КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра теорії та технології програмування

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття ступеня бакалавра

за спеціальністю 122 Комп’ютерні науки

на тему:

ВЕБЗАСТОСУНОК ДЛЯ РОЗМІЩЕННЯ ОГОЛОШЕНЬ ПРО ПРОДАЖ АВТОМОБІЛІВ

Виконав студент 4-го курсу
Гліб ПЕТРУК _____

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Людмила ОМЕЛЬЧУК _____

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Роботу розглянуто й допущено до захисту на засіданні кафедри теорії та технології програмування
«05» червня 2023р.,
протокол № 18
Завідувач кафедри
Микола. НІКІТЧЕНКО _____

Київ – 2023

РЕФЕРАТ

Загальний обсяг роботи становить 56 сторінок, з них основна частина складає 45 сторінок, 26 ілюстрацій, 19 таблиць, 16 джерел посилань, 7 додатків.

БАЗА ДАНИХ, ВЕБЗАСТОСУНОК, ПРОДАЖ АВТОМОБІЛІВ, ANGULAR, ASP.NET CORE, ONION-АРХІТЕКТУРА.

Об’єктом роботи є процес розміщення та перегляду оголошень про продаж автомобілів. Предметом є вебзастосунок для розміщення та перегляду оголошень про продаж автомобілів.

Метою кваліфікаційної роботи є створення вебзастосунку для власників автомобілів, що хочуть продати свій автомобіль, мати можливість переглядати власні оголошення в одному місці та аналогічні пропозиції на рику, та для користувачів, що хочуть придбати автомобіль або переглянути наявність у продажу певного транспортного засобу та інформацію про нього.

Методи розроблення: аналіз наявних застосунків, розробка бази даних, проєктування власного застосунку. Інструменти розроблення: інтегроване середовище розробки Visual Studio 2022, Visual Studio Code, мови програмування C#, TypeScript, система керування базами даних Microsoft SQL Server Manager Studio.

Результат роботи: проведено аналіз наявних на ринку аналогічних застосунків, розроблено власний застосунок для зручного розміщення та перегляду оголошень про продаж автомобілів.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AAA – Arrange, Act, Assert, принцип тестування;

API – Application Programming Interface, інтерфейс прикладного програмування, який представляє собою набір готових класів, процедур, функцій, структур і констант;

CSS – Cascading Style Sheets, спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду;

DI – Dependency Injection, шаблон проєктування програмного забезпечення, що передбачає надання зовнішньої залежності програмному компоненту;

HTTP – HyperText Transfer Protocol, протокол передачі даних, що використовується в комп'ютерних мережах;

IDE – Integrated Design Environment, інтегроване середовище розробки;

IoC – Inversion of Control, принцип побудови програми, при якому її частини викликаються із загальної бібліотеки;

SPA – Single Page Application, односторінковий застосунок;

SQL – Structured Query Language, мова програмування для взаємодії користувача з базами даних;

VIN – Vehicle Identification Number, унікальний серійний номер транспортного засобу.

ВСТУП

Оцінка сучасного стану об’єкта розробки. У сучасному світі автомобілі є невід’ємною частиною життя людини. Завдяки ним, люди мають швидкий і зручний спосіб пересування на великі відстані, що дозволяє ефективно використовувати свій час.

Незважаючи на зростаючі проблеми з екологією та забрудненням повітря, з кожним роком автомобілів у світі стає більше [1]. Вони допомагають розвивати промисловість та торгівлю, що позитивно впливає на економіку.

На українському ринку існує декілька вебзастосунків, призначених для розміщення оголошень про продаж автомобілів.

Актуальність роботи та підстави для її виконання. Як було зазначено вище, кількість автомобілів у світі зростає з кожним роком [1], і Україна не є виключенням. Станом на 2021-й рік, кількість зареєстрованих автомобілів в Україні складала 10,2 мільйонів одиниць, з яких легкових – 8,8 мільйонів. За даними київської мерії від січня 2022 року, лише в Києві кількість активних автомобілів щодоби становить 1,13 мільйонів [2].

Якщо зіставити вищезазначену кількість автомобілів та кількість населення України, то отримаємо близько 250 одиниць автомобілів на тисячу населення. Ця цифра менша у порівнянні з країнами Євросоюзу, у яких цей показник становить 500 одиниць. Тому виникає необхідність створення зручного сервісу для продажу автомобілів, де користувач зможе швидко знайти необхідну модель та переглянути актуальну інформацію про неї.

Мета й завдання роботи. Метою кваліфікаційної роботи є розробка вебзастосунку для розміщення та перегляду оголошень про продаж автомобілів. Для досягнення цієї мети поставлено наступні завдання:

- проведення аналізу предметної області;

- огляд наявних на ринку вебзастосунків, призначених для продажу автомобілів;
- проєктування бази даних;
- реалізація вебзастосунку.

Об’єкт, методи та засоби розроблення. Об’єктом розроблення є процес розміщення та перегляду оголошень про продаж автомобілів. Предметом роботи є вебзастосунок для автоматизації процесу розміщення та перегляду оголошень про продаж автомобілів. Перед розробкою застосунку проведено аналіз наявних на ринку вебзастосунків, що розв’язують подібну проблему.

Для розробки програмної частини застосунку було обрано мову програмування C# та інтегроване середовище розробки (IDE) Visual Studio 2022. Для розробки клієнтської частини було використано фреймворк Angular, написаний на мові TypeScript та інтегроване середовище розробки (IDE) Visual Studio Code.

Можливі сфери застосування. Розроблений застосунок може бути використаний як власниками автомобілів для розміщення оголошень про продаж, зручного перегляду власних оголошень та аналогічних пропозицій на ринку, так і покупцями для перегляду наявності у продажу автомобіля або інформації про певний транспортний засіб.

РОЗДІЛ 1.

АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Огляд проблеми

На сьогоднішній день автомобільна індустрія є однією з найбільш швидкозростаючих галузей, тому попит на автомобілі зростає постійно [1].

Більшість покупців віддають перевагу інтернет-ресурсам, щоб знайти необхідну їм інформацію. Однак проблемою є те, що багато сайтів, які надають послуги з розміщення оголошень про продаж автомобілів, є досить складними та незручними в користуванні. Вони можуть бути заповнені рекламою або мати функціонал, який не дозволяє користувачам легко знайти необхідну інформацію.

На початковому етапі розробки було проведено аналіз наявних на українському ринку застосунків. Розглянемо деякі з них.

1.2 Застосунок «AUTO.RIA»

AUTO.RIA є одним з найбільших в Україні інтернет-ресурсів, який присвячений темі автомобілів та їх продажу. На рис. 1 зображено головну сторінку сайту AUTO.RIA [3].

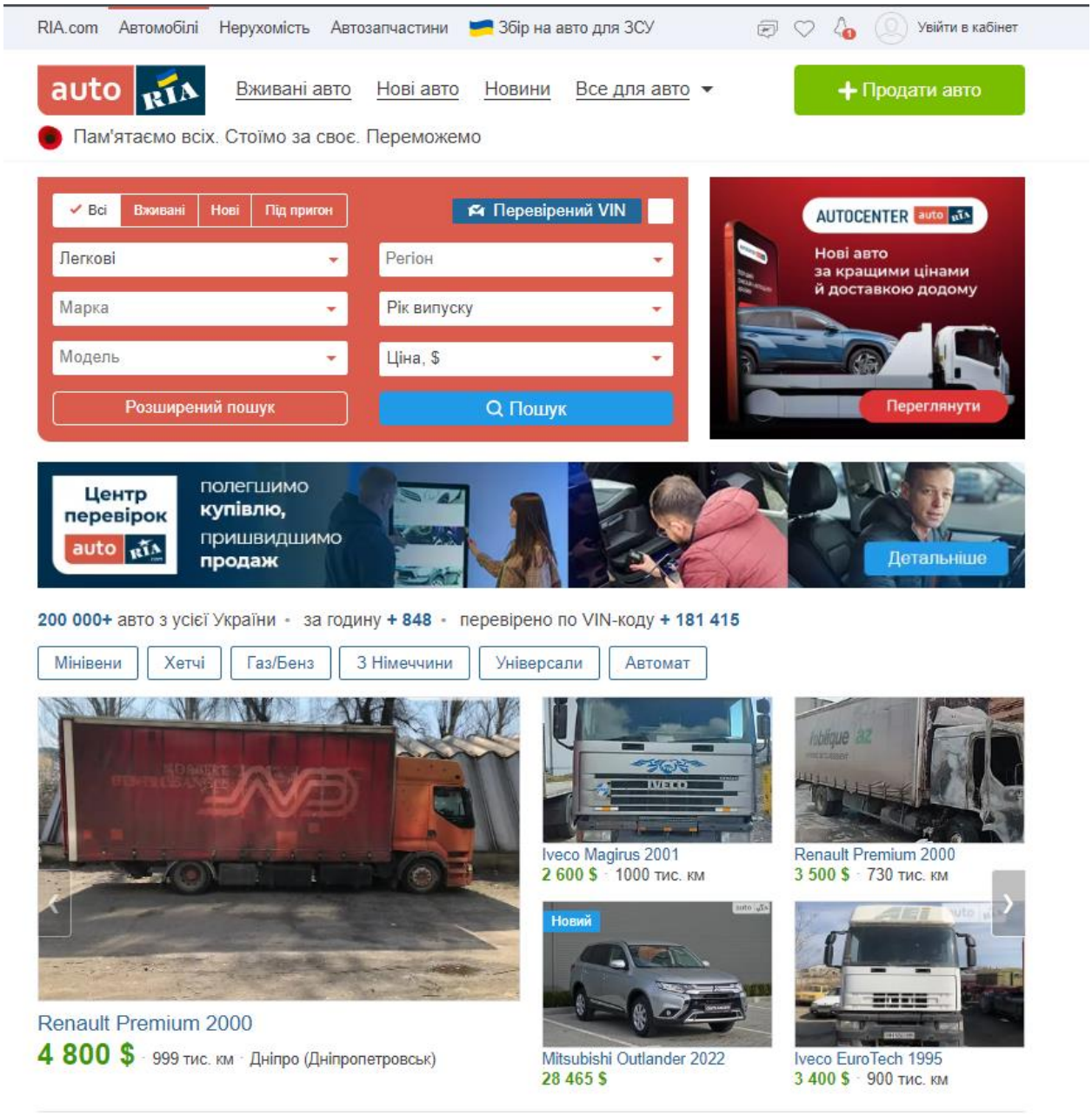


Рисунок 1 – Головна сторінка сайту «AUTO.RIA»

На сторінці знаходиться низка основних розділів, які допомагають зорієнтуватися на сайті та знайти потрібне оголошення:

1. Автомобілі – основна сторінка сайту, на якій можна переглянути активні оголошення, а також застосувати фільтр для пошуку.
2. Вживані авто – сторінка з інформацією про автомобілі з пробігом.
3. Нові авто – сторінка з інформацією про нові автомобілі.

4. Новини – розділ, який містить останні новини зі світу автомобілів, новітні технології та тренди у автомобільній індустрії.

Однією з найбільших переваг головної сторінки AUTO.RIA є те, що користувачі можуть здійснювати пошук автомобілів за різними параметрами, від базових, таких як рік випуску та ціна, до детальних, таких як наявність та тип камери заднього виду, тип приводу, максимальна швидкість та інші. Також наявна можливість авторизації, після якої можна перейти в особистий кабінет користувача (див. рис. 2).

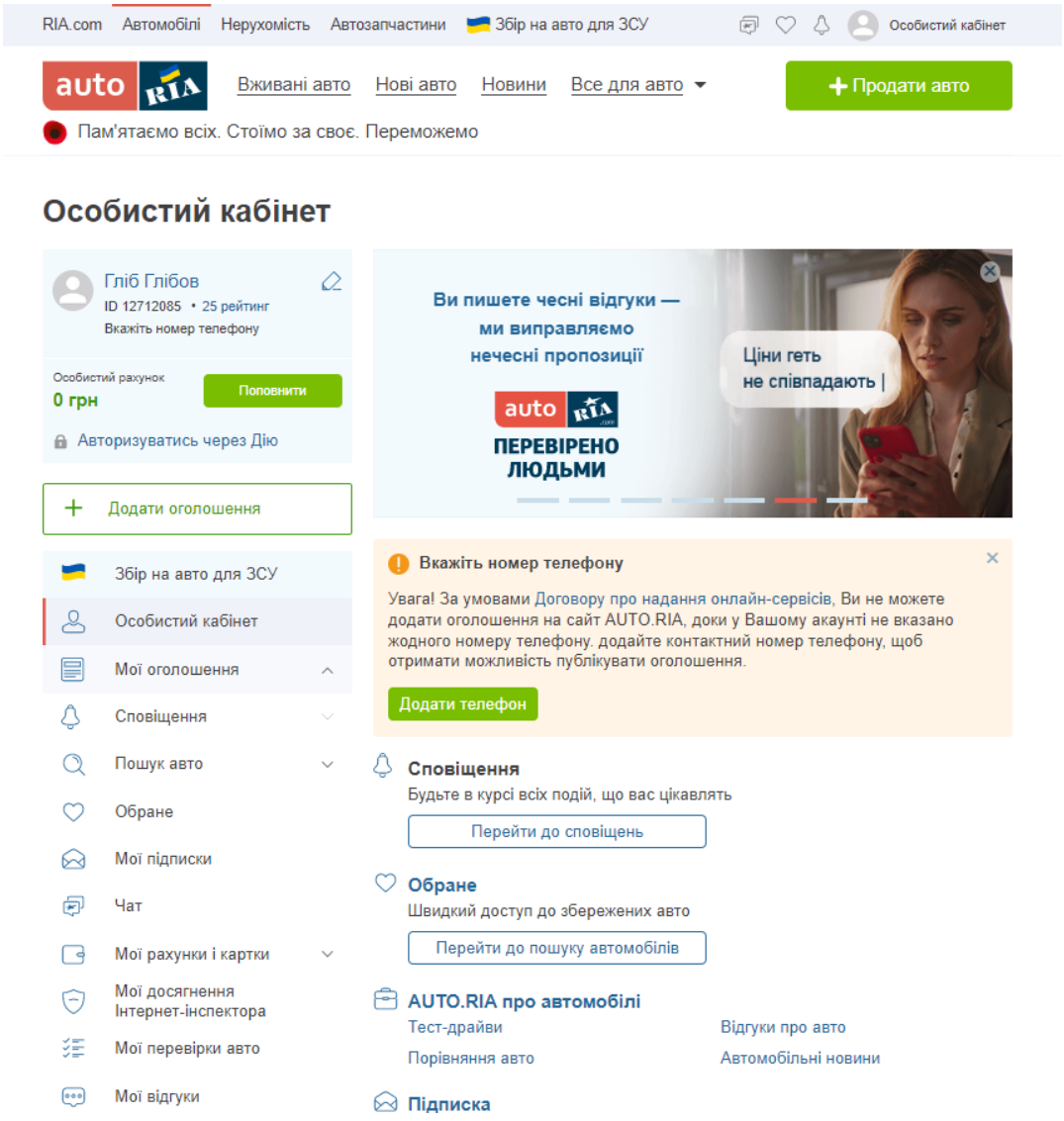


Рисунок 2 – Особистий кабінет на сайті «AUTO.RIA»

На рис. 3 зображено сторінку додавання нового оголошення.

