Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

Навчально-науковий інститут деревообробних та
комп'ютерних технологій і дизайну
(повне найменування інституту, назва факультету (відділення))

Кафедра інформаційних технологій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до дипломної роботи

другий (магістерський)
(рівень вищої освіти)

на тему: "Розроблення інтелектуальної системи для підбору, порівняння та аналізу автомобільного ринку в Україні"

Виконав: студент 6 курсу групи КН(м)
спеціальності
122 "Комп'ютерні науки"
(шифр і назва напряму підготовки, спеціальності)

Гарасимчук М. В.
(прізвище та ініціали)
Керівник Карашецький В.П.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Львів – 2022 року

Національний лісотехнічний університет України
(повне найменування вищого навчального закладу)

ННІ деревооброблювальних та комп'ютерних технологій і дизайну
Кафедра інформаційних технологій
Рівень вищої освіти другий (магістерський)
Спеціальність 122 "Комп'ютерні науки"

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Крошний І.М.
"____" _____2022 року

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Гарасимчук Максим Васильович
(прізвище, ім'я, по батькові)

1. Тема магістерської роботи Розроблення інтелектуальної системи для підбору, порівняння та аналізу автомобільного ринку в Україні

керівник роботи Карашецький Володимир Петрович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від "13" 12 2021 року № С-617

2. Термін подання студентом проекту (роботи) 12 грудня 2022 року

3. Вихідні дані до проекту (роботи) Спроектувати інтелектуальну систему для підбору, порівняння та аналізу автомобільного ринку в Україні

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

4.1. Стан проблемної області.
4.2. Інформаційне забезпечення
4.3. Математичне забезпечення
4.4. Програмне забезпечення
4.5. Розроблення стартап-проекту
4.6. Висновки

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

6. Дата видачі завдання 20 грудня 2021 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	Примітка
1	Огляд літературних та інших джерел згідно досліджуваної теми	9.03.22 – 29.03.22	виконано
2	Аналіз досліджуваної теми та вибір відповідних варіантів її розробки	3.04.22 – 22.04.22	виконано
3	Постановка задачі та її формалізація	22.04.22 – 29.04.22	виконано
4	Вибір та обґрунтування методів і засобів проведення дослідження	2.05.22 – 24.05.22	виконано
5	Розроблення концептуальної схеми реалізації завдання	26.06.22 – 29.06.22	виконано
6	Програмна реалізація завдання	2.07.22 – 29.09.22	виконано
7	Тестування програмного продукту та отриманих результатів	4.10.22 – 29.10.22	виконано
8	Розробка пояснювальної записки та презентації дипломної роботи	1.11.22 – 10.12.22	виконано

Студент _____ Гарасимчук М. В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Карашецький В.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська дипломна робота складається із 93 стор., 16 рис., 4 табл., 2 додатків, 11 джерел.

Об’єкт дослідження – методи реалізації та програмні засоби проєктування інтелектуальних систем.

Предмет дослідження – інтелектуальна система для підбору, порівняння та аналізу автомобільного ринку на території України.

В даному дипломному проекті досліджено можливості платформи .NET для розробки інтелектуальних систем. Спроектовано та розроблено інтелектуальну систему для підбору, порівняння та аналізу автомобільного ринку України, яка дозволяє користувачам переглядати інформацію про моделі автомобілів та завантажувати її з можливістю поширення, а адміністратору – керувати даними.

Ключові слова: інтелектуальна система, автомобільний ринок, розробка, .NET, React, TypeScript, SQLite, Entity Framework.

ANNOTATION

Master's thesis consists of 93 pages, 16 figures, 4 tables, 2 appendices, 11 sources.

The object of the study is implementation methods and software tools for designing intelligent systems.

The research subject is an intelligent system for selecting, comparing, and analyzing the automobile market in Ukraine.

This thesis project explores the possibilities of the .NET platform for developing intelligent systems are investigated. An intelligent system for the selection, comparison, and analysis of the car market of Ukraine was designed and developed, which allows users to view information about car models and uplooad it with the possibility of expansion and the administrator to manage the data.

Keywords: an intelligent system, an automobile market, development, .NET, React, TypeScript, SQLite, Entity Framework.

ТЕХНІЧНЕ ЗАВДАННЯ

Дослідити можливості, які надає платформа .NET для розробки інтелектуальних систем. Спроектувати інтелектуальну систему для підбору, порівняння та аналізу автомобільного ринку в Україні, яка дозволяє користувачам переглядати інформацію про моделі автомобілів та завантажувати її з можливістю поширення. Розроблена система повинна надавати адміністратору можливість контролювати внесені дані.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ГІК – графічний інтерфейс користувача;

ООП – об’єктно-орієнтоване програмування;

ПЗ – програмне забезпечення;

ОС – операційна система;

СКБД – система керування базами даних;

C# – об’єктно-орієнтована мова програмування;

CRUD – Create-Read-Update-Delete;

IDE – Integrated Development Environment;

JS – JavaScript;

JSX – JavaScript Syntax Extension;

API - Application Programming Interface;

JWT – JSON Web Token;

MVC – Model-View-Controller;

ORM – Object-Relational Mapping;

REST – Representational State Transfer;

TDD – Test Driven Development;

TS – TypeScript;

.NET – платформа від компанії Microsoft.

ВСТУП

Основною перевагою інтелектуальних веб-систем є універсальність, завдяки якій їх можна запустити буквально на любому пристрої незалежно від його операційною системи, обчислювальних можливостей, чи, наприклад, розміру екрану. Важливо також зазначити, що веб-системи вимагають меншої підтримки та надають більше безпеки для даних клієнта, що суттєво впливає на вартість продукту, як для розробників, так і кінцевих клієнтів. Отже, розробка веб-систем вносить лише позитивний вплив на будь-який вид бізнесу, зменшуючи витрати на нього.

Об'єкт дослідження – методи реалізації та програмні засоби проєктування інтелектуальних систем.

Предмет дослідження – інтелектуальна система для підбору, порівняння та аналізу автомобільного ринку на території України.

Наукова новизна – проведення дослідження можливостей платформи .NET для розробки інтелектуальних систем.

Практична значимість. Спроектована система дозволяє розробнику та користувачам наповнювати і доповнювати базу даних інформацією у реальному часі.

Актуальність роботи. Продажі та купівлі автомобілів на вторинному ринку України останнім часом значно виросли в зв'язку із зростанням попиту населення на вживані транспортні засоби, що завозяться з-за кордону за відносно невисокими цінами. Використання інтелектуальних веб-систем дає можливість користувачам отримати повну та достовірну інформацію про той чи інший транспортний засіб, його переваги та недоліки.

Мета дослідження – розробка інтелектуальної системи-порталу для підбору, порівняння та аналізу автомобільного ринку на території України з використанням технологій .NET та TypeScript.

РОЗДІЛ 1 СТАН ПРОБЛЕМНОЇ ОБЛАСТІ

1.1 Стан автомобільного ринку України

Останні два роки в Україні, як і у всьому світі є дуже непростими. На обмеження, пов'язані з пандемією коронавірусу, наклалася ще й катастрофічна нестача мікросхем та війна в Україні, через що велика кількість автовиробників змушені призупиняти та скорочувати випуск своє продукції. В цей час покупці вже відчули затримки з доставкою нових автомобілів. Інколи трапляється, що деякі автовиробники мають порожні склади, а в наш побут приходить слово "черга".

Автомобільний ринок – це сукупність економічних відносин, завдяки яким відбувається взаємодія суб'єктів ринку з метою обміну готових автомобілів на грошові кошти або їх еквіваленти.

Автомобілі є товарами тривалого користування, на які попит визначається за наступними факторами:

- об'єктивними – доходи населення, ціни на автомобілі, валютний курс, процент за кредит, ставки за депозитами;
- суб'єктивними – сприйняття споживачами поточної ситуації, індекс споживчого настрою, готовність взяти кредит, очікування змін в майбутньому.

Ситуація з виробництом нових автомобілів є досить непростою. Велика кількість учасників ринку та експертів розділяють думку, що проблема з нестачею мікросхем ще не є остаточно вирішеною. Також варто зазначити, що купівельні можливості громадян знилися із зростанням інфляції. Виходячи з цих факторів, кількість бажаючих купити новий автомобіль у 2022-2023 роках суттєво знизиться, а охочі це зробити будуть обмежені в комплектаціях та повинні набратися терпіння, адже для деяких брендів очікування замовленого автомобіля затягнеться на пів року, а то й довше.

Якщо купити новий автомобіль є певною мірою проблематично, вторинний ринок, який і без того домінує в Україні, у 2022-2023 роках має всі шанси на зростання. Наприклад, для більшості людей може стати привабливою можливість

купити за ціною нового автомобіля середнього цінового сегменту певну модель класом вище. Виходячи з цього варто зазначити ще одну не менш важливу тенденцію, яка давно наростає, а в період війни взагалі виходить на перший план. Маю на увазі де та як потенційні покупці шукають вживані автомобілі – звичайно, в Інтернеті, на сайтах онлайн-оголошень. Автобазари у класичному вигляді йдуть у минуле. В цьому немає нічого дивного, оскільки Інтернет проник у всі сфери нашого життя, де пошук, чи навіть покупка автомобілів не повинна бути виключенням. Також варто зазначити, що онлайн-пошук авто є дійсно зручним. Завжди можна використати фільтри та відсортувати бажані автомобілі дивлячись на свій бюджет та інші вимоги, переглянути фотографії, чи порівняти з іншими варіантами на ринку. Важко переоцінити економію часу у сучасному світі. Виходячи з даних сайту OLX, кожного місяця понад 9 мільйонів разів виконується пошук легкових автомобілів в категорії А. Отже бажання мати власне авто в українців не зменшується.

Звичайно, потрібно відзначити, які саме автомобілі тепер користуються найбільшою популярністю на вторинному ринку. Дивлячись на те, про що говорилося вище, легко спрогнозувати ще більший попит на автомобілі, які завозяться із США. На вторинному ринку користуються також попитом преміум бренди з великим пробігом. Особливо це стосується німецької трійки – BMW, Audi та Mercedes-Benz. Також, цілком можливий стрибок популярності електромобілів, оскільки їх ціни поступово знижуються, а переваги їх використання усім відомі. Коли ж говорити цифрами, то, за прогнозами, у 2022-2023 роках співвідношення покупок нових та вживаних автомобілів буде відповідати 1:10, тобто на нові автомобілі припаде не більше 10% ринку, що цілком віддзеркалює кризу в автомобільній галузі, так і рівень достатку населення.

1.2 Аналіз існуючих систем

На даний момент, в Україні є декілька інформаційних систем для пошуку вживаних транспортних засобів, основними з яких є OLX та Auto.RIA. Проте,

важливим фактором є те, що вищезгадані системи являються вебсистемами оголошень. Сайт OLX надає можливість лише розміщувати оголошення без створення дискусій між користувачами, та унеможливлює висловлення їх власної думки щодо представлених автомобілів. Auto.Ria представляє комплексну систему-портал із значною кількістю можливостей. До прикладу, вона має власний блог, на ній проводиться опитування, створюються підбірки автомобілів за цінами, призначенням та роками, використовуючи значну кількість фільтрів. Важливою особливістю даної системи є можливість коментування оголошень, та ведення дискусій. Але дана платформа є комерційною, і обговорення проводяться виключно на конкретних оголошеннях, що змушує користувачів висловлюватись з метою покращення ціни на транспортний засіб, а не його об'єктивних характеристик.

Розроблена мною інтелектуальна система призначена не для купівлі-продажу транспортних засобів, а для можливості користувачів висловлювати власну думку про переваги та недоліки конкретних автомобілів, чи шукати поради шляхом обговорення щодо вирішення проблем з власним транспортним засобом.

