Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ПЛАТФОРМИ ДЛЯ РОЗМІЩЕННЯ ОГОЛОШЕНЬ ПРО ПРОДАЖ АВТОМОБІЛІВ З ВИКОРИСТАННЯМ ANGULAR, ASP.NET, MVC ТА ENTITY FRAMEWORK

DEVELOPMENT OF A PLATFORM FOR POSTING CAR SALES ADVERTISEMENTS USING ANGULAR, ASP.NET, MVC AND ENTITY FRAMEWORK

спеціальність 121 «Інженерія програмного забезпечення»
освітня програма «Інженерія програмного забезпечення»

Виконала: здобувач вищої освіти
групи ІПЗз-41
Баканіна Анна Олександрівна

Керівник:
Повстяна Юлія Славомирівна

Кваліфікаційну роботу
допущено до захисту
«10» 06 2025 р.
Гарант освітньої програми:
к.т.н., доцент
Ліщина Наталія Миколаївна

_____

Луцьк – 2025 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *інженерії програмного забезпечення*
Ступінь вищої освіти *бакалавр*
Галузь знань: *12 «Інформаційні технології»*
Спеціальність: *121 «Інженерія програмного забезпечення»*
Освітня програма: «*Інженерія програмного забезпечення»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«20» 12 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ВИЩОЇ ОСВІТИ

Баканіній Анні Олександрівні
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *«Розробка платформи для розміщення оголошень про продаж автомобілів з використанням Angular, ASP.NET, MVC та Entity Framework».*
Керівник роботи: *к.т.н., доцент Повстяна Ю. С.*
затверджені наказом закладу вищої освіти від *« 19 » грудня 2024 р.* № *474/01-02*
2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *« 10 » червня 2025 р.*
3. Вихідні дані до роботи *редактор коду Visual Studio Code, середовище розробки для C# JetBrains Rider, додаток Postman для перевірки HTTP запитів та для створення API документації, SQL Server та база даних Azure SQL Database, система контролю версій Git.*
4. Зміст пояснювальної записки (перелік питань, що потрібно розробити): *у межах кваліфікаційної роботи передбачено виконання переліку завдань, спрямованих на реалізацію веб-платформи для продажу автомобілів. Зокрема, аналіз предметної області та формування вимог до програмного забезпечення, розробка UML-діаграми. Створення користувацького інтерфейсу за допомогою Angular, HTML і CSS, а також реалізування API з підтримкою CRUD-операцій. Реалізація бази даних на основі SQL Server, впровадження механізмів аутентифікації та авторизації.*
5. Перелік графічного (ілюстративного) матеріалу: *кількість рисунків - 23, кількість таблиць - 1, додатки - 2.*

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз предметної області	*Повстяна Ю. С.*		
Специфікація вимог до розробленої системи	*Повстяна Ю. С.*		
Розробка об'єкта проєктування	*Повстяна Ю. С.*		
Нормоконтроль	*Вознюк А. В.*		
Гарант ОП	*Ліщина Н. М.*		
Показник запозичень тексту	3,3 %		
Академічна доброчесність	*Повстяна Ю. С.*		

7. Дата видачі завдання: *« 20 » грудня 2024 р.*

№ з п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1.	Огляд літературних джерел по темі кваліфікаційної роботи бакалавра	до 04.02.2025 р.	вик
2.	Аналіз проблеми розробки та впровадження об'єкту проєктування	до 01.03.2025 р.	вик
3.	Обгрунтування вибору шляхів, технологій і засобів вирішення поставленого завдання	до 15.03.2025 р.	вик
4.	Розробка функціонально-структурної схеми роботи об'єкта проєктування та проєктування бази даних	до 29.03.2025 р.	вик
5.	Практична реалізація об'єкта проєктування та розробка бази даних	до 26.04.2025 р.	вик
6.	Тестування та налагодження об'єкта проєктування	до 03.05.2025 р.	вик
7.	Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру	до 10.06.2025 р.	вик

Здобувач вищої освіти _____ Анна БАКАНІНА

Керівник кваліфікаційної роботи _____ Юлія ПОВСТЯНА

АНОТАЦІЯ

Баканіна А. О. Розробка платформи для розміщення оголошень про продаж автомобілів з використанням Angular, ASP.NET, MVC та Entity Framework. Рукопис. Кваліфікаційна робота бакалавра ОП «Інженерія програмного забезпечення» спеціальності «Інженерія програмного забезпечення». Луцький національний технічний університет. Луцьк, 2025.

Кваліфікаційна робота бакалавра складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків.

У першому розділі здійснено аналіз предметної області, досліджено сучасні веб-технології для створення платформ з оголошеннями про продаж автомобілів, а також сформульовано завдання кваліфікаційної роботи. В другому розділі розроблено архітектуру програмного забезпечення, визначено технічні вимоги, створено UML-діаграму. У третьому розділі реалізовано функціонал платформи: спроєктовано базу даних, описано реалізацію користувацького інтерфейсу з використанням Angular і серверної частини з використанням ASP.NET MVC та Entity Framework, реалізовано фільтрацію, сортування та перегляд оголошень, а також механізми аутентифікації, авторизації. У висновках узагальнено результати виконаної роботи, підтверджено досягнення поставлених завдань та наведено рекомендації щодо подальшого вдосконалення платформи.

Ключові слова: веб-платформа, Angular, ASP.NET MVC, Entity Framework, продаж автомобілів, електронна комерція, фільтрація, сортування, безпека, аутентифікація, оголошення.

ABSTRACT

Bakanina A. Development of a Platform for Posting Car Sales Advertisements Using Angular, ASP.NET, MVC and Entity Framework. Manuscript. Qualification work of the bachelor's program "Software Engineering" in the specialty "Software engineering". Lutsk National Technical University. Lutsk, 2025.

The bachelor's qualification thesis consists of an introduction, three chapters, conclusions, a list of references, and appendices.

The first chapter presents an analysis of the subject area, explores modern web technologies for creating platforms for posting car sale advertisements, and formulates the objectives of the qualification thesis. The second chapter develops the software architecture, defines the technical requirements, and creates a UML diagram. The third chapter implements the functionality of the platform: the database is designed, the implementation of the user interface using Angular and the server side using ASP.NET MVC and Entity Framework is described, filtering, sorting, and viewing of advertisements are implemented, as well as authentication and authorization mechanisms. The conclusions summarize the results of the completed work, confirm the achievement of the set objectives, and provide recommendations for further improvement of the platform.

Keywords: web platform, Angular, ASP.NET MVC, Entity Framework, car sales, e-commerce, filtering, sorting, security, authentication, advertisements.

ЗМІСТ

ВСТУП

Актуальність теми. Автомобільний ринок є обширною галуззю економіки багатьох країн, яка активно розвивається під впливом цифрових технологій. Онлайн-платформи для розміщення оголошень є зручним інструментом для купівлі та продажу багатьох речей і зокрема транспортних засобів. Саме через це вони набули популярності. Однак, існуючі сервіси не завжди пропонують зручний, привабливий дизайн, або не забезпечують необхідний рівень гнучкості, ефективності пошуку. Це створює потребу у розробці нових технологічних рішень, які враховують сучасні тенденції та вимоги ринку.

Світові тенденції розвитку програмного забезпечення для електронної комерції свідчать про зростання популярності веб-додатків, побудованих на основі передових технологій. Все більше компаній обирають сучасні фронтенд-фреймворки, такі як Angular, React або Vue.js, що забезпечують високу продуктивність, модульність та зручність у розробці динамічних інтерфейсів. З точки зору серверної логіки, популярністю користуються такі технології, як ASP.NET MVC, Node.js та Django, які забезпечують масштабованість, безпеку та можливість гнучкої інтеграції з іншими сервісами. Використання GraphQL та REST API дозволяє створювати ефективні та гнучкі механізми взаємодії між фронтендом та бекендом. Для роботи з базами даних активно використовуються ORM-фреймворки, такі як Entity Framework, Sequelize або SQLAlchemy, які спрощують маніпулювання даними та підвищують продуктивність розробки. Крім того, NoSQL рішення, наприклад, MongoDB або Firebase, набувають все більшої популярності завдяки своїй гнучкості та швидкості у роботі з великими обсягами даних.

Актуальність даної роботи зумовлена необхідністю створення ефективної, зручної платформи для розміщення оголошень про продаж автомобілів, яка буде:

- ефективна і легка у використанні;
- відповідатиме сучасним технологічним вимогам;

– відповідатиме потребам як кінцевого користувача так і адміністратора, або продавця, які будуть користуватися даною платформою.

Основною підставою для виконання цієї роботи є відсутність у наявних рішеннях достатньої інтеграції між сучасними веб-технологіями, що дозволило б значно покращити досвід користувачів та розширити функціональні можливості системи.

Метою кваліфікаційної роботи бакалавра є розробка веб-платформи для розміщення оголошень про продаж автомобілів, що базується на технологіях Angular, ASP.NET MVC та Entity Framework. Очікується, що ця платформа забезпечить простий і швидкий доступ до оголошень, можливість фільтрації та сортування за різними критеріями, перегляд графічних звітів.

Об’єктом кваліфікаційної роботи бакалавра є процес розробки веб-платформи для розміщення оголошень про продаж автомобілів, що базується на сучасних веб-технологіях.

Предметом кваліфікаційної роботи бакалавра є програмні та архітектурні рішення, що використовуються для створення веб-платформи, включаючи застосування Angular для фронтенду, ASP.NET MVC для серверної логіки, Entity Framework для роботи з базою даних, а також механізми забезпечення безпеки, фільтрації, сортування та інтеграції з зовнішніми сервісами.

Завданням на кваліфікаційну роботу є реалізація низки завдань, спрямованих на реалізацію функціональних і технічних аспектів розробки, а саме:

– проаналізувати та дослідити галузь розробки веб-платформ для продажу автомобілів;

– розробити вимоги до програмного забезпечення;

– розробити UML діаграму;

– розробити UI-інтерфейс додатку та реалізувати його за допомогою фреймворка Angular та HTML, CSS;

– розробити API та впровадити CRUD операцій;

– спроєктувати і створити бази даних за допомогою SQL Server;

– реалізувати логіку для запобігання CSRF-атакам;

– додати методи обробки та логування помилок;

– реалізувати методи аутентифікації та авторизації;

– додати функції сортування, фільтрування та пагінації для оголошень також зробити можливим завантаження фото та відображення звітів для адміністратора.

Галузь застосування даної платформи охоплює здебільшого приватних осіб, які продають і купують автомобілі, та компанії, що спеціалізуються на автомобільному бізнесі, включаючи дилерів та автосалони також можуть нею користуватися.

Апробацію результатів роботи здійснено шляхом участі в міжнародній науково-практичній конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)», тези доповіді наведені у додатку А.

РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ ВЕБ-ПЛАТФОРМ ДЛЯ РОЗМІЩЕННЯ ОГОЛОШЕНЬ ПРО ПРОДАЖ АВТОМОБІЛІВ І ПОСТАНОВКА ЗАВДАНЬ НА КВАЛІФІКАЦІЙНУ РОБОТУ

1.1 Аналіз сучасного стану проблеми

Розвиток цифрових технологій суттєво вплинув на процес купівлі та продажу транспортних засобів на автомобільному ринку, зробивши цей процес більш зручним, швидким і гнучким. Автовиробники почали активно інвестувати у створення власних веб-платформ, де потенційні покупці можуть не лише переглядати модельний ряд, а й налаштовувати автомобіль під свої потреби завдяки вбудованим конфігураторам. Це дає змогу користувачам отримати унікальний досвід взаємодії з брендом ще до покупки та детально ознайомитися з всіма доступними опціями, спростити процес кастомізації тієї чи іншої моделі авто, додати до цього процесу елементи гейміфікації.

Паралельно з цим продовжували свій розвиток й онлайн-платформи для оголошень, що надають зручні інструменти як для продавців, так і для покупців. Вони дозволяють користувачам швидко та ефективно розміщувати оголошення, взаємодіяти з потенційними клієнтами, використовувати вбудовані механізми аналітики. Багато з них в тому числі пропонують можливість розміщення оголошень про продаж автомобілів, що супроводжується додатковими функціями, такими як:

- історія експлуатації транспортного засобу;
- оцінювання вартості на основі ринкових даних;
- перевірка VIN коду;
- калькулятор витрат, наприклад, для розрахунку вартості розмитнення або приблизного розрахунок витрат на пальне.

Тобто онлайн-платформи для оголошень фактично є аналогом інтернет-магазинів, де кожен користувач може створювати власні пропозиції, взаємодіяти з потенційними покупцями та використовувати різні функції для

спрощення продажу. Розробка ефективної та зручної платформи вимагає розуміння основних принципів електронної комерції, зокрема організації каталогу товарів, механізмів пошуку та фільтрації, а також забезпечення безпеки та довіри між учасниками угод [1]. Саме ці аспекти є ключовими для створення конкурентоспроможного продукту, що відповідатиме сучасним очікуванням користувачів.

Розглянемо детальніше принципи електронної комерції наведені вище.

Каталог товарів – це основний елемент платформ електронної комерції, який дозволяє покупцям швидко знайти потрібний товар або послугу. Основні принципи його організації:

– кожен товар має містити детальний опис, характеристики, фото та ціну;

– використання ключових тегів спрощує пошук і робить взаємозв’язки між товарами зрозумілішими;

– візуальне оформлення каталогу має бути інтуїтивно зрозумілим і зручним для користувача.

Механізми пошуку та фільтрації – одні з найважливіших функціональних елементів онлайн-платформ, які дозволяють користувачам швидко знаходити необхідну інформацію. Критерії ефективних методів пошуку та фільтрації:

– рядок пошуку має підтримувати автозаповнення, підказки та виправлення помилок;

– користувачі повинні мати змогу відфільтрувати товари за ціною, брендом, наявністю та іншими характеристиками;

– має бути можливість сортувати товари за популярністю, ціною, рейтингом тощо.

Забезпечення безпеки є критично важливим аспектом, що впливає на довіру до сервісу та його репутацію. Основні механізми які мають бути впроваджені:

– впровадження технологічних рішень, наприклад, використання шаблону проєктування STP (Synchronizer Token Pattern), що забезпечує захист від CSRF атак [2];

– реєстрація через емейл-адресу або номер телефону для можливості проведення аутентифікації;

– перевірка адміністраторами сумнівних оголошень перед публікацією;

– відображення рекомендацій щодо безпечної купівлі/продажу;

– використання безпечних платіжних систем.

Огляд існуючих рішень на ринку є важливим етапом у розумінні сучасних тенденцій електронної комерції в сфері продажу автомобілів. В Україні є різні онлайн-платформи, які надають користувачам можливість купівлі та продажу транспортних засобів. Такі платформи як Auto.Ria, RST.ua є найпопулярнішими з них.

Auto.Ria [3] (рис. 1.1) – це одна з найбільших онлайн-платформ для купівлі та продажу як нових так і б/у транспортних засобів в Україні. Працює з 2002 року та є лідером серед авто-класифайдів. Сайт пропонує досить зручний інтерфейс, хоча головна сторінка перевантажена рекламними банерами. Однією з ключових особливостей платформи є наявність перевірених оголошень, що додає прозорості та безпеки процесу купівлі. Додатково користувачі можуть переглядати відгуки та рейтинги як продавців, так і автомобілів, що допомагає приймати більш зважені рішення.

