МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

на тему: **«Проєктування та розробка онлайн-сервісу для підбору та купівлі автомобілів вторинного ринку США»**

Виконав: студент 4 курсу, групи КН-42
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Радчук Вадим Андрійович

Керівник: ***Клебан Ю.В., старший викладач кафедри ЕММІТ***

Рецензент: *кандидат технічних наук, доцент,*
доцент кафедри прикладної математики та кібербезпеки
Донецького національного університету імені Василя Стуса
Загоруйко Любов Василівна

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від «30» травня 2024 р.

Острог, 2024

Міністерство освіти і науки України
Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу студента

Радчука Вадима Андрійовича

1. Тема роботи: ***Проєктування та розробка онлайн-сервісу для підбору та купівлі автомобілів вторинного ринку США.***

керівник роботи: Клебан Ю.В., старший викладач кафедри ЕММІТ

Затверджено наказом ректора НаУОА від 03.11.2023 р., № 98.

2. Термін здачі студентом закінченої роботи: 31 травня 2024 року.

3. Вихідні дані до роботи: для створення проєкту використовувались: Visual Studio 2022 (для створення бекенд частини), Visual Studio Code (для фронтенд частини), GitHub, API Ninjas. Стек технологій: Entity Framework Core, JWT, ASP.NET Core, Swagger, Web API, SQL Server, C#, Angular, Bootstrap, TypeScript.

4. Перелік завдань, які належить виконати: реалізувати фронтенд онлайн-сервіс з використанням Angular та Bootstrap, реалізувати бекенд додатка з використанням ASP.NET Core, реалізувати чисту архітектуру, реалізувати базу даних на SQL Server.

5. Перелік графічного матеріалу: таблиці, рисунки, лістинги, діаграми.

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Клебан Ю. В.	01.12.2023	01.12.2023
2	Клебан Ю. В.	01.12.2023	01.12.2023
3	Клебан Ю. В.	01.12.2023	01.12.2023

7. Дата видачі завдання: 01.12.2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми роботи/проєкту	до 31.10.2023 р.	
2	Постановка технічного завдання	до 01.12.2023 р.	
3	Ознайомлення з документацією	до 10.12.2023 р.	
4	Написання розділу 1	до 01.02.2024 р.	
5	Написання розділу 2	до 01.04.2024 р.	
6	Тестування системи	до 20.04.2024 р.	
7	Виправлення помилок	до 01.05.2024 р.	
8	Попередній захист та перевірка на рівень унікальності кваліфікаційної роботи/проєкту	до 31.05.2024 р.	
9	Здача кваліфікаційної роботи/проєкту на кафедру	31.05.2024 р.	

Студент: _____ **Вадим РАДЧУК**
(підпис)

Керівник кваліфікаційної роботи: _____ **Юрій КЛЕБАН**
(підпис)

АНОТАЦІЯ
кваліфікаційної роботи
на здобуття освітнього ступеня бакалавра

Тема: Проєктування та розробка онлайн-сервісу для підбору та купівлі автомобілів вторинного ринку США

Автор: Радчук Вадим Андрійович

Науковий керівник: Клебан Ю.В., старший викладач кафедри ЕММІТ

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *92 с., 31 рис., 1 табл., 7 додатків,29 джерел.*

Ключові слова: *онлайн-сервіс, бекенд, фронтенд, архітектура, API*

Короткий зміст праці:

Завданням кваліфікаційної роботи/проєкту, було розробка вебзастосунку для підбору та купівлі автомобілів вторинного ринку США "AutoMarket". Додаток повинен надавати зручний інтерфейс для пошуку авто та безпеку даних. Головними користувачами онлайн-сервісу"AutoMarket" будуть люди, які бажають придбати автомобіль з США. Фронтенд частина створювалась задопогою Angular та Bootstrap на TypeScript. Бекенд частина створювалась на ASP.NET Core. При розробці дотримувались правила і рекомендації створення чистої архітектури.

Keywords: *online service, backend, frontend, architecture, API*

Summary of the work:

The objective of the qualification work/project was to develop a web application for selecting and purchasing used cars in the USA "AutoMarket". The application should provide a user-friendly interface for car search and data security. The main users of the online service "AutoMarket" will be people who want to buy a car from the USA. The frontend part was created using Angular and Bootstrap on TypeScript. The backend part was created on ASP.NET Core. During development, the rules and recommendations for creating a clean architecture were followed.

ЗМІСТ

ВСТУП

В наш час, коли технології стрімко розвиваються, автомобільний ринок має значний вплив на наше життя, пропонуючи нові можливості та рішення. З появою Інтернету процес вибору та купівлі автомобіля став значно простішим та зручнішим. Проте, велика кількість інформації та безліч варіантів можуть ускладнити цей процес для покупців. Розробка онлайн-сервісу для підбору та купівлі вживаних/нових автомобілів на вторинному ринку США полегшує цю задачу, роблячи процес вибору авто зручним та ефективним.

Світовий ринок вживаних автомобілів налічує мільярди доларів, а ринок США є одним з найбільших та найдинамічніших. Для українців, які прагнуть придбати вживаний автомобіль в США, існує ряд проблем, таких як мовний бар'єр, незнання особливостей американського ринку, складність оформлення документів та ризик шахрайства. Ця кваліфікаційна робота прагне розробити онлайн-сервіс, який спростить та покращить процес підбору та купівлі вживаних автомобілів для українців, зробивши його більш доступним, зручним та безпечним.

Мета написання роботи полягає у проектуванні та розробці онлайн-сервісу під назвою “AutoMarket”, який полегшить та покращить процес підбору та купівлі вживаних автомобілів в США для українців. Для досягнення мети потрібно виконати ряд завдань:

- Провести аналіз ринку онлайн-сервісів для підбору та купівлі вживаних автомобілів в США;
- Спроектувати систему:
  - Вибір технологій для розробки вебдодатку
  - Спроектувати базу даних;
  - Спроектувати API;
  - Спроектувати інтерфейс користувача (UI);
- Розробити систему згідно плану.

Об’єкт дослідження є онлайн-сервіс для підбору та купівлі вживаних автомобілів з США.

Предметом дослідження є проєктування та розробка онлайн-сервісу для підбору та купівлі автомобілів на вторинному ринку з США, який полегшить та покращить процес вибору та купівлі вживаних авто. Основна увага приділяється використанню ASP.NET Core Web API, як серверна частина, що дає змогу для розробки надшвидких API та Angular, як клієнтська частина для створення функціональності, яка враховує потреби користувачів та забезпечує ефективний процес вибору автомобілів.

Отже, створення онлайн-сервісу для підбору та купівлі вживаних авто з США стає значним кроком на шляху до спрощення процесу вибору авто для споживачів. Цей сервіс, забезпечуючи зручність, економію часу та доступ до американського авторинку, відкриває нові можливості для покупців, роблячи їх вибір більш обґрунтованим та приємним. Використання сучасних технологій, таких як ASP.NET Core Web API та Angular, гарантує високий рівень функціональності та покращує користувацький досвід. Успішне впровадження цього сервісу відкриває шлях до ефективного вибору авто на вторинному ринку США.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ОНЛАЙН-СЕРВІСУ ДЛЯ ПІДБОРУ ТА КУПІВЛІ АВТОМОБІЛІВ

1.1. Дослідження вторинного ринку автомобілів з США

Згідно з дослідженням вторинного ринку авто в Україні, спостерігається значне зростання попиту на американські автомобілі. Ось факт: 23,5% складають імпортовані автомобілі вторинного ринку в Україні і це авто, яким більше 5 років [11]. Цього року ціни на американських аукціонах знизилися від 20% до 50% [12] ,що робить придбання авто з США ще більш вигідним. На першому графіку (Рис. 1.1.) можна констатувати, що імпорт вживаних легкових авто віком понад 5 років в Україну з 2018 року почав збільшуватися.

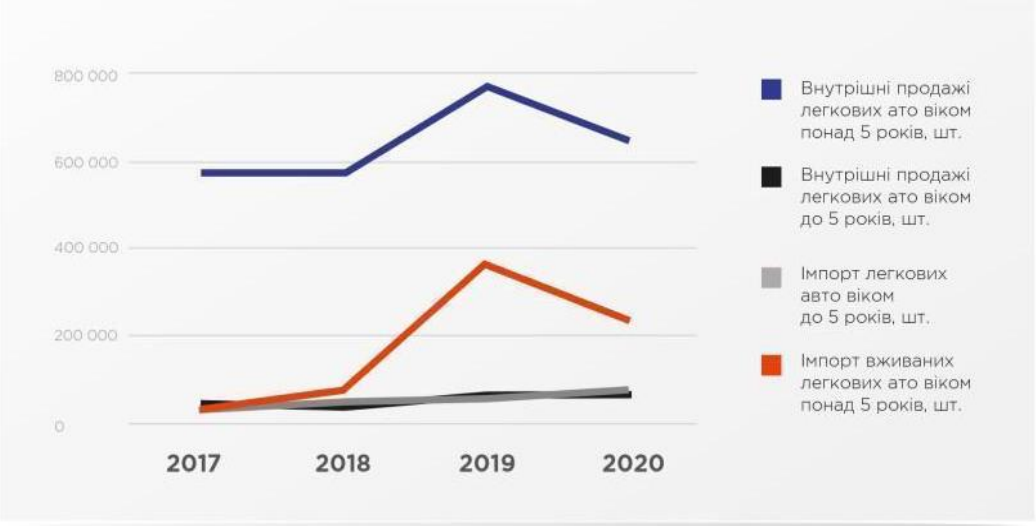

Рис. 1.1. Графік динаміки вторинного ринку легкових автомобілів в Україні з 2017 по 2020 роки
Джерело:[11]

Графік 2 (Рис. 1.2.) демонструє значні коливання обсягів імпорту автомобілів в Україну протягом 2019 і 2020 років. Зосередимося на детальному аналізі цих даних та ймовірних причин динаміки.

Зменшення імпорту у 2020 році:

- Спостерігається суттєве зниження обсягів імпорту з 440 176 авто у 2019 році до 385 853. Цей спад зумовлений комплексом факторів, зокрема:
    - **Економічні умови:** світова економіка пережила рецесію у 2020 році, що негативно вплинуло на купівельну спроможність та інвестиційні можливості.
    - **Глобальні торговельні та логістичні обмеження:** торгові війни, введення мит та інші бар'єри для торгівлі ускладнили та подорожчали міжнародні перевезення.
    - **Пандемія COVID-19:** карантинні заходи та обмеження на пересування суттєво вплинули на виробництво, логістику та дилерські мережі по всьому світу.

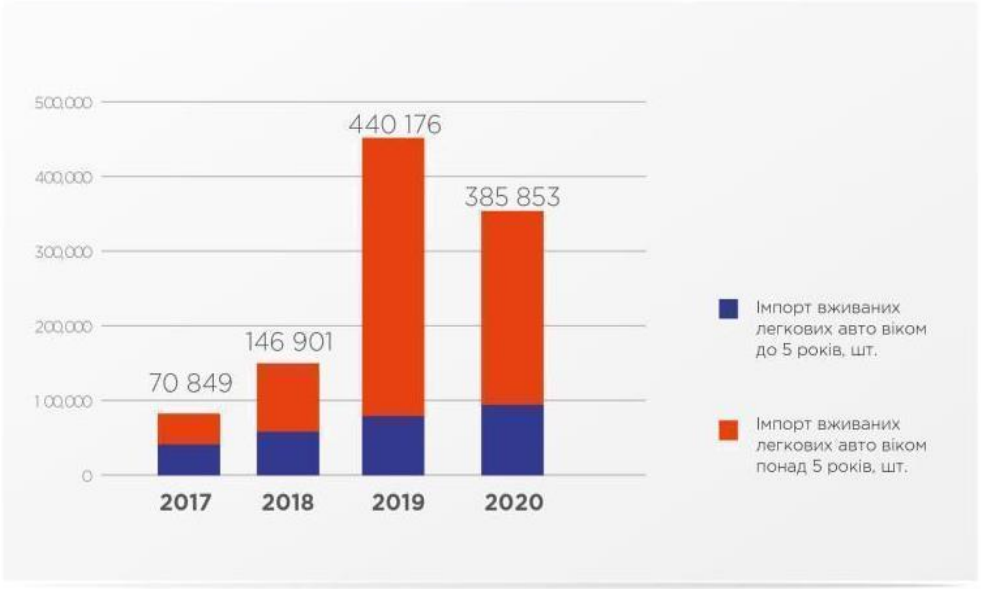

Рис. 1.2. Графік динаміки імпорту вживаних авто в Україні протягом 2017-2020 років

Джерело:[11]

Важливо зазначити, що з 2022 року, у зв'язку з повномасштабним вторгненням РФ в Україну, структура та обсяги імпорту авто зазнали значних змін:

- Зростає попит на вживані позашляховики та броньовані авто для потреб ЗСУ та тероборони.
- Багато волонтерів імпортують авто з-за кордону, зокрема з США.

- Ці дані не відображені на графіку 2, оскільки представляють собою нову тенденцію, що виникла вже після 2020 року.

Лише за перші два місяці війни для потреб українських захисників імпортували 10 700 автомобілів [17].

Одним з ключових факторів, що дають перевагу автомобілів з США є те, що на американському ринку найновіші моделі з'являються раніше ніж в решті світу і тому американські автовласники змінюють машини найчастіше тому з огляду на це на вторинному ринку США є безліч пропозицій вживаних автомобілів різних марок, станів та віку [12].

Отже, ринок вживаних авто в Україні демонструє стрімке зростання попиту на американські авто, що зумовлено низкою факторів: широкий вибір то доступ до нових моделей, якість та вигідна ціна, часта зміна авто американцями. Авторинок США стає все більш привабливим для українських покупців, пропонуючи широкий вибір якісних авто за вигідними цінами.

1.2. Клієнт-серверна архітектура, як основа для розробки вебсервісів

Клієнт-серверний додаток під собою означає клієнт-серверну архітектуру. Термін "клієнт-сервер" є однією з архітектурних моделей програмного забезпечення та найкращою концепцією для створення розподілених мережевих додатків, що передбачають взаємодію та обмін даними між клієнтом та сервером [21]. Простими словами клієнтська частина робить запит, а сервер відповідає. Популярність архітектури клієнт-сервер зумовлена динамічним розвитком Інтернету та концепцією великих обсягів інформації в базах даних на серверах [22].

Вебсервер (сервер) – це об'єкт, що дає сервіс іншим об'єктам мережі за їх запитами. Після виконання кожного завдання сервер надсилає отримані результати клієнту, який відправив завдання [1].

Сервер обробляє завдання клієнтської частини та керує виконанням завдань. Коли кожне завдання виконано, сервер надсилає результати для клієнтської частини, яка надіслала завдання. Сервер приймає HTTP запити та віддає HTTP відповіді, які можуть містити в собі текст, зображення та інше [4].

В розробленому онлайн-сервісі сервер виступає, як подання REST API для отримання інформації на базі ASP.NET Core Web API.

REST API – це стиль архітектури API для розподілених систем. REST визначає, як дані надаються клієнтській частині в зручному для нього форматі. Обмін даними відбувається у форматі JSON або XML (хоча сьогодні більш популярний формат JSON) [16].

Основна ідея цього стану полягає в тому, що ресурси передаються разом з їхнім станами та зв’язками за допомогою заздалегідь визначених операцій.

Функції, які реалізуються на стороні сервера:

- Відправлення результатів на сторону клієнтської частини
- Оброблення запитів від клієнтської частини
- Зберігання, захист та доступ до даних

Отже, сервер – це не просто одна з деталей вебзастосунку, а його основа, без якої неможливо уявити сучасний Інтернет. Його роль полягає в тому, щоб зберігати дані, обробляти запити та забезпечувати доступ до вебсайтів для користувачів. Наступна частина клієнт-серверної архітектури є клієнтська частина.

Клієнтська частина (клієнт) – комп’ютер на стороні користувача, який використовує ресурси сервера і надає зручний інтерфейс користувача. Інтерфейси користувача - це процедури взаємодії користувача з системою або мережею [1]. Для розробки ми будемо використовувати front-end фреймворк Angular.

Angular – написаний на TypeScript front-end фреймворк з відкритим кодом [5], який розробляється під керівництвом Angular Team у компанії Google, а також спільнотою приватних розробників та корпорацій.

TypeScript – мова програмування, представлена Microsoft восени 2012; позиціонується, як засіб розробки вебзастосунків, що розширює можливості JavaScript [8].

Angular CLI (cli – command-line interface) – npm-модуль, реалізуючий інтерфейс командного рядку для створення, розробки і підтримка Angular онлайн-сервісів [23].

Функції, які реалізуються на стороні клієнта:

- Надання користувацького інтерфейсу
- Формулювання запиту до сервера і його відправка
- Отримання результатів запиту від сервера таких як (GET, POST, PUT, PATCH, DELETE)

Таким чином, архітектура клієнт-сервер описує всю взаємодію між клієнтом, який створює запити та сервером, який обробляє створені запити клієнтом і відправляє відповіді. Цей підхід став основою для розробки розподілених мережевих додатків, особливо в міру того, як розширюється Інтернет і збільшується кількість інформації в БД на серверах.

1.3. Конкуренти розроблюваного сервісу

У світі великої кількості інформації та технологій, що стрімко розвиваються, існують різні онлайн-сервіси що полегшують процес купівлі автомобіля, особливо на вторинному ринку. Деякі онлайн-сервіси зосереджуються на аналізі технічних характеристик інші – на історії та технічному стані автомобіля.

“AutoMarket” пропонує комплексний підхід, що поєднує аналіз технічних параметрів автомобілів, аналіз історії експлуатації та оцінку потреб індивідуальних користувачів. Підхід має на меті надати повну інформацію та врахувати всі аспекти при виборі автомобіля.

На ринку є багато онлайн-сервісів для купівлі авто з США таких як: W8 Shipping, AUTO RIA, BOSS AUTO, які мають свої переваги та недоліки. Для забезпечення конкурентоспроможності для власного проєкту було зроблено невеликий аналіз всіх сайтів, які є зазначені вище.

W8 Shipping (Рис. 1.3.), онлайн сервіс з продажу авто з вторинного ринку США, має багато функцій такі як:

- Функціонал, який включає фільтрацію;
- гарячу лінію;
- калькулятор доставки, дозволяє забезпечити зручність користувачів.

Як конкурент на ринку продажу автомобілів з США, W8 Shipping має низку переваг і недоліків порівняно з іншими онлайн-сервісами для продажу автомобілів.

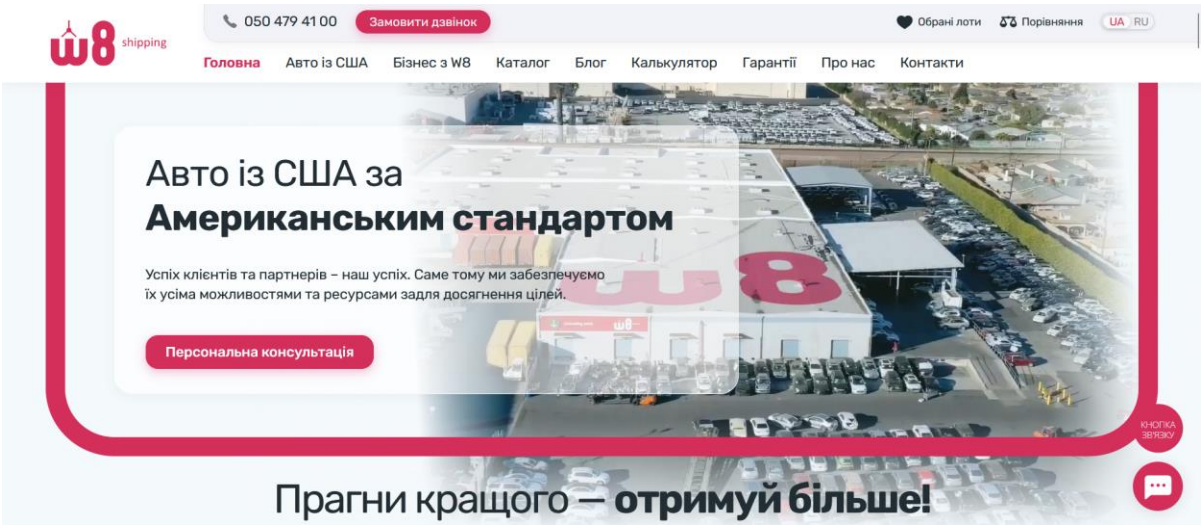

Рис. 1.3. UI дизайн сайту “W8 Shipping”

Джерело:[13]

Переваги W8 Shipping на ринку полягають у наступному:

- Потужний функціонал: W8 Shipping надає калькулятор для розрахунку доставки з США та розраховує ціну ремонту “під ключ”.
- Широкий вибір транспортних засобів: W8 Shipping пропонує широкий вибір транспортних засобів на будь-який бюджет і потребу.
- Прозорий процес покупки: W8 Shipping прагне забезпечити прозорий процес покупки, надаючи користувачам всю інформацію, необхідну для прийняття обгрунтованого рішення.

Недоліки W8 Shipping на ринку полягають у наступному:

- Складність знайти потрібну інформацію: при переході на сайт перед нами з’являється занадто багато тексту на головній сторінці і при перечитуванні користувач може просто втратити інтерес і відчувати себе перевантаженим.

Отже, W8 Shipping складає велику конкурентоспроможність і пропонує потужний функціонал з калькуляторами доставки та ремонту «під ключ», широкий вибір авто та прозорий процес покупки. Але занадто багато тексту на головній сторінці ускладнює пошук потрібної інформації.