Висновки. Проаналізовано стан автомобільного ринку України за 2022-2023 роки. Розглянуто існуючі інформаційні системи для пошуку та порівняння автомобільного ринку в Україні. Проаналізувавши їх переваги та недоліки, мною вирішено розробляти інформаційну систему без можливості продажу автомобілів з метою отримати повну та достовірну інформацію про той чи інший транспортний засіб, його переваги та недоліки.

РОЗДІЛ 2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вибір платформи розробки

На даний момент існує безліч платформ, що дозволяють розробляти інформаційні системи. Для даної роботи я вирішив використовувати саме .NET.

Платформа .NET від Microsoft і мова програмування C# були офіційно представлені приблизно в 2002 році і швидко стали опорою сучасної розробки програмного забезпечення. Платформа .NET забезпечує велику кількість мов програмування (включаючи C#, VB.NET і F#), щоб взаємодіяти одна з одною. Програма, написана на C#, може комунікувати з програмої написаною на VB.NET.

У 2016 році корпорація Microsoft офіційно запустила платформу .NET Core, яке більше не обмежується операційною системою Windows, а також може працювати і на iOS та Linux. Ця незалежність від платформи відкрила C# для значно більшого кола розробників, осільки кросплатформне використання C# раніше не підтримувалось на пряму, а лише через різні інші платформи, такі як проект Mono.

.NET – це програмна платформа для створення вебсистем і систем сервісів на Windows, iOS та Linux, а також програми WinForms і WPF на операційній системі Windows. До основних переваг платформи можна віднести наступні:

– Сумісність із існуючим кодом. Існуюче програмне забезпечення (ПЗ) .NET Framework може взаємодіяти з новішим програмним забезпеченням .NET Core, і навпаки, через .NET Standard.

– Підтримка численних мов програмування: програми .NET можуть бути створені на мовах програмування C#, F# і VB.NET (при цьому C# і F# є основними мовами для ASP.NET Core).

– Мовна інтеграція: .NET підтримує міжмовне успадкування, обробку винятків та налагодження коду. Наприклад, можна визначити базовий клас у C# і розширити цей тип у Visual Basic.

– Повна бібліотека базових класів: ця бібліотека надає тисячі попередньо визначених типів, що дозволяють будувати бібліотеки коду, прості термінальні застосунки, графічні настільні програми та вебсайти корпоративного рівня.

– Спрощена модель розгортання: бібліотеки .NET не зареєстровані в системний реєстр. Крім того, платформа .NET дозволяє використовувати декілька версій фреймворків, а також програми, щоб гармонійно існувати на одній машині.

– Розширена підтримка командного рядка: інтерфейс командного рядка .NET (CLI) кросплатформний пакет інструментів для розробки застосунків .NET. Додаткові інструменти можуть бути встановлені (глобально чи локально), крім стандартних інструментів, які постачається з пакетом SDK .NET.

Базова платформа містить компоненти, які застосовуються до всіх типів програм. Додаткові фреймворки, такі як ASP.NET, розширюють .NET компонентами для створення конкретних типів програм.

Ось деякі речі, які ASP.NET додає до платформи .NET:

– базовий фреймворк для обробки вебзапитів на C# або F#;

– синтаксис шаблону вебсторінки, відомий як Razor, для створення динамічних вебсторінок за допомогою C#;

– бібліотеки для поширених вебшаблонів, наприклад, контролер перегляду моделі (MVC);

– система аутентифікації, яка включає бібліотеки, бази даних (БД) і шаблонні сторінки для обробки входу, включаючи багатофакторну автентифікацію та зовнішню аутентифікацію за допомогою Google, Twitter тощо;

– розширення редактора для забезпечення виділення синтаксису, доповнення коду та інших функцій спеціально для розробки вебсторінок.

ASP.NET пропонує декілька платформ для створення інтернет-систем. Усі фреймворки стабільні та дозволяють створювати інтелектуальні системи. Кожний з фреймворків надає всі переваги та можливості ASP.NET та націлений на інший стиль розробки. Вибір фреймворку залежить від поєднання програмних активів (знань, навичок та досвіду) роробника, типу програми, яка створюється, і підходу до розробки.

З допомогою ASP.NET Web Forms можна створювати динамічні вебсайти, використовуючи звичну модель перетягування, керовану подіями. Сотні елементів керування та компонентів дозволяють швидко створювати складні, потужні сайти на основі інтерфейсу користувача з доступом до даних.

ASP.NET MVC надає потужний, заснований на шаблонах спосіб створення динамічних вебсайтів, що дозволяє чітко розділяти проблеми та дає повний контроль над розміткою для приємної та швидкої розробки. ASP.NET MVC містить багато функцій, які забезпечують швидку, TDD-дружню розробку для створення складних програм, які використовують новітні вебстандарти.

ASP.NET Web Pages та синтаксис Razor забезпечують швидкий, доступний і легкий спосіб поєднання коду сервера з HTML для створення динамічного веб-вмісту. Під'єднанн до БД і багато інших функцій допомагають створювати сайти, які відповідають останнім вебстандартам.

ASP.NET Web API – це структура, яка дозволяє легко створювати HTTP-сервіси, які охоплюють широкий спектр клієнтів, включаючи браузери та мобільні пристрої. ASP.NET Web API є ідеальною платформою для створення REST-застосунків на .NET.

Усі фреймворки ASP.NET засновані на .NET Framework і спільно використовують основні функції .NET і ASP.NET. Наприклад, усі три платформи пропонують модель безпеки входу і мають однакові засоби для керування запитами, обробкою сеансів тощо, які є частиною основної функціональності ASP.NET [1,11].

Крім того, ці три структури не є абсолютно незалежними, і вибір однієї не виключає використання іншої. Оскільки фреймворки можуть співіснувати в одній вебсистемі, нерідко можна побачити окремі компоненти програм, написані за допомогою різних фреймворків. Наприклад, частини програми, орієнтовані на клієнта, можуть бути розроблені в MVC для оптимізації розмітки, тоді як частини доступу до даних та адміністративні частини можуть бути представлені у вигляді вебформ, щоб скористатися перевагами контролю даних і простотою доступу до даних.

2.2 Вибір методу автентифікації та авторизації

Однією з функцій, яку потрібно включати в інтелектуальні системи, є можливість обмежувати доступ до певних ресурсів у системі лише авторизованим користувачам. Для цього ми повинні мати можливість автентифікувати користувачів, дозволивши їм зареєструватися та увійти. На ринку доступно багато підходів аутентифікації, але найпопулярнішим підходом є “Автентифікація на основі токенів”.

Простими словами можна сказати, що автентифікація – це перевірка користувача за допомогою облікових даних особи.

На ринку доступно багато відкритих стандартів для впровадження аутентифікації на основі токенів, але JSON Web Token (JWT) є найпопулярнішим серед них.

Аутентифікація на основі JWT токенів (рис. 2.1) базується на наступних концептах:

– клієнт відправив запит на сервер з обліковими даними;

– сервер перевіряє облікові дані, створює токен доступу та надсилає його назад клієнту;

– усі запити підпослідовності містять цей токен до закінчення терміну його дії.

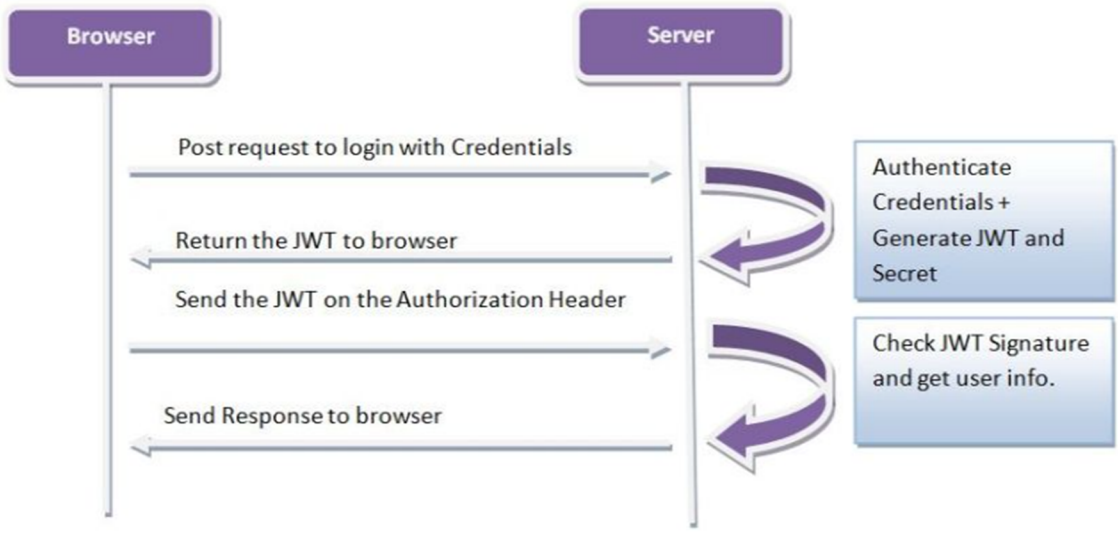

Рис. 2.1. Схема роботи JWT токена

JWT – це відкритий стандарт, який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами у вигляді об'єкта JSON. Цю інформацію можна перевірити, їй можна довіряти, оскільки вона має цифровий підпис. JWT можна підписати за допомогою секретного алгоритму HMAC або пари відкритих і закритих ключів RSA чи ECDSA.

2.3 Вибір бази даних

Реляційна система керування база даних (СКБД) – це набір елементів даних із попередньо визначеними зв'язками між ними. Ці елементи організовані як набір таблиць зі стовпцями та рядками. Таблиці використовуються для зберігання інформації про об'єкти, які будуть представлені в БД. Кожен стовпець таблиці містить певний тип даних, а поле зберігає фактичне значення атрибута. Рядки в таблиці представляють набір пов'язаних значень одного об'єкта або сутності. Кожен рядок у таблиці може бути позначений унікальним ідентифікатором, який називається первинним ключем, а рядки між кількома таблицями можуть бути пов'язані за допомогою зовнішніх ключів. До цих даних можна отримати доступ багатьма різними способами без реорганізації самих таблиць БД [6].

SQLite – це бібліотека, яка в процесі роботи реалізує автономний, безсерверний, транзакційний механізм СКБД SQL з нульовою конфігурацією. Код для SQLite є загальнодоступним і, таким чином, безкоштовний для будь-яких цілей, комерційних чи приватних. SQLite – це одна з поширених СКБД.

SQLite – це вбудований механізм СКБД. На відміну від більшості інших СКБД, SQLite не має окремого серверного процесу. Читання та запис інформації в SQLite виконується безпосередньо на звичайні дискові файли. БД з кількома таблицями, індексами, тригерами та представленнями міститься в одному дисковому файлі. Формат файлу БД є кросплатформним, тому можна вільно копіювати БД між 32-розрядними та 64-розрядними системами або між архітектурами великого та малого порядків.

SQLite – це компактна бібліотека, розмір якої з усіма увімкненими функціями може бути меншим за 750 КБ, залежно від цільової платформи та

налаштувань оптимізації компілятора. Деякі оптимізації компілятора, такі як агресивне вбудовування функцій і розгортання циклу, можуть призвести до значного збільшення об'єктного коду. Існує компроміс між використанням пам'яті та швидкістю. SQLite, як правило, працює тим швидше, чим більше пам'яті їй виділяється. Тим не менш, її продуктивність зазвичай досить хороша навіть в середовищі з низьким рівнем пам'яті. Залежно від її використання, вона може бути швидшою, ніж прямий ввід-вивід файлової системи.