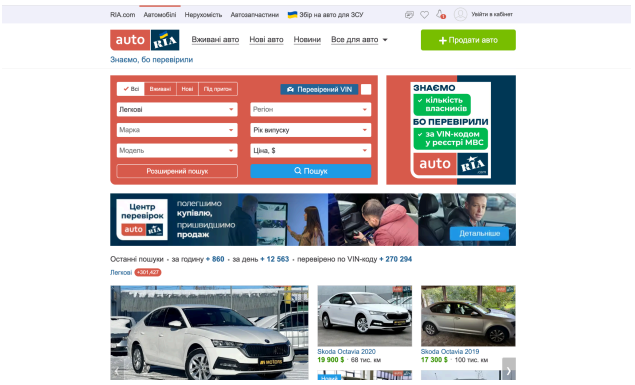

Рисунок 1.1 – Інтерфейс Auto.Ria [3]

RST.ua [4] (рис. 1.2) – це сайт для купівлі та продажу автомобілів в Україні, який є гарною альтернативою іншим онлайн-платформам. Платформа пропонує користувачам широкий вибір нових і вживаних автомобілів, а також корисні сервіси для аналізу ринку та перевірки авто. Сайт має простий і зручний інтерфейс за рахунок чого на ньому дуже легко орієнтуватися. При переході на платформу користувач одразу опиняється на сторінці перегляду наявних оголошень з можливістю їх сортування та фільтрування. Особливістю платформи є можливість оформлення підписки, що дає користувачеві доступ до нових оголошень за годину до того як вони будуть відображатися для всіх інших користувачів і також доступ до сервісу для автовикупу.

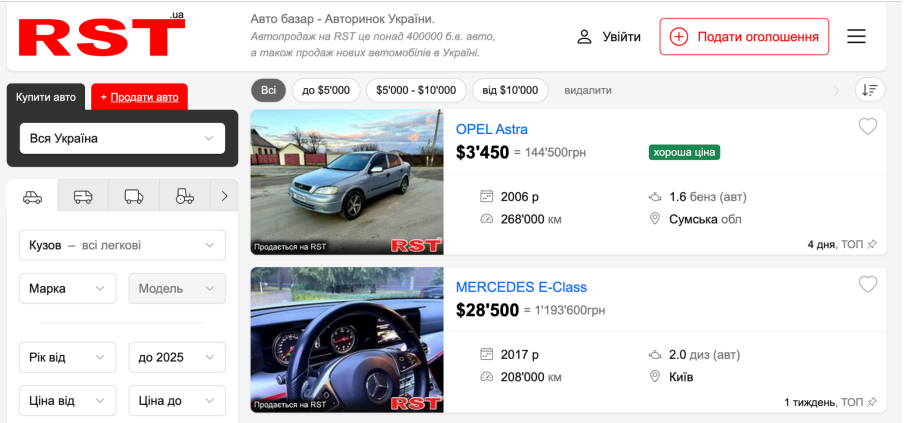

Рисунок 1.2 – Інтерфейс RST.ua [4]

Щодо європейського ринку, то там є як популярні загальноєвропейські платформи так і ті, які поширені тільки в певних країнах. Кожна з цих платформ має свої особливості, зокрема відмінності в інтерфейсі, способах перевірки авто та можливостях для покупців і продавців.

AutoScout24.com [5] (рис. 1.3) – є загальноєвропейською платформою для купівлі та продажу нових і вживаних автомобілів. Сайт охоплює більшість країн Європи, таких як Італія, Нідерланди, Бельгія, Іспанія, Франція, та надає доступ до великої бази оголошень від приватних продавців і автодилерів. Платформа пропонує зручний пошук за різними критеріями, включаючи марку, модель, рік випуску, пробіг та ціну. Крім того, AutoScout24 надає інструменти для перевірки

технічного стану авто, фінансування та страхування, що робить процес купівлі-продажу більш безпечним та зручним для користувачів.

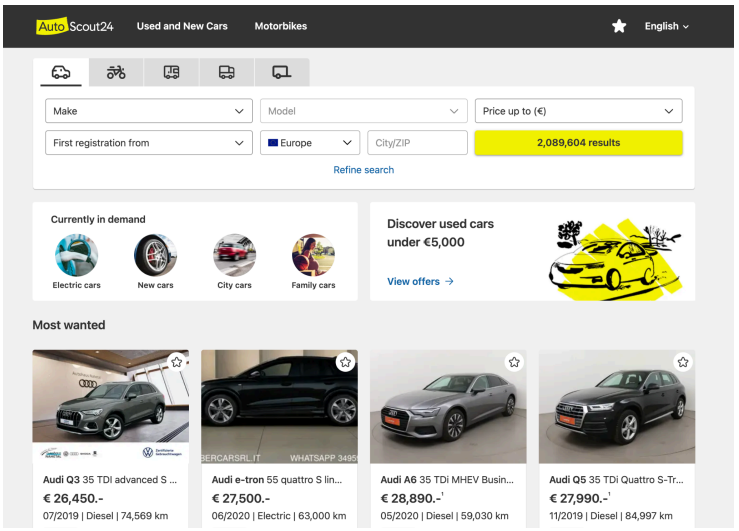

Рисунок 1.3 – Інтерфейс AutoScout24.com [5]

Otomoto.pl [6] (рис. 1.4) – це найбільша онлайн-платформа в Польщі для купівлі та продажу нових і вживаних транспортних засобів. Сайт належить до міжнародної групи OLX Group і є найпопулярнішим авто-класифайдом у Польщі, де щоденно публікуються тисячі оголошень від приватних осіб, автосалонів та дилерів.

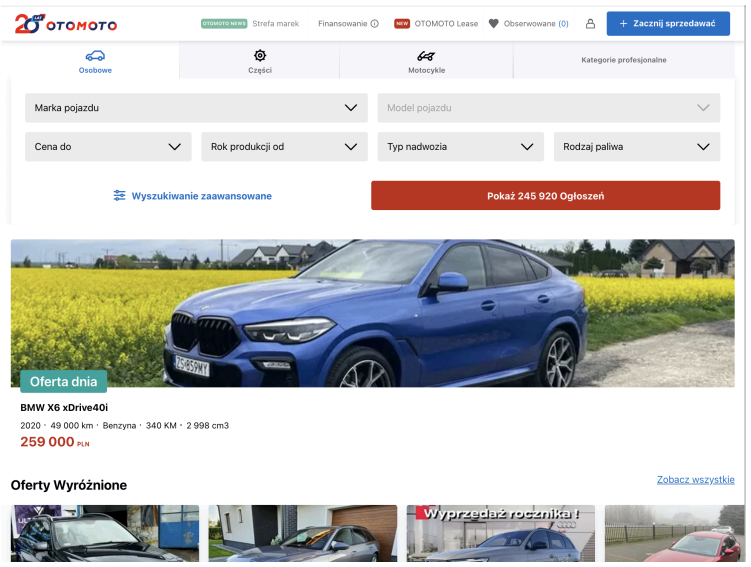

Рисунок 1.4 – Інтерфейс Otomoto.pl [6]

AutoTrader [7] (рис. 1.5) – це найбільший онлайн-майданчик для купівлі та продажу нових і вживаних автомобілів у Великій Британії. Платформа існує з 1977 року, спочатку як друкований журнал, а потім повністю перейшла в онлайн. AutoTrader є ключовим ресурсом для британських автолюбителів, дилерів та приватних продавців. Також сайт надає корисні сервіси, такі як оцінка вартості автомобіля, перевірка історії ТЗ та фінансові інструменти для кредитування чи страхування.

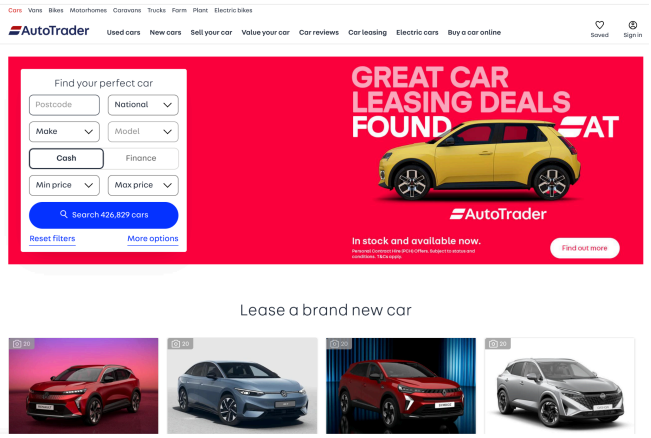

Рисунок 1.5 – Інтерфейс AutoTrader [7]

Le Centrale [8] (рис. 1.6) – це одна з найбільших онлайн-платформ для купівлі та продажу автомобілів у Франції. Вона була заснована в 1969 році як друкований автомобільний каталог, а згодом стала провідним цифровим майданчиком для продажу нових і вживаних транспортних засобів.

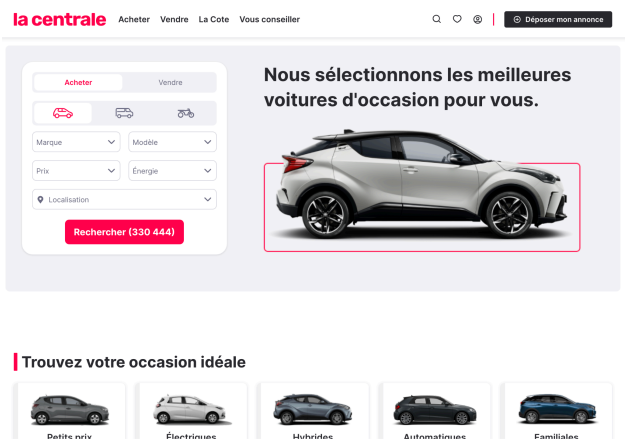

Рисунок 1.6 – Інтерфейс AutoTrader [8]

Наведені вище українські та європейські платформи надають різноманітні можливості для покупців, а саме: пошук за різними параметрами, перевірку технічного стану автомобіля, різні можливості зв’язку з продавцями. Варто зазначити, що європейські додатки часто характеризуються мінімалістичним інтерфейсом, меншою кількістю реклами та використанням кастомізованих шрифтів, що покращує користувацький досвід.

Аналіз області розробки показує, що багато існуючих платформ для купівлі-продажу транспортних засобів мають закритий вихідний код, що обмежує можливості їхньої адаптації під індивідуальні потреби користувачів. Це може включати відсутність можливості додавання специфічних функцій, зміни інтерфейсу або інтеграції з іншими сервісами. Крім того, деякі відкриті рішення не підтримують глибоку інтеграцію з сучасними технологіями, що може негативно впливати на їхню продуктивність та масштабованість.

Огляд сучасних тенденцій у розробці веб-платформ для купівлі-продажу автомобілів свідчить про активне впровадження новітніх технологій для покращення користувацького досвіду та оптимізації роботи з даними. Детальна інформація про технологічний стек платформ, таких як Auto.Ria, RST.ua, AutoScout24, Otomoto.pl, AutoTrader UK та Le Centrale, не є публічно доступною, оскільки більшість компаній не розкривають такі технічні деталі з міркувань безпеки та конкурентоспроможності. Однак, аналізуючи наукові дослідження у сфері веб-розробки, можна визначити загальні тенденції вибору технологій у цій галузі.

Одним із ключових напрямків досліджень є впровадження сучасних фронтенд-фреймворків, зокрема Angular, для створення динамічних та інтерактивних інтерфейсів користувача. У роботі «Comparative Analysis of Angular, React, and Vue.js in Single Page Application Development» (Phani Sekhar Emmanni) автор аналізує переваги та недоліки популярних фронтенд-фреймворків, підкреслюючи ефективність Angular у розробці масштабованих веб-додатків [9]. З точки зору серверної логіки, використання ASP.NET MVC та Entity Framework забезпечує надійну архітектуру та

ефективну роботу з базами даних. У дослідницькій статі «A Comprehensive analysis of architectural patterns in ASP.NET Core WEB application development» (F. Mushica, A. Memeti) розглядаються переваги використання ASP.NET MVC для побудови масштабованих веб-додатків, підкреслюючи багато корисних додаткових можливостей, таких як створення API, полегшення тестування за рахунок використання модульної архітектури MVC [10]. Крім того, додаткове використання ORM-фреймворків, таких як Entity Framework, надає додаткові переваги у вигляді спрощення маніпуляцій з даними та підвищення продуктивності розробки.

Аналіз популярності технологій веб-розробки в Україні та Європі свідчить про певні відмінності у виборі інструментів для фронтенду та бекенду. Комітет Digital Developers Committee, який є частиною Всеукраїнської рекламної коаліції (ВРК) провів експертизу ринку веб-розробки в Україні і зробили висновок, що в Україні широко використовуються JavaScript-фреймворки та PHP, зокрема Laravel (34 %) або Symfony (13 %), а також Java для бекенду [11]. Згідно з дослідженням «The European market potential for web design and web application development services» (CBI, Ministry of Foreign Affairs of the Netherlands), в Європі найбільш популярними технологіями серед веб-розробників є HTML/CSS, JavaScript, а також такі мови програмування для бекенду, як PHP і Java [12]. Серед фреймворків особливо виділяються React, Vue.js та Node.js, що підтверджує поширеність цих технологій в Україні [12]. Водночас, сучасні тенденції веб-розробки вказують на попит на уніфіковані інтерфейси, адаптовані під різні типи пристроїв, що робить продукти більш привабливими для бізнесу та споживачів. Це може пояснювати популярність React та Vue.js у Європі, оскільки ці фреймворки забезпечують гнучкість та адаптивність інтерфейсів.

Таким чином, вибір технологій для веб-розробки в Україні та Європі залежить від специфіки проєктів та вимог ринку, що в свою чергу впливає на популярність тих чи інших інструментів у різних регіонах.

1.2 Постановка завдання на кваліфікаційну роботу бакалавра

В межах виконання кваліфікаційної роботи передбачається досягнення низки завдань, спрямованих на реалізацію функціональних і технічних аспектів розробки. Зокрема, необхідно:

– проаналізувати та дослідити галузь розробки веб-платформ для продажу автомобілів;

– розробити вимоги до програмного забезпечення;

– розробити UML діаграму;

– розробити UI-інтерфейс додатку та реалізувати його за допомогою фреймворка Angular та HTML, CSS;

– розробити API та впровадити CRUD операцій;

– спроєктувати і створити бази даних за допомогою SQL Server;

– реалізувати логіку для запобігання CSRF-атакам;

– додати методи обробки та логування помилок;

– реалізувати методи аутентифікації та авторизації;

– додати функцій сортування, фільтрування та пагінації для оголошень також зробити можливим завантаження фото та відображення звітів для адміністратора.

Висновки до розділу 1

У першому розділі було проведено аналіз предметної області та обґрунтовано актуальність обраної теми. Також здійснено огляд і аналіз сучасних онлайн-платформ для розміщення оголошень про продаж автомобілів, як на українському, так і на європейському ринках.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Аналіз, визначення вимог до розроблюваного програмного забезпечення та проєктування програмного забезпечення

Проаналізувавши наведені вище онлайн-платформи переходимо до формулювання ключових аспектів, які повинні бути враховані при розробці платформи для продажу автомобілів. Важливо звернути увагу на її функціональні та нефункціональні вимоги, адже ефективність системи залежить не лише від її можливостей, але й від того, як швидко та безпечно вона працює.

