Після надсилання заявка обробляється без перезавантаження сторінки. Для неавторизованих відвідувачів відображається пропозиція увійти до системи. Також доступна функція додавання транспортного засобу до списку обраних (Favorites).

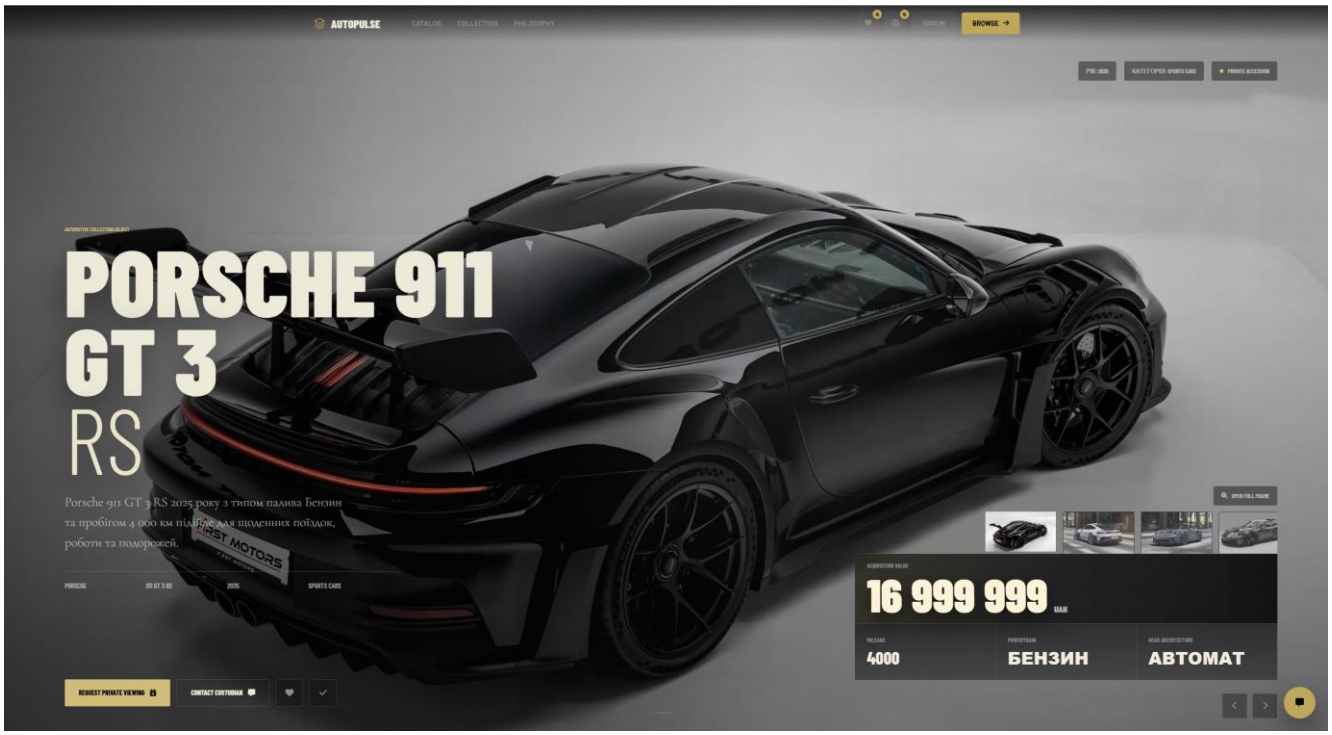

Рисунок 4.14 – Сторінка details.html

Картка транспортного засобу містить повний набір даних, необхідних для прийняття рішення щодо придбання. Наступним розділом описано особистий кабінет користувача.

Особистий профіль

Сторінка profile.html доступна лише авторизованим користувачам (рис. 4.15). Після завантаження відображаються особисті дані користувача, його роль, дата реєстрації та перелік поданих заявок із поточними статусам.