SQLite дуже ретельно тестується перед кожним випуском і має надійну репутацію. Більша частина вихідного коду SQLite присвячена виключно тестуванню та перевірці. Автоматизований набір тестів запускає тестові випадки, що включають сотні мільйонів окремих операторів SQL, і досягає 100% тестування.

2.4 Вибір архітектури для розробки клієнтської частини системи

Зараз існує три найпопулярніших архітектури для розробки інтерактивних клієнтських частин, а саме React, Angular та Vue. Якщо розглядати кожну з них в деталях, то важко визначити якісь суттєві переваги, чи недоліки, які б виділяли її серед інших. Хоча в даній роботі я використав саме React, оскільки вона була однією з перших, реліз якої відбувся в березні 2013 року, що дозволило їй набрати не аби-якої популярності серед розробників, також вона розроблена компанією Facebook, яка на даний момент називається Meta, що також впливає на її популярність.

React – це JavaScript бібліотека, завдяки якій можна створювати інтерфейси користувача.

В першу чергу бібліотека призначалась для розробки користувацьких інтерфейсів вебсайтів. Згодом з'явилася платформа React Native, що дозволила використовувати React для написання програм як для Andoid, чи IOS.

Це ідеальний інструмент для вебсистем, особливо Single Page Aplications.

React проста в вивченні, має зрозумілий синтаксис, безліч шаблонів та callback-функції для відтворення HTML. Весь результат роботи React – це HTML.

Одна з ключових особливостей React – її універсальність. Цю бібліотеку можна використовувати на сервері і на мобільних платформах за допомогою React Native.

Ще одна важлива особливість бібліотеки – декларативність. За допомогою React розробник описує, як компоненти інтерфейсу виглядають в різних станах. Декларативний підхід скорочує код і робить його зрозумілим.

React заснована на компонентах, це ще одна ключова особливість бібліотеки. Кожен компонент повертає частину призначеного для користувача інтерфейсу зі своїм станом. Об'єднуючи компоненти, програміст створює завершений інтерфейс вебзастосунку.

Важлива особливість React – використання JavaScript Syntax Extension(JSX). Це розширення синтаксису JavaScript, яке зручно використовувати для опису інтерфейсу. JSX схоже на HTML, проте це все-таки JavaScript.

До важливих особливостей React відноситься використання набору бібліотек Virtual DOM, що надає представлення, в якому зберігається інформація про стан інтерфейсу. При зміні стану, наприклад, після відправки форми або натискання кнопки, React розраховує різницю між старим і новим станом. Після цього React генерує новий стан. Використання Virtual DOM дозволяє React ефективно взаємодіяти з реальним DOM-деревом [3, 10].

Для роботи з бібліотекою React, я вважаю варто використовувати мову програмування TypeScript.

TypeScript – мова програмування з відкритим вихідним кодом, розроблена Microsoft, яка компілюється у JavaScript. З моменту виходу в 2012 році вона продовжує з кожним роком активно розвиватися і набирати популярність.

Найважливіша відмінність TypeScript від JavaScript полягає у тому, що це дві окремі мови програмування, хоча TypeScript значною мірою базується на JavaScript. Фактично, TypeScript є наднабором JavaScript, тобто весь дійсний код JavaScript також є дійсним кодом TypeScript.

TypeScript не змінює JavaScript, натомість розширюючи його новими цінними функціями. Все, що можна написати на JavaScript, також можна

написати на TypeScript.

Ключове значення, яке TypeScript приносить до JavaScript, – це статична типізація даних.

Традиційно в JavaScript одна змінна може містити текст, число або навіть цілу сутність бази даних, залежно від випадку – і це нормально. Те, що робить TypeScript, – це суворо визначає, що може містити дана змінна.

Скажімо, один із розробників пише функцію addNumbers() і використовує типи, щоб визначити, що функція може приймати лише два типи числа. TypeScript поверне помилку, якщо інший розробник спробує надати функції текстове значення, тоді як у JavaScript така операція була б цілком прийнятною.

Завдяки широкому вибору корисних типів і нових функцій TypeScript є чудовим інструментом для розробників JavaScript.

Давайте подивимося на переваги TypeScript, а також на те, як вони впливають на розробку ПЗ для інтерфейсу:

– **Точне визначення типу даних за допомогою введення.** Коли розробник робить помилку і вводить непідтримувані дані в код, він дізнається про це, перед використанням коду, тоді як JS не дає такої можливості. І навіть якщо він проігноруює цю помилку, то також є системи CI/CD, такі як Jenkins, які можуть перевіряти типи та запобігати їх подальшому використанню. Здатність знаходити ці очевидні, але часті помилки заздалегідь, значно полегшує керування кодом із типами.

– **Типи даних полегшують керування кодом.** Переваги типів, природно, виходять за рамки простого коду. Зовнішні бібліотеки також чудово працюють із TypeScript. Раніше розробникам потрібно було переглядати об’ємну документацію, щоб дізнатися, де знаходиться дана функція, яку вони можуть використовувати і як, або яку можна безпечно замінити. За допомогою TypeScript ми отримуємо всю цю інформацію безпосередньо із середовища розробки, що заощаджує багато часу.

– **Підвищується продуктивність команди розробників.** Явно визначені структури даних і анотації типів незрівнянно полегшують розуміння рішень, прийнятих інженерами [4,5].

2.5 Вибір середовища розробки

Протягом багатьох років середовище розробки Visual Studio було по суті єдиним інструментом, що пропонував функціональні можливості, які можна було використовувати для розробки .NET на рівні підприємства. Звісно, існували й інші інструменти, але вони, як правило, не досягали рівня Visual Studio.

За останні роки цей вибір значно розширився завдяки Visual Studio Code, MonoDevelop, SharpDevelop та JetBrains Rider. Не всі з них є безкоштовними або відкритими, і, загалом, це проявляється в якості інструменту або функцій, які вони пропонують.

Найбільшими конкурентами є Visual Studio від Microsoft та Rider від JetBrains. У Rider є лише платна версія, а не безкоштовна. Це відрізняється від Visual Studio, яка також пропонує безкоштовну версію, звичайно, позбавлену кількох функцій свого корпоративного аналога.

Rider походить від інших програмних продуктів компанії JetBrains, таких як ReSharper і WebStorm, але тепер є повноцінним середовищем розробки. Воно є кросплатформним, тобто може працювати як на Windows, так і на Mac, і на декількох версіях Linux, пропонуючи однаковий набір функціональних можливостей та ідентичну поведінку на всіх з них. Visual Studio підтримує лише Mac і Windows.

Rider реагує та настроюється більш гнучко, можна вибрати свою колірну схему, прив'язки клавіатури, воно швидке у порівнянні з Visual Studio. Тут можете мати декілька вікон, навіть згорнутих, а потім зберегти налаштування.

Rider пропонує декілька шаблонів для проектів .NET, .NET Core, Unity і Xamarin, які приблизно ідентичні тому, що має Visual Studio, але більше шаблонів проектів можна додати шляхом їх завантаження. Рішення та проекти, з якими

працює Rider, повністю сумісні з Visual Studio, тобто не використовують жодного власного формату.

У Visual Studio використовується статичний аналіз коду StyleCop і його перевірка, і це неймовірно корисно. Rider також включає ці правила, але він робить набагато більше, ніж просто перевірку мови, наприклад, може показувати певні конструкції коду як помилки, як-от відсутність іменованого представлення. Rider не перевіряє лише код .NET.

Останніі версії Visual Studio надають велику кількість варіантів рефакторингу, але Rider має тут значно більші можливості. Rider може інвертувати логіку умовного блоку, витягувати код до нового методу, створювати похідний тип, переміщувати методи в інший файл (часткові класи), перетворювати властивість у метод, перетворювати член екземпляра на статичний, видаляти оголошення “this”, тощо. Корисний рефакторинг використовує базовий тип замість похідних типів, коли це можливо, інший – генерує базовий тип для існуючого, за бажанням переміщуючи до нього деякі члени класу. Інший рефакторинг перевіряє члени класу на їх видимість і пропонує обмежити це, якщо це можна зробити, нічого не порушуючи. Надані рефакторинги є одним із найсильніших аспектів Rider. Саме через вищезгадані причини, даний проєкт розроблявся в середовищі Rider.

Висновки. Проаналізовано платформу .NET для розробки інтелектуальних систем, в результаті чого було вирішено використовувати саме її. Розглянуто існуючі архітектури для розробки клієнтських частин, після чого вирішено використовувати React. Переглянуто переваги використання СКБД SQLite, яка була використана в розробці інтелектуальної системи. В якості IDE для написання програмного коду вибрано Rider від компанії JetBrains.

РОЗДІЛ 3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Встановлення комунікації між клієнтською та серверною частиною системи

Враховуючи, що інтелектуальна система розроблена з використанням двох архітектур, а саме .NET, що є відповідальною за серверну частину, та React, яка відповідає за клієнтський інтерфейс, важливим етапом є встановлення комунікації між цими двома архітектурами.

Для комунікації між двома архітектурами я використав REST API. API – це набір програмного коду, який забезпечує передачу даних між одним системним продуктом та іншим, а також містить умови обміну даними.

Інтерфейс прикладного програмування (application programming interface) необхідно чітко відрізняти від інтерфейсу користувача, який приймає дані від користувачів, передає їх системі для обробки та повертає результати користувачеві. API не взаємодіє з користувачем, а обробляє дані, отримані від одного модуля програми, і передає результати назад в інший модуль.

Принцип роботи API зазвичай виражається через зв'язок запит-відповідь між клієнтом і сервером. Клієнт – це будь-яка зовнішня програма, з якою взаємодіє користувач. Сервер відповідає за серверну логіку та операції з БД. У цьому сценарії API працює як проміжний рівень між клієнтом і сервером, що дозволяє надсилати запити даних і відповіді. Схема даної архітектури зображена на рисунку 3.1.

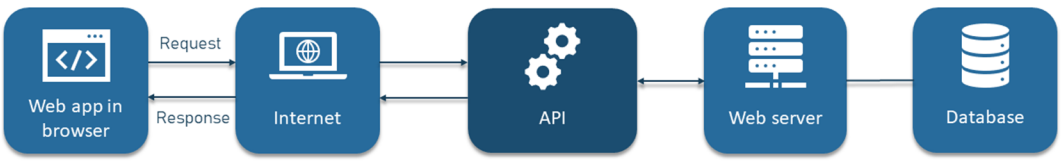

Рис. 3.1. Схема роботи API

Фахівці Red Hat зазначають, що API іноді вважається контрактами, де дані є угодою між сторонами. Інтерфейс прикладного програмування складається з двох компонентів:

- технічної специфікації, що описує варіанти обміну даними із специфікацією у формі запиту та протоколи доставки даних;
- програмного інтерфейсу, який доданий у специфікацію, яка його представляє;

Програмне забезпечення, якому потрібен доступ до інформації (наприклад, готельні номери на певні дати) або функціональних можливостей (тобто маршруту від пункту А до пункту Б на карті на основі місцезнаходження користувача) з іншим ПЗ, викликає свій API, вказуючи вимоги про те, як потрібно надавати дані/функції. Інше програмне забезпечення повертає дані/функції, запитувані попередньою програмою. API є інтерфейсом, за допомогою якого ці дві програми спілкуються.

Кожен API реалізується за допомогою викликів функцій – мовних інструкцій, які запитують ПЗ для виконання певних дій. Виклики функцій складаються з дієслів (наприклад, BEGIN, GET, DELETE тощо) та іменників (наприклад, Data, Access тощо), які дозволяють машині зрозуміти, що робити далі, наприклад:

- почати або завершити сеанс;
- отримати зручності для одномісного номера;
- відновлення або отримання об’єктів із сервера.