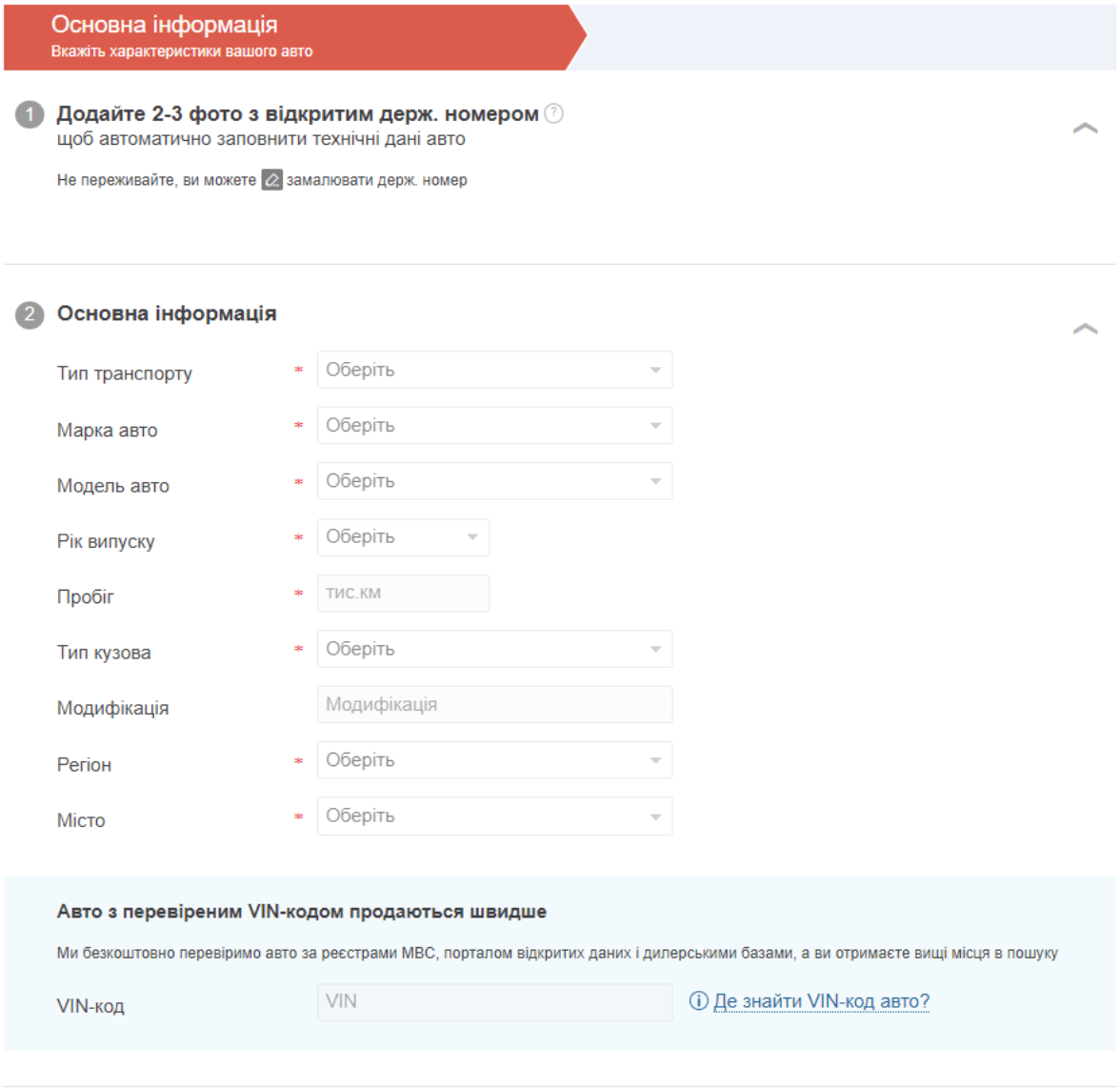


Рисунок 3 – Сторінка додавання оголошення на сайті «AUTO.RIA»

Загалом, вебзастосунок AUTO.RIA має логічну структуру та інтуїтивно зрозумілий інтерфейс, що дозволяє користувачам швидко та ефективно знайти потрібну інформацію про продаж автомобілів, однак на головній сторінці сайту присутня велика кількість рекламних блоків, які можуть заважати користувачам знайти потрібну інформацію. Також новим користувачам може бути складно зорієнтуватись на сайті через його перевантаженість великою кількістю розділів та функцій.

1.3 Застосунок «RST.ua»

На рис. 4 зображено головну сторінку застосунку RST.ua [4].

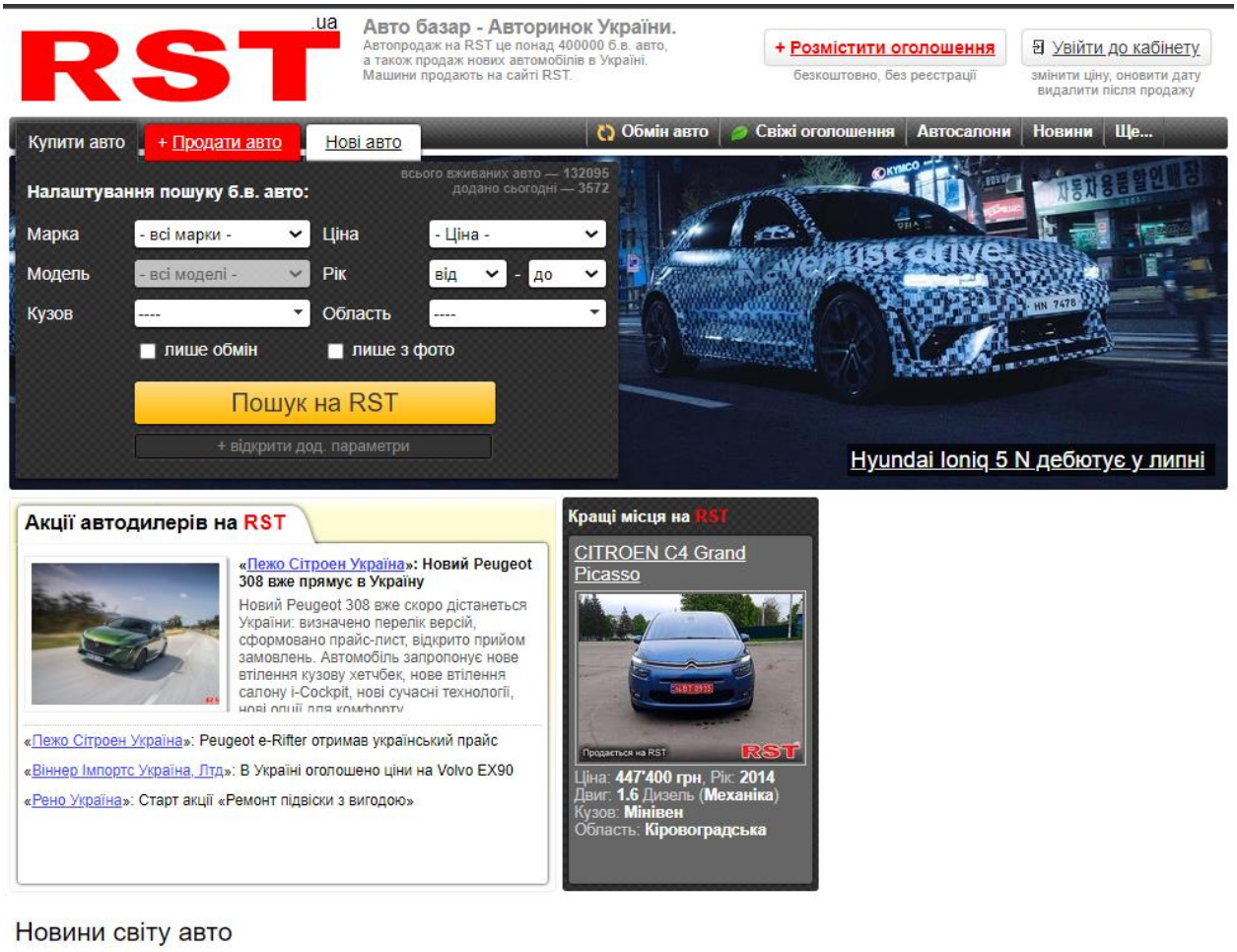


Рисунок 4 – Головна сторінка сайту «RST.ua»

Загалом, сайт має структуру, подібну до структури сайту AUTO.RIA. Тут так само наявні розділи про продаж автомобілів, пошук за фільтром, новини та авторизація. Проте відмінністю є наявність розділів «Обмін авто» та «Автосалони», на яких опубліковані оголошення про обмін автомобілями та оголошення автосалонів відповідно.

На рис. 5 зображено сторінку додавання нового оголошення. Тут наявні схожі поля, що й на сайті AUTO.RIA: кузов, марка, модель, рік випуску, вартість та інші.

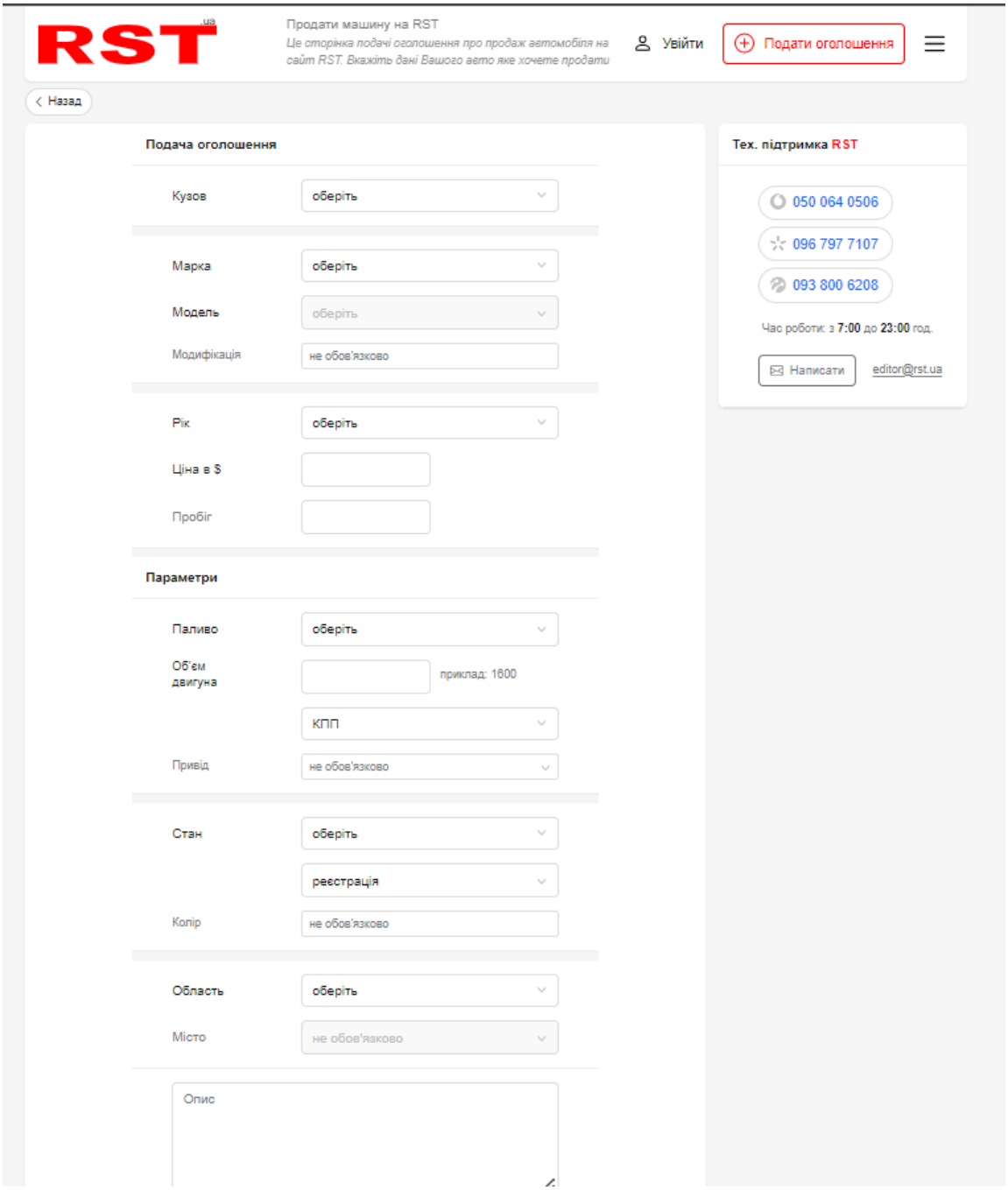


Рисунок 5 – Сторінка додавання оголошення на сайті «RST.ua»

Серед головних недоліків сайту можна виділити наступні:

1. Реклама – на головній сторінці RST.ua є досить багато рекламних банерів та посилань, які можуть відволікти користувачів від пошуку автомобілів.
2. Відсутність фотографій автомобілів – на деяких оголошеннях може бути відсутнє фото автомобіля.

1.4 Застосунок «avtobazar»

На рис. 5 зображено головну сторінку вебзастосунку «avtobazar» [5].