AUTO RIA (Рис. 1.4.) – український популярний онлайн-сервіс для продажу та купівлі автомобілів не тільки з США, а й з усієї України. Вміщує в собі багатий функціонал та має багато переваг та кілька недоліків

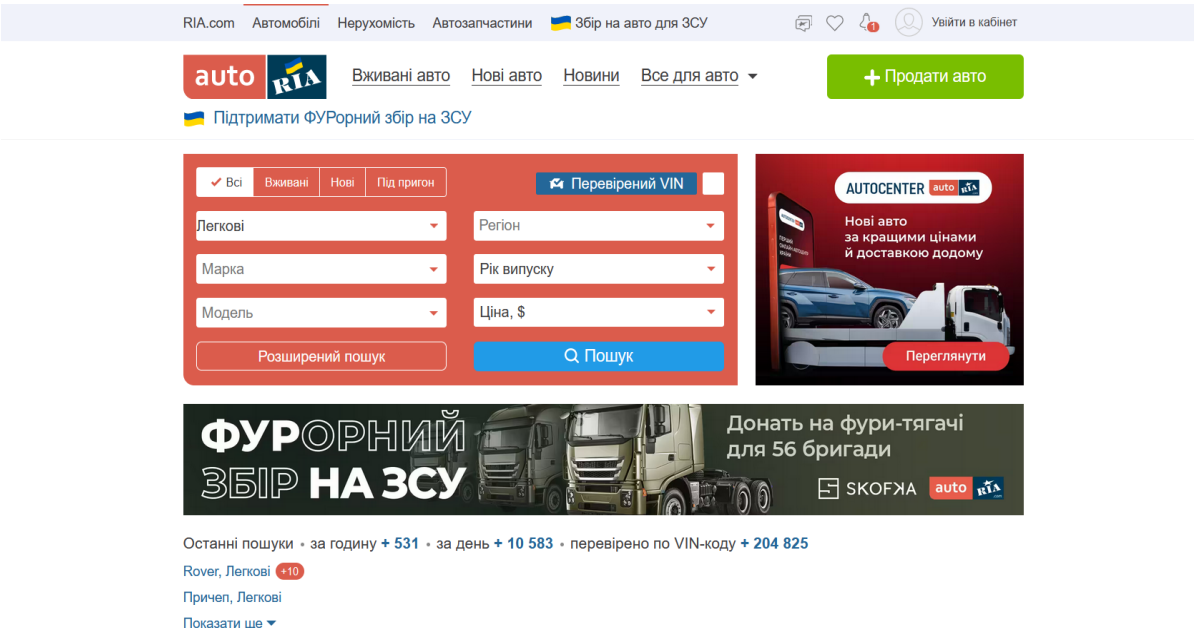

Рис. 1.4. UI дизайн сайту “AUTO RIA”

Джерело:[14]

Переваги AUTO RIA на ринку полягають у наступному:

- Розширена база даних: AUTO RIA пропонує широкий вибір вживаних автомобілів різних марок, моделей, років випуску, цін та пробігів. Ви можете знайти як популярні бюджетні авто, так і ексклюзивні та рідкісні моделі.

- Розширений функціонал пошуку та фільтрації: пошук автомобілів за різними параметрами, включаючи марку, модель, рік випуску, ціну, пробіг, тип кузова, тип трансмісії, колір, тип палива, країну походження та багато іншого.
- Велика база клієнтів: AUTO RIA має мільйони активних користувачів, які шукають вживані автомобілі. Це робить його чудовим місцем для продажу автомобіля, адже користувач може бути впевнений, що його побачать багато потенційних покупців.

Недоліки AUTO RIA на ринку полягають у наступному:

- Застарілий дизайн сайту: інтерфейс сайту може здатися не сучасним та не інтуїтивно зрозумілим.
- Вартість: AUTO RIA стягує з продавців комісію за продаж їхніх автомобілів. Ця комісія може бути значною, особливо якщо ви продаєте дорогий автомобіль.

Отже, AUTO RIA – це цінний ресурс для купівлі та продажу вживаних автомобілів в Україні. Він має багато переваг і недоліків, але в цілому AUTO RIA – дуже конкурентний ринок.

BOSS AUTO (Рис. 1.5.) – онлайн-сервіс для продажу автомобілів прямо з аукціону та ремонт машини “під ключ”. На даний момент має багатий функціонал та свої недоліки

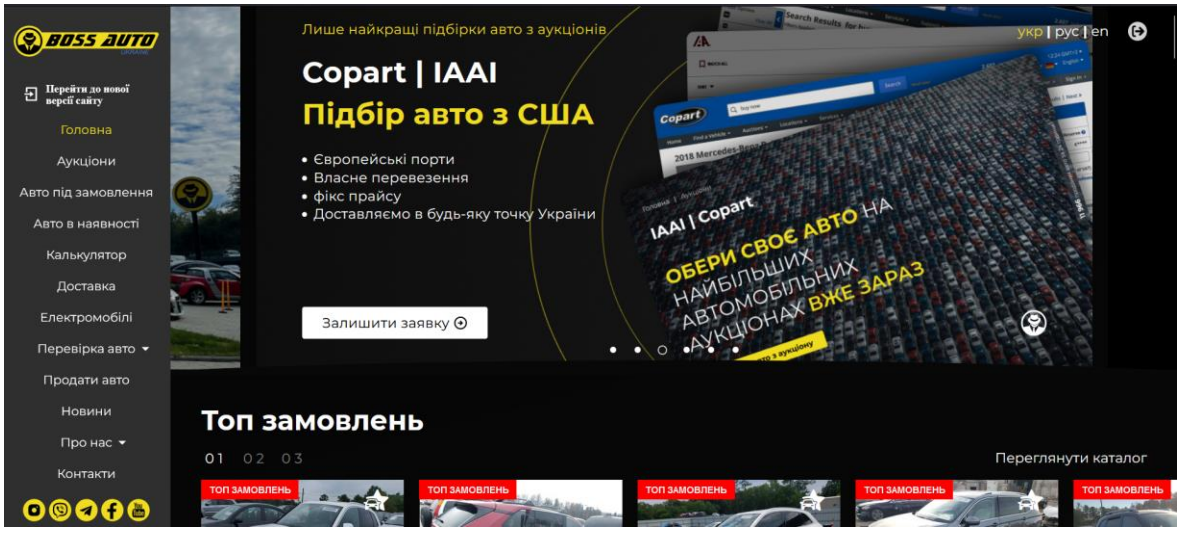

Рис. 1.5. UI дизайн сайту “BOSS AUTO”
Джерело:[15]

Переваги BOSS AUTO на ринку полягають у наступному:

- Калькулятор доставки: цей інструмент дозволяє користувачам розрахувати вартість доставки автомобіля з інших країн.

Недоліки BOSS AUTO на ринку полягають у наступному:

- Простий фільтр: відсутні фільтри за деякими важливими параметрами, такими як тип трансмісії, тип приводу, комплектація та інші.
- Багато реклами: реклама відволікає увагу користувачів від основного контенту сайту, тобто від оголошень про продаж автомобілів.
- Складний інтерфейс: навігація по сайту незручна та не інтуїтивно зрозуміла. Користувачам може бути важко знайти те, що вони шукають.

Загалом, BOSS AUTO - це потенційний конкурент, який пропонує деякі унікальні переваги. BOSS AUTO має деякі унікальні переваги, такі як калькулятор доставки та досвідчена команда. Однак він також має деякі недоліки, такі як простий фільтр, багато реклами та складний інтерфейс. Це робить його менш сильним конкурентом, ніж AUTO RIA та W8 Shipping.

Аналіз конкурентів показав, що на ринку онлайн-сервісів, які займаються продажем автомобілів з США є досить значна конкуренція. Вивчивши функціонал конкурентів, ми можемо запозичити кращі практики та вдосконалити їх, щоб створити власний унікальний продукт, який буде ще більш привабливим для клієнтів. На основі висновків та досліджень був сформований власний SWOT-аналіз оналайн-сервісу “AutoMarket” (Таблиця 1.1.)

Таблиця 1.1.

SWOT-аналіз онлайн-сервісу “AutoMarket”

Сильні сторони	Слабкі сторони
● Зручний інтерфейс ● Безпека персональних даних ● Швидкодія	● Конкуренти розвинутіші в даній сфері ● Недостатньо велика база даних ● Недостатня адаптивність під мобільні пристрої
Можливості	Загрози
● Достатня база даних ● Залучення стороннього API	● Поява нових конкурентів ● Несправність орендованого сервера(вебсайт буде недоступний)

Джерело: [створено автором]

Отже, ми проаналізували наших конкурентів і зробили SWOT-аналіз. Нашому онлайн-сервісу потрібно розробити всі потрібні функції, зробити простий інтерфейс, застосувати сторонній API, зробити найкращу швидкодію.

Висновки до розділу 1

У розділі 1 було досліджено ринок вживаних автомобілів в Україні. Ринок динамічно розвивається, зокрема, зростає попит на американські авто. Це пов'язано з низкою факторів, таких як широкий вибір та доступ до нових моделей, якість та вигідна ціна, часта зміна авто американцями.

Розробка онлайн-сервісу для підбору та купівлі авто з США має значний потенціал. Використання клієнт-серверної архітектури з REST API забезпечить зручну та безпечну взаємодію між користувачем та сервером.

Існують різні підходи до вирішення проблеми вибору та купівлі авто з США. Деякі платформи зосереджуються на аналізі технічних характеристик, інші - на історії та технічному стані автомобіля.

Проаналізовано конкурентів, таких як: W8 Shipping, AUTO RIA, BOSS AUTO. Їхні сильні та слабкі сторони допоможуть у розробці власного унікального продукту, який буде ще більш привабливим для клієнтів.

На основі висновків та досліджень проведено SWOT-аналіз онлайн-сервісу "AutoMarket". Визначено його сильні та слабкі сторони, можливості та загрози.

РОЗДІЛ 2
ПРИКЛАДНА ЧАСТИНА РОЗРОБКИ ВЕБДОДАТКУ

2.1. Серверна частина вебсервісу “AutoMarket”

2.1.1. Загальна інформація про серверну частину

Для розробки сервера, який буде спілкуватися з клієнтською частиною було обрано вебплатформу ASP.NET Core Web API, де використовується REST архітектура. Також доданий Nuget пакет Swagger, який реалізує інтерфейс для перегляду API та дозволяє документувати REST-сервіси.

У цій архітектурі кожна частина інформації унікально ідентифікується за допомогою URL-адреси. Для прикладу, якщо у нас є контролер з усіма автомобілями, то вони визначені в наступному форматі - /cars, а конкретний автомобіль - /cars{id}, де {id} – унікальний ідентифікатор автомобіля.

Для керування інформацією використовується декілька методів: GET, PUT, DELETE, POST – вони реалізують дії зчитування, оновлення, видалення та додавання.

Для розгортання та адміністрування бази даних використовується Microsoft SQL Server Management Studio (SSMS). SSMS - це система управління базами даних, яка розробляється корпорацією Microsoft [28].

Для реалізації зв’язку між базою даних та використання підходу Code first використовується технологія Entity Framework 6. Code First – це підхід для розробки баз даних в EF, який дозволяє створювати базу даних на основі коду класів на сервері. Тобто простими словами спочатку пишеться код, а потім на основі нього працює база даних.

Спочатку для курсової роботи архітектурна модель бекенду була реалізована за допомогою 3-рівневої архітектури. 3-рівнева архітектура – це

підхід, який розділяє функціональність back-end на три рівні, кожен з яких відповідає за певні задачі або взаємодії. Для кваліфікаційної роботи було прийнято рішення перейти на Чисту архітектуру, оскільки він пропонує ряд переваг, таких як незалежність, відсутність залежностей від ORM(Object-Relational Mapping) та простота використання. Також було використано дизайн DDD (Domain-Driven Design). Domain-Driven Design – це методологія розробки програмного забезпечення, яка фокусується на глибокому розумінні бізнес-домену та ефективній структуризації логіки для його вирішення [18]. Це призводить до зниження складності, підвищення гнучкості та якості коду. Основне правило для Clean Architecture - це правило залежності, тобто код на нижніх рівнях не повинен залежати від коду, що знаходиться на вищих рівнях. Основними рівнями в структуру включають: рівень домену, рівень представлення, рівень застосунку та рівень інфраструктури.

Для реалізації архітектури проєкту було використано архітектурний шаблон Generic Repository Pattern. Generic Repository Pattern – це архітектурний шаблон проектування, який використовується для створення рівня доступу до даних для програми. Цей патерн є дуже корисним при розробці онлайн-сервісів, оскільки дозволяє розробникам створювати та підтримувати код рівня даних у загальний спосіб, без повторення коду щоразу. Також був реалізований патерн CQRS(Command Query Responsibility Segregation) для покращення швидкодії та безпеки. CQRS – це стиль архітектури в якій операції читання відокремлені від операцій запису.[25]

Для реалізації CQRS було використано бібліотеку MediatR. MediatR – це бібліотека для реалізації патерну посередника в C#. Вона спрощує впровадження CQRS, надаючи зручний спосіб керування обробниками команд і запитів. MediatR виступає посередником, який направляє команди та запити до репозиторіїв.

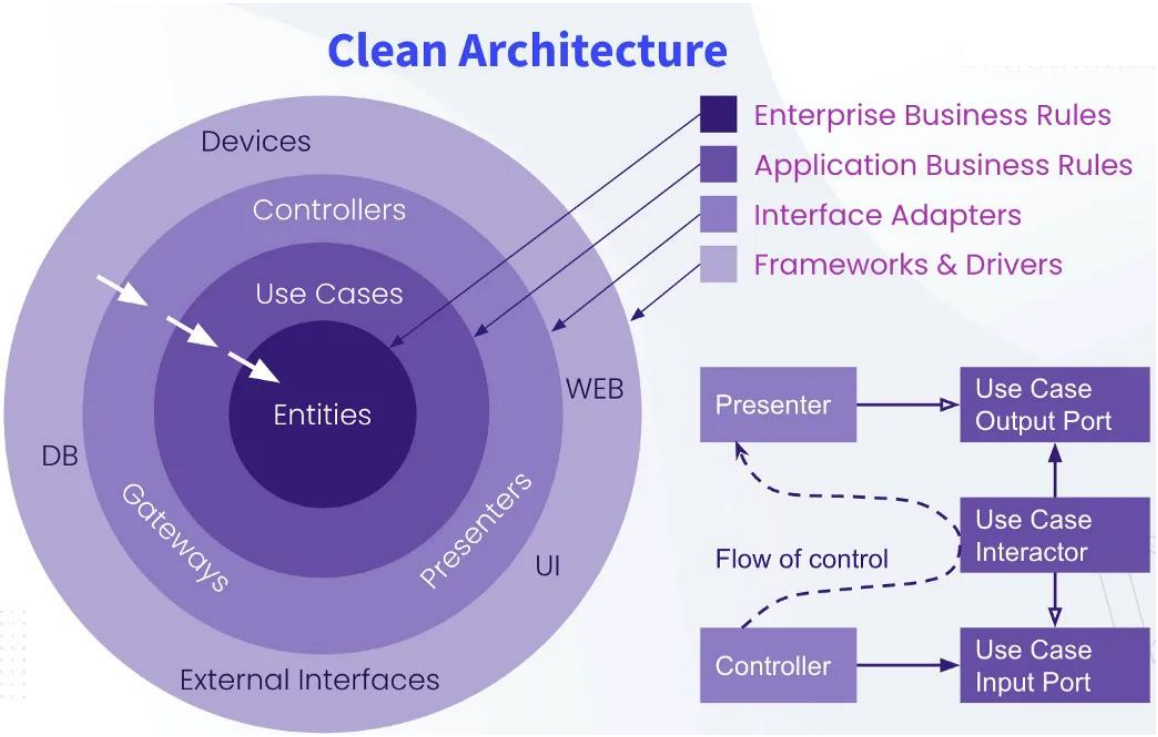

Рис. 2.1. Clean Architecture

Джерело:[19]

Для реалізації аутентифікації було використано Nuget пакети Microsot.AspNetCore.Identity.EntityFrameworkCore та для генерації JWT токенів Microsoft.AspNetCore.Authentication.JwtBearer

AutoMapper – це проста маленька бібліотека, створена для того, щоб вирішувати одну непросту задачу - реалізація співвідношення одного об'єкта до іншого [7].

2.1.2. Створення проєкту

Для розгортання та розробки серверної частини було використано IDE Visual Studio – інтегроване середовище розробки, оскільки це найзручніший інструмент для розробників [24].

При створенні проєкту було обрано .Net Framework 8 версії.

Після створення проєкту потрібно було створити нові проєкти, кожен з яких має свою роль на основі Clean Architecture, DDD та CQRS де:

- Web – рівень представлення. Цей рівень відповідає за взаємодію з користувачем та представлення даних користувачеві. Його основна функція полягає в перетворенні запитів користувачів на команди для рівня застосунку та відображення результатів у зрозумілому для користувача форматі.

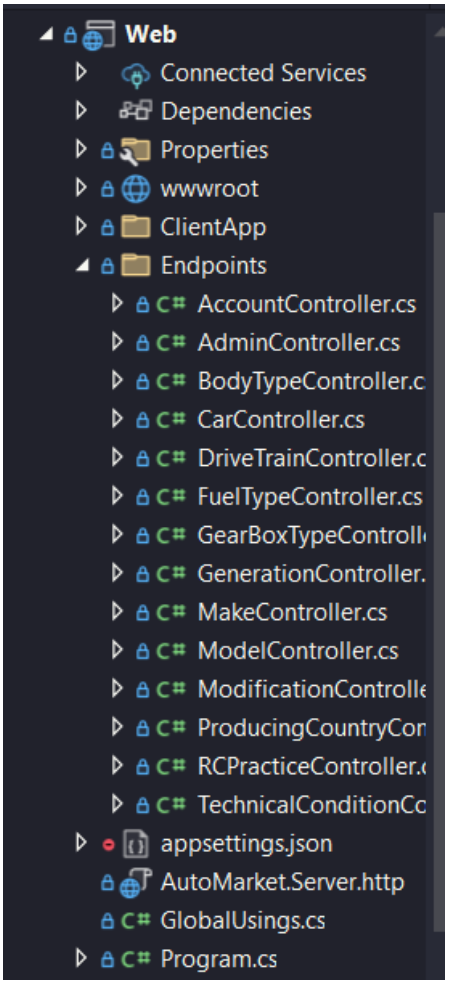

Рис. 2.2. Структура Web

- Application – рівень застосунку. Цей рівень реалізує бізнес-логіку також він не залежить від деталей реалізації інфраструктури та представлення даних. Його основна функція полягає в обробці команд від рівня представлення, виконанні правил бізнесу та генерування результатів для рівня представлення.

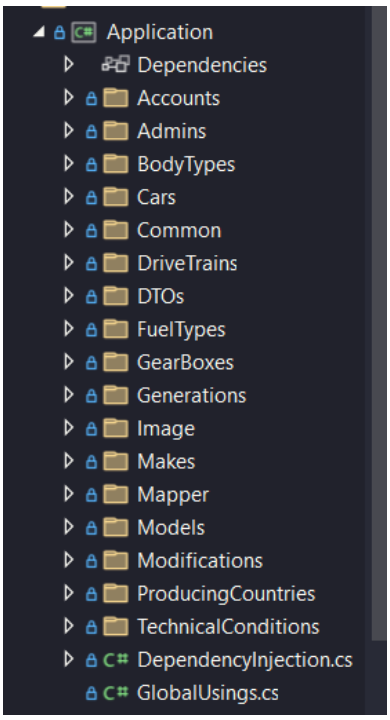

Рис. 2.3. Структура Application

- Domain – рівень домену. Цей рівень відповідає за моделювання та управління бізнес-доменом. Він містить сутності, константи і т.д. Його основна функція полягає в забезпеченні узгодженого та чіткого розуміння бізнес-домену для всіх компонентів нашого онлайн-сервісу.

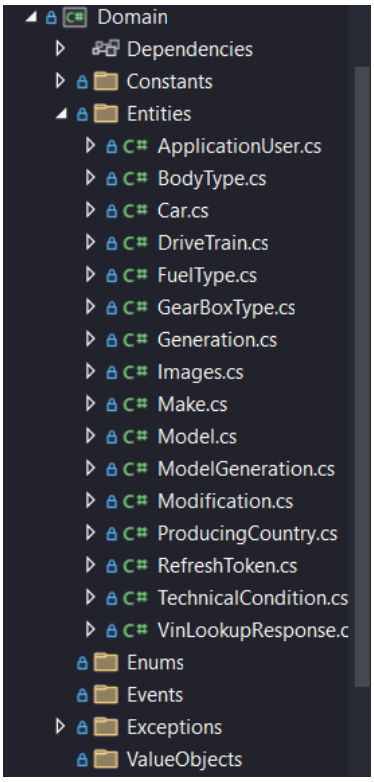

Рис. 2.4. Структура Domain

- Infrastructure – рівень інфраструктури. Цей рівень абстрагує доступ до ресурсів та сервісів, необхідних для роботи нашого сервісу. Він в собі включає БД (підключення до бази даних), репозиторії, міграції. Його основна функція полягає в наданні надійного та ефективного доступу до необхідних ресурсів для інших рівнів нашої системи.

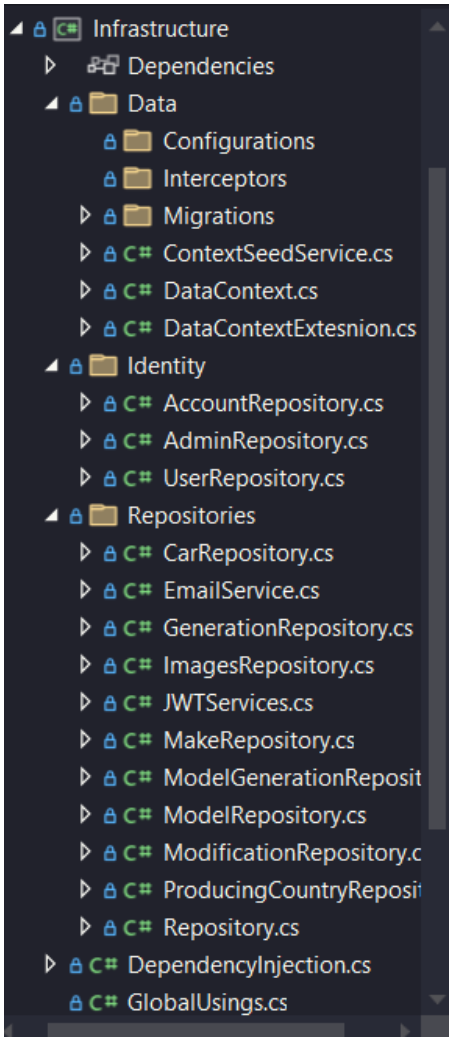

Рис. 2.5. Структура Infrastructure

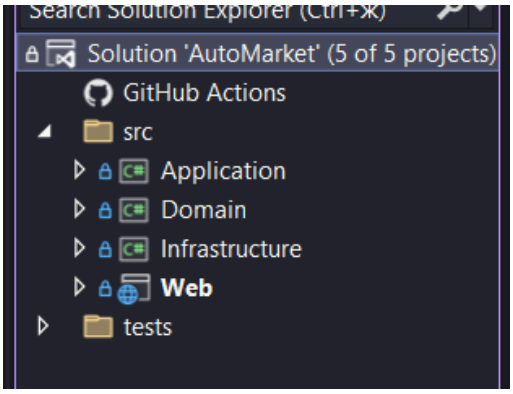

Рис. 2.6. Структура проєкту

Program.cs – це файл конфігурації основного серверу, тут міститься конфігурація аутентифікації та авторизації, підключення до бази даних. Прикладом коду в Program.cs буде додавання рядку для підключення до бази даних та налаштування IdentityCore для користувачів.