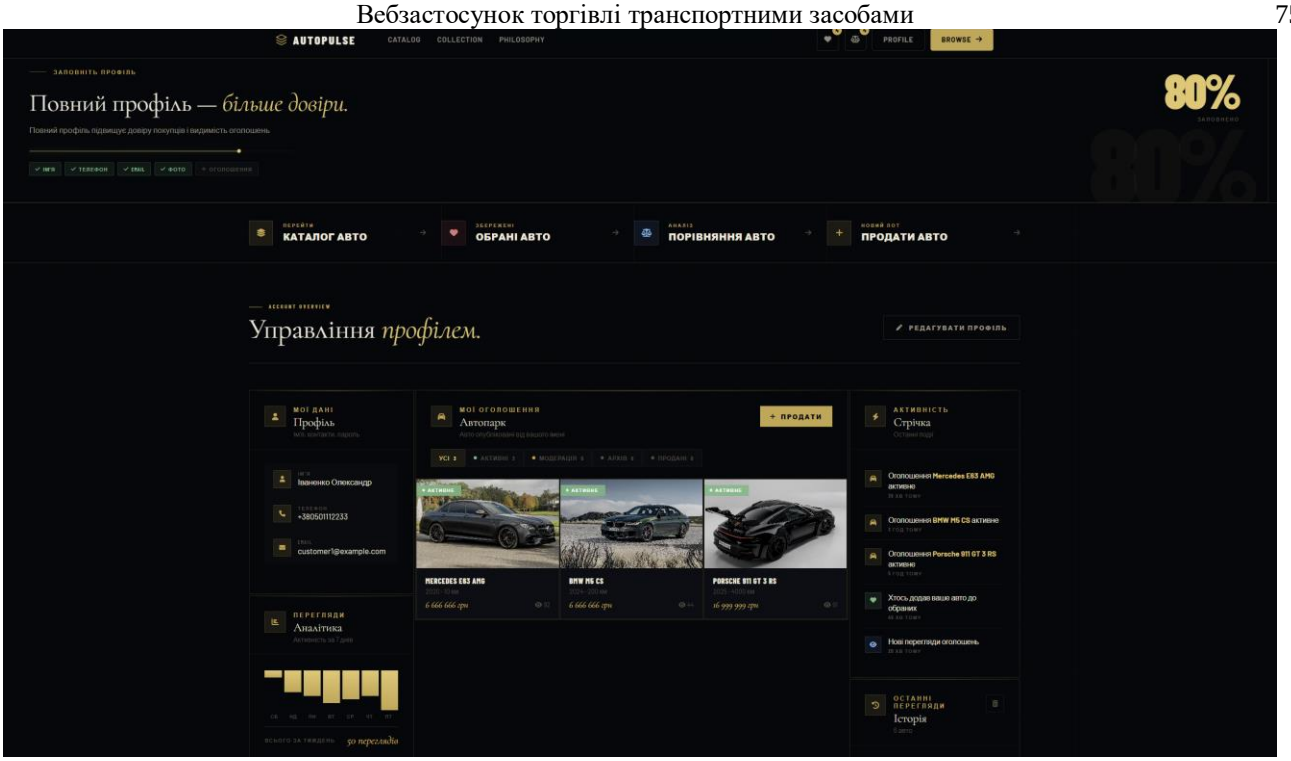

Рисунок 4.15 – Сторінка profile.html

Для користувачів із роллю продавця реалізовано сторінки sell.html і seller.html. Перша призначена для створення оголошень про продаж транспортних засобів, друга відображає статистику переглядів, додавань до обраних та отриманих заявок.