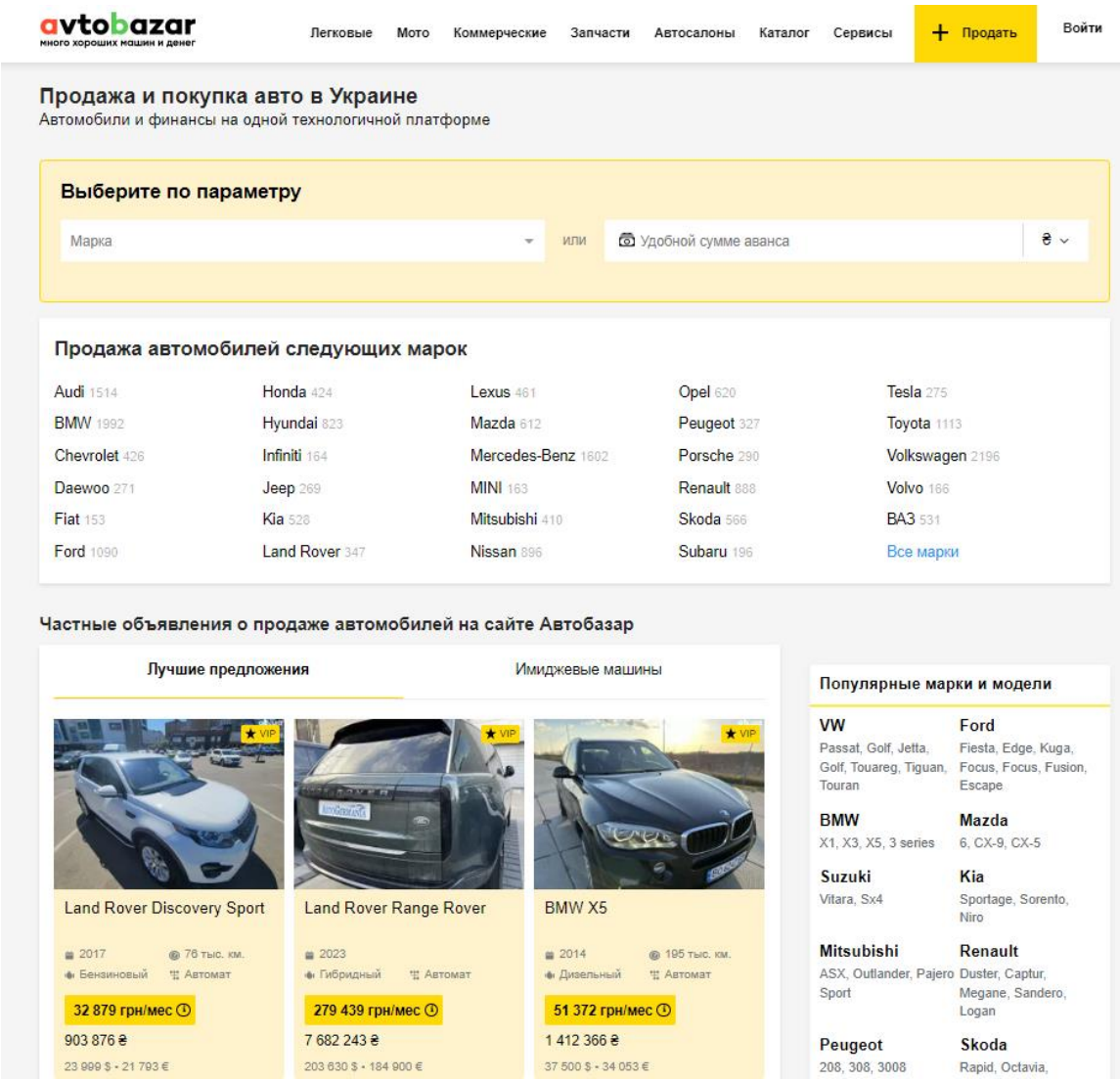


Рисунок 5 – Головна сторінка сайту «avtobazar»

Застосунок має більш сучасний та інтуїтивно зрозумілий інтерфейс у порівнянні з попередніми сайтами. Також наявний зручніший фільтр та можливість одразу подивитись оголошення про продаж автомобілів конкретної марки або кузову.

На рис. 6-7 зображено сторінку додавання нового оголошення про продаж автомобіля, сторінки змінюються поступово з вибором інформації про

автомобіль.

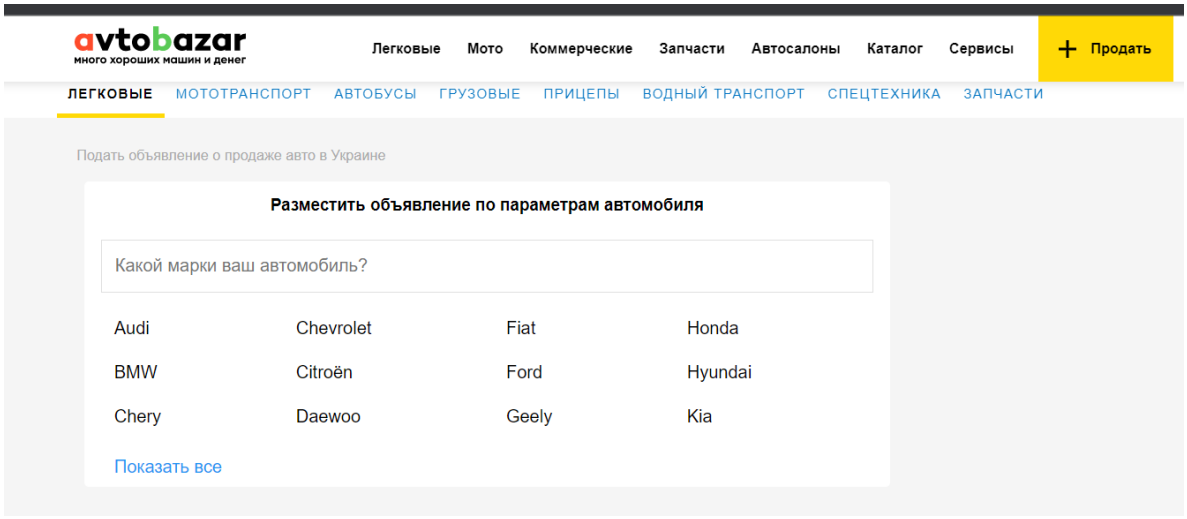


Рисунок 6 – Початкова сторінка додавання оголошення на сайті «avtobazar»

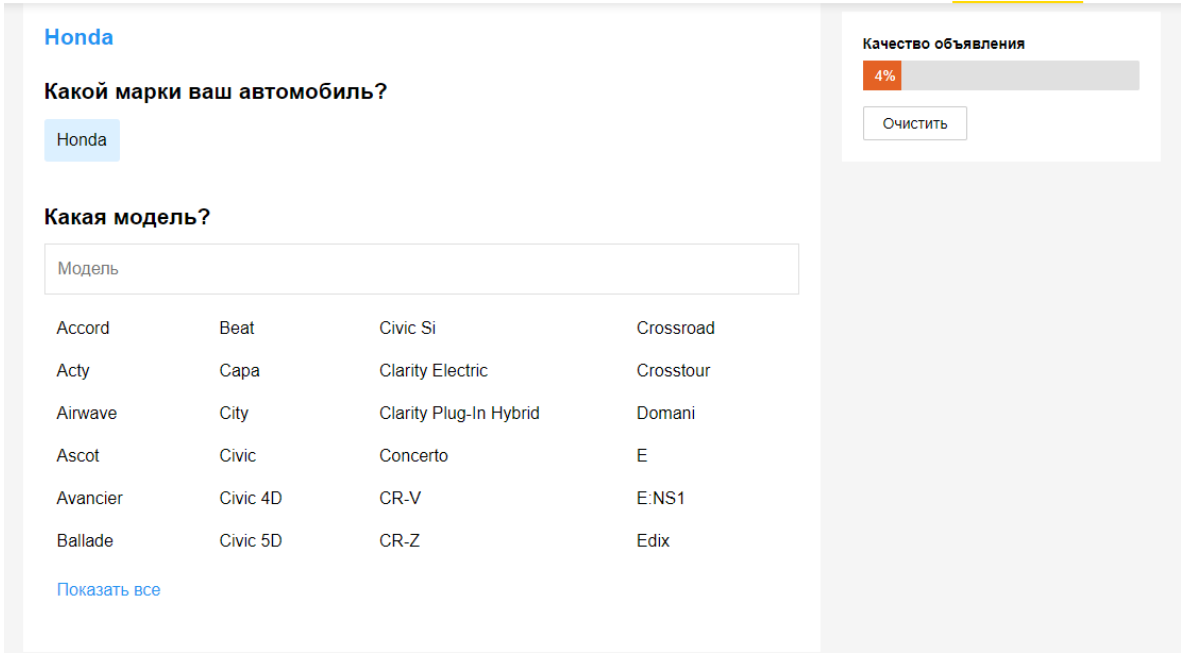


Рисунок 7 – Наступна сторінка, після вибору бренду

Основним недоліком даного застосунку є відсутність україномовної версії сайту. Серед переваг можна виділити зручність і сучасність дизайну, багатий функціонал та поступове заповнення інформації про автомобіль при додаванні нового оголошення.

Проаналізувавши наявні на ринку застосунки можна зробити висновок, що вебзастосунок для продажу автомобілів повинен бути легким в користуванні, містити широкий асортимент оголошень та мати функціональність, яка дозволить користувачам швидко та ефективно знайти потрібну інформацію. Користувачі повинні мати змогу легко знайти оголошення згідно зі своїми потребами та перевагами. Наприклад, можливість встановлення фільтрів за маркою, моделлю, роком випуску, кольором та іншими параметрами може значно полегшити пошук автомобілів для покупців.

РОЗДІЛ 2.

ОГЛЯД ВИКОРИСТАННИХ ТЕХНОЛОГІЙ

2.1 Мова реалізації C#

C# – це об'єктно-орієнтована мова програмування, яка була розроблена компанією Microsoft для розробки програмного забезпечення для платформи .NET [6-7].

Мову C# було обрано, враховуючі її наступні переваги:

1. C# є основною мовою програмування для платформи .NET.
2. Підтримує об'єктно-орієнтовану парадигму програмування, що дозволяє розробляти програмне забезпечення, яке легко розширювати та підтримувати.
3. Має простий та зрозумілий синтаксис, що дозволяє розробникам швидко розуміти код та легко виконувати різноманітні завдання.
4. Має вбудовану підтримку безпеки, що дозволяє розробляти безпечне програмне забезпечення, що містить обмеження на доступ до системних ресурсів.
5. C# підтримує розробку програм для різних платформ, включаючи веб, мобільні, настільні застосунки та інші.

2.2 Технологія ASP.NET Core

В якості платформи для розробки програмної частини веб-застосунку було обрано ASP.NET Core [8-10].

ASP.NET Core – це відкритий вебфреймворк для розробки високопродуктивних та масштабованих вебзастосунків. Основна мета ASP.NET Core полягає в створенні кросплатформного застосунку, що може працювати на будь-якій операційній системі. Основними перевагами використання ASP.NET Core є:

1. Фреймворк має високу продуктивність, так як використовує передові технології оптимізації та кешування, що дозволяє досягти високої продуктивності вебзастосунків навіть при великому навантаженні.
2. ASP.NET Core може працювати на будь-якій операційній системі, включаючи Windows, Linux та macOS та не прив'язаний до операційної системи Windows, як застарілий фреймворк ASP.NET.
3. Фреймворк підтримує багато мов програмування, зокрема C#, Visual Basic та F#.
4. ASP.NET Core підтримує вбудований принцип інверсії залежностей.
5. ASP.NET Core дозволяє використовувати лише ті компоненти, які потрібні для вебзастосунку, що робить його більш гнучким та розширюваним.

Завдяки цим перевагам ASP.NET Core є одним із найпопулярніших фреймворків серед розробників. Станом на 2022 рік, він займав 7 місце найкращих вебфреймворків [11] (див. рис 7).

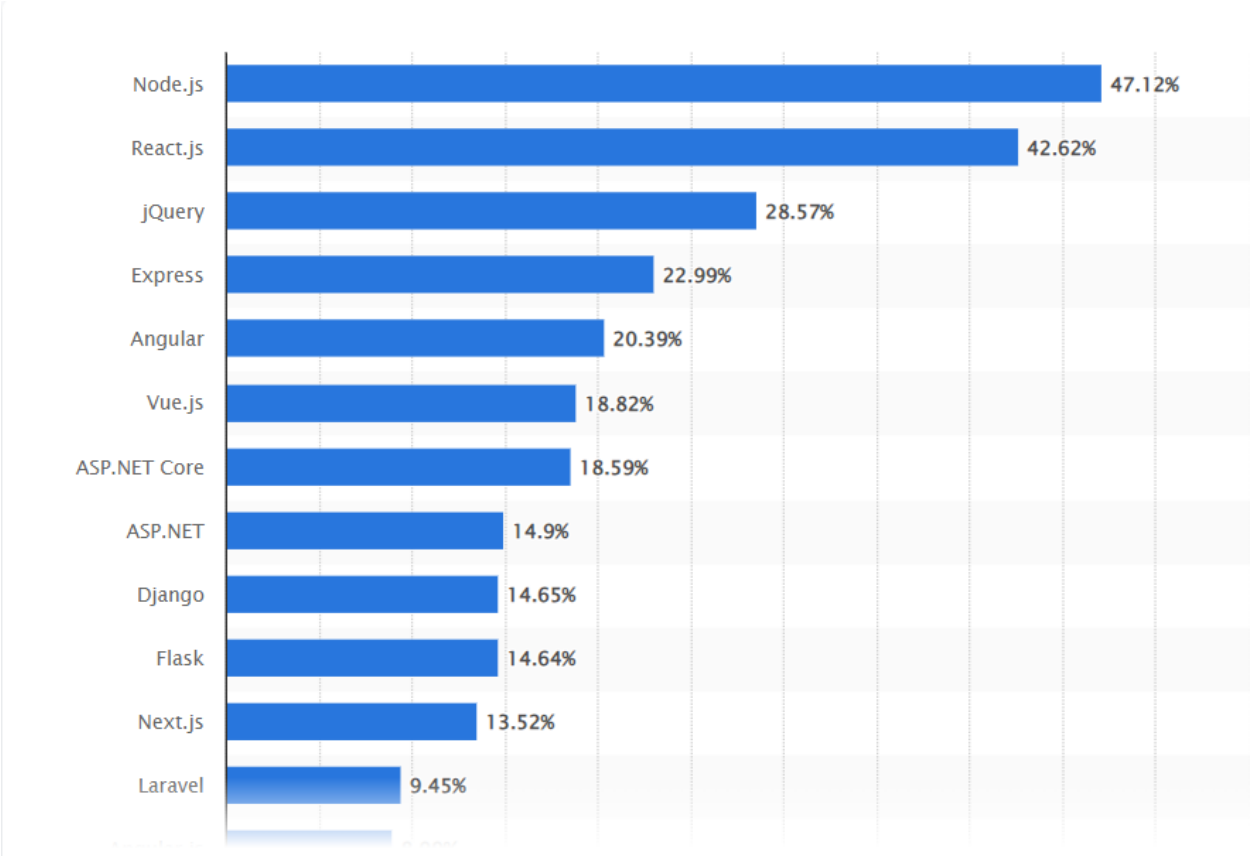


Рисунок 7 – Найбільш використовувані вебфреймворки станом на 2022 рік

В застосунку було використано Onion-архітектуру. Onion-архітектура є підходом до розробки вебзастосунків, що забезпечує гнучкість та легкість розширення. В основі цієї архітектури лежить принцип інверсії залежностей (Dependency Inversion Principle), який дозволяє розділити код застосунку на різні рівні та зробити його більш легким для тестування та модифікації.

Основна ідея Onion-архітектури полягає в тому, що застосунок повинен бути розділений на шари, які мають певний функціональний зміст та залежать один від одного відповідно до вимог бізнес-логіки (див. рис. 8) [12].