Лістинг 2.1. Код файлу Program.cs проєкту Web

```
var connectingString = builder.Configuration.GetConnectionString("AutoMarketConnection");

// Налаштування з`єднання до БД
builder.Services.AddDbContext<DataContext>(option =>
{
    option.UseSqlServer(connectingString);
});

// Налаштування IdentityCore юзера
builder.Services.AddIdentityCore<User>(option =>
{
    // Налаштування пароля
    option.Password.RequiredLength = 6;
    option.Password.RequireDigit = false;
    option.Password.RequireLowercase = false;
    option.Password.RequireUppercase = false;
    option.Password.RequireNonAlphanumeric = false;

    // Налаштування Email
    option.SignIn.RequireConfirmedEmail = true;
})
    .AddRoles<IdentityRole>()
    .AddRoleManager<RoleManager<IdentityRole>>()
    .AddEntityFrameworkStores<DataContext>()
    .AddSignInManager<SignInManager<User>>()
    .AddUserManager<UserManager<User>>()
    .AddDefaultTokenProviders();
```

Джерело: [створено автором]

2.1.3. База даних та моделі

Для реалізації моделі бази даних у нашій роботі було обрано, щоб в базі даних у проєкті було 21 таблиця з яких:

- Car (оголошення) – головна таблиця
- DriveTrain (тип приводу)
- TechnicalConditions (технічні властивості)

- BodyTypes (тип кузова)
- Images (картинки) – виніс я в окрему таблицю тому що оголошення може мати декілька картинок, а не лише одну
- FuelTypes(тип палива)
- Modifications (модифікації)
- Makes (марки)
- Models (моделі)
- ProducingCountries (країна виробник окремої марки)
- Generations (покоління)
- GearBoxes (тип коробки передач)
- Та інші таблиці які зв’язані з автентифікацією та авторизацією

Для реалізації з’єднання з базою даних було використано DataContext.cs. DataContext – клас, який використовується для управління з’єднанням з БД. Цей клас стосується Entity Framework, що дозволяє взаємодіяти з БД за допомогою класів та об’єктів і не потрібно писати SQL-запити вручну.

Цікавим захищеним методом в DataContext є OnModelCreating в якому спочатку визвано метод Seed() для заповнення БД та задано явні зв’язки між таблицями

Лістинг 2.2. Код файлу DataContext.cs проєкту Infrastructure

```
protected override void OnModelCreating(ModelBuilder builder)
{
    builder.Seed();
    builder.Entity<Car>()
        .HasOne(p => p.Modification)
        .WithMany(c => c.Cars)
        .OnDelete(DeleteBehavior.ClientSetNull);
    builder.Entity<ModelGeneration>()
        .HasKey(mg => new { mg.ModelId, mg.GenerationId });
    builder.Entity<ModelGeneration>()
        .HasOne(g => g.Model)
        .WithMany(m => m.ModelGenerations)
        .HasForeignKey(mg => mg.ModelId);
    builder.Entity<ModelGeneration>()
        .HasOne(g => g.Generation)
        .WithMany(m => m.ModelGenerations)
        .HasForeignKey(mg => mg.GenerationId);
}
```

Джерело: [створено автором]

За допомогою підходу Core First створюються класи, та створюються зв’язки між ними такі як: Один-до-Одного (1:1), Один-до-Багато (1:M) та Багато-до-Багато (M:M). Зв’язок багато-до-багато був реалізований за допомогою проміжної таблиці. Ось декілька прикладів таких класів:

Клас Model, який реалізує таблицю Model в БД та має зв’язок багато-до-багато з класом Generation, а проміжна таблиця ModelGeneration

Лістинг 2.3. Код файлу Model.cs проєкту c

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Web.Core.Models
{
    public class Model
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        public string? Name { get; set; }
        public int MakeId { get; set; }

        public Make? Make { get; set; }
        public virtual ICollection<Modification>? Modifications { get; set; }
        public virtual ICollection<ModelGeneration>? ModelGenerations { get; set;
}
        public virtual ICollection<Car>? Cars { get; set; }
    }
}
```

Джерело: [створено автором]

Лістинг 2.4. Код файлу ModelGeneration.cs проєкту Domain

```
namespace Web.Core.Models
{
    public class ModelGeneration
    {
        public int ModelId { get; set; }
        public int GenerationId { get; set; }

        public Model? Model { get; set; }
        public Generation? Generation { get; set; }
    }
}
```

Джерело: [створено автором]

Лістинг 2.5. Код файлу Generation.cs проєкту Domain

```
namespace Web.Core.Models
{
    public class Generation
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
        public string? Name { get; set; }
        public int YearFrom { get; set; }
        public int YearTo { get; set; }

        public virtual ICollection<Car>? Cars { get; set; }
        public virtual ICollection<ModelGeneration>? ModelGenerations { get; set;
}
    }
}
```

Джерело: [створено автором]

Для наповнення бази даних використовується приватний статичний метод Seed(), який викликається в DataContext(), в методі OnModelCreating(), який я зазначав вище

Лістинг 2.6. Код файлу DataContextExtension проєкту Infrastructure

```
namespace Web.Core
{
    public static class DataContextExtesnion
    {
        public static void Seed(this ModelBuilder builder)
        {
            builder.Entity<BodyType>().HasData(new BodyType
            {
                Id = 1,
                Name = "Седан",
            }, new BodyType
            {
                Id = 2,
                Name = "Хетчбек"
            }, new BodyType
            {
                Id = 3,
                Name = "Мінівен"
            }, new BodyType
            {
                Id = 4,
                Name = "Купе"
            }
}

}
```

Джерело: [створено автором]

Діаграма бази даних виглядає наступним чином:

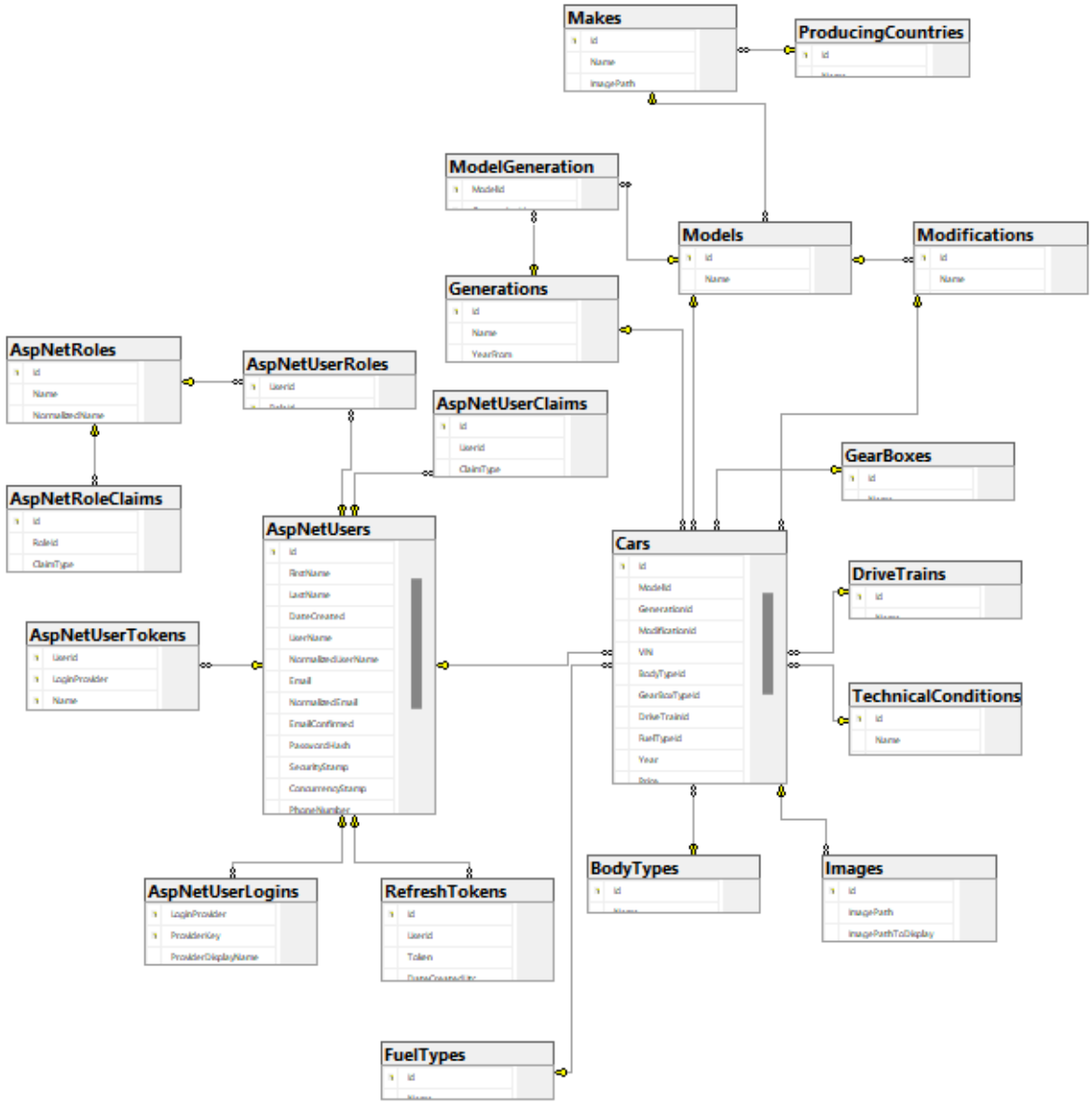

Рис. 2.7. Модель бази даних

2.1.4. Контролери та репозиторії

Контролер – це частина вебдодатку, що відповідає за обробку запитів які надходять до API з клієнтської сторони.

Репозиторій – це шаблон програмування, який відокремлює логіку доступу до даних, щоб уникнути прямого доступу до БД з різних частин коду і визначає один єдиний інтерфейс для взаємодії з цими даними [26].

Лістинг 2.7. Код файлу IRepository.cs проєкту Application

```
using System.Linq.Expressions;

namespace Web.Infrastructure
{
    public interface IRepository<T> where T : class
    {
        Task AddAsync(T entity);
        Task UpdateAsync(T entity);
        Task DeleteAsync(T entity);
        Task<T> GetByIdAsync(int id);
        T GetById(int id);
        Task<T> GetFirstAsync(Expression<Func<T, bool>> expression);
        Task<IEnumerable<T>> GetAllAsync();

    }
}
```

Джерело: [створено автором]

Тут можна побачити основні методи для обробки даних такі як: додавання, оновлення, видалення, отримання по {id}, отримати всі. Від цього інтерфейсу наслідуються всі репозиторії.

Цікавим методом є отримання марки автомобіля за id країни виробника, де параметром передається id країни і далі в тілі метода створюється змінна з назвою “makes” і для неї присвоюється пошук, де по визначеному полю марки ProducingCountryId перевіряється чи id який прийшов, як параметр дорівнює id полю марки.

Лістинг 2.8. Код файлу MakeRepository.cs проєкту Infrastructure

```
        public IEnumerable<Make> GetMakeByCountry(int id)
        {
            var makes = _ctx.Set<Make>()
                .Where(m => m.ProducingCountryId == id)
                .ToList();

            if (makes.Count == 0)
            {
                return _ctx.Set<Make>().ToList();
            }
            else
```

```
                {
                    return makes;
                }
            }
```

Джерело: [створено автором]

Як бачимо, наший репозиторій наслідується від інтерфейсу IRepository та реалізує всі методи, які були прописані в інтерфейсі. Детальніше приклад коду репозиторію можна переглянути в ДОДАТОК А.

Далі розглянемо контролер, який обробляє запити про автомобілі. Для прикладу розглянемо метод CreateWithImages(). Цей метод відповідає за створення оголошення разом з картинками. Бачимо, що метод має атрибут [HttpPost] з назвою “create-with-images”. Цей атрибут пояснює, що цей метод приймає лише HTTP-запит, як POST, а названий він для того, щоб на клієнтській частині було легше для цього методу надсилати запити. Також тут можна побачити реалізацію CQRS за допомогою MediatR. Для повного перегляду коду дивитися ДОДАТОК Б.

Лістинг 2.9. Код файлу CarController.cs проєкту Web

```
      [HttpPost("create-with-images")]
      public async Task<IResult> CreateCarWithImages([FromForm] CarCreateDTO
carDTO, List<IFormFile> images)
      {
          if (carDTO == null)
          {
              return Results.BadRequest("Invalid data");
          }

          var createCommand = _mapper.Map<CreateCar>(carDTO);

          var car = await _mediator.Send(createCommand);

          if (images != null && images.Count > 0)
          {
              foreach (var image in images)
              {
                  if (image != null && image.Length > 0)
                  {
                      var filePathToImage = AddImagesToDirectory(image);
                      var filePathToDisplay =
filePathToImage.Substring(filePathToImage.LastIndexOf("\\User Photos\\") + 1);
                      filePathToDisplay.Replace(@"\\", "/");
```

```
                    if (!string.IsNullOrEmpty(filePathToImage))
                    {
                        var imageAdded = new CreateImage { ImagePathToDisplay
= "https://localhost:7119/" + filePathToDisplay, ImagePath = filePathToImage,
CarId = car.Id, Car = car };
                        await _mediator.Send(imageAdded);
                    }
                }
            }
        }
        else
        {
            var defaultImagePath = AddImagesToDirectory(null);
            var defaultImagePathToDisplay =
defaultImagePath.Substring(defaultImagePath.LastIndexOf("\\User Photos\\") + 1);
            defaultImagePathToDisplay.Replace(@"\\", "/");

            if (!string.IsNullOrEmpty(defaultImagePath))
            {
                var defaultImage = new CreateImage { ImagePathToDisplay =
"https://localhost:7119/" + defaultImagePathToDisplay, ImagePath =
defaultImagePath, CarId = car.Id, Car = car };
                await _mediator.Send(defaultImage);
            }
        }

        return Results.Ok(new JsonResult(new { title = "Оголошення додано
успішно", message = "Ваше оголошення додано успішно", id = car.Id }));
    }
```

Джерело: [створено автором]

Розглянемо реалізацію фільтру на сервері

Лістинг 2.10. Код файлу CarController.cs проєкту Web

```
    [HttpGet("car-filter")]
    public async Task<IResult> CarFilter([FromQuery] CarFilter filter)
    {
        var query = _mapper.Map<CarAdvanceFilter>(filter);
        var car = await _mediator.Send(query);
        var carDTO = _mapper.Map<IEnumerable<CarDTO>>(car);

        return TypedResults.Ok(carDTO.ToList());
    }
```

Джерело: [створено автором]

Бачимо, що параметром передається об’єкт CarFilter з атрибутом [FromQuery], що означає, що ці дані будуть братися з рядка запиту.

Лістинг 2.11. Код файлу CarRepository.cs проєкту Infrastructure

```
        public async Task<IEnumerable<Car>> CarFilter(CarFilter filter)
        {
            var query = _ctx.Set<Car>().AsQueryable();

            if (filter.MakeId.HasValue)
            {
                query    =    query.Include(src    =>    src.Model).ThenInclude(src    =>
src.Make).Where(car => car.Model.Make.Id == filter.MakeId);
            }

            if (filter.ModelId.HasValue)
            {
                query = query.Where(car => car.ModelId == filter.ModelId);
            }

            if (filter.GenerationId.HasValue)
            {
                query = query.Where(car => car.GenerationId == filter.GenerationId);
            }

            if (filter.ModificationId.HasValue)
            {
                query = query.Where(car => car.ModificationId == filter.ModificationId);
            }

            if (filter.BodyTypeId.HasValue)
            {
                query = query.Where(car => car.BodyTypeId == filter.BodyTypeId);
            }

            if (filter.GearBoxTypeId.HasValue)
            {
                query = query.Where(car => car.GearBoxTypeId == filter.GearBoxTypeId);
            }

            if (filter.DriveTrainId.HasValue)
            {
                query = query.Where(car => car.DriveTrainId == filter.DriveTrainId);
            }

            if (filter.TechnicalConditionId.HasValue)
            {
                query     =     query.Where(car     =>     car.TechnicalConditionId     ==
filter.TechnicalConditionId);
            }

            if (filter.FuelTypeId.HasValue)
            {
                query = query.Where(car => car.FuelTypeId == filter.FuelTypeId);
            }

            if (filter.YearFrom.HasValue)
            {
                query = query.Where(car => car.Year >= filter.YearFrom);
            }

            if (filter.YearTo.HasValue)
```

```
            {
                query = query.Where(car => car.Year <= filter.YearTo);
            }

            if (filter.MileageFrom.HasValue)
            {
                query = query.Where(car => car.Mileage >= filter.MileageFrom);
            }

            if (filter.MileageTo.HasValue)
            {
                query = query.Where(car => car.Mileage <= filter.MileageTo);
            }

            if (filter.PriceFrom.HasValue)
            {
                query = query.Where(car => car.Price >= filter.PriceFrom);
            }

            if (filter.PriceTo.HasValue)
            {
                query = query.Where(car => car.Price <= filter.PriceTo);
            }

            query = query.Include(m => m.Model).ThenInclude(m => m.Make).ThenInclude(c =>
c.ProducingCountry)
                .Include(m => m.Modification)
                .Include(g => g.Generation)
                .Include(b => b.BodyType)
                .Include(g => g.GearBoxType)
                .Include(d => d.DriveTrain)
                .Include(t => t.TechnicalCondition)
                .Include(f => f.FuelType)
                .Include(u => u.User)
                .Include(i => i.Images);

            var filteredCars = await query.ToListAsync();

            return filteredCars;
        }
```

Джерело: [створено автором]

Логіка коду дуже проста, спочатку створюється базовий запит до БД для отримання автомобілів. *AsQueryable* означає, що буде можливість динамічно додавати якісь умови до запиту. Далі для кожного можливого фільтру (марка, тип палива, ціна, тощо) перевіряється чи передано значення в параметрі CarFilter, якщо так, то до запиту додається умова.

2.1.5. Аутентифікація та авторизація

Аутентифікація та авторизація є двома фундаментальними поняттями у сфері безпеки та контролю доступу в інформаційних системах.

Аутентифікація – це процес перевірки особи користувача при вході в систему [27].

Авторизація – це надання та підтвердження права на виконання певних дій в системі [27].

Одним з підходів до аутентифікації та авторизації в ASP.NET Core Web API є використання токенів JWT (JSON Web Token) – це вебстандарт, який визначає спосіб шифрування та передачі даних користувача у форматі JSON [29].

Спочатку в конфігурації сервера налаштовуємо IdentityCore.

Лістинг 2.12. Код файлу Program.cs проєкту Web

```
// Налаштування IdentityCore юзера
builder.Services.AddIdentityCore<User>(option =>
{
    // Налаштування пароля
    option.Password.RequiredLength = 6;
    option.Password.RequireDigit = false;
    option.Password.RequireLowercase = false;
    option.Password.RequireUppercase = false;
    option.Password.RequireNonAlphanumeric = false;

    // Налаштування Email
    option.SignIn.RequireConfirmedEmail = true;
})
    .AddRoles<IdentityRole>()
    .AddRoleManager<RoleManager<IdentityRole>>()
    .AddEntityFrameworkStores<DataContext>()
    .AddSignInManager<SignInManager<User>>()
    .AddUserManager<UserManager<User>>()
    .AddDefaultTokenProviders();
```

Джерело: [створено автором]

Також додаємо авторизацію.

Лістинг 2.13. Код файлу Program.cs проєкту Web

```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(option =>
```

```
        {
            option.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuerSigningKey = true,
                IssuerSigningKey                          =                          new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(builder.Configuration["JWT:Key"])),
                ValidIssuer = builder.Configuration["JWT:Issuer"],
                ValidateIssuer = true,
                ValidateAudience = false,
                ValidateLifetime = true,
                ClockSkew = TimeSpan.Zero
            };
        });
```

Джерело: [створено автором]

`["JWT:Key"]` – це згенерований секретний ключ, який використовується для перевірки підпису JWT (Лістинг 2.14.)

`["JWT:Issuer"]` – вказує, які емітенти токенів вважаються дійсними.

Згенерований секретний ключ та елементи токенів – вразлива інформація, яка була винесена в окремий файл конфігурації задля безпеки і цей файл не комітиться на GitHub.

Лістинг 2.14. Файл формату JSON проєкту Web

```
{
  "JWT": {
    "Key": "G6ZjGyuKnKabrdrPf27oj7RQHWl2kxl6G6ZjGyuKnKabrdrPf27oj7RQHWl2kxl6",
    "ExpiresInDays": 15,
    "Issuer": "http://localhost:5052",
    "ClientUrl": "http://localhost:4200"
  }
}
```

Джерело: [створено автором]

Реалізація коду в контролері для аутентифікації та авторизації можна побачити в ДОДАТОК В.

Реалізація JWT сервісу є головним етапом в аутентифікації та авторизації з декількох ключових причин: безпека та підписи токенів, легкість обміну інформацією, простота використання та декодування.

Для прикладу переглянемо метод CreateJWT(). Цей метод генерує JWT-токен, який містить інформацію про користувача та його ролі, забезпечуючи при

цьому безпеку за допомогою підпису. Повну реалізацію JWT-сервісу можна переглянути в ДОДАТОК Г.

Лістинг 2.15. Код файлу JWTSercice.cs проєкту Infrastructure