Функціональні вимоги до системи охоплюють ключові аспекти її роботи, які забезпечують зручність для користувачів та ефективне управління оголошеннями. Система повинна підтримувати реєстрацію та авторизацію користувачів, а також використовувати JWT-токени для безпечної аутентифікації. Передбачена рольова модель включає три ролі: адміністратор, продавець та покупець, кожна з яких має визначені права доступу та можливості в системі.

Оголошення є основним об’єктом роботи платформи, тому будь-який користувач має мати змогу додавати оголошення, але можливість редагувати, оновлювати статус та видаляти оголошення буде тільки в авторизованого користувача. Для полегшення пошуку та навігації передбачені можливості фільтрації та сортування оголошень за різними критеріями, такими як бренд авто, модель, технічний стан чи ціна. Окрім цього, додаткові параметри, наприклад тип кузова або пробіг, можуть бути використані для більш детальної фільтрації.

Користувачі отримують доступ до особистого кабінету, де можуть переглядати свої опубліковані оголошення та керувати ними, редагувати особисті дані. Для адміністраторів реалізована адміністративна панель, що

дозволяє переглядати звіти та виконувати інші функції з управління платформою такі як керування оголошеннями, аккаунтами користувачів.

Нефункціональні вимоги також є важливими, вони визначають якість роботи системи. Висока продуктивність досягається завдяки оптимізації бази даних для швидкого виконання запитів. Безпека забезпечується використанням механізмів захисту даних користувачів, таких як хешування паролів, впровадженням захисту від потенційних атак, зокрема SQL-ін’єкцій та CSRF-атак. Масштабованість системи гарантує її стабільну роботу при збільшенні навантаження, що передбачає можливість горизонтального масштабування за потреби.

Не треба також забувати про юзабіліті і те як користувач взаємодіятиме з платформою. Інтерфейс повинен бути інтуїтивно зрозумілим, логічно структурованим, адаптивний дизайн забезпечує коректне відображення на мобільних пристроях, а використання стилів і компонентів Bootstrap сприяє зручності та естетиці інтерфейсу.

Для візуалізації архітектури системи та взаємодії її компонентів розроблено UML-діаграму прецедентів (Use Case Diagram) (рис. 2.1). Вона допомагає зрозуміти структуру застосунку, визначити зв’язки між об’єктами та процесами, а також забезпечує чітке бачення механізму роботи платформи для всіх учасників розробки.

Рисунок 2.1 – UML-діаграма прецедентів

2.2 Вибір засобів, методів і алгоритмів вирішення поставленого завдання

На основі аналізу проведеного в розділі 1.1 було визначено наступний технологічний стек для розробки платформи.

Для фронтенду буде використано фреймворк Angular, який працює на базі TypeScript, що додає статичну типізацію та покращену підтримку коду. Angular пропонує два підходи до структурування додатків:

– NgModules – це класичний, більш старий підхід, який все ще широко використовується. В цьому підході всі компоненти, сервіси та директиви мають бути частиною модуля;

– bootstrapApplication – це сучасний підхід, що використовує standalone-компоненти. Він дозволяє компонентам існувати окремо.

Як перший так і другий підходи мають свої переваги та недоліки. Наприклад, структура NgModules сприяє кращій організації великого проєкту та спрощує його подальше масштабування, тоді як структура bootstrapApplication зменшує кількість зайвого коду тим самим покращуючи продуктивність.

Однак, підхід standalone-компонентів дещо поступається в аспекті перспективи масштабування, але різниця не є критичною. Враховуючи це, було обрано саме bootstrapApplication, оскільки він дозволяє зменшити кількість зайвого коду та покращити продуктивність, що є пріоритетним.

Для бекенду обрано ASP.NET Core MVC на C#, що забезпечує надійну та масштабовану серверну частину. Для ефективного доступу та маніпулювання даними з бази використовується Entity Framework Core як ORM. База даних буде розміщена на SQL Server у вигляді Azure SQL Database, що гарантує стабільну та безпечну роботу з даними, забезпечує зручність розробки з різних пристроїв з різними операційними системами. Azure SQL Database – це керована хмарна реляційна база даних від Microsoft, яка працює в межах хмарної платформи Azure. Вона підтримує більшість функціональних

можливостей Microsoft SQL Server, але з перевагами хмарної інфраструктури, а саме:

– наявність керованої інфраструктури, а саме підтримка автоматичного оновлення, резервного копіювання та моніторинг без необхідності ручного адміністрування;

– висока доступність та відмовостійкість. Вбудовані механізми забезпечують безперервну роботу та захист від збоїв;

– автоматичне масштабування надає можливість динамічного збільшення або зменшення ресурсів залежно від навантаження;

– вбудовані механізми захисту, такі як шифрування даних, автентифікація за допомогою Azure AD забезпечують безпеку та виявляють загрози;

– інтеграція з іншими сервісами Azure, а саме просте підключення до аналітичних, машинного навчання та DevOps-сервісів.

Для аутентифікації користувачів застосовуватимуться JWT-токени, що забезпечать безпечну та ефективну авторизацію. Для реалізації цього механізму буде використано сервіс Auth0. Це хмарне рішення, яке спрощує керування користувачами та обробку токенів. Сервіс є дуже гнучким і підтримує різні рівні інтеграції: від базового використання як бази даних користувачів із можливістю налаштування ролей та дій (actions), до повноцінного застосування Management API, підключення власної бази користувачів та створення кастомізованих скриптів для автоматичного додавання нових облікових записів чи присвоєння ролей за заданими правилами. Окрім гнучкості, ключовими перевагами Auth0 є висока безпека, простота інтеграції з сучасними фреймворками, можливість масштабування системи без необхідності самостійного управління інфраструктурою.

Цей технологічний стек був обраний через свою здатність забезпечити високу продуктивність, масштабованість та безпеку, що є ключовими вимогами для розробки стабільної та ефективної платформи для розміщення оголошень.

Висновки до розділу 2

У цьому розділі було проведено аналіз вимог до розроблюваної системи, визначено функціональні та нефункціональні вимоги, а також обрано відповідні технології для реалізації проєкту. Запропонована архітектура забезпечить високу продуктивність, безпеку та масштабованість системи. Вибір Angular для фронтенду, ASP.NET Core MVC для бекенду та використання Azure SQL Database дозволить створити гнучку та ефективну платформу для розміщення оголошень про продаж автомобілів. Додатково, використання Bootstrap забезпечить сучасний та адаптивний дизайн інтерфейсу.

РОЗДІЛ 3
РОЗРОБКА ПЛАТФОРМИ ДЛЯ РОЗМІЩЕННЯ ОГОЛОШЕНЬ ПРО ПРОДАЖ АВТОМОБІЛІВ З ВИКОРИСТАННЯМ ANGULAR, ASP.NET, MVC ТА ENTITY FRAMEWORK

3.1 Практична реалізація об’єкта проєктування

Для розробки інтерфейсу використано фреймворк Angular і реалізація починається з розробки базових компонентів додатку, таких як: сторінка каталогу, форма додавання нових оголошень, сторінка користувача, налаштування навігації.

В Angular компоненти є основними будівельними блоками інтерфейсу, що дозволяє розділяти додаток на логічні частини, які легко керуються та можуть бути використані повторно. Для створення нового компонента використовується Angular CLI команда «ng generate component ім’я-компонента --standalone» або скорочено «ng g c ім’я-компонента». Вона автоматично створює весь набір необхідних файлів (рис. 3.1).

```
annabakanina@Annas-MacBook-Pro components % ng generate component catalog --standalone
CREATE src/app/components/catalog/catalog.component.css (0 bytes)
CREATE src/app/components/catalog/catalog.component.html (22 bytes)
CREATE src/app/components/catalog/catalog.component.spec.ts (599 bytes)
CREATE src/app/components/catalog/catalog.component.ts (218 bytes)
```

Рисунок 3.1 – Приклад генерування файлів компоненту

Також в компонентах використовується Bootstrap фреймворк переважно для стилів та класів, що дозволяє легко оформлювати інтерфейс без додаткового CSS-коду. Наприклад в лістингу 3.1 показано використання Bootstrap класів для оформлення кнопки редагування на формі додавання оголошення, спадного списку та для вказування відступів елементів.

В компонентах активно використовується механізм зв’язування даних, або data binding, що забезпечує синхронізацію між моделлю та шаблоном, дозволяючи оновлювати інтерфейс у відповідь на зміни в даних. Приклад цього

можна побачити на лістингах 3.1 та 3.2. В лістингу 3.1 у тегу option за допомогою синтаксису Angular використовується масив brands, який визначений в лістингу 3.2.

Лістинг 3.1 – Фільтр для вибору бренду авто з використанням стилів Bootstrap

```
<div class="mb-3 position-relative">
  <label for="carType" class="form-label">Бренд авто:</label>
  <div class="d-flex align-items-center border-bottom">
    <select class="form-select border-0 shadow-none px-0 ps-3"
id="brandId" [(ngModel)]="vehicle.brandId" name="brandId">
      <option *ngFor="let type of brands" [value]="type.id">{{ type.name
}}</option>
    </select>
    <button type="button" class="btn btn-link p-0 ms-2"
aria-label="Edit">
      <i class="bi bi-pencil"></i>
    </button>
  </div>
</div>
```

кінець лістингу 3.1

Таким чином це дає змогу швидко створювати зручні та красиві UI-елементи, зберігаючи чистоту коду та спрощуючи підтримку стилізації.

Також на лістингу 3.2 можна побачити, що для отримання списку брендів автомобілів та їх моделей використовується один з методів сервісу BrandService. В Angular сервіси забезпечують зв’язок між фронтендом та бекендом за рахунок HTTP-запитів.

Лістинг 3.2 – Приклад моделі компонента форми для додавання оголошення

```
export class VehicleFormComponent implements OnInit {
  brands: any[] = [];
  models: any [] = [];
  constructor( private brandService: BrandService ) {}
  ngOnInit() {
    this.brandService.getBrands().subscribe((data: any) => {
    this.brands = data;
  });
}
```

кінець лістингу 3.2

В додатку сервіси використано для того щоб відокремити логіку отримання даних від компонентів, що робить код чистішим, зручнішим для тестування та підтримки.

Для кращого розуміння загального вигляду, на рисунку 3.2 представлено інтерфейс форми додавання оголошення, реалізований із використанням описаного вище підходу.

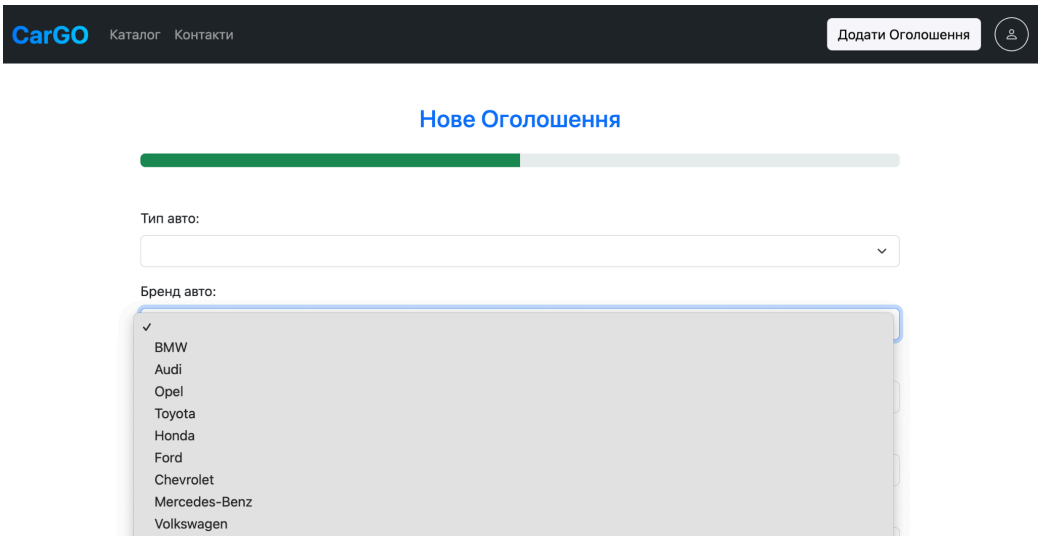

Рисунок 3.2 – Форма додавання нового оголошення

Сервіс BrandService використовується тільки для отримання брендів авто та моделей авто. Сервіс VehicleService має більше функціональностей, наприклад, він використовується для додавання нових оголошень та їх редагування, для їх видалення (ліст. 3.3).

Лістинг 3.3 – Приклад реалізації сервісу VehicleService

```
export class VehicleService {
  constructor( private http: HttpClient) { }
  create(vehicle: any) {
      return this.http.post(this.vehicleEndpoint, vehicle);
  }
  getVehicle(id: number) {
      return this.http.get(this.vehicleEndpoint + id);
  }
  updateVehicle (id: number, body: any) {
      return this.http.put(this.vehicleEndpoint + id, body);
  }
```

```
  delete(id: number) {
      return this.http.delete(this.vehicleEndpoint + id)
  }
}
```

кінець лістингу 3.3

Ще однією важливою функціональністю Angular є налаштування маршрутизації. Вона дозволяє користувачам переміщатися між сторінками без повного перезавантаження застосунку. Файл app.routes.ts використовується для визначення маршрутизації і він містить конфігурацію шляхів, які відповідають певним компонентам. Приклад декількох налаштованих маршрутів продемонстровано в лістингу 3.4.

Лістинг 3.4 – Приклад реалізації маршрутів

```
export const routes: Routes = [
  { path: 'vehicle/new', loadComponent: () =>
  import('./components/vehicle-form/vehicle-form.component').then(m =>
  m.VehicleFormComponent) },
  { path: 'catalog', loadComponent: () =>
  import('./components/catalog/catalog.component').then(m =>
  m.CatalogComponent) },
  { path: 'home', loadComponent: () =>
  import('./components/home/home.component').then(m =>
  m.HomeComponent)}
];
```

кінець лістингу 3.4

В Angular можна як створити окремий компонент для всієї сторінки, так і розбити її на дрібніші компоненти – саме такий підхід був використаний для реалізації сторінки користувача. Її вигляд відрізняється для звичайного користувача та для адміністратора. Для зручності підтримки та повторного використання коду, елементи сторінки були винесені в окремі підкомпоненти з чітко визначеною відповідальністю. На рисунку 3.3 зображено як виглядає сторінка «Мій Профайл» для адміністратора.