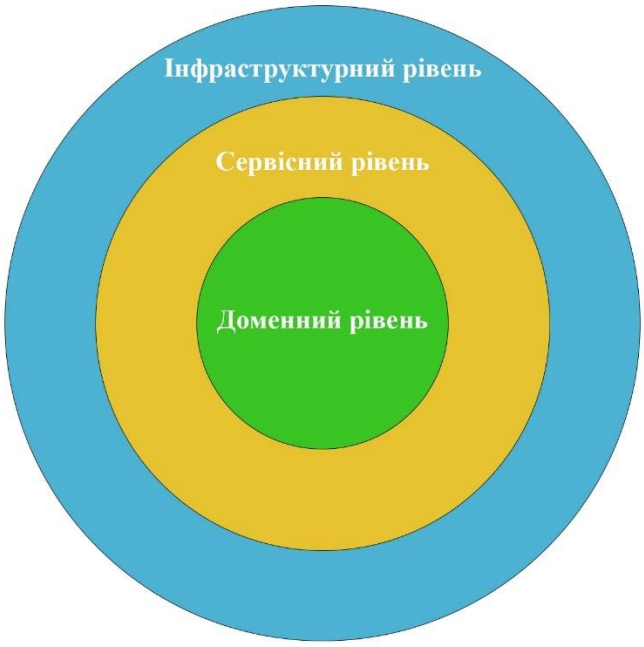


Рисунок 8 – Схема Onion-архітектури

У центрі архітектури лежить доменний рівень, що містить сутності застосунку, які використовуються для створення таблиць бази даних при підході розробки code first.

Сервісний рівень знаходиться безпосередньо над доменним рівнем та містить бізнес-логіку застосунку та інтерфейси для взаємодії з інфраструктурним рівнем.

Найвищим рівнем є інфраструктурний рівень або рівень представлення, який містить контролери та передає дані за допомогою HTTP-запитів.

Onion-архітектура в ASP.NET дозволяє зменшити залежність від фреймворку та забезпечує більшу гнучкість в розробці вебзастосунків.

2.3 Фреймворк Angular

Для розробки клієнтської частини вебзастосунку було використано фреймворк Angular [13-15].

Angular – це вебфреймворк, який написаний на мові програмування TypeScript та використовується для створення односторінкових (SPA - Single Page Applications) та багатосторінкових застосунків.

Так як Angular написаний на TypeScript, він реалізує основну та додаткову функціональність у вигляді набору бібліотек TypeScript.

Angular-застосунок є модульним та складається з набору модулів (NgModules), які в свою чергу складаються з компонентів (Components). Модулі дозволяють розділити додаток на логічні частини та забезпечити більш просту та зрозумілу структуру. У кожному додатку завжди є кореневий модуль, який забезпечує механізм завантаження, що запускає додаток.

Компоненти – це основні будівельні блоки Angular, які відповідають за рендеринг та управління відображенням даних у вебзастосунку. Angular створює, оновлює та знищує компоненти, коли користувач переміщується застосунком.

Іншою важливою складовою Angular-застосунку є сервіси (Services). Сервіси – це класи, які забезпечують різну функціональність у застосунку.

Наприклад, сервіси можуть надсилати HTTP запити до сервера, обробляти дані та відправляти їх на компоненти.

Основні переваги використання Angular:

1. Має велику кількість вбудованих функцій та може бути легко розширений за допомогою сторонніх бібліотек та модулів.
2. Angular є кросплатформним, він може бути використаний для розробки застосунків для різних платформ, як для веб, так і для мобільних та настільних застосунків.
3. Побудований на принципі компонентної архітектури, що дозволяє розбити застосунок на невеликі, самодостатні та повторно використовувані компоненти.
4. Angular забезпечує високу швидкість та ефективність завдяки вбудованій оптимізації та підтримці TypeScript.
5. Фреймворк використовує механізм інверсії контролю (IoC) та Dependency Injection (DI), що дозволяє забезпечити більш гнучку та просту архітектуру застосунку.

2.4 Фреймворк Bootstrap

Bootstrap – це один з найпопулярніших фреймворків для розробки вебзастосунків [16].

Фреймворк базується на HTML, CSS та JavaScript і надає готові компоненти, які можна використовувати для швидкої розробки сучасних вебзастосунків.

Bootstrap містить великий готовий набір компонентів, що дозволяє швидко створювати вебзастосунки без необхідності розробляти все з нуля.

Серед переваг використання фреймворку можна виділити наступні:

1. Містить багато тем та стилів, що дозволяє відокремити логіку та дизайн застосунку.
2. Bootstrap підтримує більшість сучасних браузерів та сумісний з більшістю фреймворків та бібліотек JavaScript.
3. Фреймворк забезпечує кросбраузерність, що дозволяє вебзастосунком працювати однаково на різних браузерах.
4. Bootstrap має простий та логічний API.
5. Має адаптивний дизайн, що дозволяє розробляти вебзастосунки, які коректно відображаються на різних пристроях.

РОЗДІЛ 3.

РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Реалізація бази даних

У якості серверу бази даних використовувався Microsoft SQL Server.

База даних AutoHubDB складається з 8 таблиць, короткий опис яких наведено у табл. 1. Логічну структуру таблиць зображено на діаграмі бази даних у додатку А.

Таблиця 1 – Опис таблиць бази даних

Назва	Опис
Adverts	Таблиця для збереження інформації про оголошення.
Bodies	Таблиця для збереження інформації про кузови.
Brands	Таблиця для збереження інформації про бренди.
Cities	Таблиця для збереження інформації про міста.
Colors	Таблиця для збереження інформації про кольори.
Models	Таблиця для збереження інформації про моделі.
Regions	Таблиця для збереження інформації про регіони.
Users	Таблиця для збереження інформації про користувачів.

Таблиця dbo.Adverts (див. табл 2) містить інформацію про оголошення, а саме ідентифікатор оголошення, пробіг авто, VIN-код, опис, ціну, шлях до зображення та ідентифікатори для моделі, міста, кольору та користувача.

Таблиця 2 – Опис таблиці dbo.Adverts

Атрибут	Тип	Опис
Id	int	Ідентифікатор оголошення
Mileage	float	Пробіг авто
VinCode	nvarchar(50)	VIN-код

Description	nvarchar(300)	Опис
Price	float	Ціна
ModelId	int	Ідентифікатор моделі
CityId	int	Ідентифікатор міста
ColorId	int	Ідентифікатор кольору
UserId	int	Ідентифікатор користувача – власника
ImagePath	nvarchar(MAX)	Шлях до файлу із зображенням автомобіля

Таблиця dbo.Bodies (див. табл. 3) містить інформацію про кузови, а саме ідентифікатор кузову та його назву.

Таблиця 3 – Опис таблиці dbo.Bodies

Атрибут	Тип	Опис
Id	int	Ідентифікатор кузову
Name	nvarchar(50)	Назва кузову

Таблиця dbo.Brands (див. табл. 4) містить інформацію про бренди, а саме ідентифікатор бренду та його назву.

Таблиця 4 – Опис таблиці dbo.Brands

Атрибут	Тип	Опис
Id	int	Ідентифікатор бренду
Name	nvarchar(50)	Назва бренду

Таблиця dbo.Cities (див. табл. 5) містить інформацію про міста, а саме ідентифікатор міста, його назву та ідентифікатор регіону, до якого відноситься це місто.

Таблиця 5 - Опис таблиці dbo.Cities

Атрибут	Тип	Опис
Id	int	Ідентифікатор міста
Name	nvarchar(50)	Назва міста
RegionId	int	Ідентифікатор регіону

Таблиця dbo.Colors (див. табл. 6) містить інформацію про кольори, а саме ідентифікатор кольору та його назву.

Таблиця 6 – Опис таблиці dbo.Colors

Атрибут	Тип	Опис
Id	int	Ідентифікатор кольору
Name	nvarchar(50)	Назва кольору

Таблиця dbo.Models (див. табл. 7) містить інформацію про моделі автомобілів, а саме ідентифікатор моделі, її назву, рік випуску, потужність двигуна, ідентифікатори бренду та кузову.

Таблиця 7 – Опис таблиці dbo.Models

Атрибут	Тип	Опис
Id	int	Ідентифікатор моделі
Name	nvarchar(50)	Назва моделі
Year	int	Рік випуску
EnginePower	Float	Потужність двигуна
BrandId	int	Ідентифікатор бренду
BodyId	int	Ідентифікатор кузову

Таблиця dbo.Regions (див. табл. 8) містить інформацію про регіони, а саме ідентифікатор регіону та його назву.

Таблиця 8 – Опис таблиці dbo.Regions

Атрибут	Тип	Опис
Id	int	Ідентифікатор регіону
Name	nvarchar(50)	Назва регіону

Таблиця dbo.Users (див. табл. 9) містить інформацію про користувачів застосунку, а саме ідентифікатор користувача, його ім’я, електрону пошту, номер телефону, хешований пароль, токен авторизації та роль.

Таблиця 9 – Опис таблиці dbo.Regions

Атрибут	Тип	Опис
Id	int	Ідентифікатор користувача
Name	nvarchar(50)	Ім’я користувача
Email	nvarchar(50)	Електрона пошта
Phone	nvarchar(50)	Номер телефону
Password	nvarchar(50)	Хешований пароль
Token	nvarchar(MAX)	Токен авторизації
Role	nvarchar(MAX)	Роль користувача

3.2 Реалізація програмного інтерфейсу

Застосунок містить 3 шари, згідно з Onion-архітектурою: AutoHub.Data, AutoHub.Business та AutoHub.API.

Перший рівень містить сутності, які використовуються для створення таблиць бази даних (див. табл. 10).

Таблиця 10 – Опис сутностей застосунку

Назва	Опис
Advert	Містить інформацію про оголошення
BaseEntity	Базова сутність, від якої наслідуються інші
Body	Містить інформацію про кузови

Brand	Модель для збереження інформації про бренди
Car	Модель для опису інформації про автомобілі
City	Зберігає інформацію про міста
Color	Модель для опису інформації про кольори
Region	Модель для збереження інформації про регіони
User	Модель для опису інформації про користувачів

Другий рівень містить усю бізнес-логіку застосунку, а саме інтерфейси та відповідні класи-сервіси, що використовуються для взаємодії між базою даних та інфраструктурним рівнем (див. табл. 11).

Таблиця 11 – Опис класів-сервісів застосунку

Назва	**Опис**
AdvertService	Клас для обробки інформації про оголошення
BodyService	Клас для обробки інформації про кузови
BrandService	Клас для обробки інформації про бренди
CarService	Клас для обробки інформації про автомобілі
CityService	Клас для обробки інформації про міста
ColorService	Клас для обробки інформації про кольори
RegionService	Клас для обробки інформації про регіони
UserService	Клас для обробки інформації про користувачів

Перелік та опис методів класу AdvertService наведено в таблиці 12 (додаток Б).

Таблиця 12 – Опис методів класу AdvertService

Назва	**Опис**
GetAdverts	Повертає список усіх оголошень
GetAdvertById	Повертає оголошення з відповідним Id

GetUserAdverts	Повертає список оголошень користувача
AddAdvert	Метод для додавання нового оголошення
UpdateAdvert	Метод для оновлення інформації про оголошення
DeleteAdvert	Метод для видалення оголошення

Третій рівень містить контролери та моделі, які використовуються для взаємодії з інтерфейсом користувача. Перелік та опис основних методів контролера BodiesController наведено в таблиці 13 (додаток В).

Таблиця 13 – Опис методів контролера BodiesController

Назва	**Опис**
Get	Повертає список усіх кузовів
Get (int id)	Повертає кузов з відповідним Id
Post	Додає новий кузов
Put	Оновлює кузов
Delete	Видаляє кузов

3.3 Реалізація інтерфейсу користувача

Застосунок для інтерфейсу користувача містить 3 основні частини: компоненти, моделі та сервіси.

Кожна група компонентів знаходиться в окремій папці. Перелік груп компонентів наведено в таблиці 14.

Таблиця 14 – Компоненти застосунку

Назва	Опис
adverts	Містить компоненти для взаємодії з оголошеннями
bodies	Містить компоненти для взаємодії з кузовами
brands	Містить компоненти для взаємодії з брендами
cars	Містить компоненти для взаємодії з автомобілями
cities	Містить компоненти для взаємодії з містами
colors	Містить компоненти для взаємодії з кольорами
login	Компоненти для авторизації
regions	Містить компоненти для взаємодії з регіонами
signup	Компоненти для реєстрації
user	Містить компоненти для взаємодії з користувачами

У кожній групі наявні компоненти для взаємодії з моделями. Компоненти групи bodies наведено в таблиці 15.

Таблиця 15 – Компоненти групи bodies

Назва	Опис
add-body	Компонент для додавання нового кузову
bodies-list	Компонент для відображення списку усіх кузовів
body-cars-list	Компонент для відображення списку автомобілів конкретного кузову
edit-body	Компонент для зміни інформації про кузов

Кожен компонент містить 3 основних файли:

- файл з кодом TypeScript;
- HTML-файл сторінки;
- CSS-файл зі стилями для сторінки.

Фрагменти коду для компонента add-body зображено на рис. 9-10.

```
export class AddBodyComponent implements OnInit {

  addBodyRequest: Body = {
    id: 0,
    name: ''
  };

  addBodyForm!: FormGroup;

  constructor(private bodiesService: BodiesService, private router: Router, private formBuilder: FormBuilder) {}

  ngOnInit(): void {
    this.addBodyForm = this.formBuilder.group({
      bodyName: ['', Validators.required]
    })
  }

  onSubmit() {
    if(this.addBodyForm.valid) {
      this.addBody();
    }
    else {
      FormValidator.validateFormFields(this.addBodyForm);
    }
  }

  addBody() {
    this.bodiesService.addBody(this.addBodyRequest)
    .subscribe({
      next: (body) => {
        this.router.navigate(['bodies']);
      }
    })
  }
}
```

Рисунок 9 – Фрагмент коду файлу TypeScript компонента add-body

```
<div class="container my-5">
    <h2 class="mb-3">Додати новий кузов</h2>

    <div class="row">
        <div class="col-6">
            <form [formGroup]="addBodyForm" (ngSubmit)="onSubmit()">
                <div class="mb-2">
                    <input formControlName="bodyName" type="text" class="form-control" id="name" name="name"
                    placeholder="Назва" [(ngModel)] = "addBodyRequest.name">
                </div>
                <small *ngIf="addBodyForm.controls['bodyName'].dirty && addBodyForm.hasError('required', 'bodyName')"
                class="text-danger">Це поле обов'язкове! <br/> </small>
                <button type="submit" class="btn btn-outline-primary mt-2" style="margin-right: 5pt; width: 75pt;">Додати</button>
                <a class="btn btn-outline-primary mt-2" routerLink="/bodies" style="width: 75pt;">Назад</a>
            </form>
        </div>
    </div>
</div>
```

Рисунок 10 – Фрагмент коду HTML-файлу компонента add-body

У таблиці 16 наведено перелік та опис моделей застосунку.

Таблиця 16 – Опис моделей застосунку

Назва	Опис
advert	Модель для збереження інформації про оголошення
body	Модель для збереження інформації про кузови
brand	Модель для збереження інформації про бренди
car	Модель для збереження інформації про автомобілі
city	Модель для збереження інформації про міста
color	Модель для збереження інформації про кольори
region	Модель для збереження інформації про регіони
user	Модель для збереження інформації про користувачів

Сервіси використовуються для взаємодії з програмною частиною застосунку через надсилання HTTP-запитів. Перелік та опис сервісів наведено в таблиці 17.