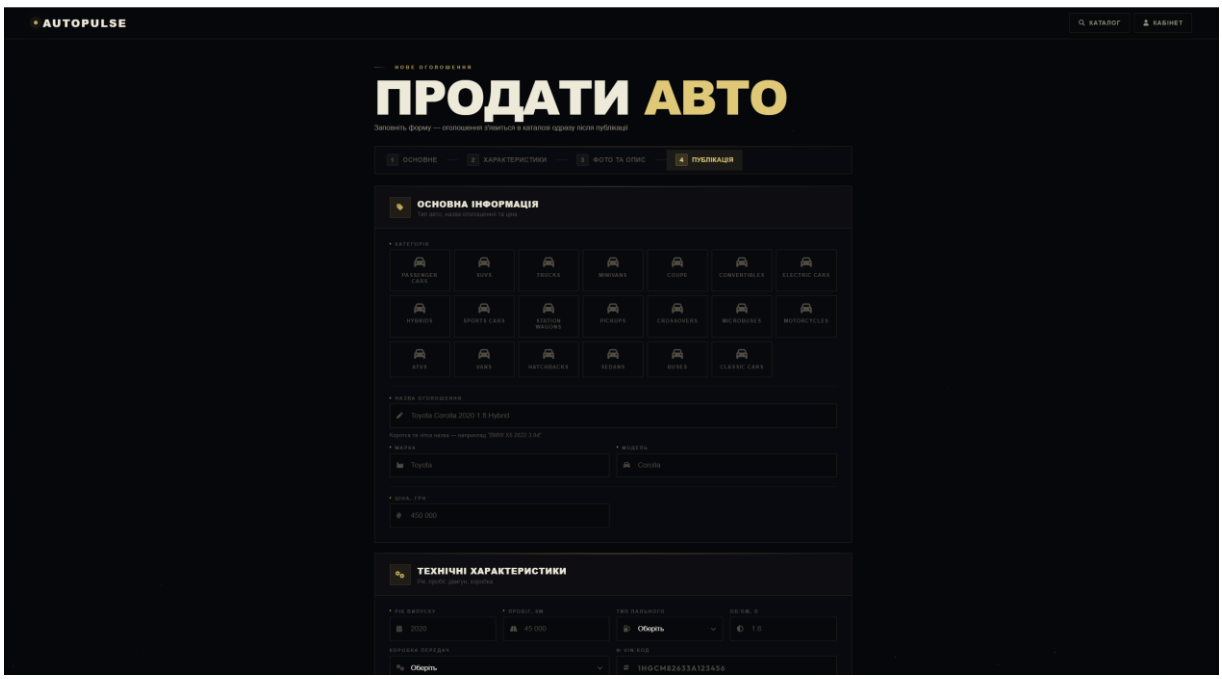

Рисунок 4.16 – Сторінка sell.html

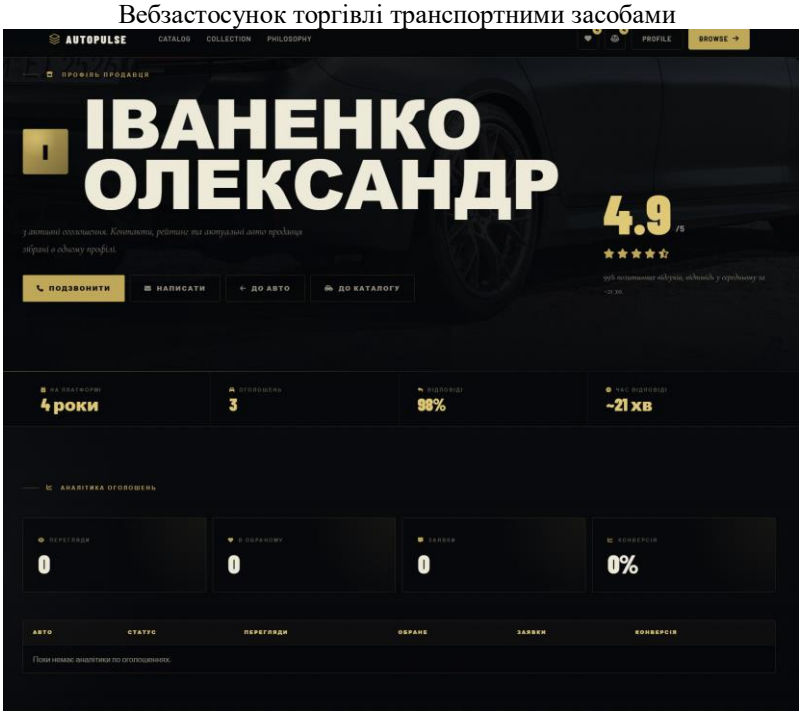

Рисунок 4.17 – Сторінка seller.html

Розглянуті сторінки формують повний набір користувацьких інтерфейсів вебзастосунку AUTOPULSE. Далі наведено результати функціонального тестування системи.