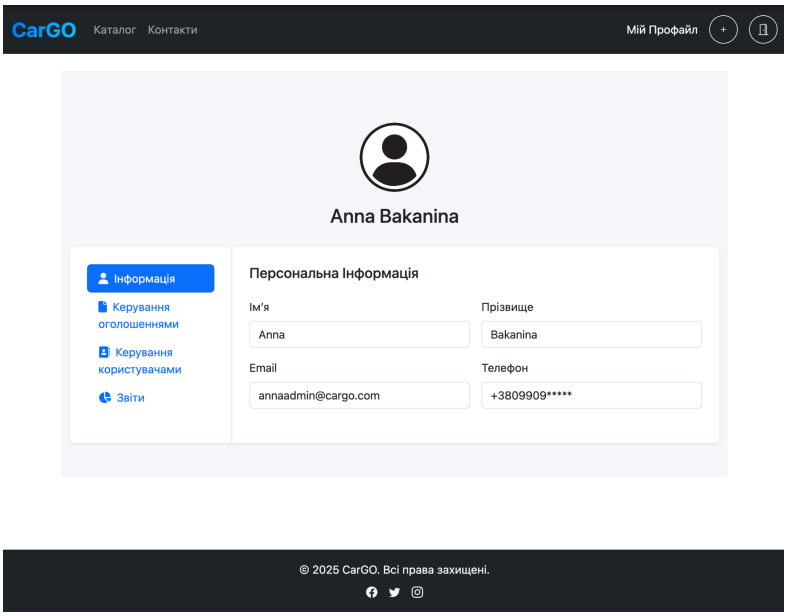

Рисунок 3.3 – Сторінка користувача-адміністратора

На рисунку 3.4 показано цю ж сторінку, але для звичайного користувача та на мобільному пристрої.

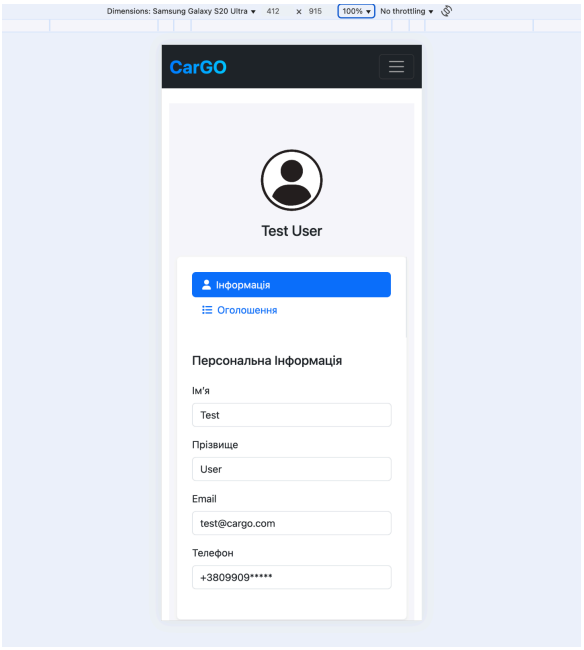

Рисунок 3.4 – Сторінка користувача на мобільному пристрої

В залежності від обраного пункту меню на сторінці відображається відповідний контент, який реалізовано у вигляді окремих Angular-компонентів (ліст. 3.5).

Лістинг 3.5 – Приклад динамічного відображення різної інформації за допомогою компонентів в залежності від активної вкладки

```
<div class="col-lg-9">
 <div class="p-4">
  <div class="mb-4">
    <app-user-info *ngIf="activeTab === 'info'"></app-user-info>
    <app-user-listing *ngIf="activeTab ===
'announcements'"></app-user-listing>
    <app-vehicle-table-listing *ngIf="activeTab ===
'manageAnnoucements'"></app-vehicle-table-listing>
    <app-user-table-listing *ngIf="activeTab ===
'manageUsers'"></app-user-table-listing>
    <app-admin-form *ngIf="activeTab === 'reports'"></app-admin-form>
  </div>
 </div>
</div>
```

кінець лістингу 3.5

Також у додатку реалізована форма контакту (рис. 3.5), яка дає змогу користувачам зв’язатися з командою підтримки. Ця форма побудована з використанням стандартних HTML-елементів, таких як текстові поля для імені, електронної пошти, повідомлення, і інтегрована зі стороннім сервісом EmailJS.

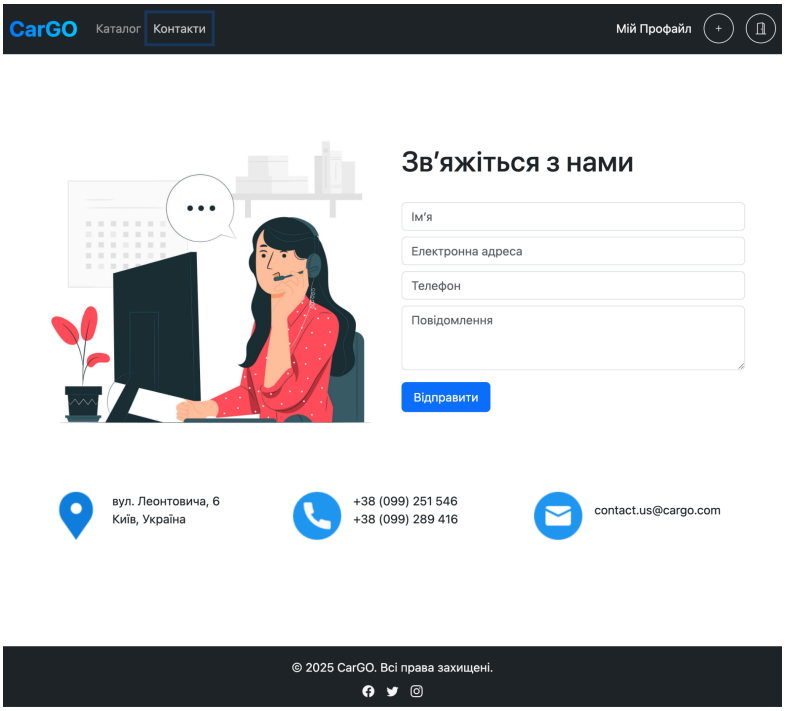

Рисунок 3.5 – Сторінка форми контакту

EmailJS дозволяє надсилати листи безпосередньо з фронтенду, минаючи власний бекенд. Після заповнення форми й натискання кнопки «Надіслати» дані автоматично передаються до сервісу, який формує електронний лист за заданим шаблоном і надсилає його на вказану електронну адресу. У результаті надсилання формується лист з усіма введеними користувачем даними. Приклад отриманого листа зображено нижче на рисунку 3.6.

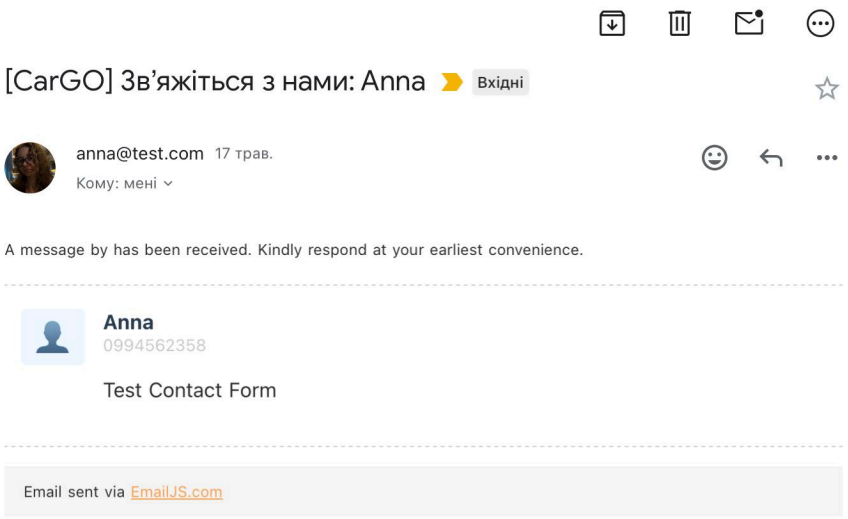

Рисунок 3.6 – Приклад електронного листа з форми контакту

На основі описаних вище компонентів формується загальний вигляд інтерфейсу користувача. Головна сторінка додатку є вхідною точкою, що забезпечує перше враження про сервіс та дозволяє зручно орієнтуватися в основних функціях платформи. На ній розміщено меню, яке надає доступ до каталогу, реєстрації та авторизації, а також навігаційні елементи для переходу до ключових розділів сайту. На рисунку 3.7 зображено зовнішній вигляд стартової сторінки платформи.

Для бекенду застосовується ASP.NET Core MVC у поєднанні з Entity Framework. Архітектура MVC передбачає поділ логіки програми на три основні компоненти: Model, View, Controller. Model описує структуру даних, а Controller керує поведінкою та обробкою запитів.

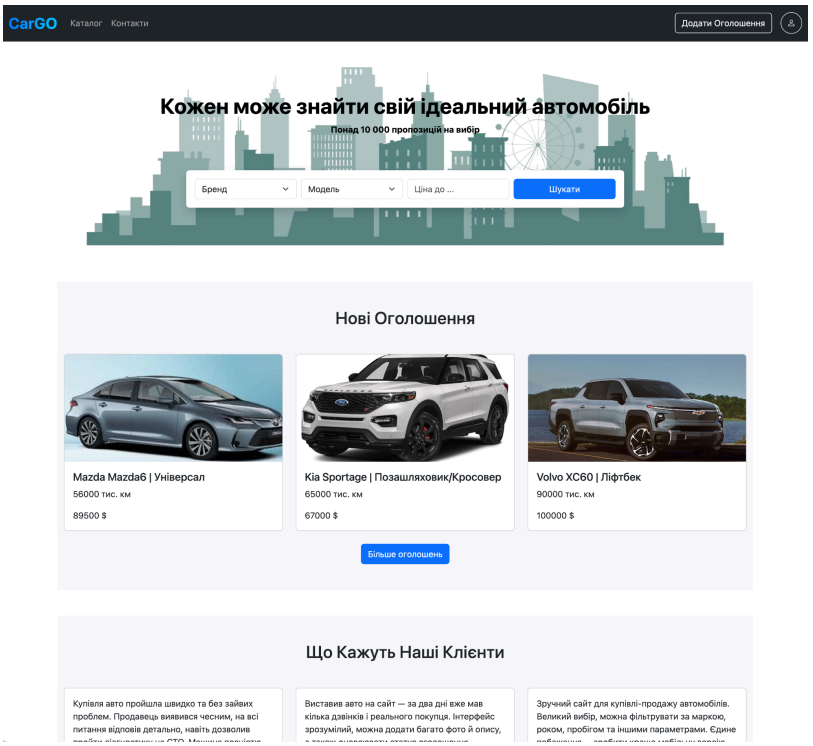

Рисунок 3.7 – Стартова сторінка платформи

Розглянемо реалізацію зберігання інформації про бренди та моделі автомобілів. У рамках моделі даних бренди CarBrands та моделі авто CarModels зберігаються в окремих таблицях бази даних, які пов’язані між собою. При виконанні запиту для отримання всіх брендів повертається json певного формату (ліст. 3.6).

Лістинг 3.6 – Приклад респонсу для ендпоінту GET /CarBrands

```
{
 "id": 4,
 "name": "BMW",
 "carModel": [
   {
   "id": 4,
   "name": "X5"
   },
   {
   "id": 7,
   "name": "3 Series"
   }
 ]
}
```

кінець лістингу 3.6

Цей формат досягається завдяки встановленню зв’язку між моделями CarBrands та CarModels. У класі CarBrands описано навігаційну властивість CarModel, яка представляє колекцію відповідних моделей авто. Детальну реалізацію наведено в лістингу 3.7.

Лістинг 3.7 – Приклад моделі CarBrands

```
public class CarBrands {
  public int Id { get; set; }
  [Required]
  [StringLength(255)]
  public string Name { get; set; }
  public ICollection<CarModel> CarModel { get; set; }
  public CarBrands() {
     CarModel = new Collection<CarModel>();
  }
}
```

кінець лістингу 3.7

У подальшій структурі моделей для встановлення зв’язків між сутностями використовується підхід із збереженням зовнішніх ключів. Наприклад, у моделі Vehicle (ліст. 3.8) передбачені як навігаційні властивості, так і відповідні зовнішні ключі. Такий підхід дозволяє зберігати як повноцінні об’єкти, так і їх ідентифікатори.

Лістинг 3.8 – Приклад моделі Vehicle

```
public CarType CarType { get; set; }
public int CarTypeId { get; set; }
public TechState TechState { get; set; }
public int TechStateId { get; set; }
```

кінець лістингу 3.8

Щоб ізолювати внутрішню логіку додатку від зовнішнього API було використано шаблон DTO (Data Transfer Object), який у контексті ASP.NET Core реалізується через так звані ресурси [13]. Вони дозволяють відокремити домену модель від моделі API, уникати передачі зайвих полів та реалізовувати різні

представлення однієї і тієї ж сутності [13]. Наприклад, сутності Vehicle, при створенні, оновленні, або перегляді. Різниця полягає у тому, що коли потрібно додати нове оголошення, береться SaveVehicleResource, який використовує тільки прості типи даних, такі як int, string. Натомість для повернення детальної інформації про оголошення клієнту використовується VehicleResource ресурс. Він містить вкладені об’єкти-ресурси. В лістингу 3.9 наведено приклад того як в SaveVehicleResource передаються дані про технічний стан авто та його місцезнаходження.

Лістинг 3.9 – Приклад ресурсу SaveVehicleResource

```
public int TechStateId { get; set; }
public int CityId { get; set; }
```

кінець лістингу 3.9

Тоді як лістинг 3.10 демонструє як ті самі дані передаються в VehicleResource.

Лістинг 3.10 – Приклад ресурсу VehicleResource

```
public TechStateResource TechState { get; set; }
public KeyValuePairResource Region { get; set; }
public KeyValuePairResource City { get; set; }
```

кінець лістингу 3.10

Такий розподіл дозволяє ефективно розділяти логіку створення та відображення даних.

Для перетворення між доменними моделями та ресурсами використовується бібліотека AutoMapper. Вона дозволяє централізовано описати правила трансформації між різними об’єктами. Наприклад, нижче в лістингу 3.11 наведено мапінг з API-ресурсу SaveVehicleResource у доменну модель Vehicle.

Лістинг 3.11 – Мапінг з API-ресурсу SaveVehicleResource у модель Vehicle

```
CreateMap<SaveVehicleResource, Vehicle>()
 .ForMember(v => v.Id, opt => opt.Ignore())
 .ForMember(v => v.UserId, opt => opt.MapFrom(vr => vr.UserId))
 .ForMember(v => v.ModelId, opt => opt.MapFrom(vr => vr.ModelId))
 .ForMember(v => v.CarTypeId, opt => opt.MapFrom(vr => vr.CarTypeId))
 .ForMember(v => v.TechStateId, opt => opt.MapFrom(vr =>
vr.TechStateId))
 .ForMember(v => v.YearOfRelease, opt => opt.MapFrom(vr =>
vr.YearOfRelease))
 .ForMember(v => v.VINNumber, opt => opt.MapFrom(vr => vr.VINNumber))
 .ForMember(v => v.CarMileage, opt => opt.MapFrom(vr => vr.CarMileage))
 .ForMember(v => v.Description, opt => opt.MapFrom(vr =>
vr.Description))
.ForMember(v => v.LastUpdated, opt => opt.MapFrom(_ =>
DateTime.UtcNow));
```

кінець лістингу 3.11

Контролери відіграють ключову роль у побудові API. Вони відповідають за обробку HTTP-запитів, виклик відповідних методів сервісів і репозиторіїв, а також за повернення даних у вигляді DTO-ресурсів. Контролери реалізуються як класи, що наслідують ControllerBase, і містять методи-ендпоінти, кожен з яких позначається відповідними HTTP-атрибутами, наприклад, [HttpGet], [HttpPost], [HttpPut]. Також атрибути використовуються для обмеження доступу до певних методів контролера відповідно до рівня автентифікації або ролі користувача. Наприклад, атрибут [Authorize], як зображено в лістингу 3.12, забезпечує доступ до методу лише для авторизованих користувачів.