Таблиця 17 – Опис сервісів застосунку

Назва	Опис
adverts.service	Сервіс для роботи з оголошеннями
auth.service	Сервіс для автентифікації користувачів
bodies.service	Сервіс для роботи з кузовами
brands.service	Сервіс для роботи з брендами
cars.service	Сервіс для роботи з автомобілями
cities.service	Сервіс для роботи з містами
colors.service	Сервіс для роботи з кольорами
regions.service	Сервіс для роботи з регіонами
user-store.service	Сервіс для роботи з користувачами

Перелік методів сервісу adverts.service наведено в таблиці 18, фрагмент коду наведено в додатку Г.

Таблиця 18 – Методи сервісу adverts.service

Назва	Опис
getAllAdverts	Надсилає запит на отримання списку всіх оголошень
addAdvert	Надсилає запит на додавання нового оголошення
getAdvert	Надсилає запит на отримання оголошення з відповідним Id
getUSerAdvert	Надсилає запит на отримання списку оголошень користувача
updateAdvert	Надсилає запит на оновлення інформації про оголошення
deleteAdvert	Надсилає запит на видалення оголошення
uploadFile	Надсилає запит на завантаження файлу

3.4 Тестування

Для програмної частини застосунку було проведено тестування контролерів за допомогою фреймворку xUnit. Unit-тести було написано згідно принципу AAA (Arrange, Act, Assert).

У блоці Arrange оголошуються на налаштовуються всі необхідні для тесту параметри. У блоці Act виконується метод, для якого проводиться тестування. Assert – це заключна частина тесту, де порівнюються очікувані результати з фактичними результатами виконання тестового методу.

У табл. 19 наведено методи для тестування контролера BrandsController, у додатку Д наведено фрагмент коду класу для тестування.

Таблиця 19 – Методи для тестування контролера BrandsController

Назва	**Опис**
GetAllBrandsTest	Метод для тестування Get методу
GetBrandByIdTest	Метод для тестування Get(int id) методу
AddBrandTest	Метод для тестування Post методу
DeleteBrandTest	Метод для тестування Delete методу
UpdateBrandTest	Метод для тестування Put методу

На рис. 11 зображено результат виконання тестів.

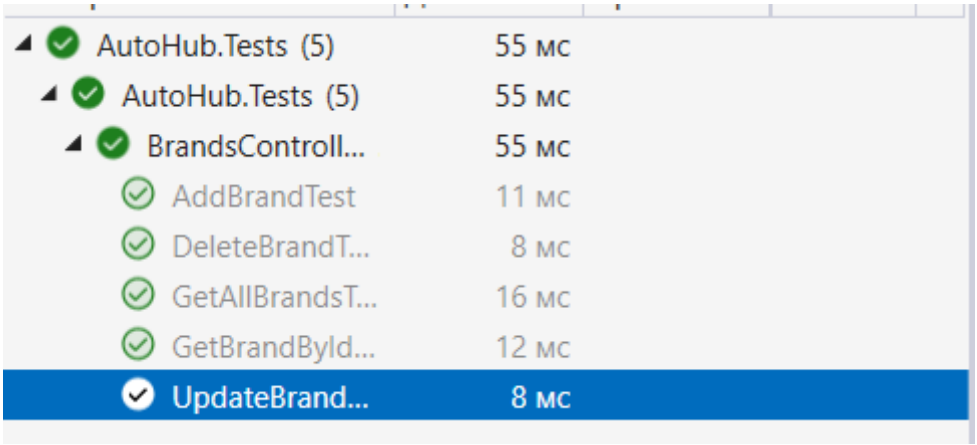


Рисунок 11 – Результат виконання тестів

РОЗДІЛ 4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Інструкція звичайного користувача

Коли користувач заходить на сайт, він потрапляє на сторінку авторизації (див. рис. 12), де також є можливість зареєструватись (див. рис. 13). Use-case діаграму для користувача наведено в додатку Е.

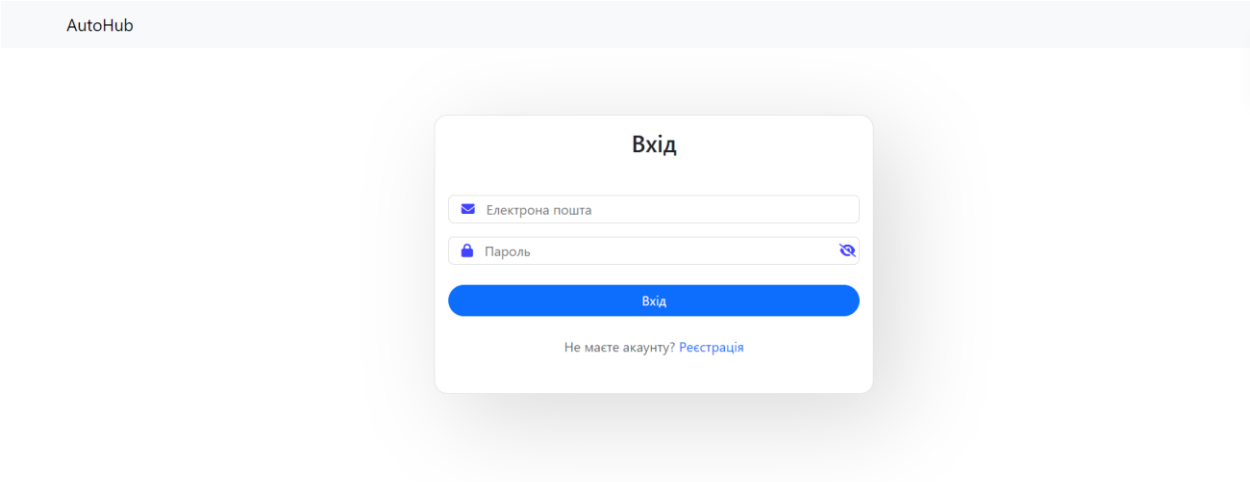


Рисунок 12 – Сторінка авторизації на сайті

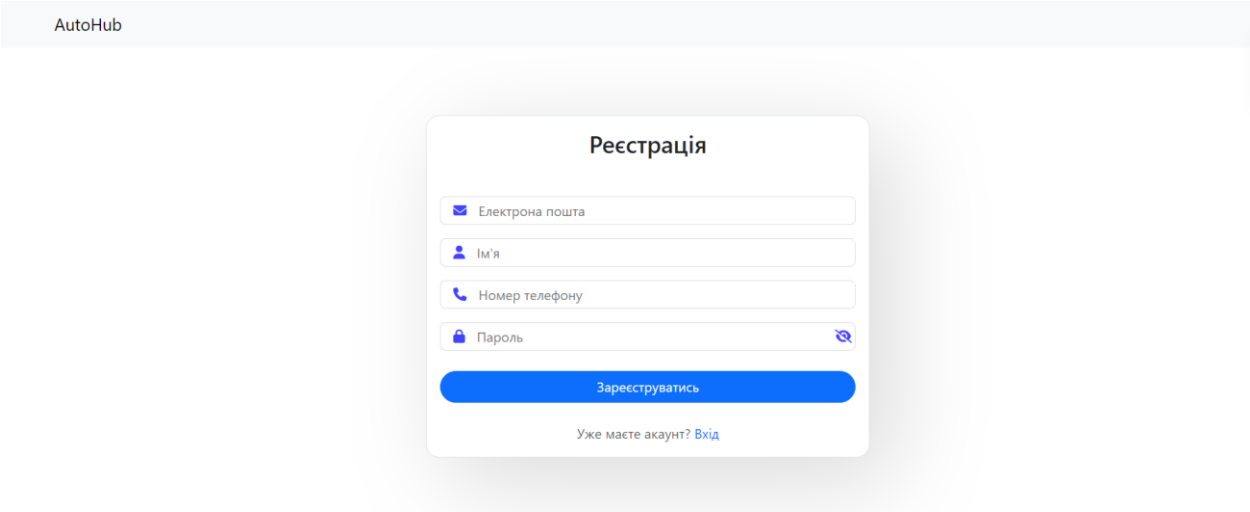


Рисунок 13 – Сторінка реєстрації на сайті

Якщо при авторизації користувача з такою адресою електронної пошти не існує, або введено неправильний пароль, система повідомить про це (див. рис. 14-15).

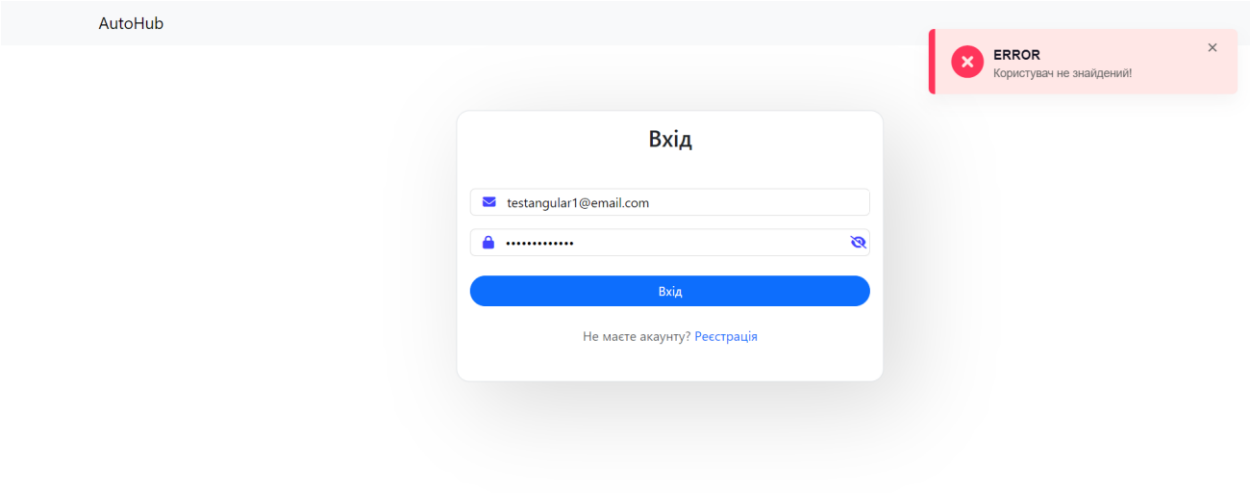


Рисунок 14 – Повідомлення помилки при введені незареєстрованої електронної пошти

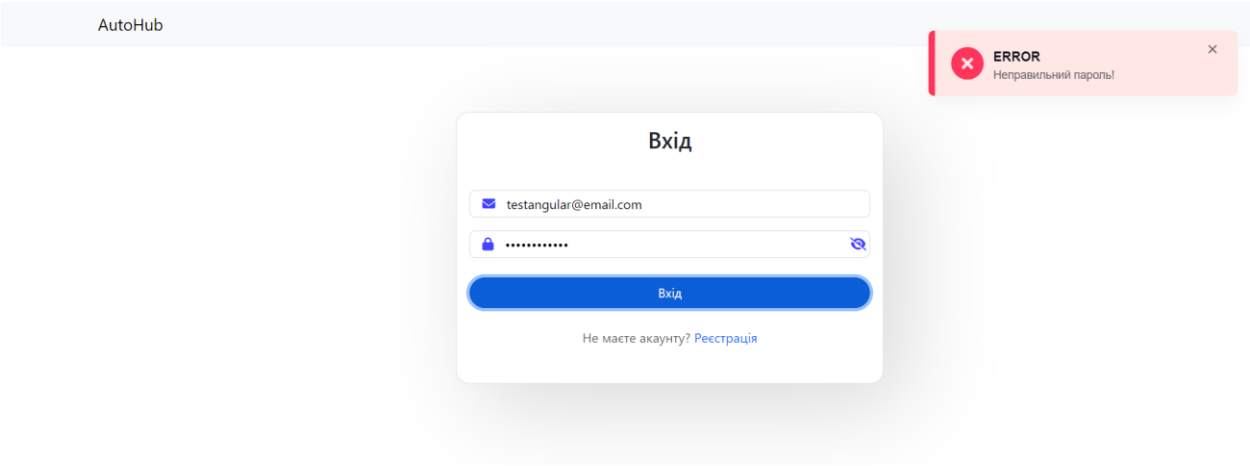


Рисунок 15 – Повідомлення при введені неправильного паролю

При реєстрації, якщо введено вже зареєстровану електронну пошту або неправильний формат паролю, система повідомить про це (див. рис. 16-17). Пароль повинен містити мінімум 8 символів, принаймні 1 маленьку, велику літеру, цифру та символ.

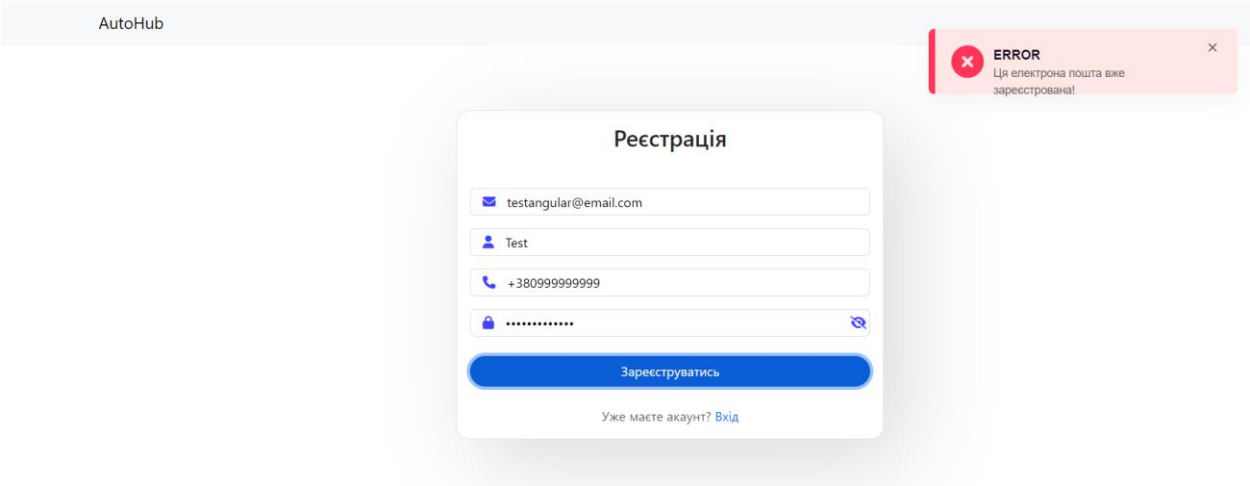


Рисунок 16 – Повідомлення помилки при спробі реєстрації на вже зареєстровану електронну пошту

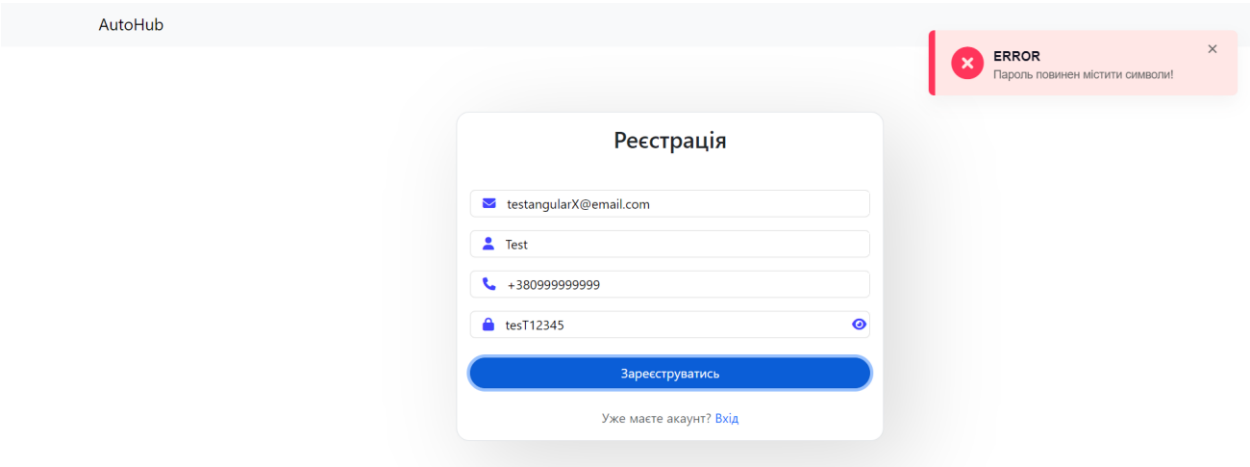


Рисунок 17 – Повідомлення помилки при неправильно введеному паролі

Після успішної авторизації користувач потрапляє на головну сторінку сайту (див. рис. 18).