4.4 Тестування вебзастосунку

Функціональну перевірку REST API виконано шляхом інтеграційного тестування через ручне надсилання HTTP-запитів до API-ендпоінтів із подальшим аналізом відповідей сервера. Для кожного тестового сценарію оцінювали три критерії: відповідність HTTP-коду статусу вимогам REST-семантики, структурну коректність JSON-відповіді та відповідність фактичної поведінки системи вимогам специфікації, наведеним у розділі 2. Такий підхід дозволив перевірити не лише правильність обробки запитів, а й коректність реалізації бізнес-логіки вебзастосунку.

Таблиця 4.1 – Тест–кейси вебзастосунку AUTOPULSE

№	Назва тесту	Передумови	Кроки тестування
1	Реєстрація: валідні дані	Email не зареєстровано	POST /api/customers/register з {name, email, password}

Кінець Таблиці 4.1 – Тест–кейси вебзастосунку AUTOPULSE

2	Реєстрація: дублікат email	Акаунт із таким email вже є	POST /api/customers/register з існуючим email
3	Вхід: коректні дані	Акаунт зареєстровано	POST /api/customers/login з {email, password}
4	Вхід: невірний пароль	Акаунт зареєстровано	POST /api/customers/login з неправильним паролем
5	Отримання каталогу	БД містить ТЗ	GET /api/vehicles
6	Фільтрація каталогу	БД містить ТЗ	GET /api/vehicles?make=Porsche&yearFrom=2020
7	Перегляд картки ТЗ	ТЗ із Code=1 є в БД	GET /api/vehicles/1
8	Додавання ТЗ	Валідні дані автомобіля та існуюча категорія	POST /api/Vehicles або POST /api/Vehicles/add з AddVehicleDTO
9	Доступ до захищеного ресурсу без автентифікації	Користувач не виконав вхід	GET /api/Customers/me
10	Фіксація додавання ТЗ до обраного	Існує ТЗ із Code=1	POST /api/Vehicles/1/favorite
11	Отримання статистики продавця	Існує користувач-продавець та його оголошення	GET /api/Sellers/{sellerId}/dashboard

Результати тестування підтвердили успішне проходження всіх 11 тест-кейсів. Під час перевірки позитивних сценаріїв система коректно виконувала реєстрацію користувачів, автентифікацію, фільтрацію каталогу транспортних засобів і подання заявок. Окрему увагу приділено перевірці ситуацій, що можуть спричиняти помилки або порушення логіки роботи застосунку. Зокрема, повторна

реєстрація з уже зареєстрованою адресою електронної пошти повертала код помилки 400, звернення покупця щодо наявного транспортного засобу успішно зберігалося в системі з формуванням підтвердження про виконання операції, а спроби доступу до захищених ресурсів без попередньої автентифікації завершувалися помилкою авторизації. Крім того, система правильно застосовувала рольові обмеження та надавала доступ лише до тих функцій, які відповідають правам конкретного користувача. Отримані результати свідчать про коректну реалізацію механізмів контролю доступу та відповідність їх вимогам, визначеним на етапі проєктування.

Додатково перевірено роботу механізмів Cookie Authentication та Claims-based Authorization. Проведені випробування підтвердили, що модель розмежування доступу функціонує відповідно до закладеної логіки. Відомості про роль користувача зберігаються у claims та використовуються під час визначення доступних функцій і операцій. Окремий тест стосувався механізму приховування транспортного засобу за допомогою параметрів IsVisible = false та Status = archived. Після виконання запиту POST /api/Vehicles/{code}/hide відповідний автомобіль переставав відображатися у публічному каталозі, який формується через GET /api/Vehicles, проте сам запис залишався доступним у базі даних. Це підтвердило правильність реалізації логіки архівування, за якої дані не видаляються фізично, а лише виключаються з переліку активних оголошень.

Висновки до розділу 4

У четвертому розділі реалізовано вебзастосунок AUTOPULSE на платформі ASP.NET Core (.NET 9) та досліджено основні складові його програмної архітектури. Особливу увагу приділено структурі проєкту, побудованій на принципі розподілу відповідальності між клієнтською частиною, контролерами, сервісним шаром і компонентами передачі даних. Обрана архітектурна організація спрощує підтримку програмного забезпечення, забезпечує можливість його масштабування та створює передумови для подальшого розвитку функціоналу.