```
            public async Task<string> CreateJWT(User user)
            {
                var userClaims = new List<Claim>
                {
                    new Claim(ClaimTypes.NameIdentifier, user.Id),
                    new Claim(ClaimTypes.Email, user.Email),
                    new Claim(ClaimTypes.GivenName, user.FirstName),
                    new Claim(ClaimTypes.GivenName, user.LastName),
                };

                var roles = await _userManager.GetRolesAsync(user);

                userClaims.AddRange(roles.Select(role => new Claim(ClaimTypes.Role,
role)));

                var credentials = new SigningCredentials(_jwtKey,
SecurityAlgorithms.HmacSha256Signature);
                var tokenDescriptor = new SecurityTokenDescriptor
                {
                    Subject = new ClaimsIdentity(userClaims),
                    Expires =
DateTime.UtcNow.AddDays(int.Parse(_config["JWT:ExpiresInDays"])),
                    SigningCredentials = credentials,
                    Issuer = _config["JWT:Issuer"]
                };

                var tokenHandler = new JwtSecurityTokenHandler();
                var jwt = tokenHandler.CreateToken(tokenDescriptor);

                return tokenHandler.WriteToken(jwt);
            }
```

Джерело: [створено автором]

Також реалізована політика авторизації для обробки запитів для різних кінцевих точок відповідно до ролей, політик і правил пов’язаних з клеймами користувачів. Для прикладу переглянемо модетод ролі AdminRole() та політику AdminPolicy(). Для повного перегляду коду дивитися в ДОДАТОК Е.

Ці два методи контролера використовують атрибут [Authorize] для забезпечення доступу лише тим користувачам, які мають певні політики або ролі. В даному випадку до першого методу доступ має лише той користувач в

якого є роль “Admin”. Для другого методу доступ до нього має тільки користувач з певною політикою “AdminPolicy”

Лістинг 2.16. Код файлу RCPracticeController.cs проєкту Web

```
[HttpGet("admin-role")]
[Authorize(Roles = "Admin")]
public IActionResult AdminRole()
{
    return Ok("admin role");
}

[HttpGet("admin-policy")]
[Authorize(policy: "AdminPolicy")]
public IActionResult AdminPolicy()
{
    return Ok("admin policy");
}
```

Джерело: [створено автором]

Не авторизований користувач може зареєструватися ввівши ім’я, прізвище, e-mail та пароль двічі. Всі з цих полів є обов’язковими. Після цього на пошту, яку він зазначив при реєстрації йому прийде лист з підтвердженням реєстрації його акаунту. Зареєстрований користувач повинен вводити тільки e-mail та пароль.

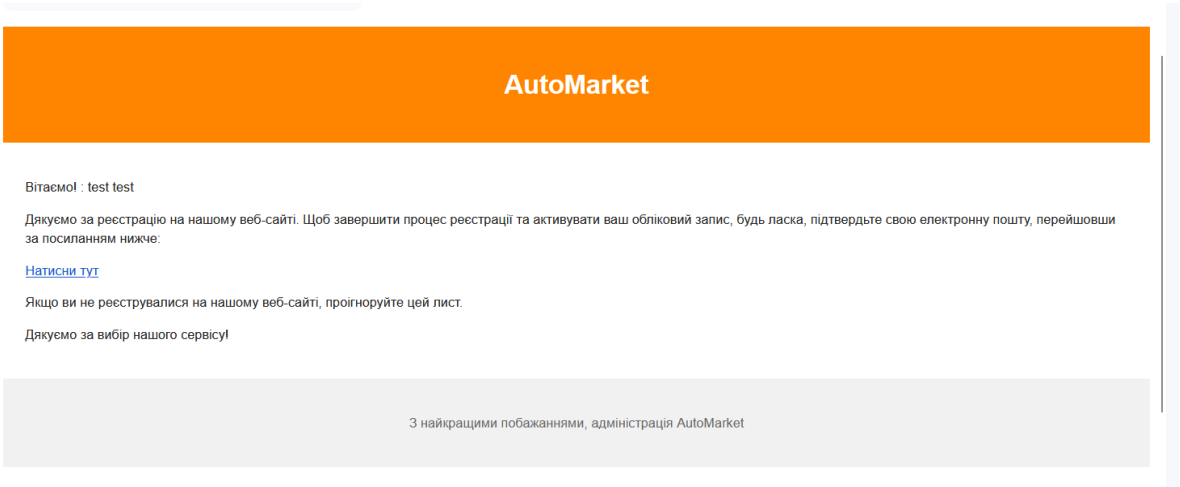

Рис. 2.8. Приклад листа з підтвердженням email

/api/account/refresh-token – створює новий токен для користувача

/api/account/refresh-page – зберігає сесію користувача при оновленні сторінки

/api/account/login – дає змогу увійти користувачу в особистий кабінет

/api/account/register – дає змогу користувача створити новий акаунт

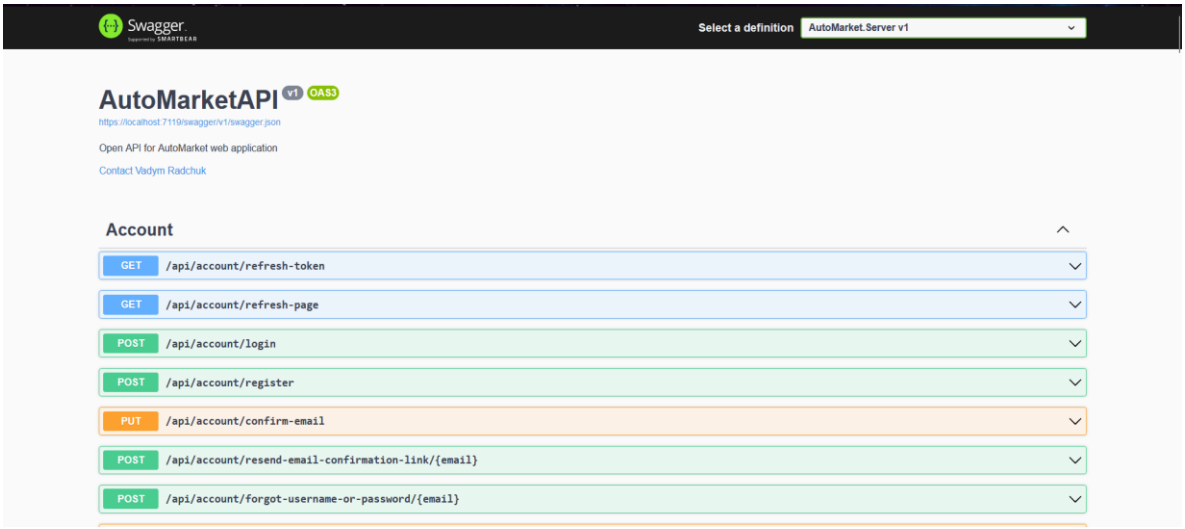

Рис. 2.9. Графічний інтерфейс Swagger (API)

2.1.6. Інтеграція стороннього API

Для перевірки VIN коду використовується сторонній сервіс API Ninjas. [20] API Ninjas простий у використанні та надійний сервіс, який надає швидкі, точні та безкоштовні запити.

Для початку, щоб отримати API-ключ потрібно створити обліковий запис. Після отримання API-ключа створюється окремий метод в репозиторію, який буде отримувати VIN код з клієнтської частини та відправляти його на сторонній API. API Ninjas відправляє відповідь в JSON файлі тому ми спочатку конвертуємо JSON в string. Далі ми робимо запит до нашої БД, щоб отримати оголошення за VIN кодом, який був надісланий з клієнтської частини і перевіряємо за полями: марка та рік. Якщо оголошення збігається з відповіддю від стороннього API, то на клієнтську частину ми відправляємо відповідь “VIN Перевірений”, якщо є проблема з VIN кодом і перевірку оголошення не пройшло, то відправляємо відповідь “Проблема з оголошенням, будьте обережні при купівлі цього автомобіля”. Візуальну реалізацію можна буде переглянути в підрозділі 2.2

Лістинг 2.17. Код файлу CarRepository.cs проєкту Infrastructure

```
        public async Task<bool> CheckVin(string vin)
        {
            using (var client = new HttpClient())
            {
                try
                {
                    var request = new HttpRequestMessage()
                    {
                        RequestUri = new Uri(@"https://api.api-
ninjas.com/v1/vinlookup?vin=" + vin),
                        Method = HttpMethod.Get,
                    };
                    request.Headers.Add("X-Api-Key", _config["VinLookup:ApiKey"]);

                    var response = await client.SendAsync(request);

                    if (response.IsSuccessStatusCode)
                    {
                        var responseContent = await
response.Content.ReadAsStringAsync();
                        var vinLookupResponse =
JsonConvert.DeserializeObject<VinLookupResponse>(responseContent);

                        var car = await _ctx.Set<Car>().FirstOrDefaultAsync(c =>
c.VIN == vinLookupResponse.Vin);

                        if (car == null)
                            return false;
                        bool matchesAllFields = car.Model?.Make?.Name ==
vinLookupResponse.Manufacturer &&
                            car.Year.ToString() ==
vinLookupResponse.Years.FirstOrDefault();

                        return matchesAllFields;
                    }
                    else
                    {
                        return false;
                    }
                }
                catch (HttpRequestException e)
                {
                    throw;
                }
                catch (Exception e)
                {
                    throw;
                }
            }
```

```
    }
```

Джерело: [створено автором]

Лістинг 2.18. Код файлу CarController.cs проєкту Web

```
    [HttpGet("check-vin/{vin}")]
    public async Task<IResult> CheckVin(string vin)
    {
        var isValid = await _mediator.Send(new CheckCarByVin { VIN = vin });

        var response = new VinCheckResponse
        {
            IsValid = isValid,
            Message = isValid ? "Перевірений VIN" : "Проблема з оголошенням,
будьте обережні при купівлі цього автомобіля"
        };

        return TypedResults.Ok(response);

    }
```

Джерело: [створено автором]

2.2. Клієнтська частина вебсервісу “AutoMarket”

2.2.1. Загальна інформація про клієнтську частину

Після створення повноцінного сервера з базою даних, яка керує ресурсами нам потрібно відобразити ці ресурси користувачам за допомогою клієнтського додатку на платформі Angular. У попередньому розділі було пояснено, що дані між сервером та клієнтською частиною передаються у форматі JSON.

Для відображення даних користувачеві використовується HTML(Hyper Text Markup Language) – мова розмітки гіпертексту, CSS(Cascading Style Sheets) – каскадні таблиці стилів, для яких була використана бібліотека Bootstrap. А логіку отримання даних з сервера та відправки даних на сервер реалізується за допомогою TypeScript

2.2.2. Структура клієнтської частини

Усі Angular додатки будуються з модулів, які включають компоненти, сервіси та різні утиліти. Іншими словами додаток базується на модулях, як глобальній структурі, в якій міститься вся логіка програми.

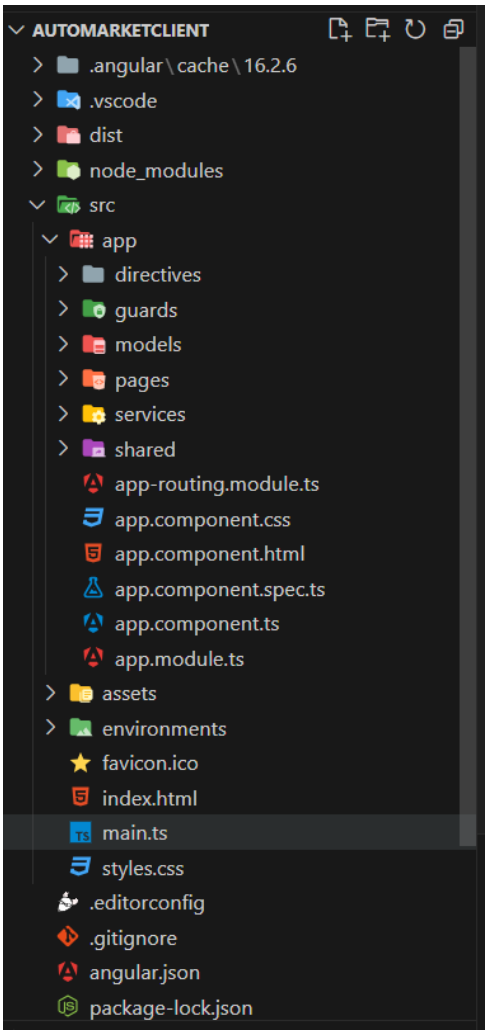

Рис. 2.10. Структура клієнтської частини

Тут є основні папки такі як app, assets, environment, directives, guards, models, pages, services, shared.

Directives – папка в якій знаходиться директива перевірки чи є в користувача якась роль.

Guards – папка в якій знаходиться директива, яка захищає окремі шляхи до представлень в які може зайти тільки користувач з певною роллю.

Models – папка в якій розміщуються основні інтерфейси класів, які використовуються для відправки, отримання та маніпулюванням даних з сервера.

Pages – папка в якій знаходяться всі сторінки сайту такі як: головна сторінка, footer, navbar.

Services – папка в якій розміщені сервісу для підключення до серверу та основні методи: POST, UPDATE, GET тощо.

Shared – папка в якій розміщуються компоненти які можуть повторюватися в коді по декілька разів, тому щоб код не повторювався була створена ця папка

Модулі в Angular використовуються для імпортування зовнішніх та внутрішніх модулів для використання, також кожен модуль може містити в собі різні компоненти, утиліти, тощо

Лістинг 2.19. Код файлу app.module.ts проєкту AutoMarketClient

```
      import { HomeComponent } from './pages/home/home.component';
      import { FooterComponent } from './pages/footer/footer.component';
      import { NavbarComponent } from './pages/navbar/navbar.component';
      import { NgModule } from '@angular/core';
      import { BrowserModule } from '@angular/platform-browser';
      import { AppRoutingModule } from './app-routing.module';
      import { AppComponent } from './app.component';
      import { BrowserAnimationsModule } from '@angular/platform-
browser/animations';
      import { SharedModule } from './shared/shared.module';
      import { HTTP_INTERCEPTORS } from '@angular/common/http';
      import { JwtInterceptor } from './shared/interceptors/jwt.interceptor';
      import { CarouselModule } from 'ngx-owl-carousel-o';
      import { UserHasRoleDirective } from './directives/user-has-
role.directive';

      @NgModule({
      declarations: [
        AppComponent,
        NavbarComponent,
        FooterComponent,
        HomeComponent,
        UserHasRoleDirective,
      ],
```

```
    imports: [
      BrowserModule,
      AppRoutingModule,
      SharedModule,
      BrowserAnimationsModule,
      CarouselModule
    ],
    providers:
      [
      { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true}
    ],
    bootstrap: [AppComponent]
  })
  export class AppModule { }
```

Джерело: [створено автором]

Тут існує об’єкт, в якому задається декларація компонента, імпорти інших модулів, провайдери тощо.

Angular дарує розробникам зручний інструмент для швидкого генерування структур проєкту. Завдяки команді ng generate {структура} {назва} можна створювати компоненти, директиви, сервіси та інші елементи ввівши їх тип та ім’я в терміналі. Після виклику даної команди буде створено папку з назвою файла, якщо для прикладу хочемо створити компонент, то можемо прописати скорочено: ng g c example. Після виклику створено папку Example та в ній є: example.component.html, example.component.css та example.component.ts

2.2.3. Компоненти додатку

Компонент – відокремлена частина функціоналу зі своєю логікою, HTML-шаблоном та CSS-стилями.

Будь-який елемент в Angular додатку розбивається на модулі. До прикладу є модуль навігаційної системи, модуль для головної сторінки, модуль для відображення машин на сайті тощо – усі ці модулі містять в собі або можуть використовувати ще менші модулі, наприклад, модальне вікно для підтвердження або відхилення.

Всередині компоненту усі файли між собою пов’язані type scripts, css, html. Через type script прописується зв’язка з різними даними, які прийшли з іншого компоненту чи з сервера та які вводяться з інтерфейсу користувача повністю описаний в компоненті html.

Для прикладу візьмемо код додавання оголошення на сайт. Компонент typescript який реалізує метод додавання автомобіля addCar(). Цей код виконує декілька важливих завдань: відправляє дані про нове оголошення на сервер, обробляє успіх та помилки, якщо вони прийшли з сервера та відображає відповідні повідомлення в компоненті HTML. Для перегляду повного коду дивтися ДОДАТОК Е.

Лістинг 2.20. Код файлу car-add.component.ts проєкту AutoMarketClient

```
      addCar() {
          this.submitted = true;
          this.errorMessages = [];

          if (this.addCarForm.valid) {
            const carData = this.addCarForm.value;

            this.carService.addCar(carData, this.selectedImages).subscribe({
              next: (response: any) => {
                this.sharedService.showNotification(true, response.value.title,
response.value.message);
                if (response.value.id) {
                  this.router.navigateByUrl(`/car/car-
details/${response.value.id}`)
                }
                console.log(response);
              },
              error: error => {
                console.log('Error object:', error);
                if (error.error.error) {
                  this.errorMessages = error.error.error;
                } else {
                  this.errorMessages.push(error.error)
                }
              }
            })
          }
        }
```

Джерело: [створено автором]

Модель(інтерфейс) додавання автомобіля виглядає так:

Лістинг 2.21. Код файлу car-add.ts проєкту AutoMarketClient

```
export interface CarAdd {
    id: number;
    modelId: number;
    generationId: number;
    modificationId: number;
    vin: string;
    bodyTypeId: number;
    gearBoxTypeId: number;
    driveTrainId: number;
    technicalConditionId: number;
    fuelTypeId: number;
    year: number;
    price: number;
    mileage: number;
    description: string;
    userId: string;
}
```

Джерело: [створено автором]

За допомогою реактивних форм спочатку отримуємо для кожного поля в моделі своє значення, а потім передаємо ці значення в компонент ts, а далі відправлямо це в сервіс для подальшої обробки цих даних сервером.

Приклад повної реалізації інтерфейсу користувача за допомогою HTML та бібліотеки Bootstrap можна переглянути в ДОДАТОК Ж.

Лістинг 2.22. Код файлу car-add.component.html проєкту AutoMarketClient

```
<form [formGroup]="addCarForm" (ngSubmit)="addCar()" autocomplete="off">
                        <div class="row">
                            <div class="form-floating mb-3 col-md-12">
                                <select formControlName="countryId"
class="form-select shadow-none" >
                                    <option selected value="">Країна
виробник</option>
                                    <option *ngFor="let county of
countries" [value]="county.id">{{county.name}}</option>
                                </select>
                            </div>
                        </div>
                    </form>
```

Джерело: [створено автором]

Інтерфейс користувача для додавання оголошення виглядає так:

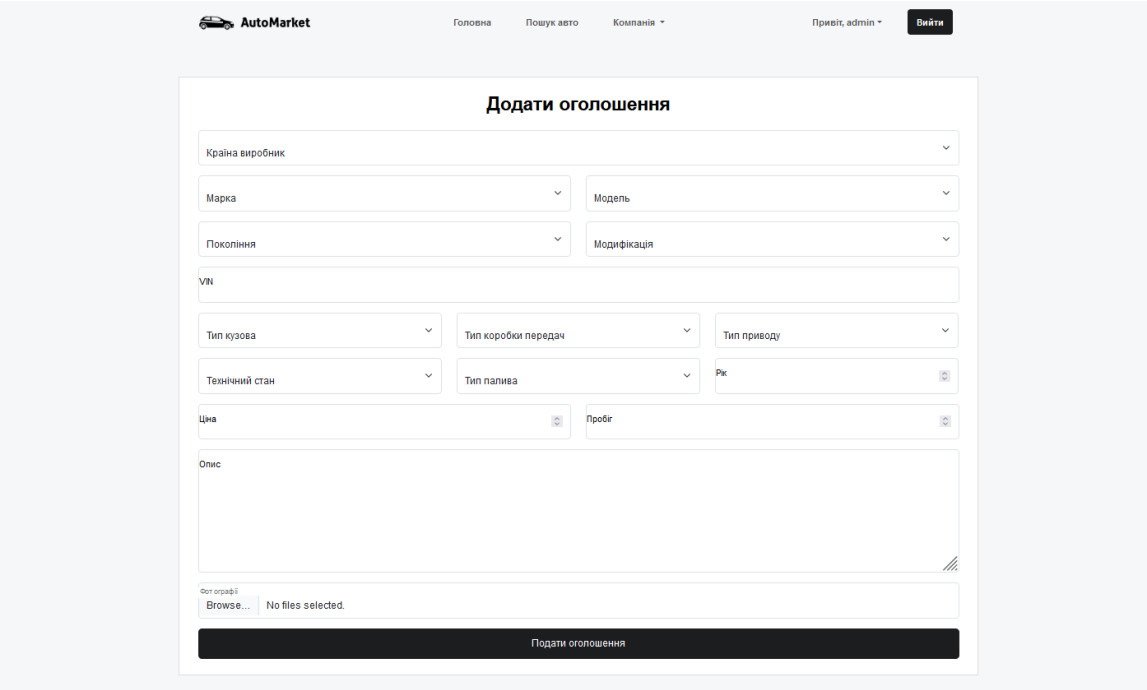

Рис. 2.11. Інтерфейс користувача

Також приклад валідації полів, які є обов’язковими, а користувач їх не заповнив.

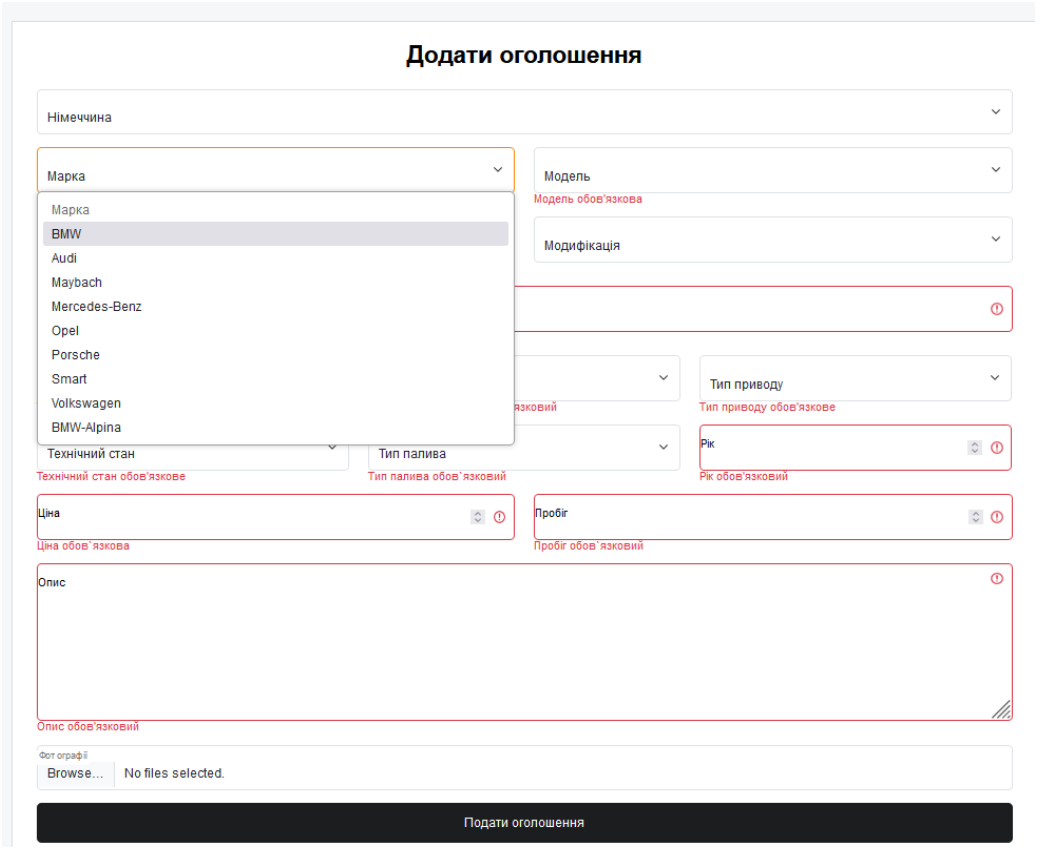

Рис. 2.12. Приклад валідації полів

2.2.4. Сервіси

Вся бізнес-логіка додатку, пов'язана з клієнтською частиною Angular-додатку знаходиться в сервісах. Сервіс – місце, де в них приходять та відправляються запити на сервер.

Показаний у минулому розділі метод для додавання оголошення про автомобіль використовував сервіс *car.service.ts*. Приклад методу в сервісі, який використовується для додавання оголошення про автомобілі та відправки HTTP-запиту POST на сервер.

Лістинг 2.23. Код файлу car.service.ts проєкту AutoMarketClient