API можна класифікувати відповідно до систем, для яких вони розроблені. API бази даних забезпечують зв’язок між програмою та СКБД. Розробники працюють із БД, пишучи запити для доступу до даних, змінення таблиць тощо. API бази даних Drupal 7, наприклад, дозволяє користувачам писати уніфіковані запити для різних БД, як пропрієтарних, так і відкритих (Oracle, MongoDB, PostgreSQL, MySQL, CouchDB та MSSQL).

API операційних систем (ОС) визначає, як програми використовують ресурси та служби операційних систем. Кожна ОС має свій набір API, наприклад Windows API або Linux API (API простору користувача та внутрішній API ядра).

Віддалені API визначають стандарти взаємодії для програм, що працюють на різних машинах. Іншими словами, один програмний продукт отримує доступ до ресурсів, розташованих за межами пристрою, який їх запитує, що пояснює назву. Оскільки дві віддалені програми з'єднані через комунікаційну мережу, зокрема через Інтернет, більшість віддалених API написані на основі веб-стандартів. Java Database Connectivity API та Java Remote Method Invocation API є двома прикладами інтерфейсів віддаленого програмування прикладних програм.

Найпоширенішим є веб API. Вебінтерфейси API забезпечують передачу машинно-зчитуваних даних і функцій між вебсистемами, які представляють архітектуру клієнт-сервер. Ці API в основному доставляють запити від веб-застосунків і відповіді від серверів за допомогою протоколу передачі гіпертексту (HTTP).

Більшість систем використовують більше одного API для підключення застосунків і обміну інформацією. Деяким зрештою потрібен інструмент керування API, який допоможе контролювати, поширювати та аналізувати різні API.

Також, API поділяються за специфікаціями. Метою специфікацій API є стандартизація обміну даними між вебслужбами. У цьому випадку стандартизація означає здатність різноманітних систем, написаних на різних мовах програмування та/або що працюють на різних ОС чи використовують різні технології, безперебійно спілкуватися одна з одною.

Вебінтерфейси API можуть дотримуватися принципів обміну ресурсами на основі віддаленого виклику процедури або RPC. Цей протокол визначає взаємодію між застосунками на основі архітектури клієнт-сервер. Одна система (клієнт) запитує дані або функції в іншої системи (сервера), розташованої на іншому комп'ютері в мережі, і сервер надсилає необхідну відповідь.

SOAP – це легкий протокол для обміну структурованою інформацією в децентралізованому розподіленому середовищі відповідно до визначення Microsoft, яка його розробила. Загалом ця специфікація містить правила синтаксису для повідомлень запитів і відповідей, які надсилаються вебзастосунками. API, які відповідають принципам SOAP, дозволяють обмінюватися XML повідомленнями між системами через HTTP або через простий протокол передачі пошти (SMTP). SOAP здебільшого використовується з корпоративним ПЗ для забезпечення високої безпеки переданих даних.

API даної інтелектуальної системи було розроблено за класифікацією REST. Термін REST був введений комп'ютерним науковцем Роєм Філдінгом у дисертації в 2000 році. На відміну від SOAP, REST є архітектурним стилем програмного забезпечення з шістьма обмеженнями для створення програм, які працюють через HTTP. Інтернет є найпоширенішою реалізацією та застосуванням цього архітектурного стилю.

REST вважається простішою альтернативою SOAP, яку багатьом розробникам важко використовувати, оскільки для виконання кожного завдання потрібно писати багато коду та дотримуватись структури XML для кожного надісланого запиту. В цей час REST дотримується іншої логіки, оскільки робить дані доступними як ресурси. Кожен ресурс представлено унікальною URL-адресою, за якою можна зробити запит до нього.

Веб-інтерфейси API, які відповідають архітектурним обмеженням REST, називаються RESTful API. Ці API використовують HTTP-запити (відомі як методи) для роботи з ресурсами: GET, PUT, HEAD, POST, PATCH, CONNECT, TRACE, OPTIONS та DELETE.

Системи REST підтримують обмін повідомленнями в різних форматах, таких як звичайний текст, HTML, YAML, XML і JSON. Тоді як SOAP підтримує лише XML. Можливість підтримувати кілька форматів для зберігання та обміну даними є однією з причин, чому REST є переважним вибором для створення загальнодоступних API у наш час.

3.2 Встановлення доступу до стороннього сховища медіа даних

Відомо, що вартість хостингу є значною, особливо коли потрібен великий розмір сховища для мультимедії. В таких випадках доцільно використовувати сервіси, які дозволять отримати таке сховище за меншу вартість та з додатковими перевагами. Одним з таких сервісів є Cloudinary.

Cloudinary надає рішення для керування зображеннями та відео для веб-сайтів і мобільних застосунків, яке охоплює завантаження зображень і відео, їх зберігання, маніпуляцію, оптимізацію та доставку.

За допомогою Cloudinary можна легко завантажувати зображення та відео в хмару та автоматизувати розумні маніпуляції з цими медіа, не встановлюючи жодного іншого програмного забезпечення. Потім Cloudinary плавно передає ці медіа-файли через мережу швидкої доставки контенту CDN.

Крім того, Cloudinary пропонує комплексні API та можливості адміністрування, які можна легко інтегрувати зі своїми веб та мобільними застосунками.

Cloudinary .NET SDK надає прості, але всеосяжні можливості завантаження зображень і відео, трансформації, оптимізації та доставки, які можна реалізувати за допомогою коду, який легко інтегрується з наявною програмою .NET, як це і було зроблено в моїй веб-системі.

Наведений нижче метод на мові C# завантажує зображення в хмару:

```
public ImageUploadResult Upload(ImageUploadParams parameters);
```

Клас ImageUploadParams встановлює зображення для завантаження з додатковими параметрами, а клас ImageUploadResult надає десеріалізовану відповідь сервера.

Наприклад, асинхронне завантаження локального файлу зображення під назвою "mypicture.jpg":

```
var uploadParams = new ImageUploadParams()
{
  File = new FileDescription(@"mypicture.jpg")
```

```
};
Task<ImageUploadResult> imageUploadTask =
cloudinary.UploadAsync(uploadParams);
var uploadResult = await imageUploadTask;
```

Виклик завантаження повертає об'єкт JSON із вмістом, подібним до такого:

RESPONSE (ImageUploadResult):


```
{
   "public_id":"tquyfignx5bxcbsupr6a",
   "version":1375302801,
   "signature":"52ecf23eeb987b3b5a72fa4ade51b1c7a1426a97",
   "width":1920,
   "height":1200,
   "format":"jpg",
   "resource_type":"image",
   "created_at":"2017-07-31T20:33:21Z",
   "bytes":737633,
   "type":"upload",
   "url": https://res.cloudinary.com/demo/image/upload/v1375302801/
   tquyfignx5bxcbsupr6a.jpg",
   "secure_url":
   "https://res.cloudinary.com/demo/image/upload/v1375302801/
  tquyfignx5bxcbsupr6a.jpg"
}
```


Варто звернути особливу уваги на поля secure_url, та public_id, саме ці два поля використовуються для отримання зображення на стороні клієнтської частини.

Висновки. Описано встановлення комунікації між клієнтською та серверною частиною розроблюваної системи. Розглянуто можливі альтернативи та реалізацію доступу до стороннього сховища медіа файлів [7].

РОЗДІЛ 4 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Реалізація серверної частини системи

Розроблений проект представляє собою Web API, який реалізує CRUD (Додаток Б).

Web API є способом побудови програми ASP.NET, який спеціально передбачений для роботи в стилі Representation State Transfer (REST). REST архітектура передбачає застосування наступних методів або типів запитів HTTP для взаємодії із сервером: GET, POST, PUT, DELETE.

Найчастіше REST-стиль особливо зручний при створенні різноманітних Single Page Application, які нерідко використовують спеціальні JavaScript-фреймворки типу Angular, React або Vue.js. По суті Web API є веб-службою, до якої можуть звертатися інші програми, які можуть представляти будь-яку технологію та платформу.

Визначення контролера починається з атрибута ApiController, що дозволяє додати до контролера деяку додаткову функціональність. Але насправді він необов'язковий для роботи api-контролера [2].

В даному проєкті реалізований загальний контролер з вищезгаданим атрибутом, який наслідується розробленими контролерами (рис. 4.1).

C# BaseApiController.cs

```
1   using Microsoft.AspNetCore.Mvc;
2
3   namespace CourseProject.Controllers
4   {
5       [ApiController]
6       [Route( template: "api/[controller]")]
7       public class BaseApiController : ControllerBase
8       {
9
10      }
11  }
```

Рис. 4.1. Приклад програмного коду загального контролера

Для даного контролера визначений один загальний маршрут

[Route("api/[controller]")]. Тобто замість “controller”, буде використовуватися саме той контролер, до якого будемо звертатись.

Приклад контролера “statistics”, що дозволяє отримати статистику кількості автомобілів за їх маркою представлено на рис. 4.2.

```
[HttpGet(template: "statistics")]
public async Task<IActionResult> GetCarsStatistics()
{
    var popularity :{Manufacturer,Count}[] = await _context.Cars // DbSet<Car>
        .GroupBy(vehicle :Car => vehicle.Manufacturer.ToLower()) // IQueryable<IGrouping<...>>
        .Select(g :IGrouping<string,Car> => new
        {
            Manufacturer = g.Key,
            Count = g.Count()
        }).ToArrayAsync(); // Task<{Manufacturer,Count}[]>

    return Ok(new{popularity});
}
```

Рис. 4.2. Приклад програмного коду контролера “statistics”

До єдиного методу контролера застосовується спеціальний атрибут [HttpGet("statistics")], який вказує, який саме тип запиту оброблятиметься методом GetCarsStatistics() і повертає у відповідь клієнту певний набір даних у вигляді об’єкта JSON.

З інших особливостей проекту Web API слід зазначити вміст класу Startup. Перш за все, оскільки в даному випадку не використовуються представлення, то підключення в методі ConfigureServices() сервісів MVC, необхідних для роботи контролерів Web API, здійснюється за допомогою методу services.AddControllers()

Під час використання маршрутизації у методі Configure() не визначається жодних маршрутів. Разом з цим просто викликається метод endpoints.MapControllers(), який дозволяє зіставляти запити з контролерами. Деякі маршрути задаються локально з допомогою атрибутів контролера.

Також важливим моментом є з’єднання з БД. В проекті це реалізовано за допомогою ORM Entity Framework. Папка Models відповідає за моделі таблиці, до прикладу візьмем модель таблиці Car (рис. 4.3).

```
public class Car
{
    1 usage
    public int Id { get; set; }
    10 usages
    public string Manufacturer { get; set; }
    5 usages
    public string Model { get; set; }
    4 usages
    public string Description { get; set; }
    4 usages
    public int YearOfProduction { get; set; }
    6 usages
    public double AveragePrice { get; set; } // double
    4 usages
    public string Type { get; set; }
    4 usages
    public int QuantityInCountry { get; set; }
    4 usages
    public double EngineDisplacement { get; set; } // double
    4 usages
    public int EngineHorsepower { get; set; }
    2 usages
    public string PictureUrl { get; set; }
    6 usages
    public string PublicId { get; set; }
}
```

Рис. 4.3. Приклад програмного коду таблиці Car

Конструктор моделей перевіряє класи, якими керує контекст і використовує набір правил або умов, щоб визначити, як ці класи та відносини описують модель, і як ця модель має відображатися у БД (Додаток А).

Також важливим моментом є реалізація авторизації користувачів. Аналогічно до попередньої моделі, створена модель користувача (рис. 4.4).