Окремо розглянуто реалізацію серверної частини системи. Проаналізовано механізми автентифікації та авторизації користувачів, процес формування ClaimsPrincipal і cookie-сесій, засоби фільтрації та пошуку транспортних засобів, роботу API-контролерів і взаємодію з базою даних через Entity Framework Core. Практична реалізація цих компонентів підтвердила ефективність використання інструментів платформи .NET для безпечної обробки облікових даних, контролю прав доступу та організації роботи з реляційною базою даних.

Значну частину розділу присвячено користувацькому інтерфейсу вебзастосунку. Детально охарактеризовано головну сторінку, каталог транспортних засобів, сторінки реєстрації та авторизації, картку автомобіля, особистий профіль користувача та функціональні можливості продавця. Для створення інтерфейсу використано HTML, CSS, JavaScript і Bootstrap. Поєднання зазначених технологій забезпечило адаптивність застосунку та його коректне відображення на різних типах пристроїв без втрати функціональності.

Завершальним етапом роботи стало функціональне тестування REST API шляхом виконання HTTP-запитів до ключових ендпоінтів системи. Отримані результати підтвердили стабільність основних бізнес-процесів, коректність реалізації механізмів автентифікації та авторизації, правильність роботи каталогу транспортних засобів, а також здатність системи належним чином обробляти помилкові та граничні сценарії використання. Усі тестові випадки завершилися успішно. Це дає підстави стверджувати, що розроблене програмне забезпечення відповідає визначеним функціональним вимогам і готове до подальшої експлуатації.

ВИСНОВКИ

У межах кваліфікаційної роботи розроблено вебзастосунок для торгівлі транспортними засобами, який забезпечує перегляд каталогу автомобілів, пошук за параметрами, подання заявок на придбання та керування інформаційним наповненням системи. Реалізований функціонал охоплює основні сценарії взаємодії між покупцями, продавцями та адміністраторами. Отриманий результат відповідає поставленій меті та завданням дослідження.

Під час роботи проаналізовано предметну область електронної комерції у сфері продажу транспортних засобів, досліджено сучасні тенденції розвитку онлайн-ринку автомобілів та оцінено функціональні можливості платформ AUTO.RIA, OLX Авто, RST, AutoTrader і Cars.com. Проведений аналіз дозволив виявити характерні недоліки існуючих рішень, зокрема перевантаженість інтерфейсу, складну навігацію, недостатню адаптацію до мобільних пристроїв та обмежені можливості для автосалонів. Виявлені особливості враховано під час проєктування власного програмного рішення.

Сформовано функціональні та нефункціональні вимоги до системи, спроєктовано архітектуру вебзастосунку та структуру бази даних. Для реалізації серверної частини використано ASP.NET Core MVC і REST API, що забезпечують розмежування бізнес-логіки та ефективну взаємодію між компонентами системи. Зберігання даних організовано за допомогою Microsoft SQL Server та Entity Framework Core, які підтримують цілісність даних і спрощують виконання CRUD-операцій.

У процесі розроблення клієнтської частини створено адаптивний користувацький інтерфейс із використанням HTML5, CSS3, Bootstrap та JavaScript. Реалізовано механізми реєстрації та авторизації користувачів на основі Cookie Authentication, Claims-based Authorization і безпечного хешування паролів засобами ASP.NET Core Identity. Такий підхід забезпечує належний рівень захисту облікових даних і контроль доступу до функцій системи. Також створено адміністративну

панель для керування каталогом транспортних засобів та обробки заявок користувачів.

Результати тестування підтвердили коректну роботу основних функціональних модулів вебзастосунку. Під час перевірки не виявлено порушень у взаємодії клієнтської та серверної частин системи. Механізми автентифікації, фільтрації транспортних засобів та обробки заявок функціонували відповідно до визначених вимог.

Практична цінність розробленого рішення полягає у можливості його використання в діяльності автосалонів, дилерських компаній та онлайн-платформ із продажу транспортних засобів. Систему можна застосовувати як готовий програмний продукт або як основу для подальшого розширення функціональності.