```
      addCar (model: CarAdd | any, images: File[]) {
          const formData = new FormData();

          for (const key in model) {
            if (model.hasOwnProperty(key)) {
              formData.append(key, model[key]);
            }
          }

          for (let i = 0; i < images.length; i++) {
            formData.append(`images`, images[i]);
          }

          return this.http.post<CarAdd>(`${environment.apiUrl}/api/car/create-
with-images`, formData)
        }
```

Джерело: [створено автором]

2.2.5. Утиліти

В Angular-додатку існує багато утиліт. Основні утиліти, які я використав в розробці додатку – це guard, interceptor.

Interceptor – дозволяє перехоплювати HTTP-запити перед їх відправкою на сервер і вносити в них необхідні зміни. Прикладом мого interceptor є автоматичне додавання токенів JWT до заголовків HTTP-запитів.

Лістинг 2.24. Код файлу jwt.interceptor.ts проєкту AutoMarketClient

```
@Injectable()
export class JwtInterceptor implements HttpInterceptor {

  constructor(private accountService: AccountService) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler):
Observable<HttpEvent<unknown>> {
    this.accountService.user$.pipe(take(1)).subscribe({
      next: user => {
        if (user) {
          request = request.clone({
            setHeaders: {
              Authorization: `Bearer ${user.jwt}`
            }
          });
        }
      }
    })
    return next.handle(request);
  }
}
```

Джерело: [створено автором]

Guard в проєкті використовується для того, щоб заборонити неавторизованим користувачам або користувачам, які не мають необхідної ролі доступитись до певних даних. Для прикладу покажемо authorization.guard.ts який перевіряє чи користувач авторизований чи ні, якщо авторизований, то буде повертати true, якщо ні, то його буде відправлено на сторінку входу та показано модальне вікно.

Лістинг 2.25. Код файлу authorization.guard.ts проєкту AutoMarketClient

```
@Injectable({
  providedIn: 'root'
})
export class AuthorizationGuard {
  constructor(private accountService:AccountService,
    private sharedService: SharedService,
    private router: Router) {}

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean> {
    return this.accountService.user$.pipe(
      map((user: User | null) => {
        if (user) {
```

```
                return true;
              } else {
                this.sharedService.showNotification(false, 'Заборонена зона',
'Покиньте негайно!');
                this.router.navigate(['account/login'], {queryParams:
{returnUrl: state.url}});
                return false;
              }
            })
          );
        }
      }
```

Джерело: [створено автором]

Та приклад де саме використовується guard. Як бачимо на цьому прикладі простий користувач не матиме змоги зайти на сторінки /add-edit… тому що тут стоїть атрибут [AdminGuard].

Лістинг 2.26. Код файлу admin-routing.module.ts проєкту AutoMarketClient

```
      const routes: Routes = [
        {
          path: '',
          runGuardsAndResolvers: 'always',
          canActivate: [AdminGuard],
          children: [
            { path: '', component: AdminComponent},
            { path: 'add-edit-member', component: AddEditMemberComponent},
            { path: 'add-edit-member/:id', component: AddEditMemberComponent},
            { path: 'add-edit-generation', component:
AddEditGenerationComponent},
            { path: 'add-edit-generation/:id', component:
AddEditGenerationComponent},
          ]
        },
      ]