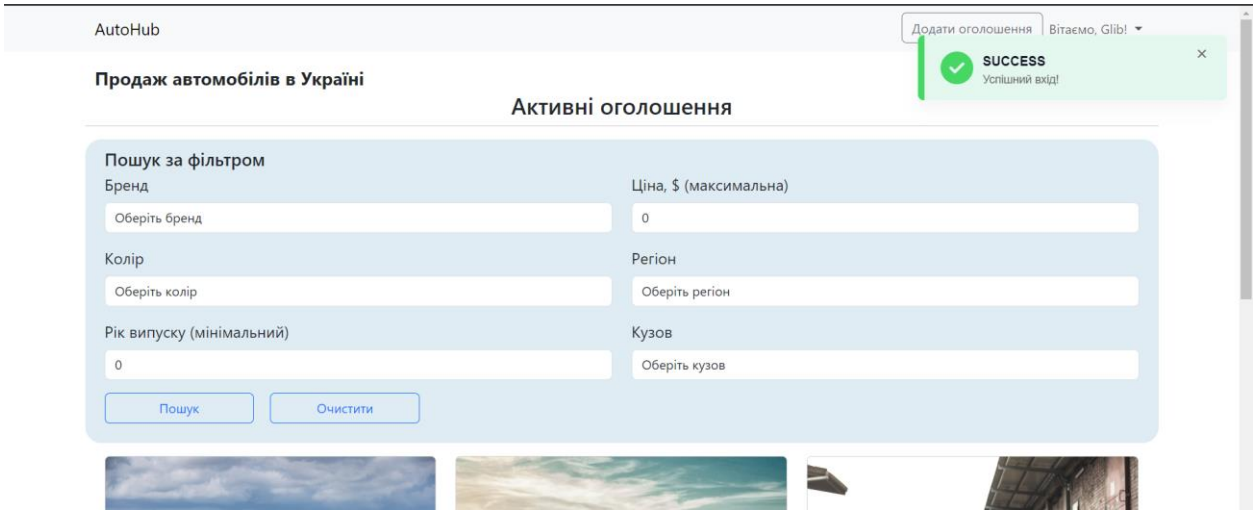


Рисунок 18 – Головна сторінка сайту

Тут він може переглянути усі активні оголошення, або застосувати фільтр за параметрами, щоб переглянути лише необхідні йому оголошення (див. рис. 19).

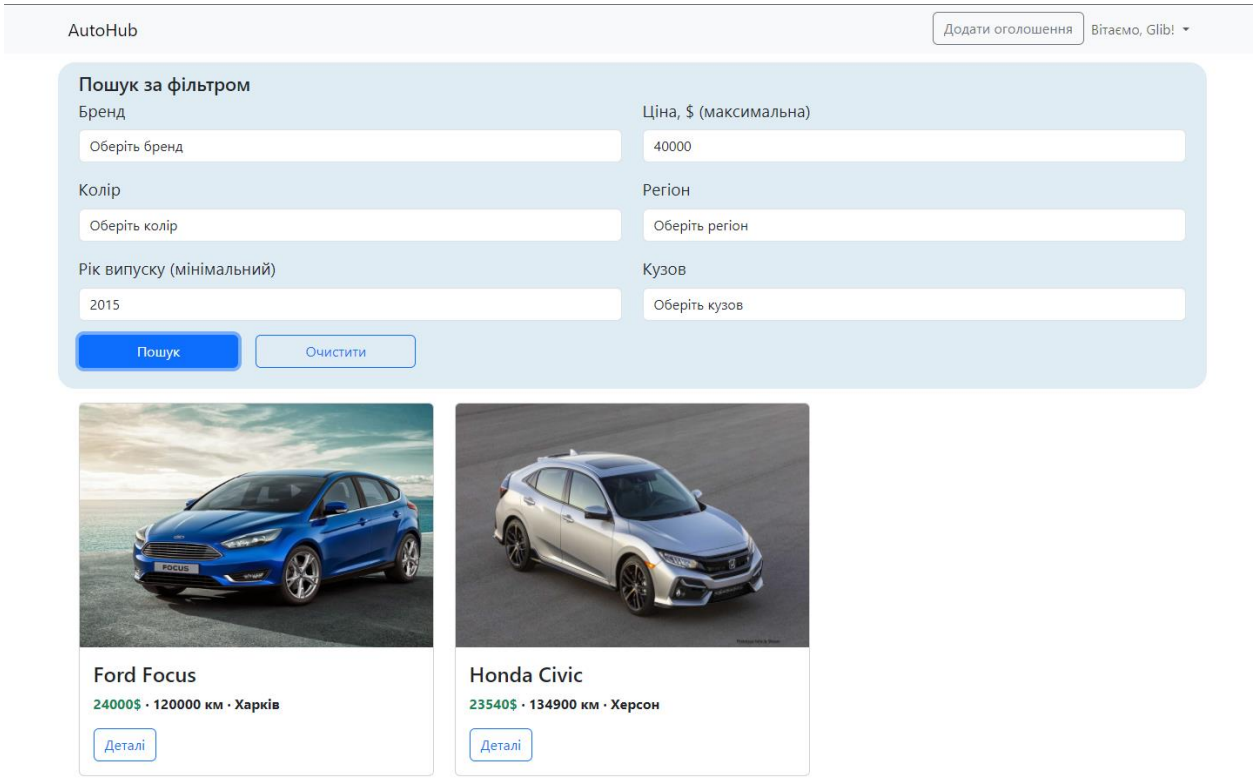


Рисунок 19 – Оголошення із застосуванням фільтру мінімальний рік випуску 2015 та максимальна ціна 40000$

На сторінці оголошення наявна детальна інформація про автомобіль (див. рис. 20).

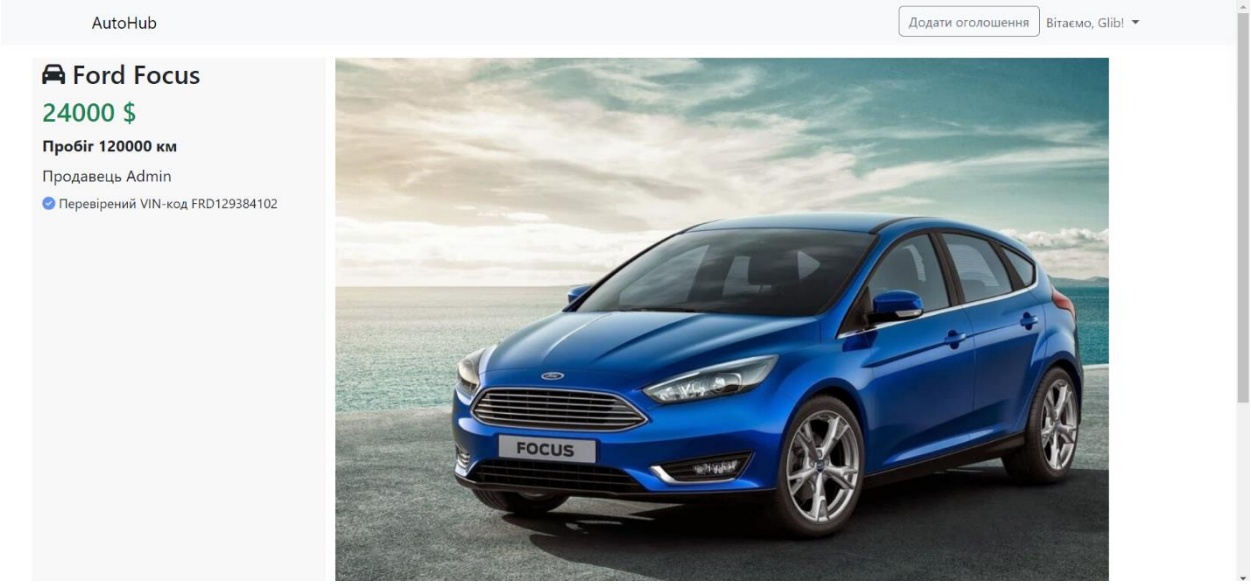


Рисунок 20 – Сторінка оголошення з детальною інформацією

Користувач може додати нове оголошення, натиснувши на кнопку «Додати оголошення». Потрібно прикріпити фото та ввести інформацію про автомобіль: бренд, модель, пробіг, VIN-код, ціну, колір, регіон, місто та опис (див. рис. 21).

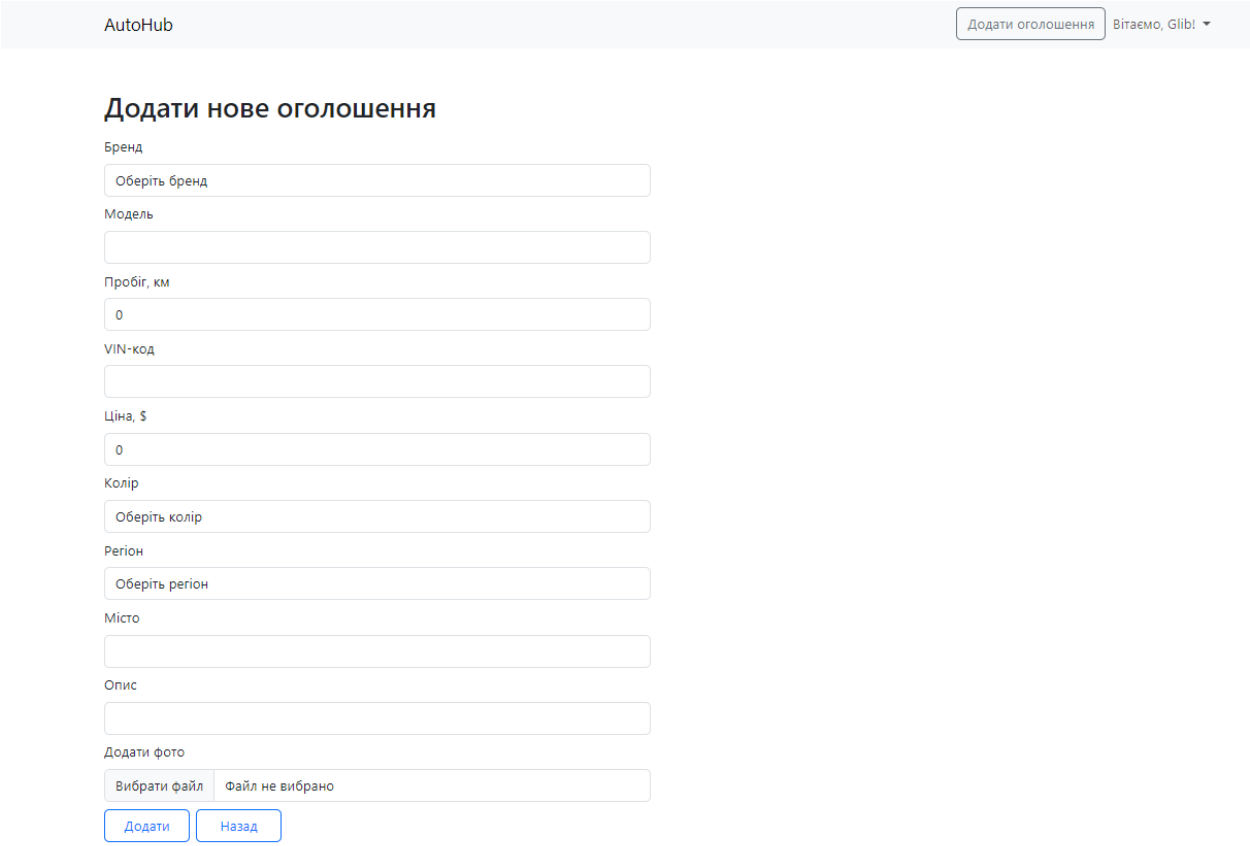


Рисунок 21 – Сторінка додавання нового оголошення

В особистому кабінеті користувач може побачити інформацію про себе та його активні оголошення (див. рис. 22), а також відредагувати їх (див. рис. 23).