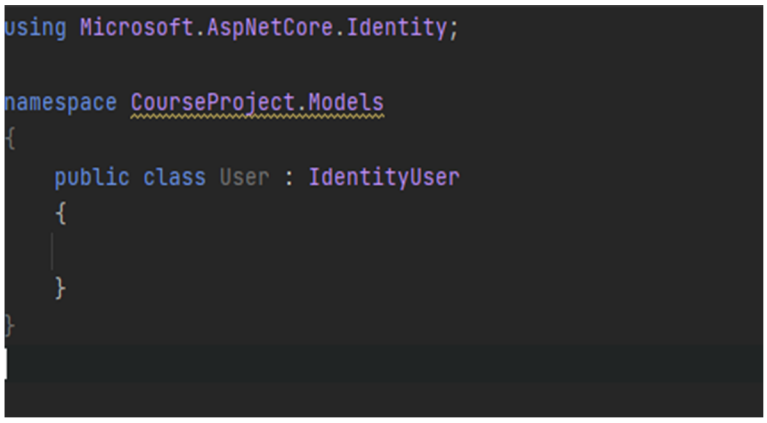

Рис. 4.4. Приклад програмного коду моделі користувача

В даній моделі не створено жодного поля, але вона наслідує клас IdentityUser. Ця модель використовує простір імен Microsoft.AspNetCore.Identity, в якому вже визначені всі потрібні значення для автентифікації. В даному випадку, використовуються лише Email та Password.

Клас контексту CarsContext наслідує клас IdentityDbContext<User>, що дозволяє визначати ролі користувачів (рис. 4.5).

```
public class CarsContext : IdentityDbContext<User>
{
    public CarsContext(DbContextOptions options) : base(options)
    {

    }

    10 usages
    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        builder.Entity<IdentityRole>() // EntityTypeBuilder<IdentityRole>
         .HasData(
             new IdentityRole { Name = "Member", NormalizedName = "MEMBER" },
             new IdentityRole { Name = "Admin", NormalizedName = "ADMIN" }
         );
    }
}
```

Рис. 4.5. Приклад програмного коду ролей користувачів.

Саме метод OnModelCreating() виконує вищезгадані дії. Після чого в клас Startup додаємо залежність AddIdentityCore<User> (рис. 4.6).

```
services.AddIdentityCore<User>( setupAction: opt :IdentityOptions =>
{
    opt.User.RequireUniqueEmail = true;
})
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<CarsContext>();
    services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(opt :JwtBearerOptions =>
    {
        opt.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8 // Encoding
                .GetBytes(Configuration["JWTSettings:TokenKey"]))
        };
    });
```

Рис. 4.6. Приклад програмного коду додавання залежності

Метод AddEntityFrameworkStores() встановлює тип сховища, яке буде використовуватись в бібліотеці Identity для зберігання даних. Як тип сховища тут вказується клас контексту даних [8].

Сервіс AddAuthentication додає у проект авторизацію та виконує валідацію JWT токену, про який було згадано раніше.

4.2. Реалізація клієнтської частини системи

Клієнтська частина системи розроблена з використання JavaScript бібліотек React, Material UI та Redux Toolkit (Додаток Б). Головна сторінка системи зображена на на рисунку 4.7.

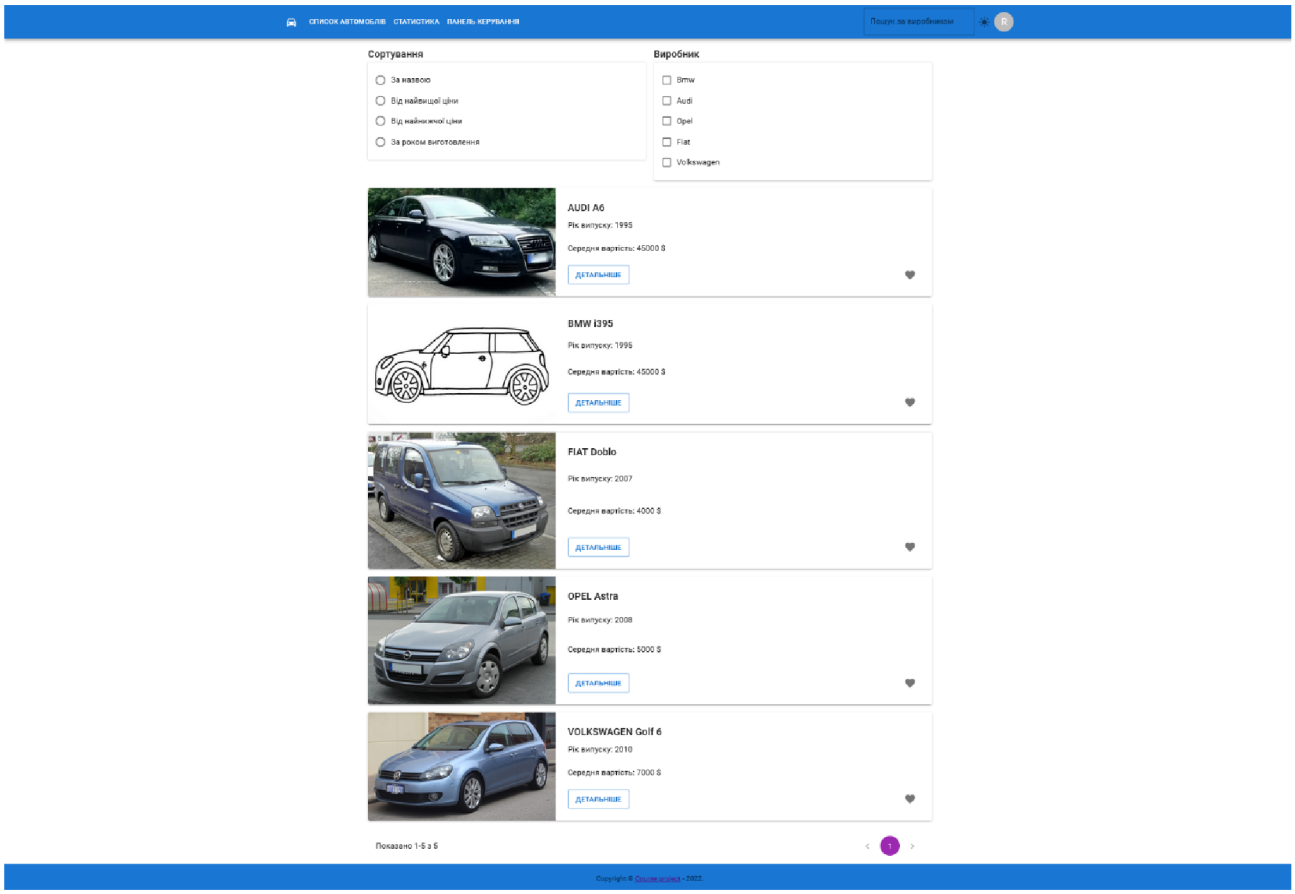

Рис. 4.7. Вигляд головної сторінки системи

На даній сторінці користувачі можуть переглянути інформацію про кожен автомобіль даного порталу, попередньо відсортувавши її за його назвою, найнижчою і найвищою ціною, за зростанням і спаданням року випуску та брендом. На цій же сторінці користувачі також мають можливість додавати моделі в списки улюблених, натиснувши на іконку у вигляді сердечка.

Програмний код цієї сторінки представлено на рис. 4.8.

```
return (
  <ThemeProvider theme={theme}>
    <Router>
      <Box
        sx={{
          display: "flex",
          flexDirection: "column",
          minHeight: "100vh",
        }}
      >
        <Header mode={darkMode} setMode={setDarkMode} />
        <Routes>
          <Route path="/" element={<HomePage />} />
          <Route path="/statistics" element={<StatisticsPage />} />
          <Route path="/account" element={<PrivateRoute />}>
            <Route path="/account" element={<Profile />} />
          </Route>
          <Route path="/account/vehicleList" element={<PrivateRoute />}>
            <Route path="/account/vehicleList" element={<VehicleItems />} />
          </Route>
          <Route path="/vehicle/:vehicleId" element={<Vehicle />} />
          <Route path="/sign-in" element={<SignIn />} />
          <Route path="/sign-up" element={<SignUp />} />
        </Routes>
        <Footer />
      </Box>
    </Router>
  </ThemeProvider>
);
}
```

Рис. 4.8. Програмний код головної сторінки системи

Батьківський елемент <ThemeProvider> бібліотеки Material UI дозволяє застосувати тему проекту. Елемент <Router> пакета React Router відповідає за сторінку, яка повинна відображатись [9].

Перехід на сторінку опису конкретного автомобіля (рис. 4.9) виконується натисканням кнопки “Детальніше” на головній сторінці системи. На цій сторінці користувачі мають можливість переглянути опис, технічні характеристики та висловити власну думку про конкретну модель.

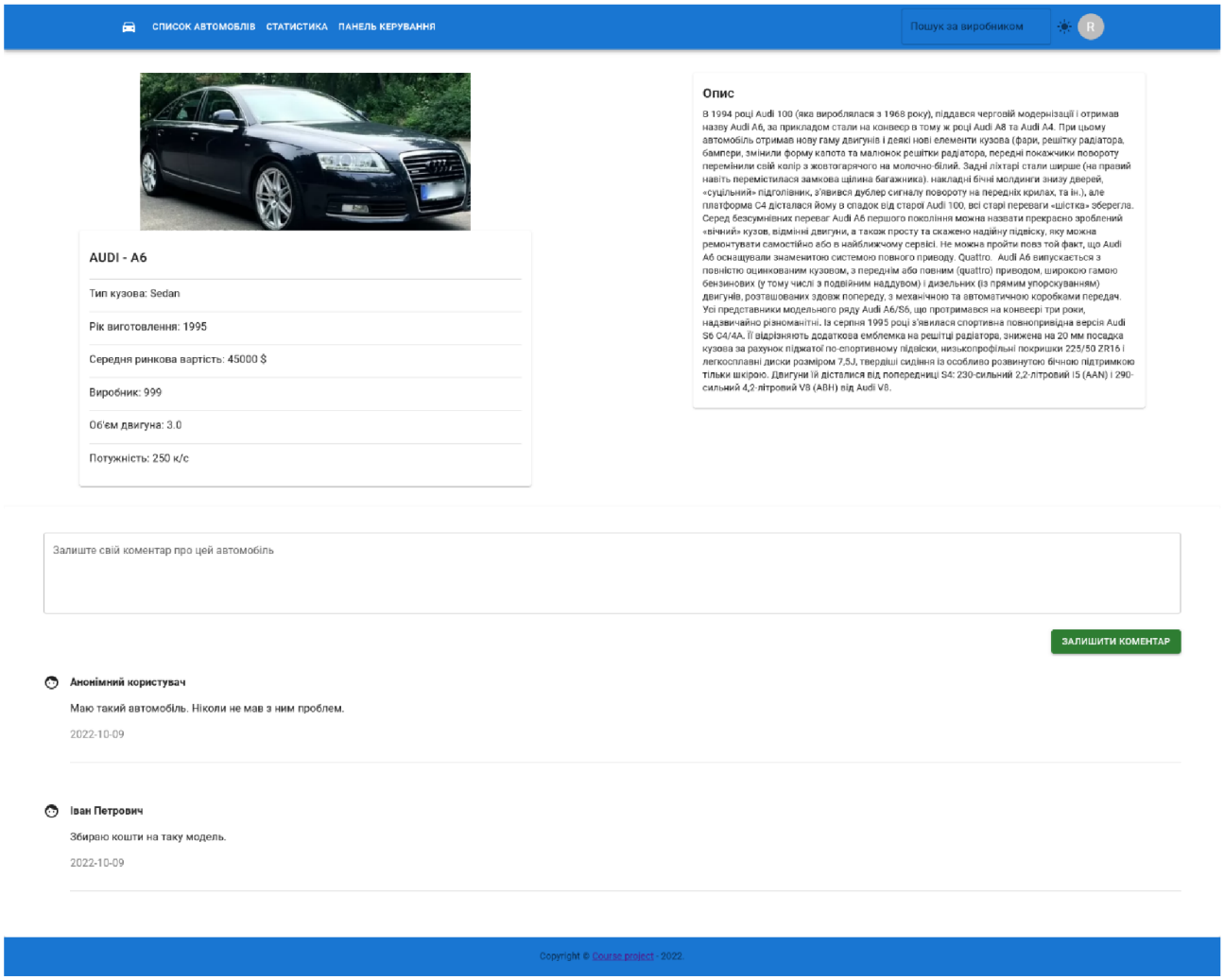

Рис. 4.9. Сторінка опису конкретного автомобіля

Для переходу на сторінку створення акаунта (рис. 4.10) та авторизації (рис. 4.11) потрібно в правому верхньому куті сторінки вибрати іконку аватару. Після цього стануть доступними опції “Увійти” та “Зареєструватися”.