      @NgModule({
        declarations: [],
        imports: [
          RouterModule.forChild(routes)
        ],
        exports: [
          RouterModule
        ]
      })
      export class AdminRoutingModule { }
```

Джерело: [створено автором]

2.2.6. Графічний інтерфейс вебсервісу

Коли користувач буде заходити на вебсервіс, перед ним відразу буде показано фільтр, зверху є навігаційна панель, в якій є кнопки реєстрації та логіну. Головна сторінка сайту виглядає наступним чином:

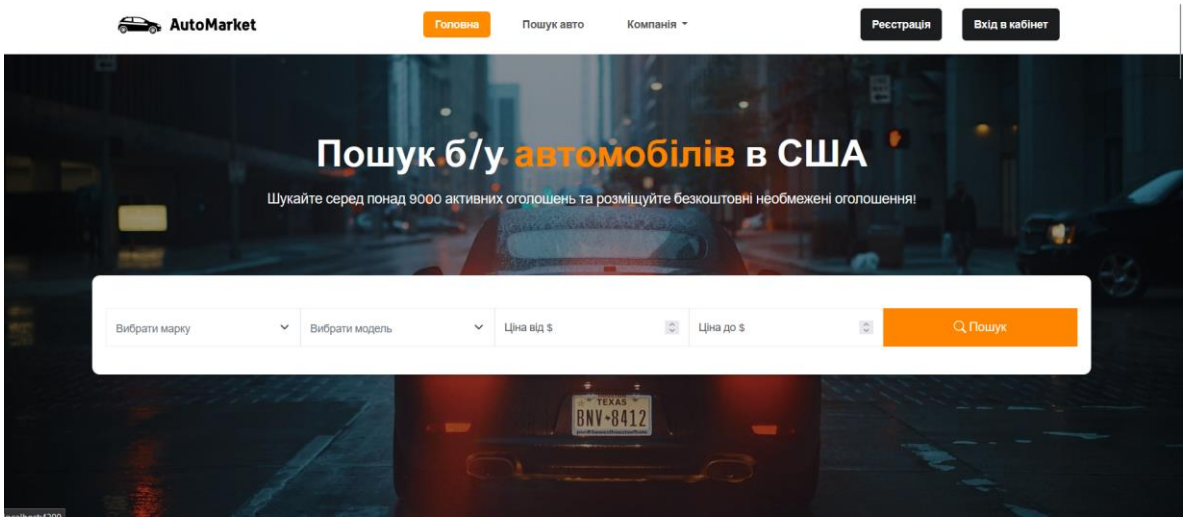

Рис. 2.13. Приклад головної сторінки онлайн-сервісу

Наступна секція це недавно створені оголошенні. Тут реалізовано карусель, яка показує останні 5 створених оголошень користувачами

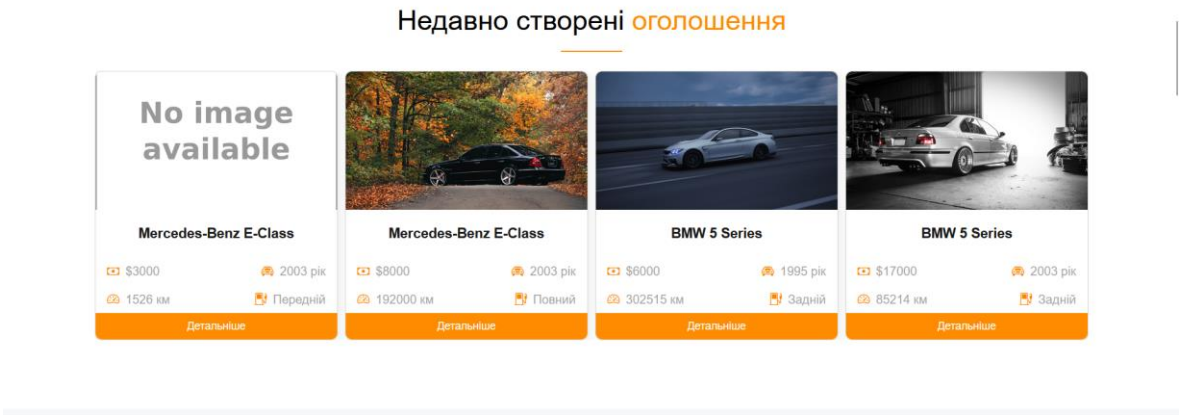

Рис. 2.14. Приклад головної сторінки онлайн-сервісу

Наступною секцією головної сторінки є пошук автомобілів по маркам. Натиснувши на будь-яку марку користувача перекидає на сторінку з автомобілями і будуть показуватися лише ті автомобілі, марки яких обрав користувач.

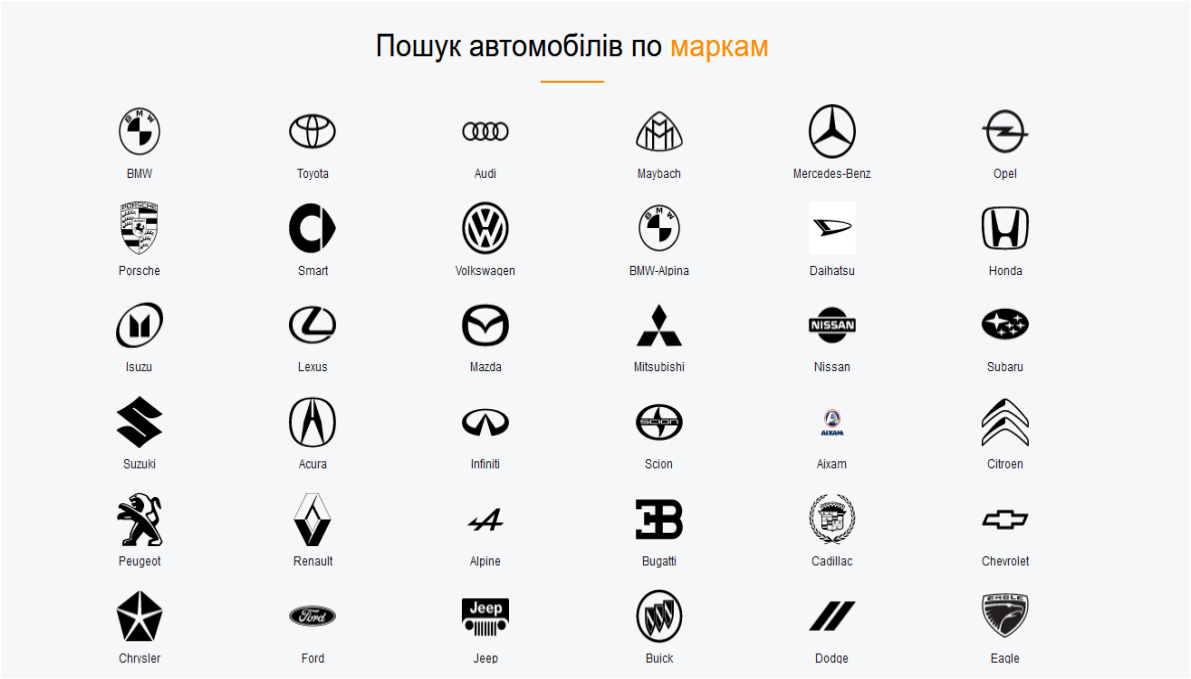

Рис. 2.15. Приклад головної сторінки онлайн-сервісу

Наступна секція це “Про нас”. Тут описується загальна інформація про наш проєкт.

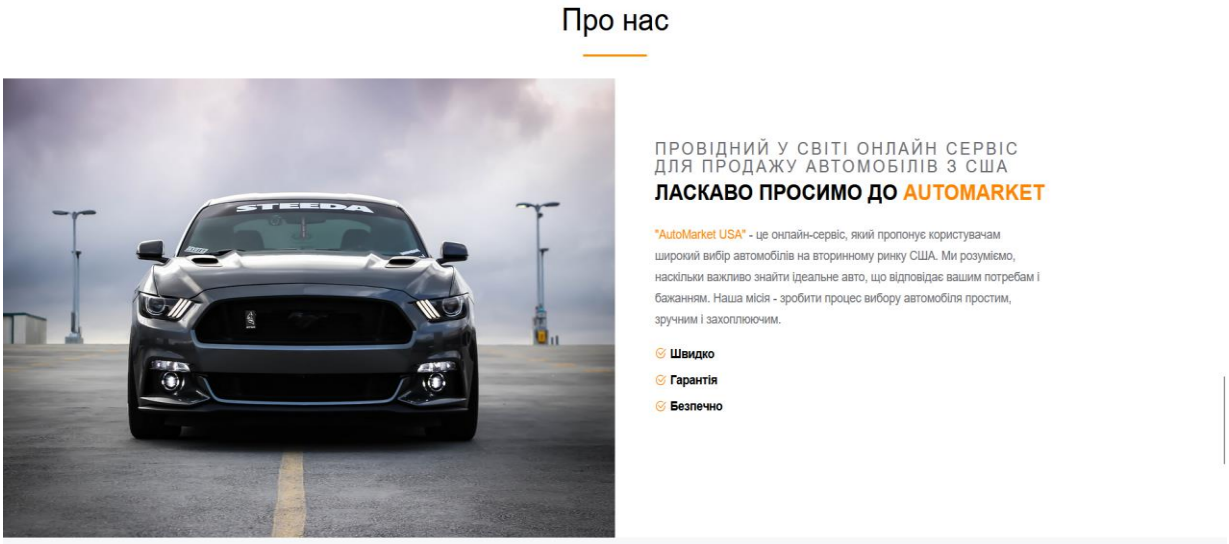

Рис. 2.16. Приклад головної сторінки

Коли користувач переходить до сторінки “Пошук авто” перед ним відкривається сторінка з усіма оголошеннями про автомобілі також на цій сторінці реалізований фільтр та сортування по ціні. Також можна переглядати оголошення у двох видах в “List”, коли оголошення йдуть один за одним та “Grid” – оголошення йдуть по блокам.

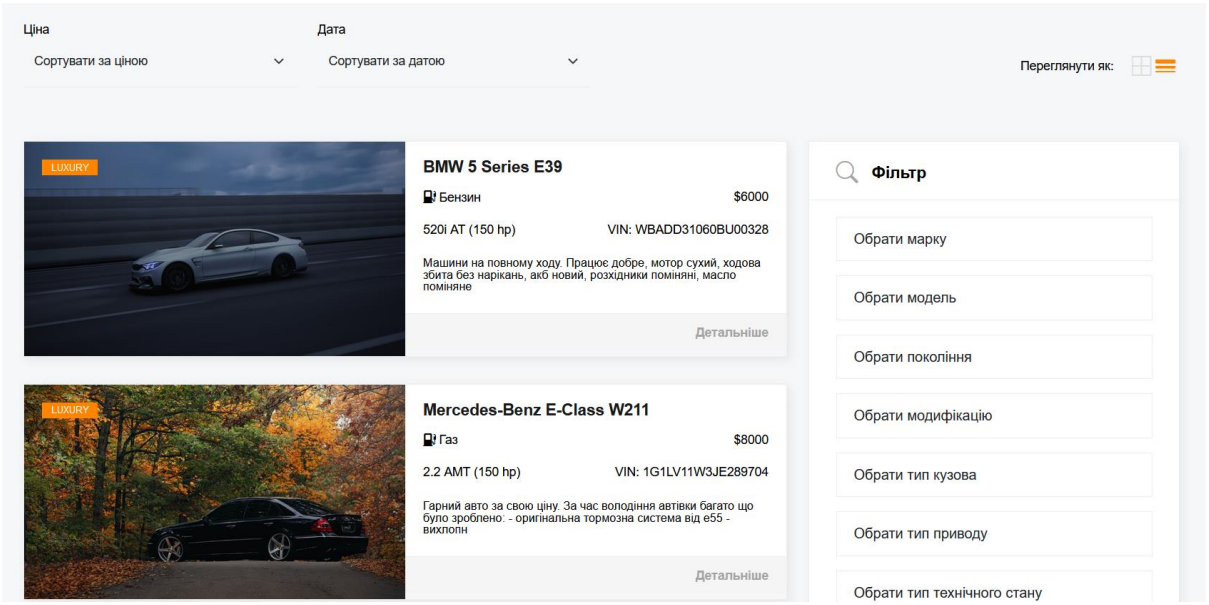

Рис. 2.17. Приклад сторінки з оголошеннями List виду

Наступним рисунком є сторінка з Grid видом.

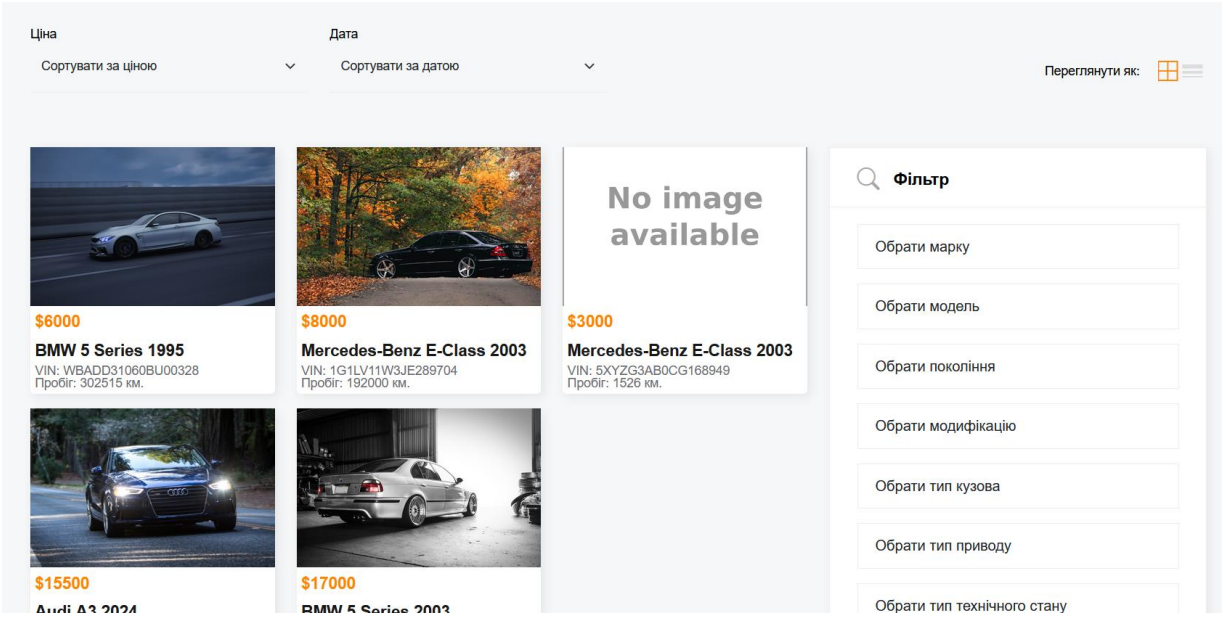

Рис. 2.18. Приклад сторінки з оголошеннями Grid виду

При натисканні на будь-яке оголошення користувача перекидає на сторінку з детальнішою інформацією про оголошення.

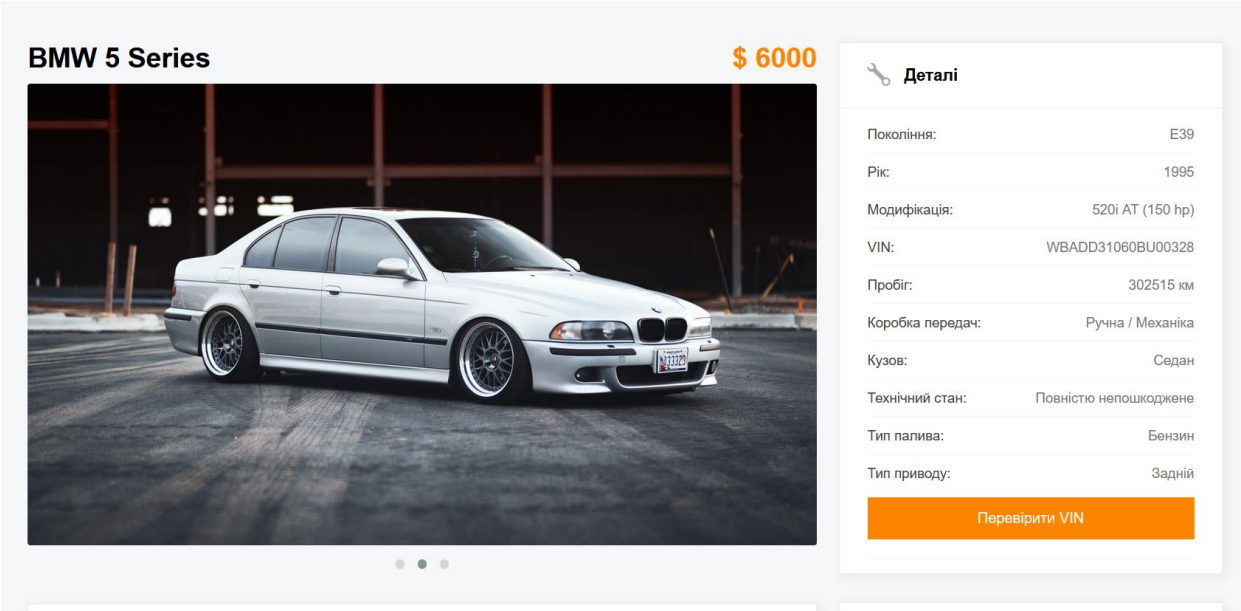

Рис. 2.19. Приклад сторінки з детальною інформацією про оголошення

Після натискання кнопки “Перевірити VIN” у нас буде з’являтися модалень вікно з текстом чи перевірений цей VIN код.

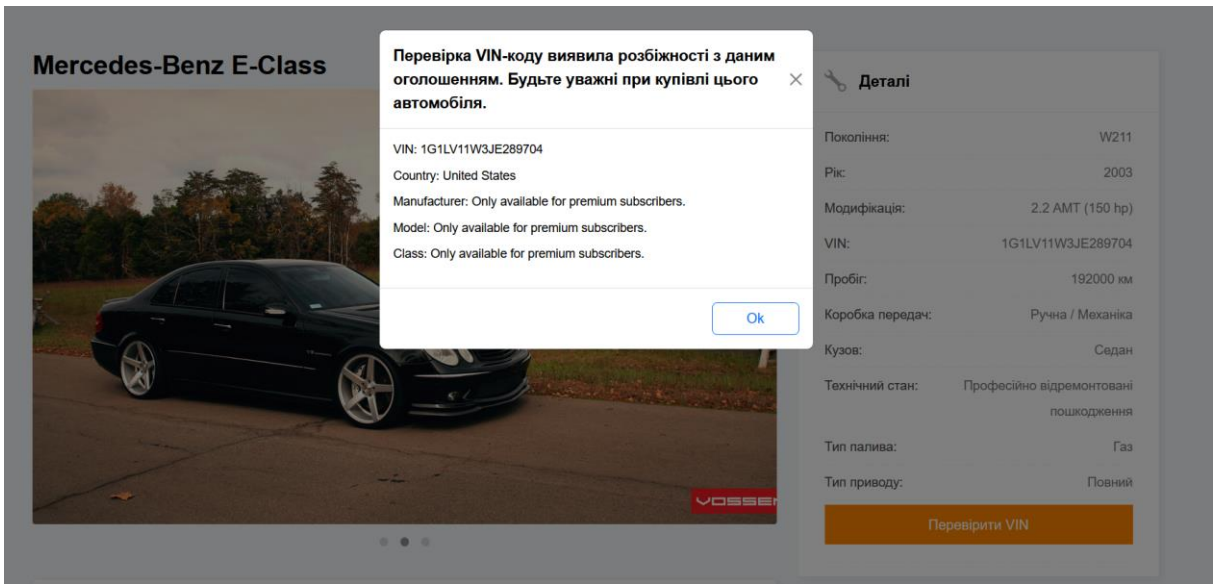

Рис. 2.20. Приклад модального вікна, коли є проблема з VIN-кодом

І також наступним рисунком є приклад, коли VIN код пройшов перевірку.

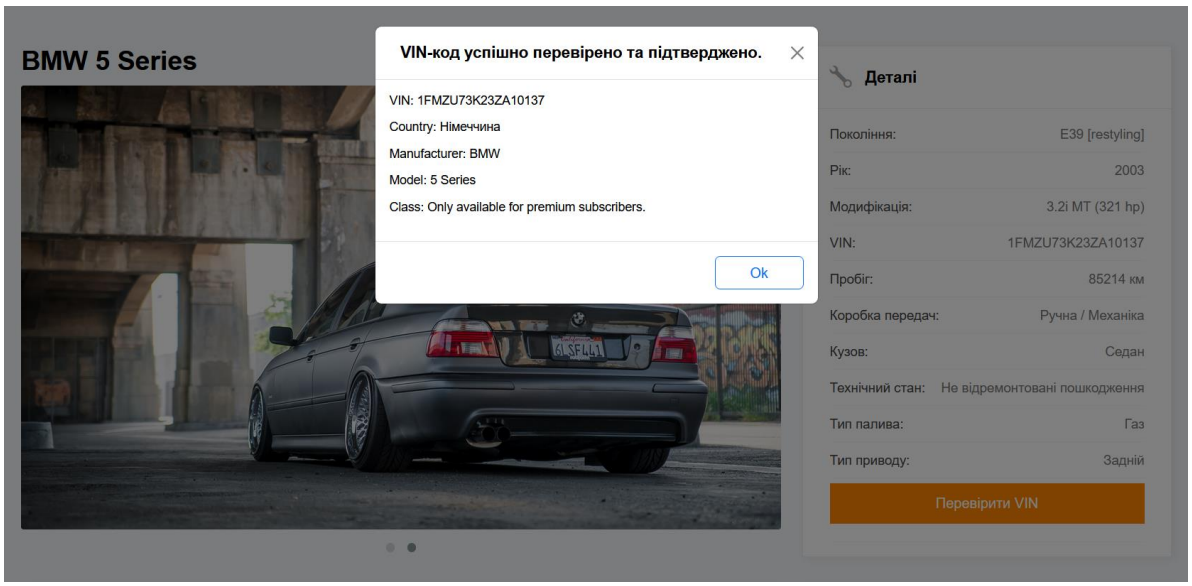

Рис. 2.21. Приклад модального вікна, коли немає проблеми з VIN-кодом

Сторінки входу/реєстрації виглядають наступним чином:

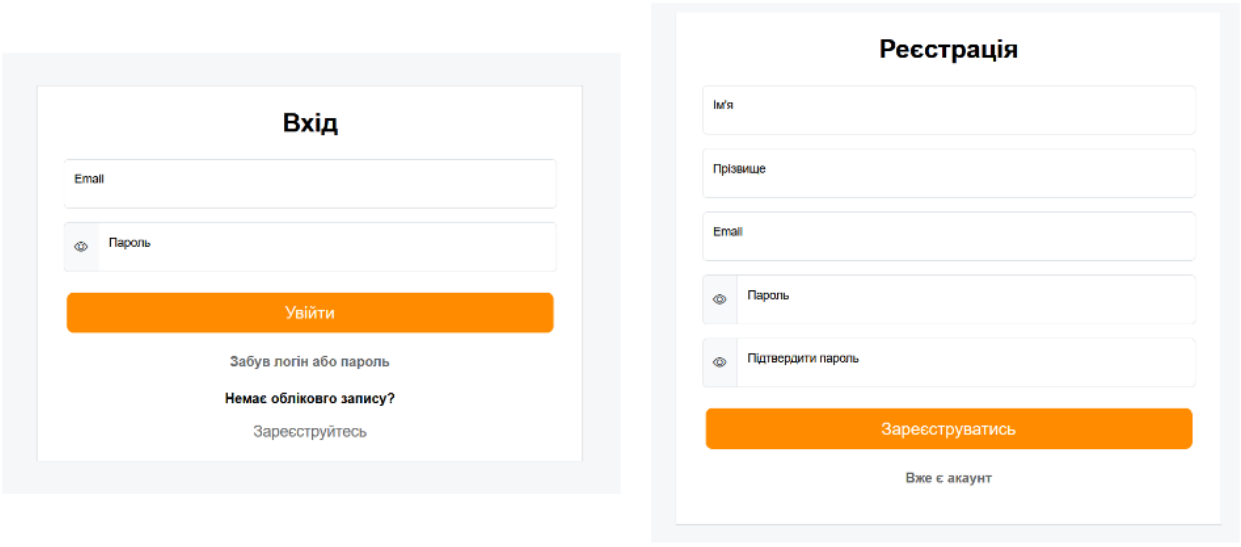

Рис. 2.22. Приклад сторінки входу та реєстрації на вебсервісі

У футері надана загальна інформація, мапа сайту та контакти:

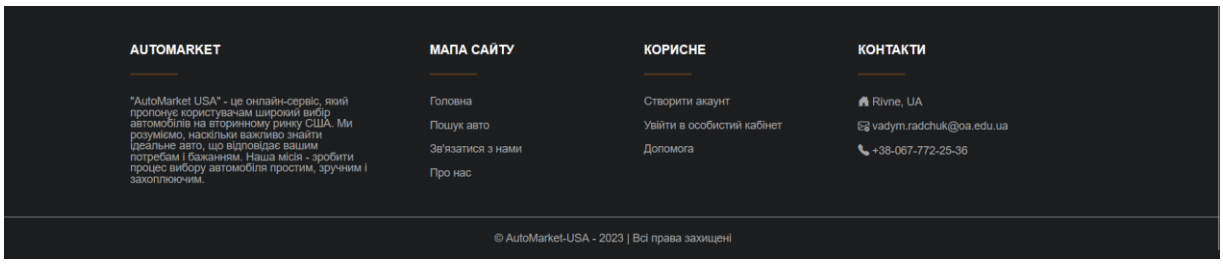

Рис. 2.23. Приклад підвалу сайту

Адмін панель – це центральний центр керування для адміністраторів вебсайтів. Вона надає широкий спектр інструментів та елементів керування для управління різними аспектами вебсайту. В нашому випадку в адмінпанелі адміністратор може переглянути всі опубліковані оголошення. Підтвердити або відхилити оголошення, які користувач додав на сайт. Переглядати усіх зареєстрованих користувачів, редагувати та видаляти їх. Додавати нові марки, моделі, модифікації і т.д.

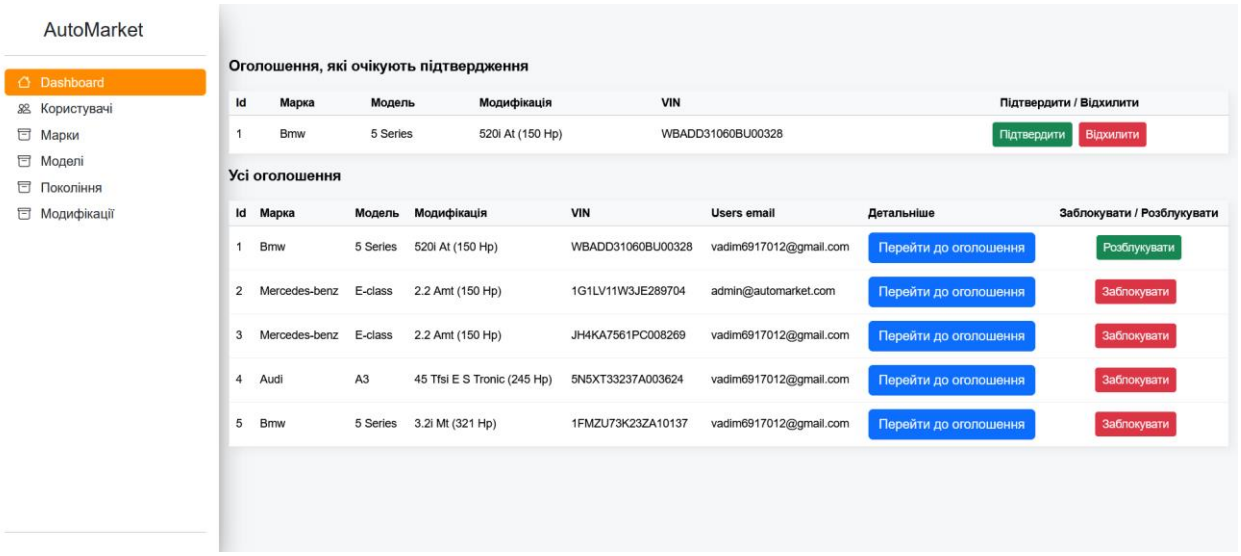

Рис. 2.24. Приклад головної сторінки адмінпанелі

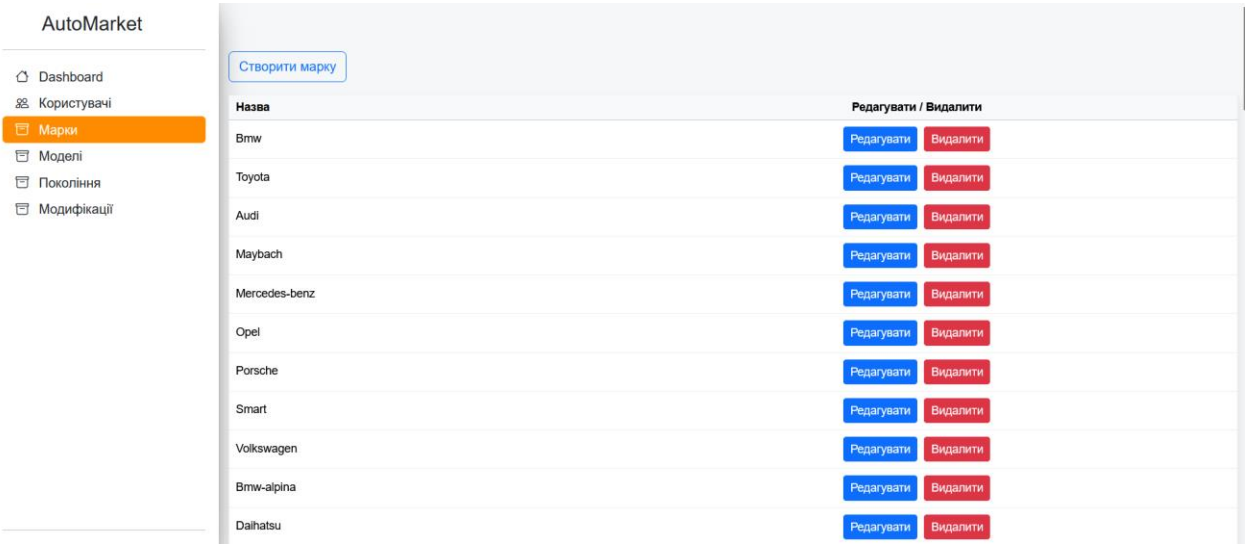

Рис. 2.25. Приклад сторінки з марками автомобілів

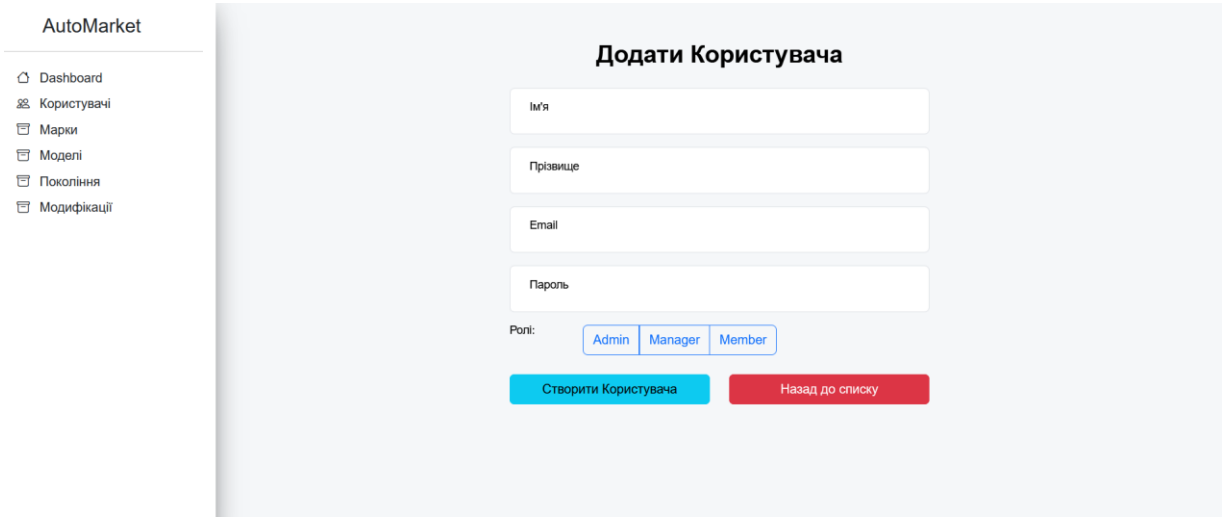

Рис. 2.26. Приклад сторінки з додаванням нового користувача

Висновки до розділу 2

Створення онлайн-сервісу “AutoMarket” вимагало ретельного планування та продуманої архітектури. У розділі 2 ми детально розглянули ключові аспекти розробки backend та frontend компонентів, а також інтеграцію стороннього API для розширення функціоналу.

Backend частина онлайн-сервісу “AutoMarket” була розроблена з використанням C# у поєднанні з фреймворком .Net 8 та платформою ASP .Net Core Web API для створення REST API, який забезпечує зручний доступ до даних сервісу. Clean Architecture також дала змогу зробити онлайн-сервіс надійним, маштабованим з модульною архітектурою, що полегшує обслуговування.

Для розробки привабливого інтерфейсу користувача на фронтенді застосовано мову програмування TypeScript, зокрема Angular, що забезпечило створення інтуїтивно зрозумілого та елегантного інтерфейсу. Angular - популярний фреймворк для розробки інтерфейсів, який пропонує розробникам широкий спектр інструментів, що полегшує створення динамічних, масштабованих та зручних у використанні інтерфейсів користувача. Для розробки більш привабливого інтерфейсу було задіяно бібліотеку Bootstrap.

Крім того, інтеграція стороннього API “API Ninjas” - додала до “AutoMarket” новий рівень функціональності, роблячи дані про оголошення більш розширеними. Користувачі “AutoMarket” можуть лише в декілька кліків дізнатися чи потрібно переглядати оголошення більш детально чи ні.

ВИСНОВКИ

В рамках даної кваліфікаційної роботи нами було розроблено серверну та клієнтську частини вебдодатку “AutoMarket”. Також ми реалізували комплекс сервісів, необхідних для його повноцінної роботи.

В рамках цього дослідження було успішно створено онлайн-сервіс “AutoMarket”, що надає користувачам можливість самостійно обирати та публікувати оголошення автомобілів. Також вдалося ефективно виконати всі поставлені завдання.

З метою розробки ефективної стратегії, було проведено аналіз конкурентів. Вивчивши їх сильні та слабкі сторони, ми створили SWOT-аналіз, який став основою для визначення ключових напрямків розвитку та прийняття обґрунтованих рішень.

Після ретельного дослідження різних архітектурних рішень, було обрано Clean Architecture, як оптимальний варіант для нашого онлайн-сервісу. Цей вибір обумовлений тим, що Clean Architecture забезпечує чіткий поділ відповідальності, що робить код більш зрозумілим, гнучким та керованим. Серверна частина реалізована на платформі .NET з використанням сучасних технологій, таких як ASP .NET Core Web API, який використовується для створення API онлайн-сервісу, JWT Authorization, що забезпечує захист API, EntityFrameworkCore, який використовується для доступу до бази даних та інші. Клієнтська частина онлайн-сервісу розроблена на базі Angular, популярного фреймворку JavaScript для створення односторінкових вебдодатків, що пропонує високу продуктивність, модульність та простоту розробки. Інтерфейс користувача створений за допомогою Bootstrap, популярної CSS-бібліотеки, що робить онлайн-сервіс зручним для використання на різних пристроях. Додатково в проєкті використовуються й інші технології, такі як NgRx, RxJS та TypeScript.

Весь код проєкту доступний на GitHub, що дає можливість іншим розробникам вивчати, використовувати та вдосконалювати його.

База даних онлайн-сервісу створена на основі SQL з використанням підходу CodeFirst. Цей підхід дозволив нам гнучко проектувати та розвивати структуру бази даних, а також генерувати код моделей даних на основі опису таблиць. Реалізовано контролери та репозиторії, які забезпечують основний функціонал онлайн-сервісу. Контролери відповідають за обробку HTTP-запитів та взаємодію з репозиторіями, а репозиторії відповідають за доступ до даних про оголошення в базі даних тощо.

З метою забезпечення безпеки користувачів та захисту їхніх даних в онлайн-сервісі реалізована авторизація на основі JWT Bearer. Завдяки шифруванню паролів та використанню токенів, JWT Bearer авторизація забезпечує надійний захист від несанкціонованого доступу до конфіденційних даних користувачів. Крім того, вона значно знижує навантаження на сервер та є простою у реалізації.

З метою розширення функціональності нашого онлайн-сервісу та надання користувачам можливості отримувати інформацію та чи потрібно переглядати оголошення, ми успішно інтегрували API Ninjas- VIN Lookup API. Це потужний інструмент, який дозволяє нам отримувати вичерпну інформацію про транспортні засоби за їх VIN-кодами. Завдяки ретельній інтеграції та тестуванню, API Ninjas- VIN Lookup API тепер безперебійно працює в нашому онлайн-сервісі, надаючи користувачам доступ до актуальної та детальної інформації про транспортні засоби. Це значно покращує досвід користувачів та робить наш онлайн-сервіс більш цінним інструментом для всіх, хто цікавиться транспортними засобами.

Нами виконано значну роботу з розробки вебдодатку, заклавши міцний фундамент для його успішного функціонування. Завдяки використанню сучасних технологій та ретельному плануванню, нам вдалося створити онлайн-сервіс з високим рівнем продуктивності, безпеки та масштабованості. Наразі ми зосереджені на розширенні функціоналу вебдодатку та його розміщенні на хостингу, щоб зробити його доступним для користувачів. Після завершення цих

кроків ми плануємо продовжувати розвивати та вдосконалювати онлайн-сервіс, щоб він максимально відповідав потребам користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1 “Архітектура клієнт-сервер”. [Електронний ресурс] – URL: http://surl.li/ueyik (дата звернення: 01.12.2023 р.).

2 Client-ServerArchitecture. [Електронний ресурс] – URL: http://surl.li/ueyiu (дата звернення: 06.12.2023 р.).

3 ”What is Swagger?” [Електронний ресурс] – URL: http://surl.li/ueyjc (дата звернення: 08.12.2023 р.).

4 Wikipedia.Вебсервер.[Електронний ресур с] – URL: http://surl.li/ueyjs (дата звернення: 10.12.2023 р.).

5 Wikipedia. Angular (фреймворк). [Електронний ресурс] – URL: http://surl.li/ueyka (дата звернення: 15.12.2023 р.).

6 Angular documentation. [Електронний ресурс] – URL: http://surl.li/ueyke (дата звернення: 25.12.2023 р.).

7 AutoMapper documentation. [Електронний ресурс] / Jimmy Bogard – URL: http://surl.li/ueykp (дата звернення: 05.01.2024 р.).

8 Wikipedia. TypeScript [Електронний ресурс] – URL: http://surl.li/ueykr (дата звернення: 11.01.2024 р.).

9 Google.Generic Repository Pattern for ASP.NET Core. [Електронний ресурс] – URL: http://surl.li/ueylb (дата звернення: 14.01.2024 р.).

10 Wikipedia.Клієнт-серверна архітектура [Електронний ресурс] – URL: http://surl.li/ueyle (дата звернення: 21.01.2024 р.).

11 Аналітичне дослідження вторинного авторинку України [Електронний ресурс] / [Олександр Онищук, Станіслав Бучацький, Остап Новицький, Вадим Малищук] – URL: http://surl.li/ueyll (дата звернення: 12.12.2023 р.).

12 Авто з США у 2023 році: ціни, тенденції, поради Електронний ресурс] – URL: http://surl.li/ueylo (дата звернення: 26.01.2024 р.).

13 АВТО З США. [Електронний ресурс] – URL: http://surl.li/ueylv (дата звернення: 27.01.2024 р.).

14 AUTO RIA. [Електронний ресурс] – URL: http://surl.li/ueyly (дата звернення: 29.01.2024 р.).

15 BOSS AUTO. [Електронний ресурс] – URL: http://surl.li/ueymh (дата звернення: 02.02.2024 р.).

16 Розуміння основ роботи API і REST API – короткий вступ. [Електронний ресурс] – URL: http://surl.li/ueymp (дата звернення: 08.02.2024 р.).

17 Державна прикордонна служба України. [Електронний ресурс] – URL: http://surl.li/ueymx (дата звернення: 12.02.2024 р.).

18 Як імплементувати Clean Architecture та Domain Driven Design на базі Nest.js [Електронний ресурс] – URL: http://surl.li/ueynb (дата звернення: 20.02.2024 р.).

19 What is the Clean Architecture? | by Semih Tekin | Medium [Електронний ресурс] – URL: http://surl.li/ueynh (дата звернення: 25.02.2024 р.).

20 API Ninjas [Електронний ресурс] – URL: http://surl.li/ueynn (дата звернення: 02.03.2024 р.).

21 Wikipedia. Клієнт-серверна архітектура.[Електронний ресурс] – URL: http://surl.li/ueynq (дата звернення: 10.03.2024 р.).

22 Клієнт-серверна архітектура та ролі серверів.| by Ivan Zmerzly| Medium [Електронний ресурс] – URL: http://surl.li/ueynw (дата звернення: 15.03.2024 р.).

23 Angular - CLI Overview and Command Reference. [Електронний ресурс] – URL: http://surl.li/ueyoc (дата звернення: 19.03.2024 р.).

24 Wikipedia. Клієнт-серверна архітектура.[Електронний ресурс] – URL: http://surl.li/ueyoh (дата звернення: 23.03.2024 р.).

25 Як, для чого і де використовувати CQRS, Event Sourcing та DDD.[Електронний ресурс] – URL: http://surl.li/ueyom (дата звернення: 28.03.2024 р.).

26 Wikipedia.Repository.[Електронний ресурс] – URL: http://surl.li/ueyou (дата звернення: 05.04.2024 р.).

27 Різниця між аутентифікацією та авторизацією. [Електронний ресурс] – URL: http://surl.li/ueyow (дата звернення: 10.04.2024 р.).

28 Wikipedia.Microsoft SQL Server.[Електронний ресурс] – URL: http://surl.li/ueyoz (дата звернення: 18.04.2024 р.).

29 Аутентифікація за допомогою JWT-токенів.[Електронний ресурс] – URL: http://surl.li/ueypd (дата звернення: 19.04.2024 р.).

ДОДАТКИ

ДОДАТОК А.

Приклад реалізації репозиторію CarRepository