Лістинг 3.12 – Приклад методу оновлення існуючого Vehicle

```
[HttpPut("{id:int}")]
[Authorize]

public async Task<IActionResult> UpdateVehicle(int id, [FromBody]
SaveVehicleResource saveVehicleResource) {
  if (!ModelState.IsValid)
   return BadRequest(ModelState);
  var vehicle = await _vehicleRepository.GetVehicleById(id,
  includeRelated: false);
  _mapper.Map(saveVehicleResource, vehicle);
  await _unitOfWork.CompleteAsync();
  vehicle = await _vehicleRepository.GetVehicleById(id);
```

```
    var result = _mapper.Map<Vehicle, VehicleResource>(vehicle);
    return Ok(result);
}
```

кінець лістингу 3.12

Метод приймає ID ідентифікатор ресурсу та DTO-об’єкт SaveVehicleResource, який надсилається у тілі запиту. Спочатку перевіряється валідність вхідних даних через перевірку ModelState.IsValid. Далі за допомогою _vehicleRepository отримується об’єкт доменної моделі Vehicle, який оновлюється даними з DTO через _mapper, бібліотеку AutoMapper, яка спрощує мапінг між об’єктами. Після оновлення викликається _unitOfWork.CompleteAsync() для збереження змін у базі даних.

Поле _vehicleRepository є об’єктом, що працює за контрактом, визначеним інтерфейсом IVehicleRepository. Цей інтерфейс описує, як саме можна взаємодіяти з даними оголошень, наприклад, отримати транспорт за ідентифікатором через метод GetVehicleById(), додати нове оголошення або видалити існуюче. Завдяки використанню інтерфейсу, контролер не залежить від конкретної реалізації репозиторію. Це спрощує заміну або тестування компонента. У юніт-тестах можна легко підставити фейкову або мок-реалізацію IVehicleRepository.

Сам _vehicleRepository реалізує шаблон Repository або Repository Pattern, який інкапсулює логіку доступу до джерела даних і надає абстракцію над ORM, а саме Entity Framework. Це дозволяє уникнути дублювання коду доступу до бази та забезпечує гнучкість – ORM-фреймворк можна замінити без змін у бізнес-логіці чи контролерах.

Аналогічно, _unitOfWork реалізує шаблон Unit of Work, який координує роботу кількох репозиторіїв і забезпечує збереження всіх змін у межах однієї операції. Це особливо важливо, коли потрібно зберегти кілька сутностей водночас, і гарантує цілісність даних.

Використання таких патернів сприяє чистій архітектурі, покращує підтримуваність проєкту й дозволяє легше проводити тестування.

3.2 Тестування та налагодження інформаційно-комп’ютерної системи

Для забезпечення якості розробленої інформаційно-комп’ютерної системи було проведено тестування, основна мета якого – це виявлення помилок, перевірка відповідності функціональності вимогам та забезпечення стабільної роботи системи. Для кожної ключової функції системи було створено набір тест-кейсів наведений у додатку Б.

У процесі тестування проводилося верифікування сценаріїв пов’язаних з реєстрацією користувача, редагуванням, додаванням та переглядом даних тощо. Однією з основних технік було функціональне тестування, яке передбачало перевірку роботи основних сценаріїв – зокрема, автентифікації через зовнішнього провайдера, реєстрації нового користувача, редагування профілю, додавання нових записів у базу даних через форму. Кожен з цих функціональних блоків тестувався на предмет відповідності очікуваному результату: коректності збереження даних, оновлення інтерфейсу та повернення правильних повідомлень користувачеві. Як приклад такого тестування можна навести перевірку вигляду навігаційного меню. Після входу користувача система має відображати додаткові кнопки «Мій профіль» та «Вийти з системи», які відсутні для неавторизованих відвідувачів. Також кнопка для додавання оголошення змінює свій вигляд (рис. 3.8). Це дозволяє оцінити коректність роботи умовного рендерингу та ролей користувачів.

Рисунок 3.8 – Приклад умовного рендерингу елементів меню

Додатково було застосовано тестування граничних значень, мета якого перевірити поведінку форми або поля введення в ситуаціях, коли вводяться мінімально допустимі, максимально допустимі або некоректні значення,

наприклад, порожнє поле, надто довгий рядок, недійсний email. Це дозволяє переконатися, що система здатна обробляти нестандартні сценарії введення даних і не призводить до помилок або збоїв. Наприклад, користувач редагує оголошення, натискає кнопку «Зберегти», але одне з полів є пустим. В такому випадку пусте поле підсвічується червоним та відображається інформаційне повідомлення (рис. 3.9).

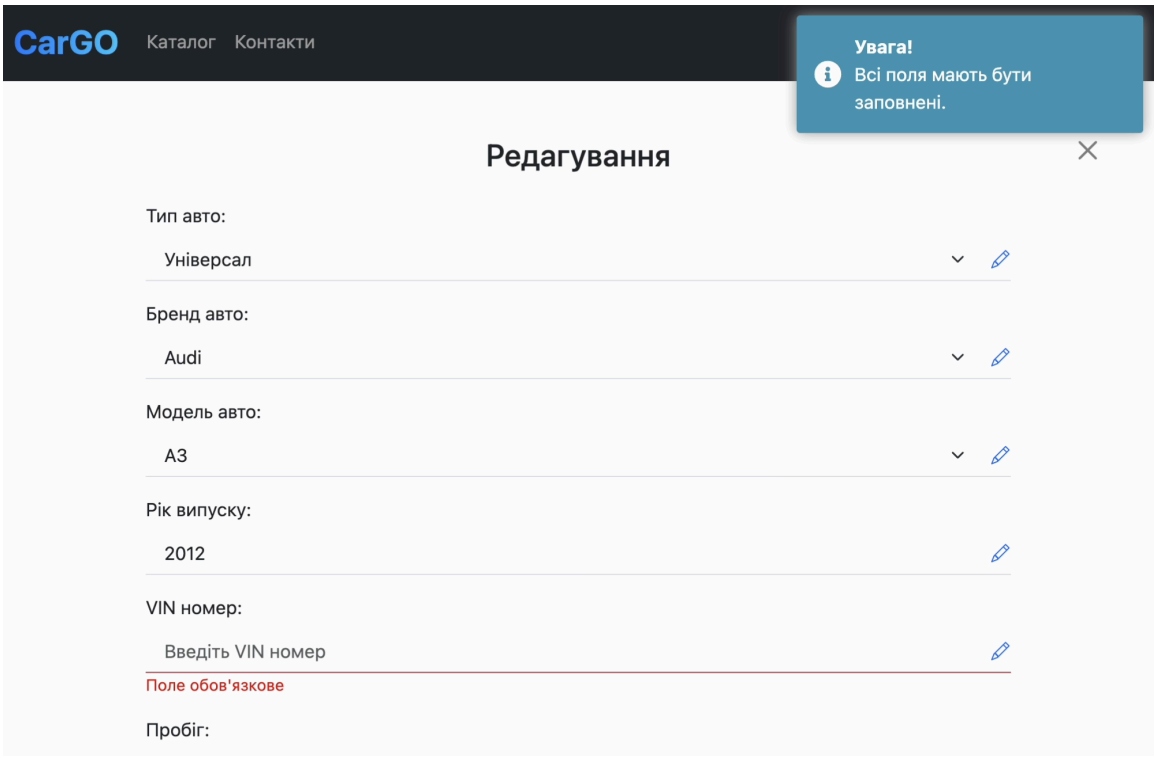

Рисунок 3.9 – Приклад валідації форми редагування оголошення

Також було виконано тестування інтерфейсу користувача, яке зосереджене на перевірці правильності відображення основних елементів інтерфейсу, їх доступності та зручності у використанні. Особливу увагу приділялося коректності адаптивного відображення інтерфейсу, логіці відображення повідомлень про помилки, індикації завантаження та доступності кнопок або посилань залежно від статусу користувача.

Не менш важливою частиною тестування було тестування аутентифікації та авторизації. Перевірка охоплювала сценарії, у яких користувач здійснює вхід до системи за допомогою сервісу Auth0, а також взаємодію з функціональністю,

доступною лише авторизованим користувачам. Наприклад, спроба відкрити профіль або переглянути список оголошень користувача повинна бути дозволена лише після входу. Усі ці сценарії були ретельно перевірені як у позитивних, так і в негативних умовах.

Крім цього, проводилось тестування негативних сценаріїв. Цей підхід дозволяє виявити слабкі місця системи, які можуть проявлятися при неправильних або некоректних діях користувача. Наприклад, спроби залишити обов’язкові поля порожніми, ввести недійсні значення, або виконати дії без авторизації повинні оброблятися системою з виведенням інформативного повідомлення та без збоїв у роботі.

Таким чином, виконане тестування дозволило комплексно оцінити якість реалізації ключового функціоналу системи та підтвердити її готовність до використання кінцевими користувачами.

3.3 Розробка бази даних

Оскільки в проєкті використовується Entity Framework може бути декілька підходів до створення таблиць бази даних та додавання даних до них. Спочатку було створено SQL Server та шаблон бази даних.

Коли SQL Server з шаблоном бази даних готові наступним кроком буде налаштування доступу до нього зі сторонніх машин, що робиться за допомогою методу безпеки whitelisting, або методу білого списку.

Whitelisting (білого спискування) – це метод безпеки, при якому доступ або дозвіл на виконання операцій надається лише визначеному списку дозволених об’єктів, таких як IP-адреси, програми, користувачі або домени [14]. Усе, що не входить до цього списку, автоматично блокується. У розроблюваному додатку доступ до сервера дозволено лише з визначених IP-адрес.

Як тільки всі ці налаштування зроблені можна підключатися до бази даних. Для цього використовується среда разработки для баз даних DataGrip,

який вбудований в Rider IDE. Підключення відбувається за допомогою SQL Authentication (рис. 3.10).

Наступним кроком буде створення таблиць, одним із методів який було використано для цього є міграції, які надає Entity Framework.

Міграції в Entity Framework – це механізм для управління схемою бази даних під час змін у коді моделі даних [15]. Вони дозволяють оновлювати структуру БД без втрати даних, додаючи або змінюючи таблиці, стовпці та інші елементи. Коли змінюється модель, наприклад, додається новий клас або поле, Entity Framework використовує міграції, щоб оновити базу даних відповідно до цих змін. Таким чином, міграції дозволяють легко створювати, оновлювати та підтримувати структуру бази даних без необхідності вручну писати SQL-запити.

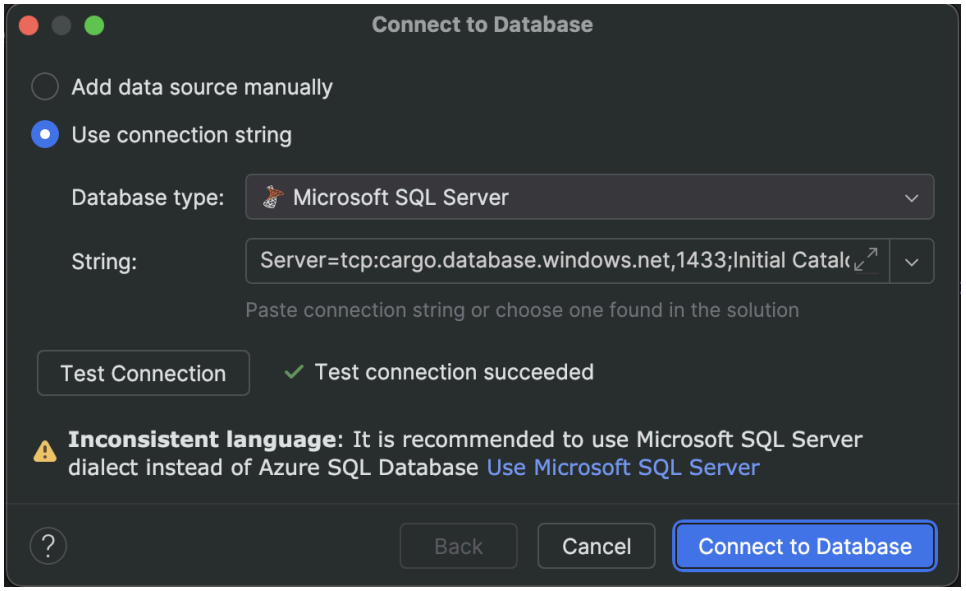

Рисунок 3.10 – Підключення до бази даних за допомогою DataGrip

Наприклад, треба створити таблиці Brands та CarModel які мають зв’язок між собою. Для цього було написано клас-модель, який відображає цей зв’язок (ліст. 3.13). Далі створюється міграція з назвою InitialModel за допомогою команди «dotnet ef migrations add InitialModel» та приміняється за допомогою команди «dotnet ef database update».

Лістинг 3.13 – Приклад моделі CarModel

```
using System.ComponentModel.DataAnnotations;
namespace Backend.Models;

public class CarModel {
  public int Id { get; set; }
  [Required]
  [StringLength(255)]
  public string Name { get; set; }
  public CarBrands Brand { get; set; }
  public int BrandId { get; set; }
}
```

кінець лістингу 3.13

Міграції рекомендується називати зрозумілими іменами, щоб в подальшому при необхідності можна було оновити базу даних до якогось з попередніх станів за допомогою попередніх міграцій. Наприклад, якщо буде створена і додана нова таблиця CarType, є можливість повернутися до минулого стану БД за допомогою команди «dotnet ef database update InitialModel».

Entity Framework дозволяє не лише змінювати структуру БД через міграції, а й наповнювати її початковими даними. Це робиться шляхом використання методу MigrationBuilder.Sql() або перевизначення методу Up() у файлі міграції. Цей спосіб підходить для додавання статичних даних, які не змінюються після міграції, наприклад, для заповнення довідкових таблиць.

В проєкті використано підхід з перевизначенням методу Up() у файлі міграції. Для цього створюється нова міграція за допомогою команди «dotnet ef migrations add SeedDatabase» і в створеному файлі замінюються методи Up() і Down() (ліст. 3.14).

Лістинг 3.14 – Приклад методу Up() в файлі міграції SeedDatabase.cs

```
public partial class SeedDatabase : Migration {
 protected override void Up(MigrationBuilder migrationBuilder) {
   migrationBuilder.Sql("INSERT INTO Brands (Name) VALUES ('BMW')");
   migrationBuilder.Sql("INSERT INTO Brands (Name) VALUES ('Audi')");
   migrationBuilder.Sql("INSERT INTO Brands (Name) VALUES ('Opel')");
   migrationBuilder.Sql("INSERT  INTO  CarModel  (Name,  BrandId)  VALUES
   ('X5',(SELECT ID FROM Brands WHERE Name = 'BMW'))");
```

```
        migrationBuilder.Sql("INSERT INTO CarModel (Name, BrandId) VALUES
        ('A4', (SELECT ID FROM Brands WHERE Name = 'Audi'))");
        migrationBuilder.Sql("INSERT INTO CarModel (Name, BrandId) VALUES
        ('Astra', (SELECT ID FROM Brands WHERE Name = 'Opel'))");
    }
}
```

кінець лістингу 3.14

Метод Down() у файлі міграції використовується для відкату міграції до попереднього стану. Його основне завдання – це скасувати зміни, внесені методом Up(), щоб можна було повернути базу даних до попередньої версії.