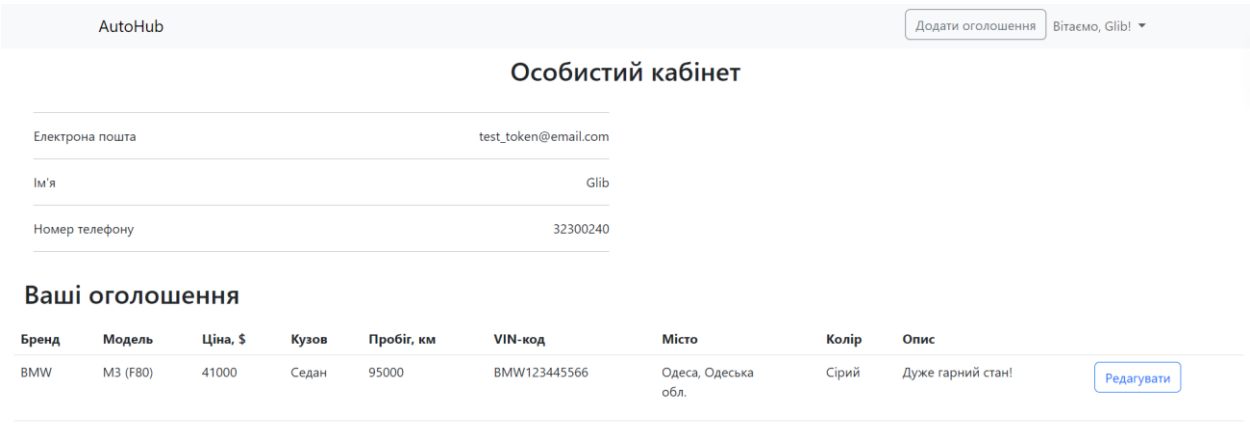


Рисунок 22 – Особистий кабінет користувача

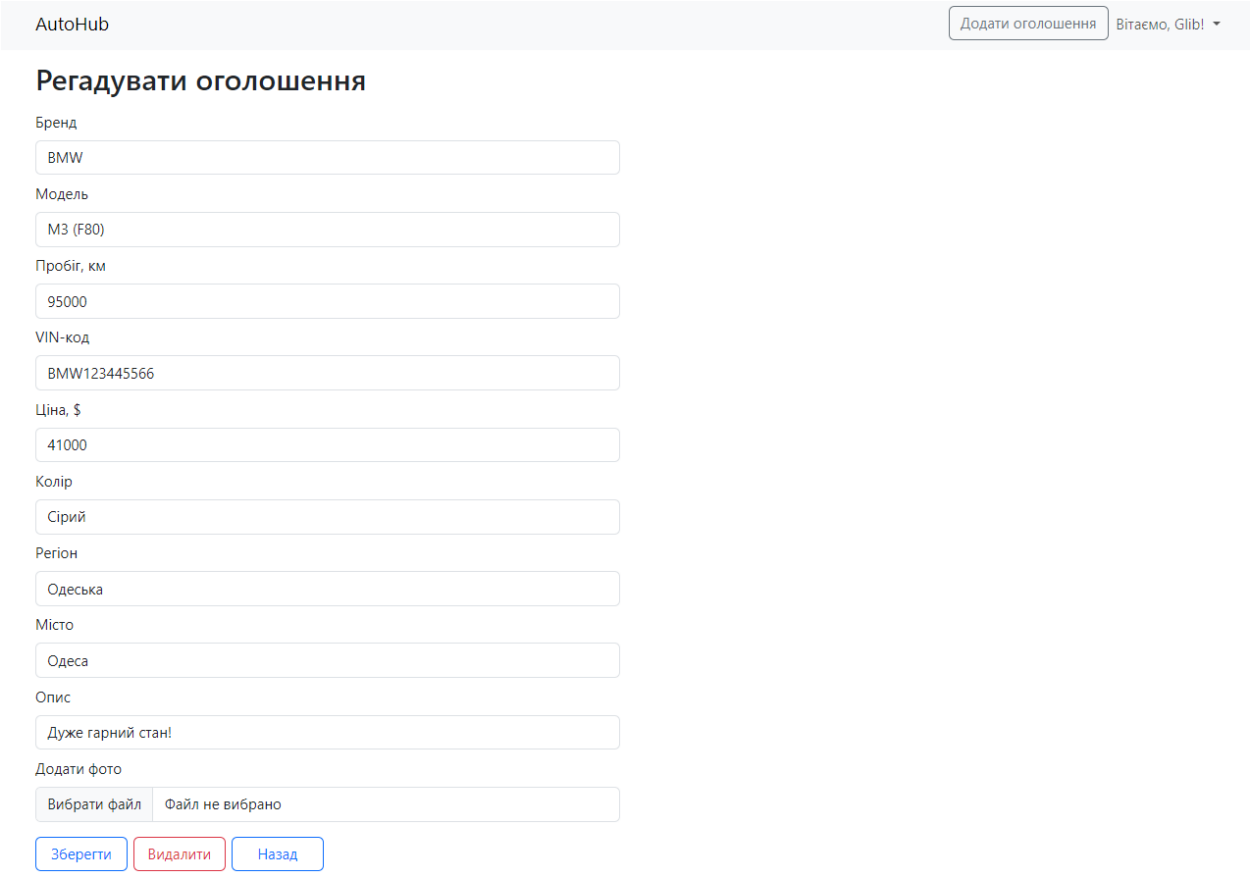


Рисунок 23 – Сторінка редагування оголошення

4.2 Інструкція адміністратора

При авторизації в ролі адміністратора відкривається головна сторінка сайту (див. рис. 18). Адміністратор має доступ до керування усіма даними застосунку, в особистому кабінеті наявні кнопки для перегляду, додавання, видалення та редагування усіх даних: брендів, автомобілів, кузовів, регіонів, міст та кольорів (див. рис. 24-25). Use-case діаграму для адміністратора наведено в додатку Ж.

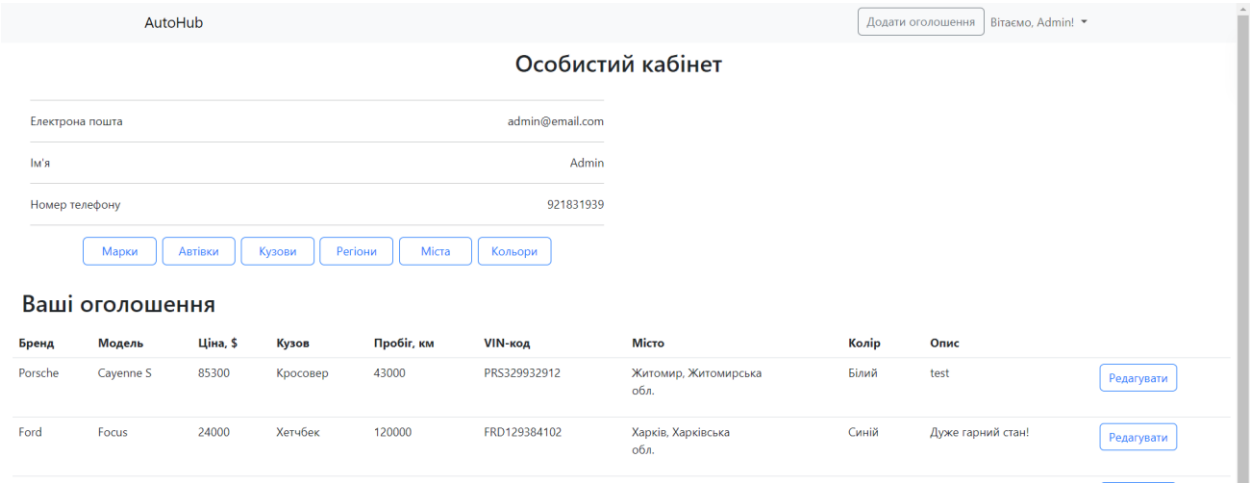


Рисунок 24 – Особистий кабінет адміністратора

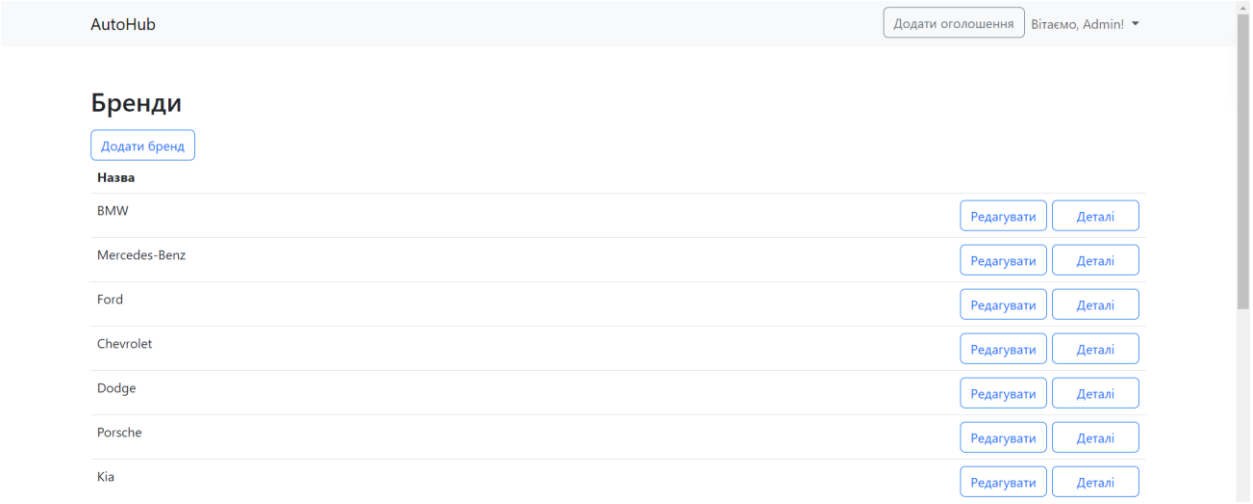


Рисунок 25 – Сторінка з інформацією про бренди

Адміністратор може переглянути усіх зареєстрованих користувачів сайту (див. рис. 26).

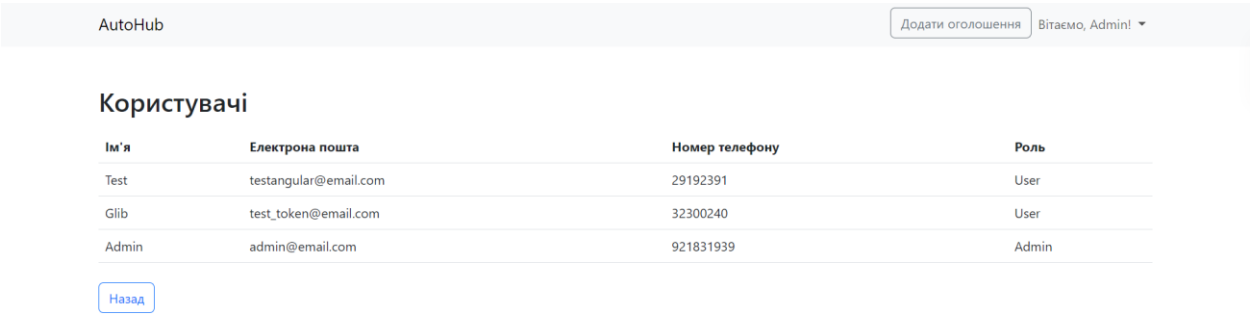


Рисунок 26 – Сторінка з інформацією про користувачів сайту

ВИСНОВКИ

У ході виконання кваліфікаційної роботи розроблено вебзастосунок, основною метою якого є розміщення та перегляд оголошень про продаж автомобілів.

На початковому етапі було розглянуто та проаналізовано наявні на ринку застосунки, визначено переваги та недоліки кожного з них, після чого було визначено основні вимоги до застосунку, яких слід дотримуватись при розробці.

Враховуючи попередній аналіз предметної області, на наступному етапі було розроблено базу даних, серверну та клієнтську частини застосунку.

Під час виконання кваліфікаційної роботи були отримані знання про Onion-архітектуру застосунку та навички роботи з технологією Angular. Були доповнені та закріплені знання у сфері розробки вебзастосунків за допомогою технологій ASP.NET Core, Bootstrap та розробки бази даних.

Результатом роботи є робочий застосунок для розміщення та перегляду оголошень про продаж автомобілів, що повністю відповідає поставленим вимогам. У якості подальшого розвитку проєкту можна визначити наступні кроки: удосконалення клієнтського інтерфейсу застосунку, реалізація додавання декількох фотографій автомобіля, реалізація списку «Обране» в особистому кабінеті користувача, реалізація алгоритму покупки автомобіля.

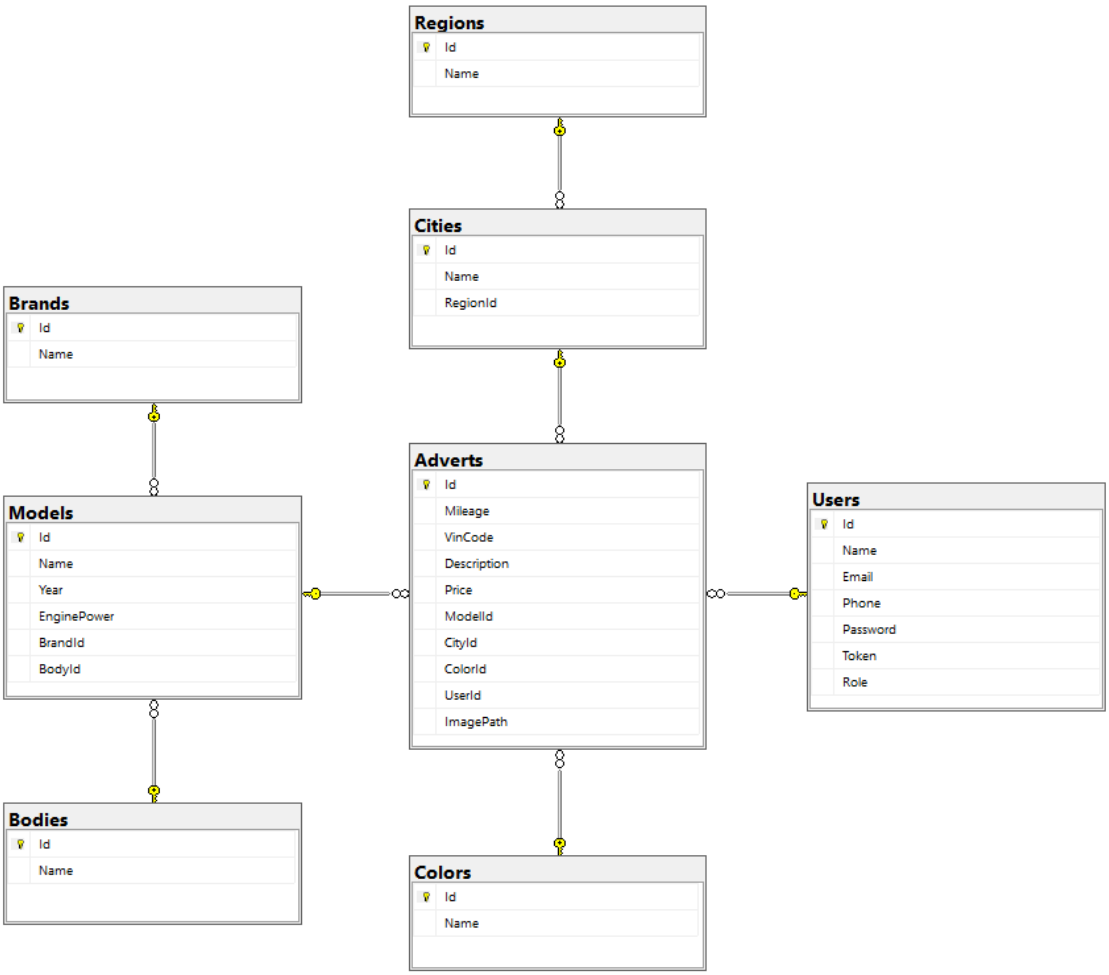
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Кількість автомобілів у світі [Електронний ресурс] – Режим доступу до ресурсу: https://www.pd.com.au/blogs/how-many-cars-in-the-world/
2. Кількість автомобілів в Україні [Електронний ресурс] – Режим доступу до ресурсу: https://auto.24tv.ua/skilky_naspravdi_mashyn_v_ukraini_bahato_chy_malo_n43694
3. Auto.RIA [Електронний ресурс] – Режим доступу до ресурсу: https://auto.ria.com/uk/
4. RST.ua [Електронний ресурс] – Режим доступу до ресурсу: https://rst.ua/ukr/
5. avtobazar [Електронний ресурс] – Режим доступу до ресурсу: https://ab.ua/
6. C# Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/
7. Albahari J. C# 9.0 in a Nutshell: The Definitive Reference, 1st Edition / Joseph Albahari., 2021. – 1058 с.
8. Lock A. ASP.NET Core in Action, Second Edition / Andrew Lock., 2021. – 832 с.
9. Peres R. Modern Web Development with ASP.NET Core 3, 2nd Edition / Ricardo Peres., 2020. – 802 с.
10. Freeman A. Pro ASP.NET Core 3 / Adam Freeman., 2020. – 1109 с.
11. Most used web frameworks among developers worldwide [Електронний ресурс] – Режим доступу до ресурсу: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/