Порівняння з наявними аналогами показало, що розроблений вебзастосунок орієнтований на швидкий пошук транспортних засобів і зручну взаємодію користувача з каталогом. На відміну від багатьох популярних платформ, система не містить надлишкових рекламних елементів та підтримує централізоване керування каталогом через адміністративну панель.

Подальший розвиток системи доцільно спрямувати на створення мобільного застосунку, інтеграцію платіжних сервісів, впровадження рекомендаційної системи на основі поведінки користувачів, підтримку онлайн-бронювання транспортних засобів, а також розширення засобів аналітики та персоналізації контенту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Іванова О. В., Коваленко В. М. Порівняльний аналіз СКБД для систем електронної комерції. Вісник Національного технічного університету «ХПІ». 2023. № 1. С. 45–52.

2. AUTO RIA: вебсайт. URL: https://auto.ria.com/uk/ Accessed: 23.05.2026.

3. AutoTrader: вебсайт. URL: https://www.autotrader.co.uk/ Accessed: 23.05.2026.

4. Beaulieu A. Learning SQL. Sebastopol : O'Reilly Media, 2020. 338 p.

5. Ben–Gan I. T–SQL Fundamentals. Redmond : Microsoft Press, 2016. 464 p.

6. Cars.ua: вебсайт. URL: https://cars.ua/ Accessed: 23.05.2026.

7. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation, and Management. Harlow : Pearson, 2015. 1440 p.

8. Delaney K., Soukup R. Inside Microsoft SQL Server 2000. Redmond : Microsoft Press, 2001. 1056 p.

9. Elmasri R., Navathe S. Fundamentals of Database Systems. Boston : Pearson, 2016. 1272 p.

10. Fielding R. T. Architectural Styles and the Design of Network–Based Software Architectures. Irvine : University of California, 2000. 162 p.

11. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison–Wesley, 2002. 533 p.

12. Freeman A. Pro ASP.NET Core MVC. New York : Apress, 2016. 832 p.

13. Freeman A., Sanderson S. ASP.NET MVC 5 Framework. New York : Apress, 2014. 600 p.

14. Galloway J., Haack P., Wilson B. et al. Professional ASP.NET MVC 4. Indianapolis : John Wiley & Sons, 2012. 624 p.

15. Hassan A., Ahmed M. RESTful API Design Patterns for E–Commerce Systems. IEEE Access. 2021. Vol. 9. P. 78432–78449. DOI: 10.1109/ACCESS.2021.3084521.

16. Kleppmann M. Designing Data–Intensive Applications. Sebastopol : O'Reilly Media, 2017. 616 p.

17. Krug S. Don't Make Me Think: A Common Sense Approach to Web Usability. Berkeley : New Riders, 2014. 216 p.

18. Kumar S., Sharma R., Patel A. MVC Architecture in Modern Web Applications: A Comparative Study. International Journal of Web Engineering and Technology. 2022. Vol. 17, No. 2. P. 145–167.

19. Masse M. REST API Design Rulebook. Sebastopol : O'Reilly Media, 2011. 114 p.

20. OLX: вебсайт. URL: https://www.olx.ua/uk/transport/legkovye–avtomobili/ Accessed: 23.05.2026.

21. Nguyen T., Tran H., Le V. Comparative Analysis of ORM Frameworks for .NET Platform. Procedia Computer Science. 2022. Vol. 204. P. 512–521.

22. Patel R., Williams J. Performance Benchmarking of Server–Side Web Frameworks for REST API Development. Journal of Systems and Software. 2023. Vol. 196. P. 111–128.

23. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. New York : McGraw–Hill, 2015. 970 p.

24. Richardson L., Amundsen M. RESTful Web APIs. Sebastopol : O'Reilly Media, 2013. 408 p.

25. RST: вебсайт. URL: https://rst.ua/ukr/ Accessed: 23.05.2026.

26. Silva M., Costa R. CSS Frameworks for Responsive Web Design: A Systematic Review. International Journal of Human–Computer Studies. 2022. Vol. 162. P. 102–118.

27. Sommerville I. Software Engineering. Boston : Pearson, 2016. 816 p.