```
      using Application.Common.Interfaces;
      using Application.DTOs.Car;
      using Domain.Entities;
      using Infrastructure.Data;
      using Microsoft.EntityFrameworkCore;
      using System.Linq.Expressions;
      using Newtonsoft.Json;
      using Microsoft.Extensions.Configuration;

      namespace Infrastructure.Repositories
      {
          public class CarRepository : ICarRepository
          {
              private readonly DataContext _ctx;
              private readonly IImagesRepository _imagesRepository;
              private readonly IConfiguration _config;

              public CarRepository(DataContext ctx, IImagesRepository
imagesRepository, IConfiguration config)
              {
                  _ctx = ctx ?? throw new ArgumentNullException(nameof(_ctx));
                  _imagesRepository = imagesRepository ?? throw new
ArgumentNullException(nameof(_imagesRepository));
                  _config = config;
              }
              public async Task<Car> AddAsync(Car entity)
              {
                  await _ctx.Set<Car>().AddAsync(entity);
                  await _ctx.SaveChangesAsync();

                  return entity;
              }

              public async Task<bool> CheckYear(Car entity)
              {
                  var isVadlidGeneration = await
_ctx.Set<Generation>().FirstOrDefaultAsync(g =>
                  entity.Year >= g.YearFrom && entity.Year <= g.YearTo);

                  if (isVadlidGeneration == null)
                  {
                      return false;
                  }
```

```
                else
                {
                    return true;
                }
            }

            public async Task<Car> UpdateAsync(Car entity)
            {
                _ctx.Set<Car>().Update(entity);
                await _ctx.SaveChangesAsync();

                return entity;
            }

            public async Task DeleteAsync(Car entity)
            {
                var imagesToDelete = _ctx.Set<Images>().Where(image =>
image.CarId == entity.Id);

                _imagesRepository.RemoveImages(imagesToDelete.ToList());

                _ctx.Set<Images>().RemoveRange(imagesToDelete);
                _ctx.Set<Car>().Remove(entity);

                await _ctx.SaveChangesAsync();
            }

            public async Task<IEnumerable<Car>> GetNotVerifiedCarsAsync()
            {
                return await _ctx.Set<Car>()
                    .Include(m => m.Model).ThenInclude(m =>
m.Make).ThenInclude(c => c.ProducingCountry)
                    .Include(m => m.Modification)
                    .Include(g => g.Generation)
                    .Include(b => b.BodyType)
                    .Include(g => g.GearBoxType)
                    .Include(d => d.DriveTrain)
                    .Include(t => t.TechnicalCondition)
                    .Include(f => f.FuelType)
                    .Include(i => i.Images)
                    .Include(u => u.User)
                    .Where(x => x.IsAdvertisementApproved == false)
                    .ToListAsync();
            }

            public async Task<Car> GetByIdAsync(int id)
            {
                return await _ctx.Set<Car>().Include(m =>
m.Model).ThenInclude(m => m.Make).ThenInclude(c => c.ProducingCountry)
                    .Include(m => m.Modification)
                    .Include(g => g.Generation)
```

```
                    .Include(b => b.BodyType)
                    .Include(g => g.GearBoxType)
                    .Include(d => d.DriveTrain)
                    .Include(t => t.TechnicalCondition)
                    .Include(f => f.FuelType)
                    .Include(u => u.User)
                    .Include(i => i.Images).FirstAsync(c => c.Id == id);
            }

            public Car GetById(int id)
            {
                return _ctx.Set<Car>().Find(id);
            }

            public async Task<bool> CheckVin(string vin)
            {
                using (var client = new HttpClient())
                {
                    try
                    {
                        var request = new HttpRequestMessage()
                        {
                            RequestUri = new Uri(@"https://api.api-
ninjas.com/v1/vinlookup?vin=" + vin),
                            Method = HttpMethod.Get,
                        };

                        request.Headers.Add("X-Api-Key",
_config["VinLookup:ApiKey"]);

                        var response = await client.SendAsync(request);

                        if (response.IsSuccessStatusCode)
                        {
                            var responseContent = await
response.Content.ReadAsStringAsync();

                            var vinLookupResponse =
JsonConvert.DeserializeObject<VinLookupResponse>(responseContent);

                            var car = await
_ctx.Set<Car>().FirstOrDefaultAsync(c => c.VIN == vinLookupResponse.Vin);

                            if (car == null)
                                return false;

                            bool matchesAllFields = car.Model?.Make?.Name ==
vinLookupResponse.Manufacturer &&
                                car.Year.ToString() ==
vinLookupResponse.Years.FirstOrDefault();
```

```
                            return matchesAllFields;
                        }
                        else
                        {
                            return false;
                        }
                    }
                    catch (HttpRequestException e)
                    {
                        throw;
                    }
                    catch (Exception e)
                    {
                        throw;
                    }
                }
            }

            public async Task<IEnumerable<Car>> HomeFilter(CarHomeFilter
filter)
            {
                var query = _ctx.Set<Car>().AsQueryable();

                if (filter.MakeId.HasValue)
                {
                    query = query.Include(src => src.Model).ThenInclude(src =>
src.Make).Where(car => car.Model.Make.Id == filter.MakeId);
                }

                if (filter.ModelId.HasValue)
                {
                    query = query.Where(car => car.ModelId == filter.ModelId);
                }

                if (filter.PriceFrom.HasValue)
                {
                    query = query.Where(car => car.Price >= filter.PriceFrom);
                }

                if (filter.PriceTo.HasValue)
                {
                    query = query.Where(car => car.Price <= filter.PriceTo);
                }

                query = query.Include(m => m.Model).ThenInclude(m =>
m.Make).ThenInclude(c => c.ProducingCountry)
                    .Include(m => m.Modification)
                    .Include(g => g.Generation)
                    .Include(b => b.BodyType)
                    .Include(g => g.GearBoxType)
                    .Include(d => d.DriveTrain)
```

```
                    .Include(t => t.TechnicalCondition)
                    .Include(f => f.FuelType)
                    .Include(u => u.User)
                    .Include(i => i.Images);

                var filteredCars = await query.ToListAsync();

                return filteredCars;
            }

            public async Task<IEnumerable<Car>> CarFilter(CarFilter filter)
            {
                var query = _ctx.Set<Car>().AsQueryable();

                if (filter.MakeId.HasValue)
                {
                    query = query.Include(src => src.Model).ThenInclude(src =>
src.Make).Where(car => car.Model.Make.Id == filter.MakeId);
                }

                if (filter.ModelId.HasValue)
                {
                    query = query.Where(car => car.ModelId == filter.ModelId);
                }

                if (filter.GenerationId.HasValue)
                {
                    query = query.Where(car => car.GenerationId ==
filter.GenerationId);
                }

                if (filter.ModificationId.HasValue)
                {
                    query = query.Where(car => car.ModificationId ==
filter.ModificationId);
                }

                if (filter.BodyTypeId.HasValue)
                {
                    query = query.Where(car => car.BodyTypeId ==
filter.BodyTypeId);
                }

                if (filter.GearBoxTypeId.HasValue)
                {
                    query = query.Where(car => car.GearBoxTypeId ==
filter.GearBoxTypeId);
                }

                if (filter.DriveTrainId.HasValue)
                {
```

```
                    query = query.Where(car => car.DriveTrainId ==
filter.DriveTrainId);
                }

                if (filter.TechnicalConditionId.HasValue)
                {
                    query = query.Where(car => car.TechnicalConditionId ==
filter.TechnicalConditionId);
                }

                if (filter.FuelTypeId.HasValue)
                {
                    query = query.Where(car => car.FuelTypeId ==
filter.FuelTypeId);
                }

                if (filter.YearFrom.HasValue)
                {
                    query = query.Where(car => car.Year >= filter.YearFrom);
                }

                if (filter.YearTo.HasValue)
                {
                    query = query.Where(car => car.Year <= filter.YearTo);
                }

                if (filter.MileageFrom.HasValue)
                {
                    query = query.Where(car => car.Mileage >=
filter.MileageFrom);
                }

                if (filter.MileageTo.HasValue)
                {
                    query = query.Where(car => car.Mileage <=
filter.MileageTo);
                }

                if (filter.PriceFrom.HasValue)
                {
                    query = query.Where(car => car.Price >= filter.PriceFrom);
                }

                if (filter.PriceTo.HasValue)
                {
                    query = query.Where(car => car.Price <= filter.PriceTo);
                }

                query = query.Include(m => m.Model).ThenInclude(m =>
m.Make).ThenInclude(c => c.ProducingCountry)
                    .Include(m => m.Modification)
```

```
                        .Include(g => g.Generation)
                        .Include(b => b.BodyType)
                        .Include(g => g.GearBoxType)
                        .Include(d => d.DriveTrain)
                        .Include(t => t.TechnicalCondition)
                        .Include(f => f.FuelType)
                        .Include(u => u.User)
                        .Include(i => i.Images);

                    var filteredCars = await query.ToListAsync();

                    return filteredCars;
                }

                public async Task<IEnumerable<Car>> GetRecentCars(int count)
                {
                    return await _ctx.Cars
                       .OrderByDescending(car => car.DateCreated)
                       .Take(count)
                       .Include(m => m.Model).ThenInclude(m =>
m.Make).ThenInclude(c => c.ProducingCountry)
                        .Include(m => m.Modification)
                        .Include(g => g.Generation)
                        .Include(b => b.BodyType)
                        .Include(g => g.GearBoxType)
                        .Include(d => d.DriveTrain)
                        .Include(t => t.TechnicalCondition)
                        .Include(f => f.FuelType)
                        .Include(i => i.Images)
                        .Include(u => u.User)
                        .ToListAsync();
                }

                public async Task<Car> GetFirstAsync(Expression<Func<Car, bool>>
expression)
                {
                    return await _ctx.Set<Car>().FirstOrDefaultAsync(expression);
                }

                public async Task<IEnumerable<Car>> GetCarByUserId(string userId)
                {
                    return await _ctx.Set<Car>().Where(u => u.UserId ==
userId).Include(m => m.Model).ThenInclude(m => m.Make).ThenInclude(c =>
c.ProducingCountry)
                        .Include(m => m.Modification)
                        .Include(g => g.Generation)
                        .Include(b => b.BodyType)
                        .Include(g => g.GearBoxType)
                        .Include(d => d.DriveTrain)
                        .Include(t => t.TechnicalCondition)
                        .Include(f => f.FuelType)
```

```
                        .Include(i => i.Images)
                        .Include(u => u.User)
                        .ToListAsync();
                }
                public async Task<IEnumerable<Car>> GetAllAsync()
                {
                    return await _ctx.Set<Car>()
                        .Include(m => m.Model).ThenInclude(m =>
m.Make).ThenInclude(c => c.ProducingCountry)
                        .Include(m => m.Modification)
                        .Include(g => g.Generation)
                        .Include(b => b.BodyType)
                        .Include(g => g.GearBoxType)
                        .Include(d => d.DriveTrain)
                        .Include(t => t.TechnicalCondition)
                        .Include(f => f.FuelType)
                        .Include(i => i.Images)
                        .Include(u => u.User)
                        .ToListAsync();
                }
                public async Task SaveChangesAsync()
                {
                    await _ctx.SaveChangesAsync();
                }
            }
        }
```

ДОДАТОК Б.

Приклад реалізації контролеру CarController

```
using Application.Cars.Commands;
using Application.Cars.Queries;
using Application.DTOs.Car;
using Application.Image.Commands;
using AutoMapper;
using MediatR;

namespace Web.Endpoints
{
    [Route("api/[controller]")]
    [ApiController]
    public class CarController : ControllerBase
    {
        private readonly IMapper _mapper;
        private readonly IMediator _mediator;
        private readonly IWebHostEnvironment _hostingEnvironment;
        public CarController(IMapper mapper, IMediator mediator,
IWebHostEnvironment hostingEnvironment)
        {
            _mapper = mapper;
            _mediator = mediator;
            _hostingEnvironment = hostingEnvironment;
        }

        [HttpGet("get-cars")]
        public async Task<IResult> GetCars()
        {
            var cars = await _mediator.Send(new GetAllCars());
            var carDTOs = _mapper.Map<IEnumerable<CarDTO>>(cars);

            return TypedResults.Ok(carDTOs.ToList());
        }

        [HttpGet("get-unverified-cars")]
        public async Task<IResult> GetUnverifiedCars()
        {
            var cars = await _mediator.Send(new GetUnverifiedCars());
            var carDtos = _mapper.Map<IEnumerable<CarDTO>>(cars);

            return TypedResults.Ok(carDtos.ToList());
        }

        [HttpGet("get-car/{id}")]
        public async Task<IResult> GetById(int id)
        {
            var car = await _mediator.Send(new GetCarById { Id = id });
            var carDTO = _mapper.Map<CarDTO>(car);
```

```
                return TypedResults.Ok(carDTO);
            }

            [HttpGet("get-car-for-update/{id}")]
            public async Task<IResult> GetByIdForUpdate(int id)
            {
                var car = await _mediator.Send(new GetCarById { Id = id });
                var carDTO = _mapper.Map<CarCreateDTO>(car);

                return TypedResults.Ok(carDTO);
            }

            [HttpGet("get-car-by-userid/{id}")]
            public async Task<IResult> GetCarByUserId(string id)
            {
                var car = await _mediator.Send(new GetCarByUserId { UserId =
id });
                var carDTO = _mapper.Map<IEnumerable<CarDTO>>(car);

                return TypedResults.Ok(carDTO.ToList());
            }

            [HttpGet("car-home-filter")]
            public async Task<IResult> CarHomeFilter([FromQuery] CarHomeFilter
filter)
            {
                var query = _mapper.Map<CarAdvanceFilter>(filter);
                var car = await _mediator.Send(query);
                var carDTO = _mapper.Map<IEnumerable<CarDTO>>(car);

                return TypedResults.Ok(carDTO.ToList());
            }

            [HttpGet("car-filter")]
            public async Task<IResult> CarFilter([FromQuery] CarFilter filter)
            {
                var query = _mapper.Map<CarAdvanceFilter>(filter);
                var car = await _mediator.Send(query);
                var carDTO = _mapper.Map<IEnumerable<CarDTO>>(car);

                return TypedResults.Ok(carDTO.ToList());
            }

            [HttpGet("get-recent-cars/{count}")]
            public async Task<IResult> GetRecentCars(int count)
            {
                var cars = await _mediator.Send(new GetRecentCars { Count =
count });
                var carDto = _mapper.Map<IEnumerable<CarDTO>>(cars);
```

```
                return TypedResults.Ok(carDto.ToList());
            }

            [HttpPost("create-with-images")]
            public async Task<IResult> CreateCarWithImages([FromForm]
CarCreateDTO carDTO, List<IFormFile> images)
            {
                if (carDTO == null)
                {
                    return Results.BadRequest("Invalid data");
                }

                var createCommand = _mapper.Map<CreateCar>(carDTO);

                var car = await _mediator.Send(createCommand);

                if (images != null && images.Count > 0)
                {
                    foreach (var image in images)
                    {
                        if (image != null && image.Length > 0)
                        {
                            var filePathToImage = AddImagesToDirectory(image);
                            var filePathToDisplay =
filePathToImage.Substring(filePathToImage.LastIndexOf("\\User Photos\\") + 1);
                            filePathToDisplay.Replace(@"\\", "/");

                            if (!string.IsNullOrEmpty(filePathToImage))
                            {
                                var imageAdded = new CreateImage {
ImagePathToDisplay = "https://localhost:7119/" + filePathToDisplay, ImagePath =
filePathToImage, CarId = car.Id, Car = car };
                                await _mediator.Send(imageAdded);
                            }
                        }
                    }
                }
                else
                {
                    var defaultImagePath = AddImagesToDirectory(null);
                    var defaultImagePathToDisplay =
defaultImagePath.Substring(defaultImagePath.LastIndexOf("\\User Photos\\") + 1);
                    defaultImagePathToDisplay.Replace(@"\\", "/");

                    if (!string.IsNullOrEmpty(defaultImagePath))
                    {
                        var defaultImage = new CreateImage {
ImagePathToDisplay = "https://localhost:7119/" + defaultImagePathToDisplay,
ImagePath = defaultImagePath, CarId = car.Id, Car = car };
                        await _mediator.Send(defaultImage);
                    }
```

```
                }

                return Results.Ok(new JsonResult(new { title = "Оголошення
додано успішно", message = "Ваше оголошення додано успішно", id = car.Id }));
            }

            [HttpPut("update-car")]
            public async Task<IResult> UpdateCar([FromBody] CarCreateDTO
carDTO)
            {
                var command = _mapper.Map<UpdateCar>(carDTO);

                var car = await _mediator.Send(command);

                return Results.Ok(new JsonResult(new { title = "Оголошення
оновлено успішно", message = "Ваше оголошення було оновлено успішно", id =
car.Id }));
            }

            [HttpDelete("delete-car/{carId}")]
            public async Task<IResult> DeleteCar(int carId)
            {
                await _mediator.Send(new DeleteCar { Id = carId });

                return Results.Ok(new JsonResult(new { title = "Оголошення
видалено успішно", message = "Ваше оголошення було видалено успішно" }));
            }

            [HttpGet("check-vin/{vin}")]
            public async Task<IResult> CheckVin(string vin)
            {
                var isValid = await _mediator.Send(new CheckCarByVin { VIN =
vin });

                var response = new VinCheckResponse
                {
                    IsValid = isValid,
                    Message = isValid ? "Перевірений VIN" : "Проблема з
оголошенням, будьте обережні при купівлі цього автомобіля"
                };

                return TypedResults.Ok(response);

            }

            #region private Helpers
            private string AddImagesToDirectory(IFormFile? images)
            {
                if (images == null || images.Length == 0)
                {
                    var uniqueFileName = "NoImage.png";
```

```
                    var uploadsFolder =
Path.Combine(_hostingEnvironment.WebRootPath, "User Photos");
                    var filePathCombained = Path.Combine(uploadsFolder,
uniqueFileName);

                    return filePathCombained;
                }
                else if (images != null && images.Length > 0)
                {
                    var uniqueFileName = Guid.NewGuid().ToString() + "_" +
images.FileName;
                    var uploadsFolder =
Path.Combine(_hostingEnvironment.WebRootPath, "User Photos");

                    if (!Directory.Exists(uploadsFolder))
                    {
                        Directory.CreateDirectory(uploadsFolder);
                    }
                    var filePathCombained = Path.Combine(uploadsFolder,
uniqueFileName);

                    using (var fileStream = new FileStream(filePathCombained,
FileMode.Create))
                    {
                        images.CopyTo(fileStream);
                    }

                    return filePathCombained;
                }
                else
                {
                    return null;
                }
            }
            #endregion
        }
    }
```

ДОДАТОК В.

Приклад реалізації коду в контролері для аутентифікації та авторизації