Після цього виконується команда «dotnet ef database update» та перевіряється база даних. На рисунку 3.11 зображено дані, які були додані в таблицю Brands після застосування міграції.

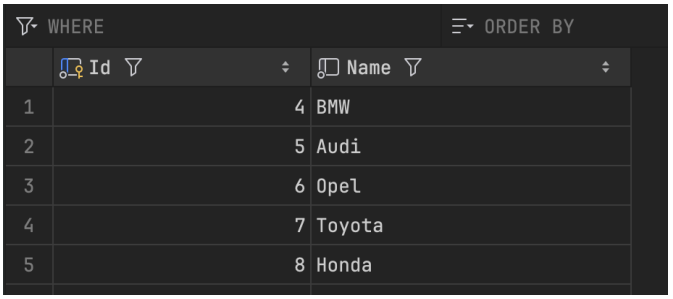

Рисунок 3.11 – Таблиця Brands з доданими даними

На рисунку 3.12 показано дані, які були додані в таблицю CarModel.

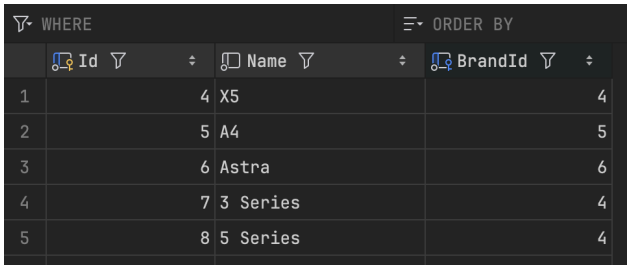

Рисунок 3.12 – Таблиця CarModel з доданими даними

Окрім використання міграцій в Entity Framework, створювати таблиці та заповнювати їх даними можна класичним способом за допомогою SQL-запитів.

Цей підхід надає більше контролю над структурою бази даних і використовується в ситуаціях, коли потрібно точно визначити схему або виконати складні операції з даними. SQL-запити також активно використовуються для додавання даних до таблиць. В лістингу 3.15 наведено приклад SQL скрипта для додавання даних в таблицю CarTypes. N на початку значення мусить бути додане щоб воно було оброблено як ASCII.

Лістинг 3.15 – SQL скрипт для додавання даних в таблицю CarTypes

```
INSERT INTO CarTypes (Name) VALUES
(N'Седан'),
(N'Позашляховик/Кросовер'),
(N'Універсал'),
(N'Хетчбек'),
(N'Мінівен'),
(N'Ліфтбек'),
(N'Купе'),
(N'Мікровен');
```

кінець лістингу 3.15

Таким чином, використовуючи як засоби міграцій Entity Framework, так і класичні SQL-запити, було реалізовано створення повної структури бази даних, зокрема таблиць, первинних та зовнішніх ключів, індексів, а також визначення обмежень цілісності даних. Крім того, за допомогою SQL-інструкцій здійснено початкове заповнення таблиць довідковою інформацією, необхідною для коректної роботи системи, включаючи дані про бренди, типи автомобілів, технічний стан, регіони та міста.

Для наочного представлення структури бази даних, її основних таблиць та зв’язків між ними була створена діаграма сутність-зв’язок (ER-діаграма), яка зображена на рисунку 3.13. Вона дозволяє краще зрозуміти логіку побудови системи, а також є корисним інструментом для подальшої розробки та підтримки проєкту. ER-діаграма дає змогу швидко оцінити структуру даних. Її використання також сприяє кращій комунікації між розробниками й тестувальниками.

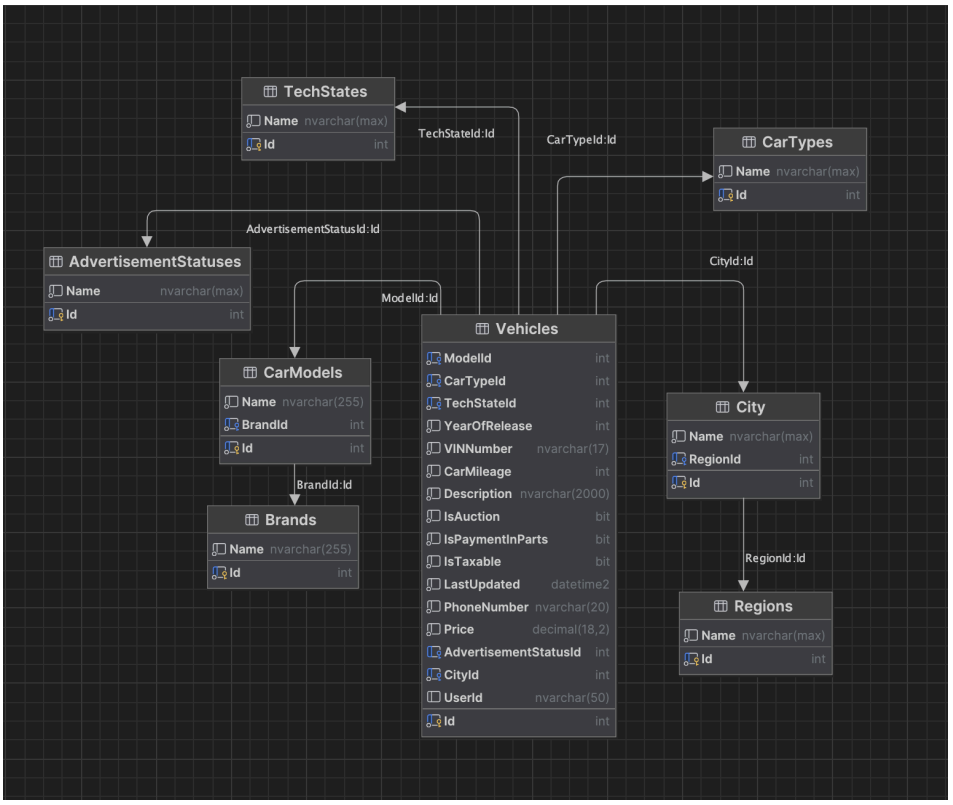

Рисунок 3.13 – Діаграма бази даних

3.4 Захист інформаційно-комп’ютерної системи

У проєкті для забезпечення аутентифікації та авторизації користувачів використовується сервіс Auth0, який надає зручні та безпечні механізми для керування доступом до інформаційно-комп’ютерної системи. Auth0 підтримує сучасні стандарти безпеки, серед яких основними є OAuth 2.0 для авторизації та OpenID Connect для автентифікації, що дозволяє інтегрувати систему входу користувачів без необхідності створювати власні механізми логіну та захисту паролів [16].

У проєкті використовується власний екземпляр платформи з власним доменом, де розміщено конфігурацію входу, користувачів, дозволів і підключень. Для взаємодії з бекендом Access Token має містити правильне значення audience, яке в нашому випадку дорівнює https://cargo.com – це ідентифікатор ресурсу, до якого запитуватиметься доступ. Це дозволяє бекенду впевнено перевірити, що токен дійсно видано для потрібного API.

В рамках реалізація платформи Auth0 також використовується як додаткова база даних для зберігання даних користувачів, їх ролей. Auth0 не зберігає паролі користувачів у відкритому вигляді. Замість цього використовується хешування, в першу чергу це використання адаптивної криптографічної функції bcrypt, яка спеціально розроблена для захисту паролів від зловмисників навіть у випадку витоку бази даних [17]. У деяких випадках, наприклад при імпорті користувачів з інших систем, можуть також використовуватись інші алгоритми, такі як PBKDF2, scrypt або argon2, які також вважаються безпечними і рекомендованими в сучасній криптографії [17]. Записи оголошень зв’язуються з користувачами за рахунок userId. Щоб оновлювати дані користувачів використовується Management API. Доступ до цього API можливий лише через аутентифікацію за допомогою спеціального access token, що видається при запиті з правильними client_id та client_secret. Вся ця логіка реалізована на бекенді, що забезпечує вищий рівень безпеки. Для взаємодії з Management API фронтенд звертається лише до власних внутрішніх ендпоінтів бекенду, які вже, у свою чергу, виконують необхідні запити до зовнішнього API. Таким чином, з боку клієнта немає жодної видимої прив’язки до Management API, а конфіденційні дані, такі як client_id і client_secret, ніколи не потрапляють на фронтенд і, користувач взаємодіє з одним уніфікованим API, що належить застосунку, а логіка інтеграції з зовнішньою системою автентифікації повністю інкапсульована на сервері.

Крім того, Auth0 використовує JWT (JSON Web Tokens) як основний формат для передачі авторизаційної інформації між клієнтом та сервером. Для підписування токенів, зокрема id_token, за замовчуванням використовує алгоритм RS256 (RSA з SHA-256), який є рекомендованим і безпечним методом. У цьому випадку Auth0 підписує токен приватним ключем, а клієнтська сторона може перевірити його достовірність за допомогою публічного ключа [18]. Після успішного входу користувача сервіс видає два основні токени: id токен, який містить інформацію про самого користувача і використовується на фронтенді

для побудови інтерфейсу, та access токен, що призначений для автентифікованого доступу до API.

Для реєстрації нового користувача на форму було додано додаткові кастомізовані поля: firstName, lastName, phoneNumber (рис. 3.14).

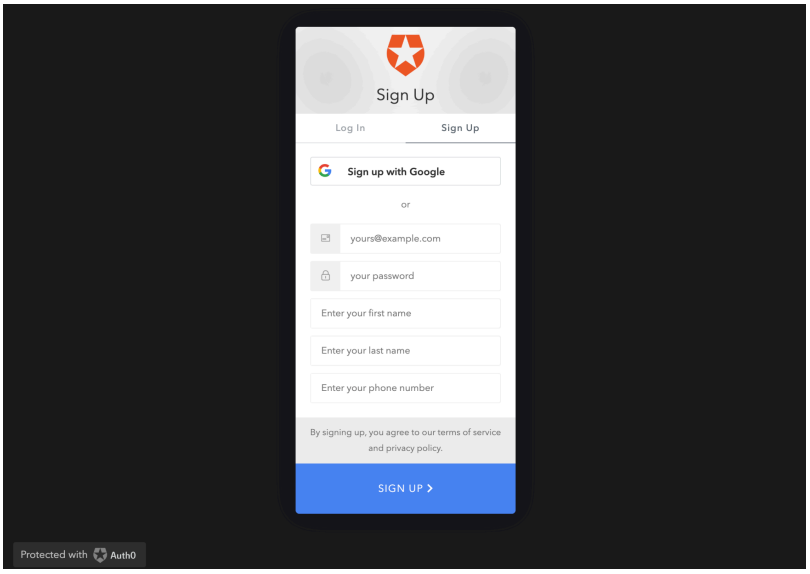

Рисунок 3.14 – Кастомізовані поля, додані до форми реєстрації користувача

Вони автоматично зберігаються в user_metadata, але вони мають бути додані до id токену за допомогою створення власної дії (action) для етапу логіну. В лістингу 3.16 продемонстровано код для цієї дії.

Лістинг 3.16 – Приклад коду дії (action) для етапу логіну в Auth0

```
exports.onExecutePostLogin = async (event, api) => {
  const userMetadata = event.user.user_metadata || {};
  const appMetadata = event.user.app_metadata || {};
  const namespace = 'https://yourdomain.com';
  api.idToken.setCustomClaim(`${namespace}/first_name`,
  userMetadata.first_name);
  api.idToken.setCustomClaim(`${namespace}/last_name`,
  userMetadata.last_name);
  api.idToken.setCustomClaim(`${namespace}/phone_number`,
  userMetadata.phone_number);
  api.idToken.setCustomClaim(`${namespace}/role`, appMetadata.roles);
};
```

кінець лістингу 3.16

Для використання сервісу він має бути інтегрований як на фронтенді так і на бекенді. На бекенді це відбувається за рахунок додавання реалізації сервісу аутентифікації в Program.cs (ліст. 3.17).

Лістинг 3.17 – Приклад реалізації сервісу аутентифікації в Program.cs

```
builder.Services.AddAuthentication(options => {
  options.DefaultAuthenticateScheme =
  JwtBearerDefaults.AuthenticationScheme;
  options.DefaultChallengeScheme =
  JwtBearerDefaults.AuthenticationScheme;
  })
  .AddJwtBearer(options => {
  options.Authority = auth0Domain;
  options.Audience = auth0Audience;
  options.TokenValidationParameters = new TokenValidationParameters
  {
    ValidateIssuer = true,
    ValidateAudience = true,
    ValidateLifetime = true,
    ValidateIssuerSigningKey = true
  };
});
```

кінець лістингу 3.17

Також на бекенді реалізується контролер UserManagementApiController для управління користувачами. Він відповідає за оновлення метаданих користувача в системі авторизації Auth0 через Management API та може бути розширений додатковими методами. Використовується залежності IHttpClientFactory для створення HTTP-клієнтів та IConfiguration для зчитування конфігураційних параметрів, таких як домен Auth0, client ID і client secret. Основний метод контролера – UpdateUserMetadata(), який доступний за маршрутом PATCH api/user/update-profile та приймає у тілі запиту об’єкт, що містить ID користувача, ім’я, прізвище, номер телефону. Метод спочатку викликає допоміжний метод GetManagementApiToken(), щоб отримати access token для доступу до Auth0 Management API. Якщо токен не вдається отримати, метод повертає помилку 500. Якщо токен був отриманий то формується PATCH-запит до Auth0 для оновлення метаданих користувача за вказаним

userId, додаючи отриманий токен у заголовки. Якщо запит не успішний, повертається код помилки й повідомлення з відповіддю Auth0. У випадку успіху повертається тіло відповіді. Таким чином, цей контролер реалізує захищене оновлення додаткової інформації про користувача в Auth0 на основі авторизованого запиту з бекенду.

Для інтеграції на фронтенді для реалізації аутентифікації використовується бібліотека @auth0/auth0-angular, яка забезпечує зручну інтеграцію з Auth0 та дозволяє керувати виходом користувача, отриманням його профілю та статусом автентифікації. Основним інструментом для роботи з користувачем є сервіс UserService. Він підписується на зміну статусу автентифікації та, у разі авторизації, отримує об’єкт користувача. Із цього об’єкта сервіс бере як стандартні поля такі як email та sub, так і кастомні поля firstName, lastName, phoneNumber, які зберігаються в Auth0 із префіксом https://yourdomain.com/. Отримані дані зберігаються у властивості userDetails, що дозволяє використовувати їх в інших частинах застосунку. Для оновлення профілю передбачений метод updateProfile(), який відправляє PATCH-запит на бекенд із відповідними даними, включаючи userId, взятий із поля sub.