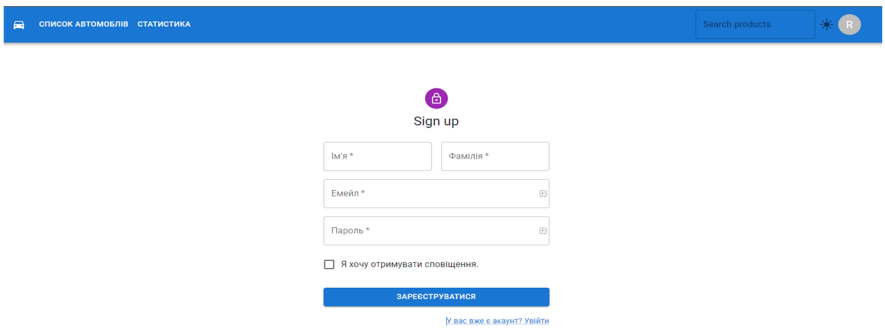

Рис. 4.10. Приклад сторінки реєстрації

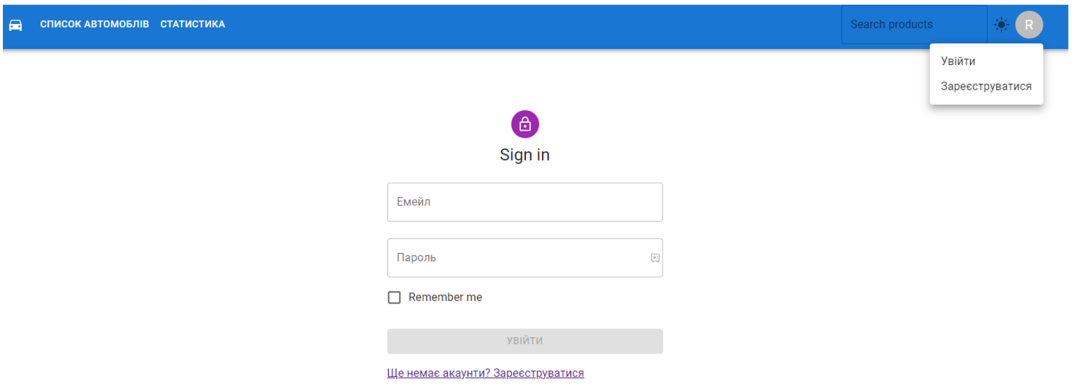

Рис. 4.11. Приклад сторінки авторизації

Адміністратор може переглядати та редагувати описи автомобілів з панелі керування вибором елемента меню “Панель керування” (рис. 4.12).

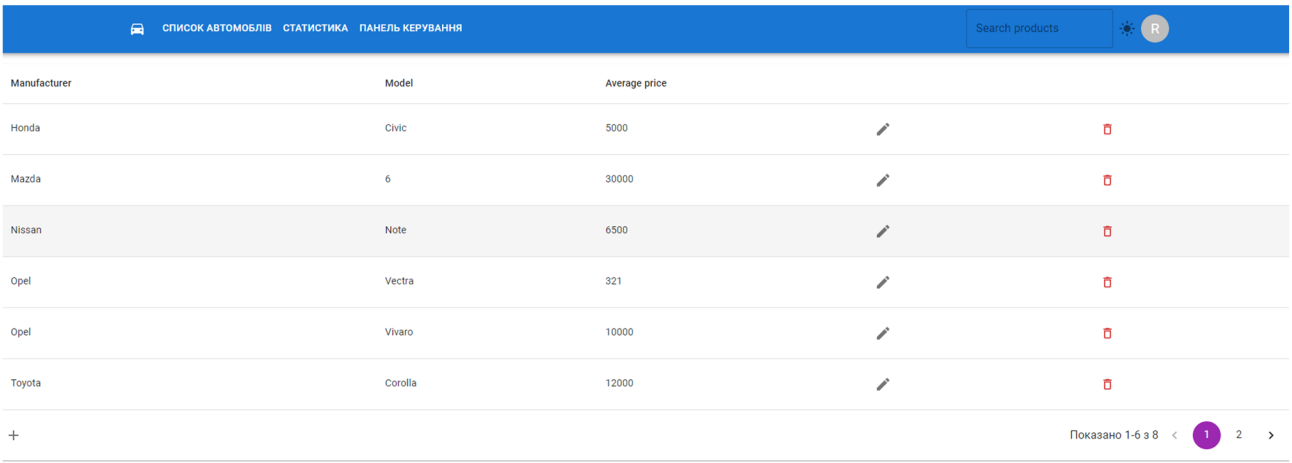

Рис. 4.12. Панель керування

Редагування опису автомобіля здійснюється натисканням на іконку у вигляді олівця, яка відкриває нове вікно (рис. 4.13).

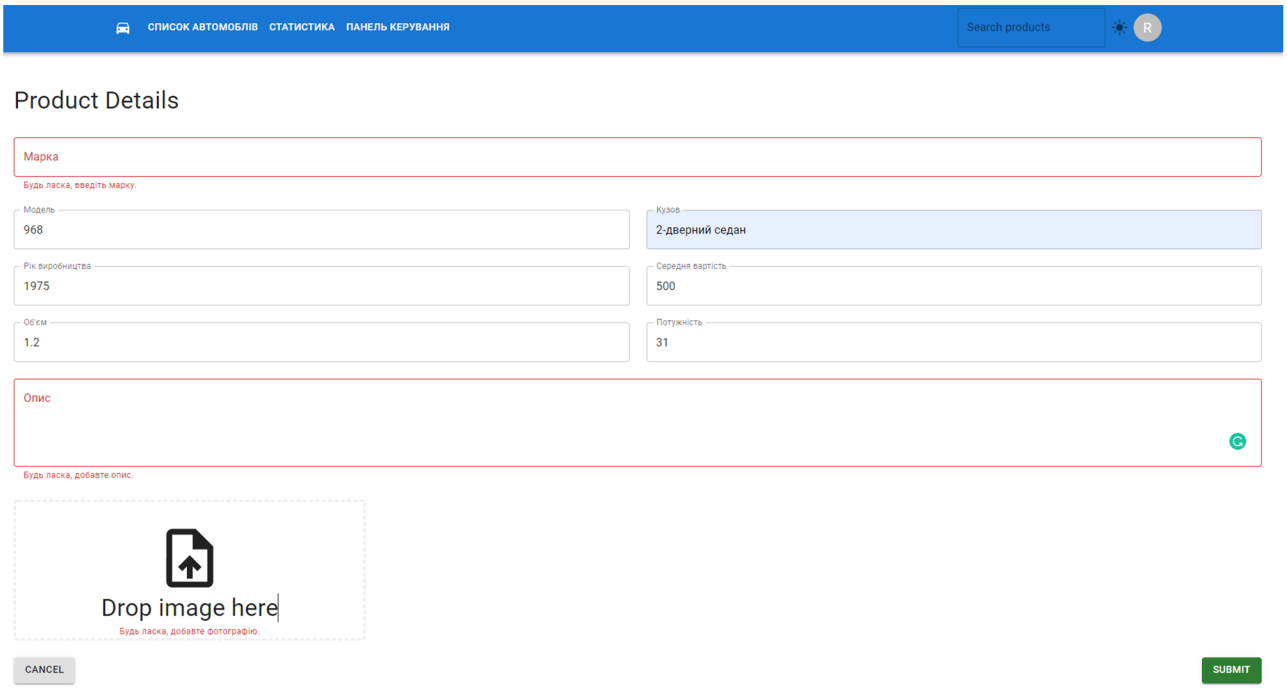

Рис. 4.13. Вікно редагування опису автомобіля

В цьому вікні передбачено валідацію усіх полів введення.

Важливою функціональною особливістю розробленої системи є можливість перегляду популярності представлених тут автомобілів (рис. 4.14).

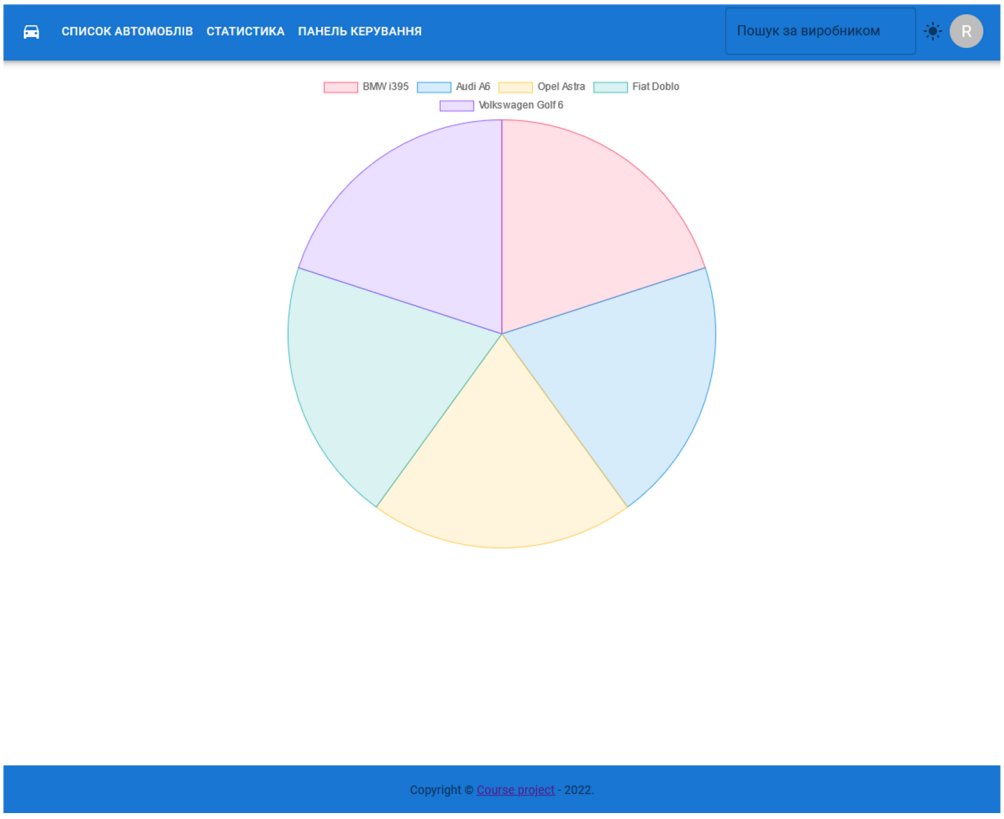

Рис. 4.14. Вікно сторінки популярності автомобілів

Висновки. Реалізовано клієнтську та серверну частини системи, описано процес її роботи та функціональність.