```
[Authorize]
[HttpGet("refresh-page")]
public async Task<ActionResult<ApplicationUserDTO>> RefreshPage()
{
    var query = new RefreshPageQuery
    {
        Email = User.FindFirst(ClaimTypes.Email)?.Value
    };

    try
    {
        var result = await _mediator.Send(query);
        return Ok(result);
    }
    catch (Exception)
    {
        return StatusCode(500, "Internal Server Error");
    }
}

[HttpPost("login")]
public async Task<IActionResult> Login(LoginDTO model)
{
    var query = new LoginUserQuery
    {
        UserName = model.UserName,
        Password = model.Password,
    };

    try
    {
        var result = await _mediator.Send(query);
        return Ok(result);
    }
    catch (UnauthorizedAccessException ex)
    {
        return Unauthorized(ex.Message);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Internal Server Error: {ex.Message}");
    }
}

[HttpPost("register")]
public async Task<IActionResult> Register(RegisterDTO model)
{
```

```
            try
            {
                await _mediator.Send(new RegisterUserCommand { RegistrationData =
model });

                return Ok(new JsonResult(new { title = "Акаунт створено успішно",
message = "Ваш акаунт створено успішно, підтвердіть, будь ласка, вашу email
пошту" }));
            }
            catch (ValidationException ex)
            {
                return BadRequest(ex.Message);
            }
            catch (Exception)
            {
                return StatusCode(500, "Internal Server Error");
            }
        }
```

ДОДАТОК Г.

Приклад реалізації JWT-сервісу

```
namespace Web.Infrastructure
{
    public class JWTServices
    {
        private readonly IConfiguration _config;
        private readonly UserManager<User> _userManager;
        private readonly SymmetricSecurityKey _jwtKey;
        public JWTServices(IConfiguration config, UserManager<User> userManager)
        {
            _config = config;
            _userManager = userManager;

            // jwt ключ використаний для шифрування та розшифрування jwt токенів
            _jwtKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["JWT:Key"]));
        }
        public async Task<string> CreateJWT(User user)
        {
            var userClaims = new List<Claim>
            {
                new Claim(ClaimTypes.NameIdentifier, user.Id),
                new Claim(ClaimTypes.Email, user.Email),
                new Claim(ClaimTypes.GivenName, user.FirstName),
                new Claim(ClaimTypes.GivenName, user.LastName),
            };

            var roles = await _userManager.GetRolesAsync(user);

            userClaims.AddRange(roles.Select(role => new Claim(ClaimTypes.Role, role)));

            var credentials = new SigningCredentials(_jwtKey, SecurityAlgorithms.HmacSha256Signature);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(userClaims),
                Expires = DateTime.UtcNow.AddDays(int.Parse(_config["JWT:ExpiresInDays"])),
                SigningCredentials = credentials,
                Issuer = _config["JWT:Issuer"]
            };

            var tokenHandler = new JwtSecurityTokenHandler();
            var jwt = tokenHandler.CreateToken(tokenDescriptor);

            return tokenHandler.WriteToken(jwt);
        }

        public RefreshToken CreateRefreshToken(User user)
        {
            var token = new byte[32];
            using var randomNumberGenerator = RandomNumberGenerator.Create();
            randomNumberGenerator.GetBytes(token);
```

```
                var refreshToken = new RefreshToken()
                {
                    Token = Convert.ToBase64String(token),
                    User = user,
                    DateExpiresUtc                                                    =
DateTime.UtcNow.AddDays(int.Parse(_config["JWT:ExpiresInDays"]))
                };

                return refreshToken;
            }
        }
    }
```

ДОДАТОК Д.

Приклад реалізації політики авторизації

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace Web.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class RCPracticeController : ControllerBase
    {
        [HttpGet("public")]
        public IActionResult Public()
        {
            return Ok("public");
        }

        #region Roles


        [HttpGet("admin-role")]
        [Authorize(Roles = "Admin")]
        public IActionResult AdminRole()
        {
            return Ok("admin role");
        }

        [HttpGet("manager-role")]
        [Authorize(Roles = "Manager")]
        public IActionResult ManagerRole()
        {
            return Ok("manager role");
        }

        [HttpGet("member-role")]
        [Authorize(Roles = "Member")]
        public IActionResult MemberRole()
        {
            return Ok("member role");
        }

        [HttpGet("admin-or-manager-role")]
        [Authorize(Roles = "Admin,Manager")]
        public IActionResult AdminOrManagerRole()
        {
            return Ok("admin or manager role");
        }

        [HttpGet("admin-or-member-role")]
        [Authorize(Roles = "Admin,Member")]
        public IActionResult AdminOrMemberRole()
        {
            return Ok("admin or member role");
        }
        #endregion
```

```
#region Policy

[HttpGet("admin-policy")]
[Authorize(policy: "AdminPolicy")]
public IActionResult AdminPolicy()
{
    return Ok("admin policy");
}

[HttpGet("manager-policy")]
[Authorize(policy: "ManagerPolicy")]
public IActionResult ManagerPolicy()
{
    return Ok("manager policy");
}

[HttpGet("member-policy")]
[Authorize(policy: "MemberPolicy")]
public IActionResult MemberPolicy()
{
    return Ok("member policy");
}


[HttpGet("admin-or-manager-policy")]
[Authorize(policy: "AdminOrManagerPolicy")]
public IActionResult AdminOrManagerPolicy()
{
    return Ok("admin or manager policy");
}

[HttpGet("admin-and-manager-policy")]
[Authorize(policy: "AdminAndManagerPolicy")]
public IActionResult AdminAndManagerPolicy()
{
    return Ok("admin and manager policy");
}

[HttpGet("all-role-policy")]
[Authorize(policy: "AllRolePolicy")]
public IActionResult AllRolePolicy()
{
    return Ok("all role policy");
}

#endregion

#region Claim Policy

[HttpGet("admin-email-policy")]
[Authorize(policy: "AdminEmailPolicy")]
public IActionResult AdminEmailPolicy()
{
    return Ok("admin email policy");
}
```

```
        [HttpGet("admin-surname-policy")]
        [Authorize(policy: "AdminSurnamePolicy")]
        public IActionResult MillerSurnamePolicy()
        {
            return Ok("miller surname policy");
        }

        [HttpGet("manager-email-and-manager-surname-policy")]
        [Authorize(policy: "ManagerEmailAndManagerSurnamePolicy")]
        public IActionResult ManagerEmailAndManagerSurnamePolicy()
        {
            return Ok("manager email and manager surname policy");
        }

        [HttpGet("vip-policy")]
        [Authorize(policy: "VIPPolicy")]
        public IActionResult VIPPolicy()
        {
            return Ok("vip policy");
        }

        #endregion
    }
}
```

ДОДАТОК Е.

Приклад реалізації компонененту на type script

Top of Form

```
import { ModelService } from './../../../services/model.service';
import { Component, OnInit } from '@angular/core';
import { AbstractControl, FormBuilder, FormControl, FormGroup, ValidatorFn, Validators } from '@angular/forms';
import { Router } from '@angular/router';
import { BodyType } from 'src/app/models/body-type/body-type';
import { DriveTrain } from 'src/app/models/drive-train/drive-train';
import { FuelType } from 'src/app/models/fuel-type/fuel-type';
import { GearBox } from 'src/app/models/gearbox/gearbox';
import { Generation } from 'src/app/models/generation/generation';
import { Make } from 'src/app/models/make/make';
import { Model } from 'src/app/models/model/model';
import { Modification } from 'src/app/models/modification/modification';
import { ProducingCountry } from 'src/app/models/producing-country/producing-country';
import { TechnicalCondition } from 'src/app/models/technical-condition/technical-condition';
import { AccountService } from 'src/app/services/account.service';
import { BodyTypeService } from 'src/app/services/body-type.service';
import { CarService } from 'src/app/services/car.service';
import { DriveTrainService } from 'src/app/services/drive-train.service';
import { FuelTypeService } from 'src/app/services/fuel-type.service';
import { GearBoxTypeService } from 'src/app/services/gear-box-type.service';
import { GenerationService } from 'src/app/services/generation.service';
import { MakeService } from 'src/app/services/make.service';
import { ModificationService } from 'src/app/services/modification.service';
import { ProducingCountryService } from 'src/app/services/producing-country.service';
import { TechnicalConditionService } from 'src/app/services/technical-condition.service';
import { SharedService } from 'src/app/shared/shared.service';

@Component({
  selector: 'app-car-add',
  templateUrl: './car-add.component.html',
  styleUrls: ['./car-add.component.css']
})
export class CarAddComponent implements OnInit {

  addCarForm: FormGroup = new FormGroup({});
  selectedImages: File[] = [];
  errorMessages: string[] = [];
  submitted = false;

  countries: ProducingCountry[] = [];
```

```
  bodytypes: BodyType[] = [];
  gearboxes: GearBox[] = [];
  fueltypes: FuelType[] = [];
  drivetrains: DriveTrain[] = [];
  technicalconditions: TechnicalCondition[] = [];

  filteredMakes: Make[] = [];
  filteredModels: Model[] = [];
  filteredGenerations: Generation[] = [];
  filteredModifications: Modification[] = [];

  currentUser = this.accountService.user$;

  constructor(private carService: CarService,
    private countryServise: ProducingCountryService,
    private makeService: MakeService,
    private generationService: GenerationService,
    private modelService: ModelService,
    private modificationService: ModificationService,
    private bodyTypeService: BodyTypeService,
    private gearBoxService: GearBoxTypeService,
    private fuelTypeService: FuelTypeService,
    private driveTrainService: DriveTrainService,
    private technicalConditionService: TechnicalConditionService,
    private formBuilder: FormBuilder,
    private sharedService: SharedService,
    private router: Router,
    private accountService: AccountService) { }

  ngOnInit(): void {
    this.initializeForm();
    this.getMakes();
    this.getCountries();
    this.getBodyTypes();
    this.getGearBoxes();
    this.getFuelTypes();
    this.getDriveTrain();
    this.getTechnicalCondition();

    this.addCarForm.get('countryId')?.valueChanges.subscribe((countryId) => {
      if (countryId) {
        this.filteredModels = [];
        this.filteredGenerations = [];
        this.filteredModifications = [];
      }

      if (countryId === 0) {
        this.getMakes();
      } else {
```

```
            this.makeService.getMakeByCountry(countryId).subscribe(data => {
              this.filteredMakes = data;
            });
          }
        });

        this.addCarForm.get('makeId')?.valueChanges.subscribe((makeId) => {

          if (makeId) {
            this.filteredModels = [];
            this.filteredGenerations = [];
            this.filteredModifications = [];
          }

          this.modelService.getModelsByMake(makeId).subscribe(data => {
            this.filteredModels = data;
          });
        });

        this.addCarForm.get('modelId')?.valueChanges.subscribe((modelId) => {

          if (modelId) {
            this.filteredGenerations = [];
            this.filteredModifications = [];
          }

          this.generationService.getGenerationByModel(modelId).subscribe(data => {
            this.filteredGenerations = data;
          });
          this.modificationService.getModificationsByModel(modelId).subscribe(data =>
{
            this.filteredModifications = data;
          });
        });

        this.currentUser.subscribe((user) => {
          if (user) {
            this.addCarForm.patchValue({ userId: user.id });
          }
        });
      }

      onImagesSelected(event: any) {
        this.selectedImages = event.target.files;
      }

      initializeForm() {
        this.addCarForm = this.formBuilder.group({
          countryId: [''],
```

```
            makeId: [''],
            modelId: ['', [Validators.required]],
            generationId: ['', [Validators.required]],
            modificationId: ['', [Validators.required]],
            vin:    ['',    [Validators.required,    Validators.pattern('^[A-HJ-NPR-Z0-
9]{17}$')]],
            bodyTypeId: ['', [Validators.required]],
            gearBoxTypeId: ['', [Validators.required]],
            driveTrainId: ['', [Validators.required]],
            technicalConditionId: ['', [Validators.required]],
            fuelTypeId: ['', [Validators.required]],
            year:        ['',        [Validators.required,        Validators.min(1000),
Validators.pattern(/^\d+$/), this.yearWithinGenerationRangeValidator()]],
            price: ['', [Validators.required, Validators.pattern(/^\d+$/)]],
            mileage: ['', [Validators.required, Validators.pattern(/^\d+$/)]],
            description: ['', [Validators.required]],
            userId: [''],
          });

          this.addCarForm.get('modificationId')?.clearValidators();
          this.addCarForm.get('modificationId')?.updateValueAndValidity();
        }

        yearWithinGenerationRangeValidator(): ValidatorFn {
          return (control: AbstractControl): { [key: string]: any } | null => {
            const year = control.value;

            const generationId = this.addCarForm.get('generationId')?.value;

            const   selectedGeneration   =   this.filteredGenerations.find(generation   =>
generation.id == generationId);

            if (selectedGeneration) {
              const generationYearFrom = selectedGeneration.yearFrom;
              const generationYearTo = selectedGeneration.yearTo;

              const currentYear = new Date().getFullYear();

              if  (year  <  generationYearFrom  ||  year  >  Math.min(generationYearTo,
currentYear)) {
                return { yearNotInValidRange: 'Рік не входить у дійсний діапазон для
вибраного покоління.' };
              }
            }
            return null;
          };
        }

        addCar() {
```

```
          this.submitted = true;
          this.errorMessages = [];

          if (this.addCarForm.valid) {
            const carData = this.addCarForm.value;

            this.carService.addCar(carData, this.selectedImages).subscribe({
              next: (response: any) => {
                this.sharedService.showNotification(true,         response.value.title,
response.value.message);
                if (response.value.id) {
                  this.router.navigateByUrl(`/car/car-details/${response.value.id}`)
                }
                console.log(response);
              },
              error: error => {
                console.log('Error object:', error);
                if (error.error.error) {
                  this.errorMessages = error.error.error;
                } else {
                  this.errorMessages.push(error.error)
                }
              }
            })
          }
        }

        getCountries() {
          this.countryServise.getProducingCountries().subscribe(data => {
            this.countries = data;
          })
        }

        getMakes() {
          this.makeService.getMakes().subscribe(data => {
            this.filteredMakes = data;
          })
        }

        getBodyTypes() {
          this.bodyTypeService.getBodyTypes().subscribe(data => {
            this.bodytypes = data;
          })
        }

        getGearBoxes() {
          this.gearBoxService.getGearBoxes().subscribe(data => {
            this.gearboxes = data;
          })
```

```
    }

    getFuelTypes() {
      this.fuelTypeService.getFuelTypes().subscribe(data => {
        this.fueltypes = data;
      })
    }

    getDriveTrain() {
      this.driveTrainService.getDriveTrains().subscribe(data => {
        this.drivetrains = data;
      })
    }

    getTechnicalCondition() {
      this.technicalConditionService.getTechnicalConditions().subscribe(data => {
        this.technicalconditions = data;
      })
    }
  }
```

ДОДАТОК Ж.

Приклад реалізації компоненту на HTML та Bootstrap

```
      <div class="wrapper">
          <div class="container">
              <div class="row">
                  <div class="col-md-12">
                      <main class="form-carAdd">
                          <form [formGroup]="addCarForm" (ngSubmit)="addCar()"
autocomplete="off">
                              <div class="text-center mb-4 col-md-12">
                                  <h3 class="mb-3 font-weight-normal">Додати
оголошення</h3>
                              </div>
                              <div class="row">
                                  <div class="form-floating mb-3 col-md-12">
                                      <select formControlName="countryId"
class="form-select shadow-none" >
                                          <option selected value="">Країна
виробник</option>
                                          <option *ngFor="let county of
countries" [value]="county.id">{{county.name}}</option>
                                      </select>
                                  </div>

                                  <div class="form-floating mb-3 col-md-6">
                                      <select formControlName="makeId"
class="form-select shadow-none">
                                          <option selected disabled
value="">Марка</option>
                                          <option *ngFor="let make of
filteredMakes" [value]="make.id">{{make.name}}</option>
                                      </select>
                                  </div>

                                  <div class="form-floating mb-3 col-md-6">
                                      <select formControlName="modelId"
class="form-select shadow-none">
                                          <option selected disabled
value="">Модель</option>
                                          <option *ngFor="let model of
filteredModels" [value]="model.id">{{model.name}}</option>
                                      </select>
                                      <span class="text-danger" *ngIf="submitted
&& addCarForm.get('modelId')?.hasError('required')">
                                          Модель обов'язкова
                                      </span>
                                  </div>

                                  <div class="form-floating mb-3 col-md-6">
```

```
                                    <select formControlName="generationId"
class="form-select shadow-none">
                                        <option selected disabled
value="">Покоління</option>
                                        <option *ngFor="let generation of
filteredGenerations" [value]="generation.id">{{generation.name}}
[{{generation.yearFrom}} - {{generation.yearTo}}]</option>
                                    </select>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('generationId')?.hasError('required')">
                                        Покоління обов`язкове
                                    </span>
                                </div>

                                <div class="form-floating mb-3 col-md-6">
                                    <select formControlName="modificationId"
class="form-select shadow-none">
                                        <option selected disabled
value="">Модифікація</option>
                                        <option *ngFor="let modification of
filteredModifications" [value]="modification.id">{{modification.name}}</option>
                                    </select>
                                </div>

                                <div class="form-floating mb-3 col-md-12">
                                    <input formControlName="vin" type="text"
placeholder="VIN" class="form-control shadow-none"
                                        [class.is-invalid]="submitted &&
addCarForm.get('vin')?.errors">
                                    <label for="vin">VIN</label>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('vin')?.hasError('required')">
                                        VIN обов`язкове
                                    </span>
                                </div>

                                <div class="form-floating mb-3 col-md-4">
                                    <select formControlName="bodyTypeId"
class="form-select shadow-none">
                                        <option selected disabled value="">Тип
кузова</option>
                                        <option *ngFor="let bodyType of
bodytypes" [value]="bodyType.id">{{bodyType.name}}</option>
                                    </select>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('bodyTypeId')?.hasError('required')">
                                        Тип кузоав обов'язковий
                                    </span>
                                </div>

                                <div class="form-floating mb-3 col-md-4">
```

```
                                    <select formControlName="gearBoxTypeId"
class="form-select shadow-none">
                                        <option selected disabled value="">Тип
коробки передач</option>
                                        <option *ngFor="let gearbox of
gearboxes" [value]="gearbox.id">{{gearbox.name}}</option>
                                    </select>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('gearBoxTypeId')?.hasError('required')">
                                        Тип коробки передач обов`язковий
                                    </span>
                                </div>

                                <div class="form-floating mb-3 col-md-4">
                                    <select formControlName="driveTrainId"
class="form-select shadow-none">
                                        <option selected disabled value="">Тип
приводу</option>
                                        <option *ngFor="let drivetrain of
drivetrains" [value]="drivetrain.id">{{drivetrain.name}}</option>
                                    </select>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('driveTrainId')?.hasError('required')">
                                        Тип приводу обов'язкове
                                    </span>
                                </div>

                                <div class="form-floating mb-3 col-md-4">
                                    <select
formControlName="technicalConditionId" class="form-select shadow-none">
                                        <option selected disabled
value="">Технічний стан</option>
                                        <option *ngFor="let technicalCondition
of technicalconditions"
[value]="technicalCondition.id">{{technicalCondition.name}}</option>
                                    </select>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('technicalConditionId')?.hasError('required')">
                                        Технічний стан обов'язкове
                                    </span>
                                </div>

                                <div class="form-floating mb-3 col-md-4">
                                    <select formControlName="fuelTypeId"
class="form-select shadow-none">
                                        <option selected disabled value="">Тип
палива</option>
                                        <option *ngFor="let fueltype of
fueltypes" [value]="fueltype.id">{{fueltype.name}}</option>
                                    </select>
```

```
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('fuelTypeId')?.hasError('required')">
                                        Тип палива обов`язковий
                                    </span>
                                </div>

                                <div class="form-floating mb-3 col-md-4">
                                    <input formControlName="year"
type="number" placeholder="Рік" class="form-control shadow-none"
                                        [class.is-invalid]="submitted &&
addCarForm.get('year')?.errors">
                                    <label for="year">Рік</label>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('year')?.hasError('required')">
                                        Рік обов'язковий
                                    </span>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('year')?.hasError('yearNotInValidRange')">
                                        {{
addCarForm.get('year')?.getError('yearNotInValidRange')}}
                                      </span>
                                </div>

                                <div class="form-floating mb-3 col-md-6">
                                    <input formControlName="price"
type="number" placeholder="Ціна" class="form-control shadow-none"
                                        [class.is-invalid]="submitted &&
addCarForm.get('price')?.errors">
                                    <label for="price">Ціна</label>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('price')?.hasError('required')">
                                        Ціна обов`язкова
                                    </span>
                                </div>

                                <div class="form-floating mb-3 col-md-6">
                                    <input formControlName="mileage"
type="number" placeholder="Пробіг" class="form-control shadow-none"
                                        [class.is-invalid]="submitted &&
addCarForm.get('mileage')?.errors">
                                    <label for="mileage">Пробіг</label>
                                    <span class="text-danger" *ngIf="submitted
&& addCarForm.get('mileage')?.hasError('required')">
                                        Пробіг обов`язковий
                                    </span>
                                </div>

                                <div class="form-floating mb-3 col-md-12">
                                    <textarea formControlName="description"
type="text" placeholder="Опис" class="form-control shadow-none"
```

```
                                        [class.is-invalid]="submitted &&
addCarForm.get('description')?.errors"></textarea>
                                        <label for="description">Опис</label>
                                        <span class="text-danger" *ngIf="submitted
&& addCarForm.get('description')?.hasError('required')">
                                            Опис обов'язковий
                                        </span>
                                    </div>

                                    <div class="form-floating mb-3 col-md-12">
                                        <input type="file" id="images"
name="images" (change)="onImagesSelected($event)" class="form-control shadow-
none" multiple>
                                        <label for="images">Фотографії</label>
                                    </div>

                                    <div class="form-floating"
*ngIf="errorMessages.length > 0">
                                        <app-validation-messages
[errorMessages]="errorMessages"></app-validation-messages>
                                    </div>

                                    <div class="d-grid col-md-12">
                                        <button class="btn btn-add-car"
type="submit">Подати оголошення</button>
                                    </div>
                                </div>
                            </form>
                        </main>
                    </div>
                </div>
            </div>
        </div>
```