Сервіс також містить методи userLogin() та userLogout(), які викликають редірект на логін або логаут через Auth0. У компоненті NavigationComponent використовується об’єкт AuthService напряму і перевіряється статус авторизації. На основі цього статусу рендеряться різні елементи інтерфейсу. Якщо користувач авторизований – показується посилання на профіль, кнопка «+» для додавання оголошення, а також кнопка виходу. Якщо ні – кнопка входу та можливість перейти на сторінку створення оголошення. Для підтримки довготривалих сесій Auth0 реалізує ротацію refresh токенів – це механізм, при якому після кожного використання refresh токен автоматично замінюється на новий [19]. Таким чином, навіть якщо токен буде викрадений, зловмисник не зможе використовувати його повторно. Це підвищує рівень безпеки та зменшує ризик несанкціонованого доступу.

Таким чином, використання Auth0 у поєднанні з токенами, ролями, audience та Management API забезпечує надійний рівень захисту інформаційної системи як на етапі автентифікації користувача, так і при контролі доступу до ресурсів.

3.5 Розробка документації для програмного продукту

Успішна розробка програмного забезпечення неможлива без створення повноцінної, структурованої та доступної документації. Документація не лише супроводжує процес розробки, але й забезпечує зрозумілу і послідовну комунікацію між усіма учасниками проєкту.

Для коректної роботи розробленої системи на локальному комп’ютері користувача або на сервері необхідно забезпечити певні мінімальні системні вимоги описані у таблиці 3.1.

Таблиця 3.1 – Список мінімальних системних вимог

№	Тип вимоги	Опис мінімальних вимог
1.	Операційна система	Windows 10 / macOS 12 / Linux (Ubuntu 20.04 або новіше)
2.	Процесор	Intel Core i5 або еквівалент ARM/M1 (Apple Silicone)
3.	Оперативна пам’ять	Не менше 8 ГБ RAM
4.	Вільне місце на диску	Не менше 2 ГБ
5.	Браузер	Google Chrome, Mozilla Firefox, Safari (останні версії)
6.	Node.js	Версія 22.12.0
7.	.NET SDK	Версія 9.0.200
8.	SQL Server	Локальна база або Azure SQL Database
9.	Інше ПЗ	npm 10.9.0 Angular CLI 19.1.7

Для запуску веб-застосунку локально користувачеві необхідно виконати низку кроків з підготовки середовища, клонування репозиторію, встановлення залежностей і запуску клієнтської та серверної частин системи.

На першому етапі потрібно встановити необхідні компоненти, а саме:

– node.js версії 22.12.0, що відповідає за виконання JavaScript-коду та управління пакетами клієнтської частини;

– .NET SDK версії 9.0.200, який забезпечує запуск серверної частини на платформі ASP.NET Core;

– також потрібно глобально встановити Angular CLI версії 19.1.7, скориставшись командою «npm install -g @angular/cli@19.1.7».

Другим кроком є клонування репозиторію з кодом проєкту. Для цього слід скористатися командою «git clone» і перейти до відповідної директорії, яка містить код.

На третьому етапі користувач переходить до налаштування клієнтської частини. Для цього потрібно перейти в кореневий каталог, встановити всі залежності за допомогою «npm install», після чого запустити клієнтську частину командою «npm run start». Після запуску веб-інтерфейс буде доступний за адресою http://localhost:4200.

Четвертий крок стосується налаштування серверної частини. Потрібно перейти до кореневий каталог backend, відновити залежності за допомогою команди «dotnet restore» і запустити сервер командою «dotnet run». У результаті серверна частина буде доступна за адресою http://localhost:5000.

На завершальному етапі слід налаштувати підключення до бази даних. Це здійснюється через редагування файлу appsettings.json, у якому необхідно вказати строку підключення до хмарної бази даних в Azur. В лістингу 3.18 наведено приклад рядка підключення до бази даних.

Лістинг 3.18 – Приклад рядка підключення до бази даних

```
"ConnectionStrings": {
"Default": "Server=tcp:cargo.database.windows.net,1433;Initial
Catalog=CarGO;Persist Security Info=False;User
```

```
ID=annaadmin;Password=pa$$w0rd;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;"
}
```

кінець лістингу 3.18

Для забезпечення взаємодії з backend частиною платформи створено колекцію запитів у Postman, яка охоплює основні аспекти роботи з API. Ця колекція слугує не лише як зручний інструмент тестування запитів, а й як форма технічної документації, що дозволяє швидко ознайомитися з можливостями системи, а зовнішнім розробникам – інтегрувати власні рішення з урахуванням API.

На рисунку 3.15 представлено структуру колекції Postman, яка має назву CarGo. Вона складається з декількох логічно згрупованих блоків.

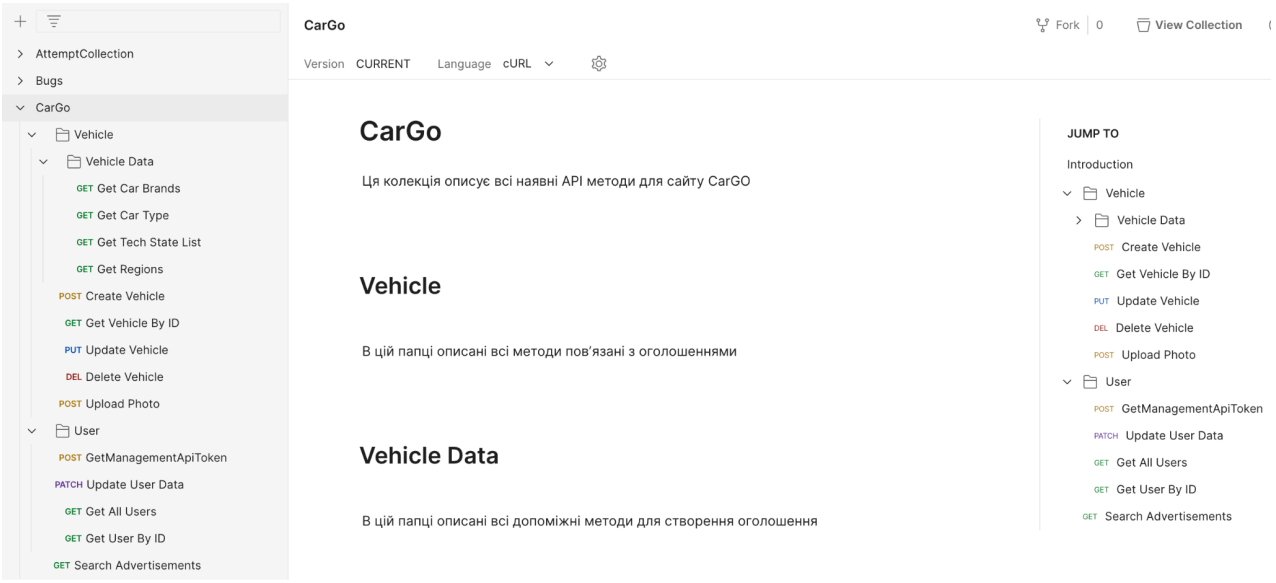

Рисунок 3.15 – Структура Postman колекції

Vehicle Data – це набір GET-запитів, які дозволяють отримати довідкову інформацію, необхідну для створення або фільтрації оголошень. У папці Vehicle зберігаються всі CRUD-операцій, що дозволяють керувати оголошеннями. Папка User включає запити, пов'язані з автентифікацією, отриманням токену, оновлення даних користувача, запити для взаємодії з Management API від Auth0.

Search Advertisements – це запит для пошуку оголошень із використанням заданих фільтрів.

Для зручності тестування API налаштовано окреме середовище, яке містить всі необхідні змінні, такі як базовий URL сервера, токени авторизації, client ID тощо. Це дає змогу швидко переключатися між різними конфігураціями та забезпечує стабільну роботу в різних умовах. Наявність такого середовища значно зменшує ризик помилок при ручному введенні параметрів та пришвидшує розробку. Усі запити згруповані логічно, що полегшує навігацію в межах проєкту та сприяє кращому розумінню його структури.

Кожен запит супроводжується описом необхідних параметрів, прикладами успішних відповідей та можливих помилок. Це дозволяє швидко зрозуміти логіку роботи API та забезпечує ефективне тестування всіх функціональних можливостей системи. На рисунку 3.16 показано приклад запиту для додавання нового оголошення.

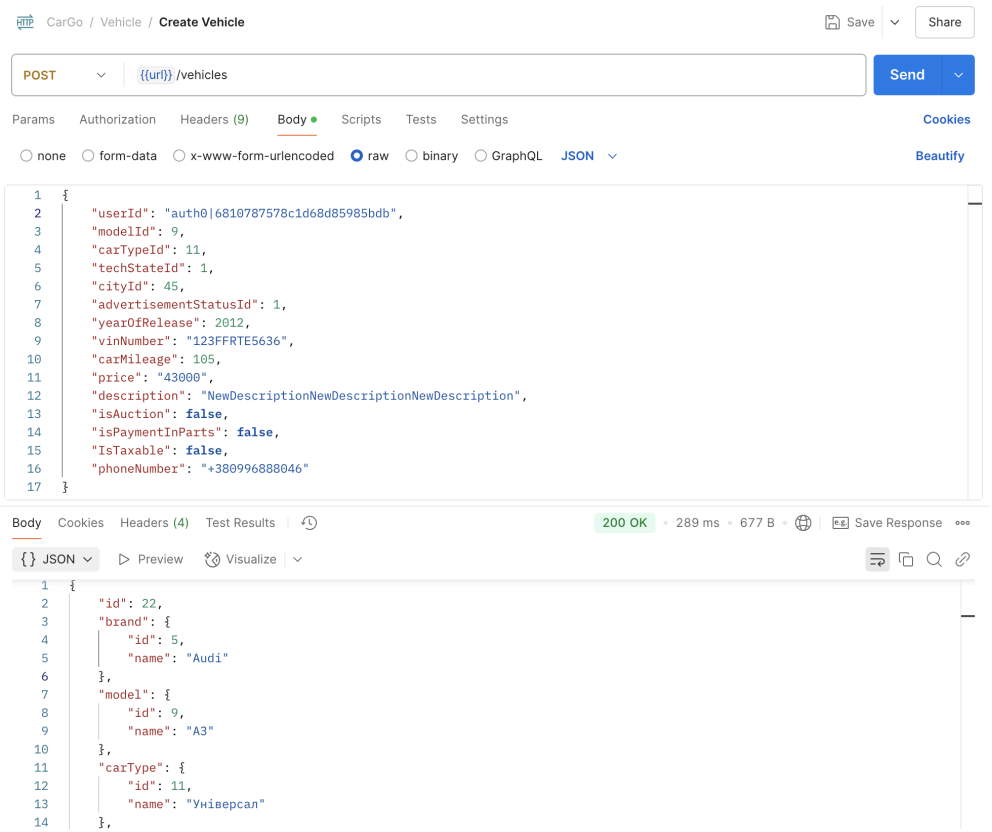

Рисунок 3.16 – Приклад запиту для додавання нового оголошення

У цьому сервісі однією з центральних функціональностей є створення та перегляд оголошень про продаж автомобілів. Для цього реалізовано два головні ендпоінти: POST /vehicles та GET /vehicles.

POST /vehicles ендпоінт використовується для створення нового оголошення про продаж авто. Його викликає користувач, який хоче додати своє авто на платформу. У тілі запиту в форматі JSON потрібно вказати основну інформацію про транспортний засіб, а саме:

– userId – ID користувача, який створює оголошення;

– modelId, carTypeId, techStateId, cityId, advertisementStatusId – це ID відповідних характеристик авто;

– yearOfRelease – рік випуску авто;

– vinNumber – VIN-код транспортного засобу;

– carMileage – пробіг авто;

– price – ціна;

– description – опис оголошення;

– isAuction – можливий торг;

– isPaymentInParts – можлива оплата частинами;

– isTaxable – чи було авто розмитнене;

– phoneNumber – номер телефону власника.

У відповіді сервер повертає повну інформацію про створене оголошення включно з розшифрованими назвами моделі, бренду, типу авто, міста тощо. Це дозволяє одразу відображати оголошення у клієнті.

GET /vehicles ендпоінт дозволяє отримати список доступних оголошень з можливістю фільтрації та сортування. У рядку запиту query parameters можна передати:

– номер сторінки (page) та розмір сторінки (pageSize) для пагінації;

– параметри для фільтрів;

– параметри для сортування.

У відповіді повертається:

– масив об’єктів з короткою інформацією про авто;

– загальна кількість знайдених оголошень;
– поточна сторінка та її розмір.

Цей ендпоінт є основою для реалізації головної сторінки з переліком авто. Усі дати подаються у форматі ISO, наприклад, 2025-05-13T18:00:00Z. Усі поля, які передаються як ID – це числові значення, що посилаються на окремі таблиці, наприклад, моделі, міста, типи кузова тощо.

Ця колекція разом з налаштуваннями середовища будуть додані до репозиторію як файл .json, який може бути імпортований у додаток Postman і одразу буде готовий до використання.

Висновки до розділу 3

У цьому розділі було детально описано етапи реалізації платформи, продемонстровано використання обраних технологій та інструментів, а також представлено застосовані підходи до розробки. Окрему увагу приділено тестуванню – виконано мануальну перевірку функціоналу за допомогою підготовлених тест-кейсів. Продемонстровано реалізацію механізмів забезпечення безпеки користувацьких даних та автентифікації, а також створено документацію стосовно загальних вимог до середовища для запуску платформи та до API, що полегшує взаємодію з системою для зовнішніх розробників і користувачів.

ВИСНОВКИ

В даній кваліфікаційній роботі розроблено веб-платформу для публікації оголошень про продаж автомобілів, яка поєднує зручність використання, сучасний адаптивний дизайн і надає надійні механізми захисту даних.

На початковому етапі було проведено аналіз предметної області – веб-платформ для продажу автомобілів. Було досліджено функціональні особливості існуючих рішень, таких як Auto.Ria, RST.ua, AutoScout24, Otomoto.pl, AutoTrader UK та Le Centrale.

На основі цього аналізу було сформульовано функціональні та нефункціональні вимоги до системи, визначено функціональності аналогічних платформ, ключові потреби користувачів і сучасні тенденції в інтерфейсах.

Також розроблено необхідну документацію, включно з UML-діаграмою, яка ілюструє основні ролі в системі, а саме: користувач, авторизований користувач та адміністратор, а також їхні взаємодії з платформою – включаючи дії з оголошеннями, управління користувачами, перегляд звітів і надсилання зворотного зв’язку.

Інтерфейс користувача спроєктовано та реалізовано із використанням HTML, CSS, фреймворків Bootstrap і Angular. Дизайн платформи враховує принципи адаптивності та зручності.

Для реалізації серверної частини застосовано ASP.NET. Створено REST API для управління оголошеннями та користувачами, реалізовано можливість фільтрації, сортування оголошень та пагінації. Архітектура проєкту побудована з використанням шаблонів проєктування Repository Pattern та Unit of Work, що забезпечує чітке розділення відповідальностей, спрощує тестування та обслуговування коду.

Базу даних створено і розміщено на SQL Server у вигляді Azure SQL Database із забезпеченням цілісності зв’язків між сутностями. Для її побудови використовувалися міграції Entity Framework, а також ручні SQL-запити.

Для захисту від CSRF-атак було реалізовано відповідну логіку на стороні клієнта і сервера. Зокрема, у запити додано JWT-токени, що перевіряються сервером перед обробкою, що дозволяє переконатися в достовірності запитів і запобігає несанкціонованому виконанню дій.