РОЗДІЛ 5 РОЗРОБЛЕННЯ СТАРТАП-ПРОЕКТУ

5.1 Опис ідеї проєкту

Таблиця 5.1. Інформаційна карта проєкту

Назва номінації	Програмна система
Назва проєкту	Інтелектуальна системи для підбору, порівняння та аналізу автомобільного ринку в Україні
Назва ВНЗ, спеціальності	НЛТУ, кафедра інформаційних технологій, “Комп’ютерні науки”
Прізвище, ім’я, по-батькові автора	Гарасимчук Максим Васильович
Цілі і задачі проєкту	Спроектовати та розробити інтелектуальну систему для підбору, порівняння та аналізу автомобільного ринку в Україні
Короткий зміст проєкту	Інтелектуальна система дозволяє користувачам переглядати інформацію про доступні моделі автомобілів та їх популярність на українському ринку.
Цільова аудиторія	Автолюбителі та люди хто підбирають собі новий автомобіль.
Терміни виконання проекту	4 місяці
Бюджет проекту	10 000 грн.

Суть проєкту полягає в тому, щоб спроєктувати та розробити інтелектуальну систему, за допомогою якої користувачі можуть легко переглядати інформацію про доступні автомобілі, а адміністратор модерувати завантажені дані. Метою даного проєкту є демонстрація можливостей, які надає платформа .NET для розробки інтелектуальних систем.

5.2 Розроблення ринкової стратегії проєкту

Розроблення ринкової стратегії передбачає визначення стратегії охоплення ринку (табл. 5.2).

Таблиця 5.2. Опис цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи	Інтенсивність конкуренції на ринку	Простота виходу на ринок
1.	Автолюбителі	Висока	Високий	Конкуренція на ринку середня	Розробка інтелектуальної системи є досить затратним процесом, проте ймовірність використання цільовою аудиторією є доволі високою
2.	Люди, що шукають новий автомобіль	Середня	Середній		

Для двох груп споживачів є важливою легкість та зручність отримування правдивої інформації, переглянувши коротку інформацію на стартовому екрані, користувач може перейти до повної інформації, вибравши потрібну модель, одразу після внесення змін в БД адміністратором системи, користувачі отримують оновлену інформацію на своїх пристроях.

5.3 Розробка маркетингової програми стартап-проекту

Таблиця 5.3. Визначення ключових переваг концепції потенційного продукту.

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Подання текстової та візуальної інформації про автомобілі та їх технічні характеристики	Використовувати систему можна з будь-якого девайсу, який дозволяє переглядати веб-системи.	Уникнення можливості продажу автомобілів на платформі, що зменшує відсоток неправдивої інформації залишеної власниками.
2	Простий та зрозумілий інтерфейс	Інтерфейс дозволяє легко знайти потрібну інформацію в декілька кліків.	Можна використовувати для подання інформації для іншої техніки

Після незначних змін в програмному коді, інтелектуальну систему можна використовувати для відображення інформації про інші види техніки, які можуть бути цікавими кінцевому користувачу.

Таблиця 5.4. Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
Невідомо	Невідомо	200$	100-200$

Висновки. Розроблена інтелектуальна система демонструє переваги платформи .NET для розробки інтелектуальних систем. Інтелектуальна система може використовуватися автолюбителями, чи просто людьми для пошуку нового

автомобіля та отримання інформації про доступні та популярні моделі. Після незначних маніпуляцій в програмному коді, систему можливо використовувати для відображення іншої техніки, що є цікавою кінцевому користувачу.

ВИСНОВКИ

В результаті виконання дипломної роботи було проаналізовано можливості, які надає платформа .NET для розробки інтелектуальних систем.

Використовуючи середовище Rider, мову програмування C#, архітектуру React та СКБД SQLite спроєктовано та розроблено інтелектуальну систему для підбору, порівняння та аналізу автомобільного ринку в Україні, що дозволяє користувачам переглядати інформацію про моделі автомобілів і залишати свої коментарі та вподобання щодо них. Розроблена система дозволяє адміністратору модерувати завантажену інформацію, що зберігається в БД.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Andrew Troelsen and Phillip Japikse, Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming, April 26 2021, pp. 1331.
2. Adam Freeman, Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications 9th ed. Edition, February 24 2022, pp. 1253.
3. Boris Cherny, Programming TypeScript: Making Your JavaScript Applications Scale, Apr 25 2019, pp 332. (дата звернення 16.11.2022). – Назва з екрана.
4. Dan Vanderkam, Effective TypeScript: 62 Specific Ways to Improve Your TypeScript, Oct 17 2019, pp. 264.
5. Stoyan Stefanov, React: Up & Running: Building Web Applications, July 14 2016, pp. 305.
6. A Relational Database Overview [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://docs.oracle.com/javase/tutorial/jdbc/overview/ database.html (дата звернення 08.11.2022). – Назва з екрана.
7. Cloudinary [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://cloudinary.com/ (дата звернення 10.11.2022). – Назва з екрана.
8. Entity Framework documentation [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://docs.microsoft.com/en-us/ef/
9. Material UI [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://material-ui.com/ (дата звернення 13.11.2022). – Назва з екрана.
10. ReactJS [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://reactjs.org/ (дата звернення 15.11.2022). – Назва з екрана.
11. .NET [Електронний ресурс] : [Веб-сайт]. – Режим доступу: https://dotnet.microsoft.com/en-us/ (дата звернення 16.11.2022). – Назва з екрана.

ДОДАТКИ

ДОДАТОК А

База даних

```
BEGIN TRANSACTION;
CREATE TABLE IF NOT EXISTS "__EFMigrationsHistory" (
    "MigrationId"    TEXT NOT NULL,
    "ProductVersion"    TEXT NOT NULL,
    CONSTRAINT "PK___EFMigrationsHistory" PRIMARY KEY("MigrationId")
);
CREATE TABLE IF NOT EXISTS "AspNetRoles" (
    "Id"    TEXT NOT NULL,
    "Name"  TEXT,
    "NormalizedName"    TEXT,
    "ConcurrencyStamp"  TEXT,
    CONSTRAINT "PK_AspNetRoles" PRIMARY KEY("Id")
);
CREATE TABLE IF NOT EXISTS "AspNetUsers" (
    "Id"    TEXT NOT NULL,
    "UserName"  TEXT,
    "NormalizedUserName"    TEXT,
    "Email" TEXT,
    "NormalizedEmail"   TEXT,
    "EmailConfirmed"    INTEGER NOT NULL,
    "PasswordHash"  TEXT,
    "SecurityStamp" TEXT,
    "ConcurrencyStamp"  TEXT,
    "PhoneNumber"   TEXT,
    "PhoneNumberConfirmed"  INTEGER NOT NULL,
    "TwoFactorEnabled"  INTEGER NOT NULL,
    "LockoutEnd"    TEXT,
    "LockoutEnabled"    INTEGER NOT NULL,
    "AccessFailedCount" INTEGER NOT NULL,
    CONSTRAINT "PK_AspNetUsers" PRIMARY KEY("Id")
);
CREATE TABLE IF NOT EXISTS "AspNetRoleClaims" (
    "Id"    INTEGER NOT NULL,
    "RoleId"    TEXT NOT NULL,
    "ClaimType" TEXT,
    "ClaimValue"    TEXT,
    CONSTRAINT "FK_AspNetRoleClaims_AspNetRoles_RoleId" FOREIGN KEY("RoleId") REFERENCES "AspNetRoles"("Id") ON
DELETE CASCADE,
    CONSTRAINT "PK_AspNetRoleClaims" PRIMARY KEY("Id" AUTOINCREMENT)
);
CREATE TABLE IF NOT EXISTS "AspNetUserClaims" (
```

```
    "Id"    INTEGER NOT NULL,
    "UserId"    TEXT NOT NULL,
    "ClaimType" TEXT,
    "ClaimValue"    TEXT,
    CONSTRAINT "FK_AspNetUserClaims_AspNetUsers_UserId" FOREIGN KEY("UserId") REFERENCES "AspNetUsers"("Id") ON
DELETE CASCADE,
    CONSTRAINT "PK_AspNetUserClaims" PRIMARY KEY("Id" AUTOINCREMENT)
);
CREATE TABLE IF NOT EXISTS "AspNetUserLogins" (
    "LoginProvider" TEXT NOT NULL,
    "ProviderKey"   TEXT NOT NULL,
    "ProviderDisplayName"   TEXT,
    "UserId"    TEXT NOT NULL,
    CONSTRAINT "FK_AspNetUserLogins_AspNetUsers_UserId" FOREIGN KEY("UserId") REFERENCES "AspNetUsers"("Id") ON
DELETE CASCADE,
    CONSTRAINT "PK_AspNetUserLogins" PRIMARY KEY("LoginProvider","ProviderKey")
);
CREATE TABLE IF NOT EXISTS "AspNetUserRoles" (
    "UserId"    TEXT NOT NULL,
    "RoleId"    TEXT NOT NULL,
    CONSTRAINT "FK_AspNetUserRoles_AspNetRoles_RoleId" FOREIGN KEY("RoleId") REFERENCES "AspNetRoles"("Id") ON
DELETE CASCADE,
    CONSTRAINT "FK_AspNetUserRoles_AspNetUsers_UserId" FOREIGN KEY("UserId") REFERENCES "AspNetUsers"("Id") ON
DELETE CASCADE,
    CONSTRAINT "PK_AspNetUserRoles" PRIMARY KEY("UserId","RoleId")
);
CREATE TABLE IF NOT EXISTS "AspNetUserTokens" (
    "UserId"    TEXT NOT NULL,
    "LoginProvider" TEXT NOT NULL,
    "Name"  TEXT NOT NULL,
    "Value" TEXT,
    CONSTRAINT "FK_AspNetUserTokens_AspNetUsers_UserId" FOREIGN KEY("UserId") REFERENCES "AspNetUsers"("Id")
ON DELETE CASCADE,
    CONSTRAINT "PK_AspNetUserTokens" PRIMARY KEY("UserId","LoginProvider","Name")
);
CREATE TABLE IF NOT EXISTS "FavouritesCars" (
    "Id"    INTEGER NOT NULL,
    "CarId" INTEGER,
    "UserId"    TEXT,
    "CarModel"  TEXT,
    "Manufacturer"  TEXT,
    CONSTRAINT "FK_FavouritesCars_AspNetUsers_UserId" FOREIGN KEY("UserId") REFERENCES "AspNetUsers"("Id"),
    CONSTRAINT "FK_FavouritesCars_Cars_CarId" FOREIGN KEY("CarId") REFERENCES "Cars"("Id"),
    CONSTRAINT "PK_FavouritesCars" PRIMARY KEY("Id" AUTOINCREMENT)
);
CREATE TABLE IF NOT EXISTS "Comments" (
```

```
    "Id"    INTEGER NOT NULL,
    "Body"  TEXT,
    "CarId" INTEGER,
    "CreatedAt" TEXT NOT NULL,
    "UserId"    TEXT,
    "UserName"  TEXT,
    CONSTRAINT "FK_Comments_AspNetUsers_UserId" FOREIGN KEY("UserId") REFERENCES "AspNetUsers"("Id"),
    CONSTRAINT "PK_Comments" PRIMARY KEY("Id" AUTOINCREMENT),
    CONSTRAINT "FK_Comments_Cars_CarId" FOREIGN KEY("CarId") REFERENCES "Cars"("Id")
);
CREATE TABLE IF NOT EXISTS "Cars" (
    "Id"    INTEGER NOT NULL,
    "AveragePrice"  REAL NOT NULL,
    "Description"   TEXT,
    "EngineDisplacement"    TEXT,
    "EngineHorsepower"  INTEGER NOT NULL,
    "Manufacturer"  TEXT,
    "Model" TEXT,
    "PictureUrl"    TEXT,
    "PublicId"  TEXT,
    "QuantityInCountry" INTEGER NOT NULL,
    "Type"  TEXT,
    "YearOfProduction"  INTEGER NOT NULL,
    CONSTRAINT "PK_Cars" PRIMARY KEY("Id" AUTOINCREMENT)
);
INSERT INTO "__EFMigrationsHistory" VALUES ('20220330114817_InitialCreate','6.0.4');
INSERT INTO "__EFMigrationsHistory" VALUES ('20220407111429_IdentityAdded','6.0.4');
INSERT INTO "__EFMigrationsHistory" VALUES ('20220505151402_PublicId','6.0.4');
INSERT INTO "__EFMigrationsHistory" VALUES ('20220926064951_CommentEntityAdded','6.0.4');
INSERT INTO "__EFMigrationsHistory" VALUES ('20220929171455_FavouritesEntityAdded','6.0.4');
INSERT INTO "__EFMigrationsHistory" VALUES ('20221030224123_AddCarModelToFavouritesDB','6.0.4');
INSERT INTO "__EFMigrationsHistory" VALUES ('20221108175636_AddManufacturer','6.0.4');
INSERT INTO "__EFMigrationsHistory" VALUES ('20221108183521_AddCarId','6.0.4');
INSERT INTO "__EFMigrationsHistory" VALUES ('20221109200829_AddUserToComments','6.0.4');
INSERT INTO "AspNetRoles" VALUES ('90062e10-8a3e-461b-9e43-95db66e3ea21','Member','MEMBER','9fc6d5df-f3f9-4dbb-a705-1bf0d5538916');
INSERT INTO "AspNetRoles" VALUES ('96df4dc4-dbcd-4d00-bdef-3285892207a7','Admin','ADMIN','f001ca6e-56fb-4c86-abd8-a74b7979b0d9');
INSERT INTO "AspNetUsers" VALUES ('c52115b9-b5d5-435e-ba1c-8d5d4467ee9f','bob','BOB','bob@test.com','BOB@TEST.COM',0,'AQAAAAEAACcQAAAAEB04cROpF+wyxUNNweZopiNQ1BIi5iExpIhj1w4fuK5uVVG9OAwv8UK6fynePKuvJg==','WOBZWKDYAXZZJB4EH4WUKJUWLCYMSAMV','1d4793dd-7f83-491d-9bf0-30bf6a10d8c7',NULL,0,0,NULL,1,0);
INSERT INTO "AspNetUsers" VALUES ('d3867778-23d8-4a1d-8e00-e21407beb564','admin','ADMIN','admin@test.com','ADMIN@TEST.COM',0,'AQAAAAEAACcQAAAAEP7xBQagfINzgFcgEsVt8QR6X0saRjlH95NPeaXJZGU7OrwNmlv3TBqX0RetmPWAew==','75ESLGCNGYI67OOXZDIKG5KNALEMWWVM','3f1a2755-3aad-4042-95c1-19fa3082eedf',NULL,0,0,NULL,1,0);
```