12. Onion Architecture in ASP.NET Core 6 Web API [Електронний ресурс] – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/onion-architecture-in-asp-net-core-6-web-api/
13. Angular Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs
14. Fain Y. Angular Development with TypeScript / Yakov Fain, Anton Moiseev., 2018. – 560 с.
15. Hussain A. Angular: From Theory to Practice: Build the web applications of tomorrow using the Angular web framework from Google / Asim Hussain., 2017. – 703 с.
16. Bootstrap Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/docs/5.3/getting-started/introduction/

ДОДАТКИ

Додаток А. Діаграма бази даних



Додаток Б. Фрагмент коду класу AdvertService

```
public class AdvertService : IAdvertService
   {
      private readonly AutoHubContext _context;

      public AdvertService(AutoHubContext context)
      {
         _context = context;
      }

      public IEnumerable<Advert> GetAdverts()
      {
         return _context.Adverts.Include(a => a.Model).Include(a =>
a.Model.Brand).Include(a => a.Model.Body).Include(a => a.Color)
            .Include(a => a.City).Include(a => a.City.Region).Include(a =>
a.User).AsNoTracking();
      }

      public async Task<Advert> GetAdvertById(int advertId)
      {
         var advert = await _context.Adverts.Where(a => a.Id == advertId).Include(a =>
a.Model).Include(a => a.Model.Brand).Include(a => a.Model.Body)
            .Include(a => a.Color).Include(a => a.City).Include(a =>
a.City.Region).Include(a => a.User).FirstOrDefaultAsync();

         if (advert is null)
         {
            throw new ArgumentNullException("Advert does not exist");
         }

         return advert;
      }

      public IEnumerable<Advert> GetUserAdverts(int userId)
      {
         var user = _context.Users.Where(u => u.Id == userId).FirstOrDefault();

         if (user is null)
         {
            throw new ArgumentNullException("Brand does not exist");
         }

         return _context.Adverts.Where(a => a.UserId == user.Id).Include(a =>
a.Model).Include(a => a.Model.Brand).Include(a => a.Model.Body)
            .Include(a => a.Color).Include(a => a.City).Include(a => a.City.Region);
      }

      public async Task<Advert> AddAdvert(Advert advert)
      {
         _context.Adverts.Add(advert);
         await _context.SaveChangesAsync();

         return advert;
```

```
    }

    public async Task<Advert> UpdateAdvert(Advert newAdvert, int advertId)
    {
        Advert advert = await GetAdvertById(advertId);

        advert.Description = newAdvert.Description;
        advert.Price = newAdvert.Price;
        advert.Mileage = newAdvert.Mileage;
        advert.VinCode = newAdvert.VinCode;
        advert.CityId = newAdvert.CityId;
        advert.ColorId = newAdvert.ColorId;
        advert.ModelId = newAdvert.ModelId;
        advert.UserId = newAdvert.UserId;
        advert.ImagePath = newAdvert.ImagePath;

        _context.Adverts.Update(advert);
        await _context.SaveChangesAsync();

        return advert;
    }

    public async Task DeleteAdvert(int advertId)
    {
        var advert = await GetAdvertById(advertId);

        _context.Adverts.Remove(advert);
        await _context.SaveChangesAsync();
    }
}
```

Додаток В. Фрагмент коду контролера BodiesController

```
[HttpGet]
        [Authorize]
        public ActionResult<BodyModel> Get()
        {
            try
            {
                var entities = _bodyService.GetBodies();
                var bodies = entities.Select(e => new BodyModel
                {
                    Id = e.Id,
                    Name = e.Name,
                });

                return Ok(bodies);
            }
            catch (ArgumentNullException ex)
            {
                return NotFound(ex.Message);
            }
        }

        [HttpGet("{id}")]
        [Authorize]
        public async Task<ActionResult<BodyModel>> Get(int id)
        {
            try
            {
                var entity = await _bodyService.GetBodyById(id);
                var body = new BodyModel
                {
                    Id = entity.Id,
                    Name = entity.Name,
                };

                return Ok(body);
            }
            catch (ArgumentNullException ex)
            {
                return NotFound(ex.Message);
            }
        }

        [HttpPost]
        [Authorize(Roles = "Admin")]
        public async Task<ActionResult<BodyModel>> Post([FromBody] BodyModel body)
        {
            try
            {
                var entity = new Body()
                {
                    Name = body.Name,
                };
```

```
            var addedEntity = await _bodyService.AddBody(entity);
            var addedBody = new BodyModel()
            {
                Id = addedEntity.Id,
                Name = addedEntity.Name,
            };

            return Ok(addedBody);
        }
        catch (ArgumentNullException ex)
        {
            return NotFound(ex.Message);
        }
    }

    [HttpPut("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult<BodyModel>> Put([FromBody] BodyModel body,
[FromRoute] int id)
    {
        try
        {
            var entity = new Body()
            {
                Name = body.Name,
            };
            var newEntity = await _bodyService.UpdateBody(entity, id);
            var newBody = new BodyModel()
            {
                Name = newEntity.Name,
            };

            return Ok(newBody);
        }
        catch (ArgumentNullException ex)
        {
            return NotFound(ex.Message);
        }
    }

    [HttpDelete("{id}")]
    [Authorize(Roles = "Admin")]
    public async Task<ActionResult> Delete(int id)
    {
        try
        {
            await _bodyService.DeleteBody(id);
            return Ok();
        }
        catch (ArgumentNullException ex)
        {
            return NotFound(ex.Message);
        }
    }
```

Додаток Г. Фрагмент коду сервісу adverts.service

```
baseApiUrl: string = "https://localhost:7077/api/Adverts";

  constructor(private http: HttpClient) {}

  getAllAdverts(): Observable<Advert[]> {
    return this.http.get<Advert[]>(this.baseApiUrl);
  }

  addAdvert(addAdvertRequest: Advert): Observable<Advert> {
    return this.http.post<Advert>(this.baseApiUrl, addAdvertRequest);
  }

  getAdvert(id: number): Observable<Advert> {
    return this.http.get<Advert>(this.baseApiUrl + '/' + id);
  }

  getUserAdvert(id: number): Observable<Advert[]> {
    return this.http.get<Advert[]>(this.baseApiUrl + '/user/' + id);
  }

  updateAdvert(updateAdvertRequest: Advert, id: number): Observable<Advert> {
    return this.http.put<Advert>(this.baseApiUrl + '/' + id, updateAdvertRequest);
  }

  deleteAdvert(id: number): Observable<Advert> {
    return this.http.delete<Advert>(this.baseApiUrl + '/' + id);
  }

  uploadFile(fileData: FormData): Observable<any> {
    return this.http.post(this.baseApiUrl + "/upload", fileData);
  }

  applyFilter(filterModel: Advert): Observable<Advert[]> {
    return this.http.post<Advert[]>(this.baseApiUrl + '/filter', filterModel);
  }
```

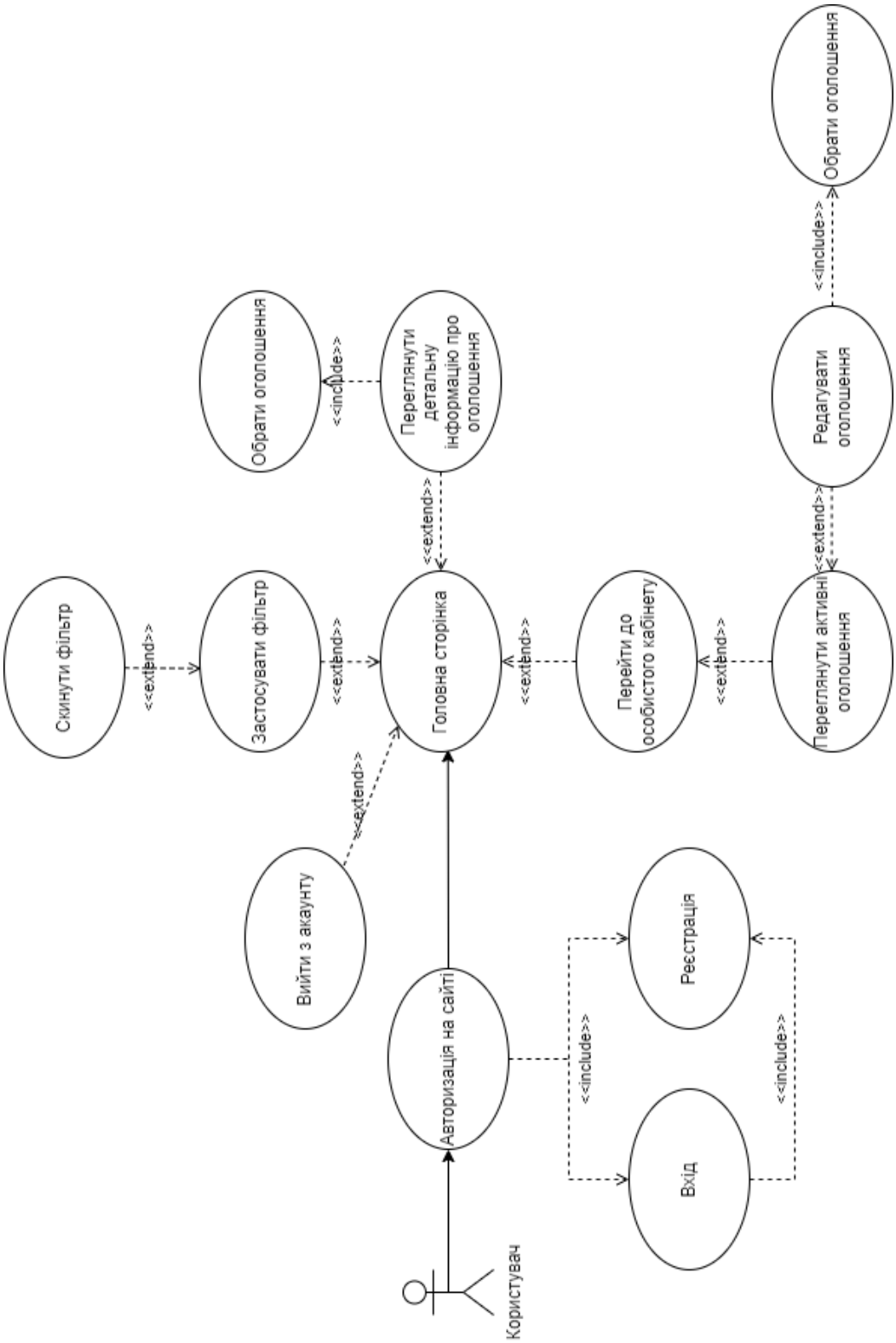
Додаток Д. Фрагмент коду класу для тестування контролера BrandsController

```
public class BrandsControllerTests
   {
      private readonly BrandsController _controller;
      private readonly IBrandService _service;

      public BrandsControllerTests()
      {
         _service = new BrandServiceFake();
         _controller = new BrandsController(_service);
      }

      [Fact]
      public void GetAllBrandsTest()
      {
         //Arrange
         //Act
         var res = _controller.Get();

         //Assert
         Assert.IsType<OkObjectResult>(res.Result);

         //Is result a list of BrandModels
         var list = res.Result as OkObjectResult;
         Assert.IsType<List<BrandModel>>(list.Value);

         //Check number of brands
         var listBrands = list.Value as List<BrandModel>;
         Assert.Equal(4, listBrands.Count);
      }

      [Fact]
      public async void GetBrandByIdTest()
      {
         //Arrange
         var validId = 4;
         var invalidId = 10;

         //Act
         var notFoundRes = await _controller.Get(invalidId);
         var okRes = await _controller.Get(validId);

         //Assert
         Assert.IsType<NotFoundObjectResult>(notFoundRes.Result);
         Assert.IsType<OkObjectResult>(okRes.Result);

         //Is result a BrandModel object
         var item = okRes.Result as OkObjectResult;
         Assert.IsType<BrandModel>(item.Value);

         //Is result a correct brand
         var brandItem = item.Value as BrandModel;
         Assert.Equal(validId, brandItem.Id);
         Assert.Equal("Volkswagen", brandItem.Name);
```

```
}

[Fact]
public async void AddBrandTest()
{
    //Arrange
    var newBrand = new BrandModel()
    {
        Name = "Nissan"
    };

    //Act
    var createdResponse = await _controller.Post(newBrand);

    //Assert
    Assert.IsType<OkObjectResult>(createdResponse.Result);

    //Value of the result
    var item = createdResponse.Result as OkObjectResult;
    Assert.IsType<BrandModel>(item.Value);

    //Check value of this brand
    var brandItem = item.Value as BrandModel;
    Assert.Equal(5, brandItem.Id);
    Assert.Equal(newBrand.Name, brandItem.Name);
}

[Fact]
public async void DeleteBrandTest()
{
    //Arrange
    var validId = 4;
    var invalidId = 10;

    //Act
    var notFoundRes = await _controller.Delete(invalidId);

    //Assert
    Assert.IsType<NotFoundObjectResult>(notFoundRes);
    Assert.Equal(4, _service.GetBrands().Count());

    //Act
    var okRes = await _controller.Delete(validId);

    //Assert
    Assert.IsType<OkResult>(okRes);
    Assert.Equal(3, _service.GetBrands().Count());
}

[Fact]
public async void UpdateBrandTest()
{
    //Arrange
    var newBrand = new BrandModel()
    {
```

```
            Name = "Opel"
        };
        var validId = 4;
        var invalidId = 10;

        //Act
        var notFoundRes = await _controller.Put(newBrand, invalidId);

        //Assert
        Assert.IsType<NotFoundObjectResult>(notFoundRes.Result);

        //Act
        var okRes = await _controller.Put(newBrand, validId);

        //Assert
        Assert.IsType<OkObjectResult>(okRes.Result);

        //Value of the result
        var item = okRes.Result as OkObjectResult;
        Assert.IsType<BrandModel>(item.Value);

        //Ckeck value of the new brand
        var brandItem = item.Value as BrandModel;
        Assert.Equal(newBrand.Name, brandItem.Name);
    }
}
```

Додаток Е. Use-case діаграма для звичайного користувача



Додаток Ж. Use-case діаграма для адміністратора