Додано методи логування та обробки помилок з відображенням відповідних повідомлень на інтерфейсі користувача. Це спрощує діагностику та дозволяє забезпечити зворотний зв’язок для користувача у випадку виникнення проблем.

Для забезпечення безпеки впроваджено автентифікацію та авторизацію з використанням JWT-токенів через сервіс Auth0, що забезпечує контроль доступу та захист персональних даних. Також була реалізована інтеграція з Management API для реалізації можливості оновлення даних користувача, отримання інформації про всіх користувачів.

В каталозі реалізована можливість сортування, фільтрування та пагінації оголошень, що забезпечує зручну взаємодію з великим обсягом даних. Користувачі мають змогу додавати фотографії до оголошень, які автоматично відображаються в інтерфейсі. Для адміністратора реалізовано окремі компоненти для керувати контентом та візуалізації звітів, що дозволяє відслідковувати активність на платформі.

Якість розробленого ПЗ перевірено шляхом тестування функціональних, граничних і негативних сценаріїв із використанням підготовлених тест-кейсів. Це дозволило виявити потенційні недоліки на ранніх етапах і підвищити стабільність роботи системи.

У результаті виконання роботи створено веб-платформу, яка відповідає сучасним вимогам до продуктивності, безпеки й зручності, має зрозумілий інтерфейс та масштабовану архітектуру. Усі поставлені завдання успішно виконано, що підтверджує досягнення мети дослідження та створення функціонуючого цифрового продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Стрілець С. В. Орлик О. В. Електронна комерція: принципи, види та перспективи розвитку в Україні. URL: https://core.ac.uk/download/pdf/147039105.pdf (дата звернення: 03.05.2025 р.).

2. Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core. URL: https://surl.li/zzjtvk (дата звернення: 20.02.2025 р.).

3. Онлайн-платформа Auto.Ria. URL: https://auto.ria.com/uk/ (дата звернення: 21.02.2025 р.).

4. Онлайн-платформа RST.ua. URL: https://rst.ua/ukr/ (дата звернення: 21.02.2025 р.).

5. Онлайн-платформа AutoScout24.com. URL: https://www.autoscout24.com/ (дата звернення: 25.02.2025 р.).

6. Онлайн-платформа Otomoto.pl. URL: https://www.otomoto.pl/ (дата звернення: 25.02.2025 р.).

7. Онлайн-платформа AutoTrader. URL: https://www.autotrader.co.uk/ (дата звернення: 02.03.2025 р.).

8. Онлайн-платформа Le Centrale. URL: https://www.lacentrale.fr/ (дата звернення: 02.03.2025 р.).

9. Emmanni P. S. Comparative Analysis of Angular, React, and Vue.js in Single Page Application Development. URL: https://surl.li/sfhxca (дата звернення: 02.03.2025 р.).

10. Mushica F., Memeti A. A Comprehensive analysis of architectural patterns in ASP.NET Core WEB application development. URL: https://surl.li/wdjoow (дата звернення: 02.03.2025 р.).

11. Digital Developers Committee. Дослідження ринку веб-розробки в Україні. URL: https://surl.li/qmzzau (дата звернення: 02.03.2025 р.).

12. Компанія Globally Cool та Klucs L. The European market potential for web design and web application development services. URL: https://surl.li/bjfvto (дата звернення: 02.03.2025 р.).

13. Filho S. N. Best practices for using DTOs (Data Transfer Objects) in a clean architecture. URL: https://surl.li/vzulxb (дата звернення: 05.05.2025 р.).

14. Куперштейн Л., Кренцін М., Малиновський В. Аналіз методів підвищення захищеності піринговых мереж. URL: https://surli.cc/qkftwf (дата звернення: 05.05.2025 р.).

15. Jayaraman K. D., Siddharth Er. Harnessing the Power of Entity Framework Core for Scalable Database Solutions. URL: https://surl.li/mphnyv (дата звернення: 05.05.2025 р.).

16. Документація Auth0. OpenID Connect Protocol. URL: https://surl.lu/ninfjn (дата звернення: 12.05.2025 р.).

17. Документація Auth0. Change Password Script Templates. URL: https://surl.li/vbdhdz (дата звернення: 12.05.2025 р.).

18. Документація Auth0. Signing Keys. URL: https://surl.li/lzehte (дата звернення: 12.05.2025 р.).

19. Документація Auth0. Continuous Session Protection. URL: https://surl.li/wsjdqx (дата звернення: 12.05.2025 р.).

ДОДАТКИ

ДОДАТОК А

Тези доповіді на науково-практичній конференції

Міністерство освіти і науки України
Волинська обласна рада
Луцька міська рада
Луцький національний технічний університет
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»
Національний університет «Львівська політехніка»
Військовий інститут телекомунікацій та інформатизації імені Героїв Крут (м. Київ)
Тернопільський національний педагогічний університет імені В. Гнатюка Рівненський державний гуманітарний університет
Навчально-науковий інститут «Українська інженерно-педагогічна академія» Харківського національного університету ім. В.Н. Каразіна
Люблінська політехніка (Польща)
Фрайберзька гірнича академія (Німеччина)
Гронінгенський університет (Нідерланди)
Кавказький університет (Грузія)
Політехнічний університет Браганси (Португалія)

ТЕЗИ ДОПОВІДЕЙ
X Міжнародної науково-практичної конференції з проблем вищої освіти і науки «Інформаційні технології в освіті, науці і виробництві (ІТОНВ-2025)

23-24 травня 2025 р.
Луцьк 2025

UDC 004.738.5

DESIGN AND DEVELOPMENT OF A WEB-PLATFORM FOR CAR SELLING USING ANGULAR, ASP.NET, MVC AND ENTITY FRAMEWORK

Bakanina Anna Oleksandrivna
Lutsk National Technical University, Student of the department of Software Engineering, bakaninaanya@gmail.com
Paweł Komada
Politechnika Lubelska, D.Sc. Ph.D. Eng
Povstiana Yullia Slavomirivna
Lutsk National Technical University, Ph.D., Associate Professor, Department of Software Engineering

This paper explores the development of a web platform for posting car sale advertisements based on Angular, ASP.NET MVC, and Entity Framework. The study outlines current industry trends, analyses existing platforms, and proposes a user-friendly, secure, and technologically advanced system that meets the expectations of both end users and administrators.

Keywords: *web application, car sale platform, Angular, ASP.NET MVC, Entity Framework, e-commerce, software development.*

The automotive market is one of the most dynamic sectors in e-commerce. In the context of the rapid development of digital technologies, online platforms for buying and selling vehicles play a significant role, offering users the ability to post advertisements, browse available offers, and access services like VIN checks or valuations. However, existing solutions don't always meet modern standards of convenience and functionality, which highlights the need for new web solutions built on up-to-date technologies.

The analysis of existing web platforms, such as Auto.Ria, RST.ua, AutoScout24.com, and Otomoto.pl, revealed that one of the main challenges is ensuring high performance, ease of use, and data security for users. Although most existing platforms have closed-source code, academic publications have made it possible to identify the key technological trends in web development.

Modern web application development relies heavily on advanced frontend frameworks. Among them, Angular stands out for enabling the creation of dynamic and interactive user interfaces. In the study "Comparative Analysis of Angular, React, and Vue.js in Single Page Application Development" by Emmanni P. S., the author examines the advantages and disadvantages of the most popular frontend frameworks, highlighting Angular's effectiveness in developing scalable web applications [1]. On the backend, ASP.NET MVC paired with Entity Framework provides a solid architectural foundation, effective database management, and support for modular, testable code. The research article "A Comprehensive Analysis of Architectural Patterns in ASP.NET Core WEB Application Development" by F. Mushica and A. Memeti discusses the benefits of using ASP.NET MVC for building scalable web systems. It emphasizes several additional features, such as simplified API creation and improved testing facilitated by the modular structure of the MVC pattern [2]. Moreover, the use of Object-Relational Mapping (ORM) frameworks such as Entity Framework brings additional advantages, including increased development efficiency, enabling streamlined CRUD operations, supporting automatic database updates in response to changes in the data model, and allowing the creation and management of migrations [3]. These technologies together provide high performance, ease of development, and scalability for the platform.

The platform architecture is based on the principles of a three-tier architecture, consisting of the following components: frontend, backend and database.

The database is implemented using Microsoft SQL Server and includes several core tables: Vehicles, TechState, CarBrands, CarModels, and AdvertisementStatuses. The Vehicles table is central and contains foreign keys to related entities (Fig. 1). This structure

ensures data consistency and allows for advanced filtering and sorting of advertisements.

The frontend is developed using Angular and is responsible for the presentation layer and user interaction. Bootstrap is used extensively for styling and layout, allowing quick and consistent design through pre-defined CSS classes. A key aspect of the frontend implementation is the use of data binding, which ensures real-time synchronization between the component's model and its HTML template. Angular services play a crucial role in separating data access logic from the component logic. For instance, BrandService is used to fetch vehicle brands and models, while VehicleService handles more complex operations, such as creating, updating, and deleting listings.

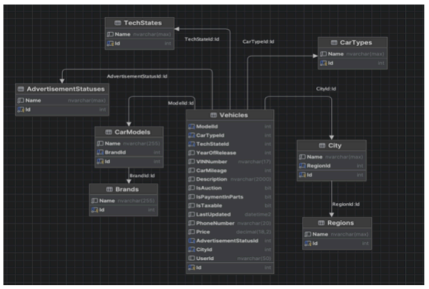

Figure 1 – Database schema

The backend is implemented using ASP.NET Core, which adheres to the Model-View-Controller (MVC) architectural pattern. To ensure clean architecture and secure data exposure, the backend uses Data Transfer Objects (DTOs). These are implemented via resource classes to decouple internal domain logic from the API interface. For example, when creating a vehicle, the SaveVehicleResource is used, which contains only primitive fields.

However, if detailed listing information needs to be fetched, the VehicleResource is returned, which contains nested DTOs. The transformation between domain models and DTOs is managed by the AutoMapper library, which provides centralized mapping logic. Controllers, inheriting from ControllerBase, define API endpoints with methods decorated by attributes such as [HttpGet], [HttpPost], or [HttpPut], and they handle all incoming HTTP requests.

To ensure secure access and role-based permissions, the platform integrates the Auth0 authentication and authorization service. Auth0 guarantees secure user identity management, supports multi-factor authentication, and protects against common vulnerabilities such as unauthorized access and session hijacking [4].

The main functional capabilities of the platform include the ability for users to post advertisements for vehicle sales by providing vehicle specifications, price, and photos. The system supports advanced filtering options, allowing users to narrow down listings based on criterias such as brand, year of manufacture, price range, and car type. Users can view detailed advertisements, explore images. An administrative panel has been developed for platform administrators, enabling them to efficiently manage the system. This panel provides access to key functionalities such as viewing reports, managing advertisements listings, and handling user accounts.

The development of a platform for posting vehicle sale advertisements based on modern web technologies such as Angular, ASP.NET MVC, and Entity Framework demonstrates a modern and scalable approach to web application development. The choice of technologies was made based on their proven effectiveness in creating responsive user interfaces, maintainable backend logic, and reliable database operations. In the future, the platform can be expanded with additional features such as chat between users, integration with payment systems, and mobile app support. The current architecture and technology stack ensure that such

improvements can be implemented with minimal rework, supporting the long-term growth and relevance of the system.

References

1. Emmanni P. S. (2023). Comparative Analysis of Angular, React, and Vue.js in Single Page Application Development. URL: https://www.researchgate.net/publication/380291017_Comparative_Analysis_of_Angular_React_and_Vuejs_in_Single_Page_Application_Development
2. Mushica F., & Memeti A. (2024). A Comprehensive analysis of architectural patterns in ASP.NET Core WEB application development. URL: https://eprints.unite.edu.mk/1949/1/revista%20-%202024-207-218.pdf
3. Jayaraman K. D., & Siddharth Er. (2025). Harnessing the Power of Entity Framework Core for Scalable Database Solutions. URL: https://www.researchgate.net/publication/390200510_Harnessing_the_Power_of_Entity_Framework_Core_for_Scalable_Database_Solutions
4. Singh J. (2022). OAuth 2.0 : Architectural design augmentation for mitigation of common security vulnerabilities. URL: https://www.sciencedirect.com/science/article/abs/pii/S2214212621002684

ДОДАТОК Б

Тест-кейси для тестування веб-платформи

№	Назва тест-кейсу	Опис дій користувача	Очікуваний результат
Функціональні тест-кейси			
1.	Реєстрація нового користувача	1. Перейти на сторінку реєстрації 2. Ввести валідні email і пароль 3. Підтвердити реєстрацію	Користувача зареєстровано, відкрито особистий кабінет
2.	Логін користувача	1. Перейти на сторінку входу 2. Ввести валідні облікові дані	Користувач авторизований, редирект до головної сторінки
3.	Редагування профілю	1. Увійти в систему 2. Перейти до особистого кабінету 3. Змінити ім’я 4. Натиснути «Зберегти»	Дані збережено, відображається повідомлення про успіх
4.	Додавання нового оголошення (авторизований користувач)	1. Увійти в систему 2. Перейти до форми створення оголошення 3. Заповніть всі обов’язкові поля 4. Натиснути «Зберегти»	Оголошення збережене, з’являється у списку оголошень користувача та в каталозі
5.	Спроба додати оголошення (не авторизований користувач)	1. Увійти в систему 2. Перейти до форми створення оголошення 3. Заповніть всі обов’язкові поля 4. Натиснути «Зберегти»	Оголошення збережене, з’являється в каталозі
6.	Перегляд профілю без входу	1. Відкрити URL профілю без авторизації	Редирект на сторінку входу
7.	Реєстрація нового користувача через Google аккаунт	1. Перейти на сторінку реєстрації 2. Натиснути «Увійти через Auth0» 3. Авторизуватися	Успішна реєстрація та вхід, відображення відповідного інтерфейсу

№	Назва тест-кейсу	Опис дій користувача	Очікуваний результат
Граничні значення			
8.	Реєстрація з порожнім email	1. Відкрити форму реєстрації 2. Залишити поле email порожнім 3. Натиснути «Зареєструватися»	Поява повідомлення про помилку: «Поле обов’язкове»
9.	Довге ім’я (256 символів)	1. В особистому кабінеті ввести ім’я з 256 символів 2. Зберегти	Помилка валідації або повідомлення про обмеження
10.	Невалідний email	1. Ввести email без @ (наприклад, test.com) 2. Зареєструватися	Відображення помилки: «Невалідний email»
Негативні сценарії			
11.	Створення оголошення без заповнення обов’язкових полів	1. Відкрити форму 2. Не заповнювати поля 3. Натиснути «Зберегти»	Відображення помилок біля кожного незаповненого поля
12.	Вхід з неправильним паролем	1. Ввести існуючий email і неправильний пароль	Відображення помилки: «Невірний email або пароль»
13.	Повторна реєстрація з тим самим email	1. Ввести email, що вже використовується 2. Натиснути «Зареєструватися»	Повідомлення: «Користувач з таким email вже існує»
Тестування інтерфейсу			
14.	Перевірка адаптивності	1. Відкрити сайт на мобільному пристрої або емуляторі	Елементи інтерфейсу адаптовані, меню доступне
15.	Відображення повідомлення про помилку	1. Виконати дію, яка викликає помилку (напр. неправильний пароль)	Виводиться інформативне повідомлення