INSERT INTO "AspNetUsers" VALUES ('d0e80c9b-de55-4842-b85a-66610eefb90f','Ivan','IVAN','Ivan@gmail.com','IVAN@GMAIL.COM',0,'AQAAAAEAACcQAAAAEF2iNp+bTJmzArN/oEOMkcgec0mmLwUPbQ4CRIyNX3jy+OflZB1R016Fc3cTht0twQ==','UY7LCFVIZXNHH7RF5DIOQT3VD6BBMGGE','25b76b3f-6a2a-4928-bc9e-a619af14cde6',NULL,0,0,NULL,1,0);
INSERT INTO "AspNetUsers" VALUES ('b75cd301-3e16-4dda-b4ff-5ae27ecdc084','Petro','PETRO','Petro@gmail.com','PETRO@GMAIL.COM',0,'AQAAAAEAACcQAAAAECFmsn8A7mbBAu36YYSw4UPs04p5FIpCpDNnXC15WE0MnRILT4fov3Juy54WnD9Img==','4GTTBSRWIXEXHFXP4DE252NAJYQ2WFVN','fc50cd8a-f891-47b4-88f5-e2214e21fb4d',NULL,0,0,NULL,1,0);
INSERT INTO "AspNetUserRoles" VALUES ('d0e80c9b-de55-4842-b85a-66610eefb90f','90062e10-8a3e-461b-9e43-95db66e3ea21');
INSERT INTO "AspNetUserRoles" VALUES ('b75cd301-3e16-4dda-b4ff-5ae27ecdc084','90062e10-8a3e-461b-9e43-95db66e3ea21');
INSERT INTO "FavouritesCars" VALUES (140,2,'0080ccef-716e-4871-b0e7-7bad39d00b2f','A6','AUDI');
INSERT INTO "FavouritesCars" VALUES (141,1,'0080ccef-716e-4871-b0e7-7bad39d00b2f','i395','BMW');
INSERT INTO "FavouritesCars" VALUES (142,8,'0080ccef-716e-4871-b0e7-7bad39d00b2f','Doblo','FIAT');
INSERT INTO "FavouritesCars" VALUES (143,6,'0080ccef-716e-4871-b0e7-7bad39d00b2f','Astra','OPEL');
INSERT INTO "FavouritesCars" VALUES (144,9,'0080ccef-716e-4871-b0e7-7bad39d00b2f','Golf 6','VOLKSWAGEN ');
INSERT INTO "FavouritesCars" VALUES (146,6,'d3867778-23d8-4a1d-8e00-e21407beb564','Astra','OPEL');
INSERT INTO "FavouritesCars" VALUES (147,8,'d3867778-23d8-4a1d-8e00-e21407beb564','Doblo','FIAT');
INSERT INTO "FavouritesCars" VALUES (148,1,'d3867778-23d8-4a1d-8e00-e21407beb564','i395','BMW');
INSERT INTO "FavouritesCars" VALUES (149,10,'0080ccef-716e-4871-b0e7-7bad39d00b2f','Vivaro','OPEL');
INSERT INTO "FavouritesCars" VALUES (150,10,'d3867778-23d8-4a1d-8e00-e21407beb564','Vivaro','OPEL');
INSERT INTO "FavouritesCars" VALUES (151,14,'d3867778-23d8-4a1d-8e00-e21407beb564','e-Golf','VOLKSWAGEN ');
INSERT INTO "FavouritesCars" VALUES (152,12,'c52115b9-b5d5-435e-ba1c-8d5d4467ee9f','Fusion','FORD');
INSERT INTO "FavouritesCars" VALUES (153,19,'c52115b9-b5d5-435e-ba1c-8d5d4467ee9f','9','LADA');
INSERT INTO "FavouritesCars" VALUES (154,19,'d3867778-23d8-4a1d-8e00-e21407beb564','9','LADA');
INSERT INTO "FavouritesCars" VALUES (155,19,'b75cd301-3e16-4dda-b4ff-5ae27ecdc084','9','LADA');
INSERT INTO "FavouritesCars" VALUES (156,16,'d3867778-23d8-4a1d-8e00-e21407beb564','428','BMW');
INSERT INTO "Comments" VALUES (15,'Це просто космос!
',11,'2022-11-09 20:17:50.4340065',NULL,'Ivan');
INSERT INTO "Comments" VALUES (16,'Економна! Ніколи не ламається
',14,'2022-11-10 14:01:07.3742874',NULL,'bob');
INSERT INTO "Comments" VALUES (17,'Ракета',2,'2022-11-10 14:02:45.5485267',NULL,'Petro');
INSERT INTO "Comments" VALUES (18,'Продається?
',2,'2022-11-10 14:02:54.5327993',NULL,'Petro');
INSERT INTO "Cars" VALUES (2,10000.0,'В 1994 році Audi 100 (яка вироблялася з 1968 року), піддався черговій модернізації і отримав назву Audi A6, за прикладом стали на конвеєр в тому ж році Audi A8 та Audi A4. При цьому автомобіль отримав нову гаму двигунів і деякі нові елементи кузова (фари, решітку радіатора, бампери, змінили форму капота та малюнок решітки радіатора, передні покажчики повороту перемінили свій колір з жовтогарячого на молочно-білий. Задні ліхтарі стали ширше (на правий навіть перемістилася замкова щілина багажника). накладні бічні молдинги знизу дверей, «суцільний» підголівник, з"явився дублер сигналу повороту на передніх крилах, та ін.), але платформа C4 дісталася йому в спадок від старої Audi 100, всі старі переваги «шістка» зберегла. Серед безсумнівних переваг Audi A6 першого покоління можна назвати прекрасно зроблений «вічний» кузов, відмінні двигуни, а також просту та скажено надійну підвіску, яку можна ремонтувати самостійно або в найближчому сервісі. Не можна пройти повз той факт, що Audi A6 оснащували знаменитою системою повного приводу. Quattro.

Audi A6 випускається з повністю оцинкованим кузовом, з переднім або повним (quattro) приводом, широкою гамою бензинових (у тому числі з подвійним наддувом) і дизельних (із прямим упорскуванням) двигунів, розташованих здовж попереду, з механічною та автоматичною коробками передач.

Усі представники модельного ряду Audi A6/S6, що протримався на конвеєрі три роки, надзвичайно різноманітні. Із серпня 1995 році з"явилася спортивна повнопривідна версія Audi S6 C4/4A. Її відрізняють додаткова емблемка на решітці радіатора, знижена на 20 мм посадка кузова за рахунок піджатої по-спортивному підвіски, низькопрофільні покришки 225/50 ZR16 і легкосплавні диски розміром 7,5J, твердіші сидіння із особливо розвинутою бічною підтримкою тільки шкірою. Двигуни їй дісталися від попередниці S4: 230-сильний 2,2-літровий I5 (AAN) і 290-сильний 4,2-літровий V8 (ABH) від Audi V8.','3.0',250,'AUDI','A6','https://res.cloudinary.com/daclkkup2/image/upload/v1665058532/hrmxpn6cj8f6xcmpvbo4.webp','hrmxpn6cj8f6xcmpvbo4',999,'Sedan',1995);

INSERT INTO "Cars" VALUES (6,5000.0,'Опель Астра','1.5',135,'OPEL','Astra','https://res.cloudinary.com/daclkkup2/image/upload/v1664890703/bjed2jgzorymsie8zkbw.jpg','bjed2jgzorymsie8zkbw',0,'Хетчбек',2008);

INSERT INTO "Cars" VALUES (8,4000.0,'Fiat Doblò (Фіат Добло) — компактний фургон, який представляє італійський автовиробник Fiat з 2000 року. ','1.5',135,'FIAT','Doblo','https://res.cloudinary.com/daclkkup2/image/upload/v1665060268/kulootbsmpcz07g2wzat.jpg','kulootbsmpcz07g2wzat',0,'Фургон',2007);

INSERT INTO "Cars" VALUES (9,7000.0,'Volkswagen Golf 6 — шосте покоління сімейства Volkswagen Golf, який був представлений на Паризькому автосалоні в жовтні 2008 року. Автомобіль зібраний на платформі Volkswagen A5 (PQ35), як і його попередник Volkswagen Golf 5. ','2.0',150,'VOLKSWAGEN ','Golf 6','https://res.cloudinary.com/daclkkup2/image/upload/v1665060367/lucimd8v5szs1edkfzaq.jpg','lucimd8v5szs1edkfzaq',0,'Хетчбек',2010);

INSERT INTO "Cars" VALUES (10,8000.0,'Працюйте ефективно та з майстерністю справжнього професіонала! Можливість складати спинки сидінь та підключати смартфони додають комфорту та зручності, а чималий вантажний простір та неабияке корисне навантаження сприяють продуктивності. ','1.5',150,'OPEL','Vivaro','https://res.cloudinary.com/daclkkup2/image/upload/v1668023826/rcyiodeempduv9b6rine.jpg','rcyiodeempduv9b6rine',0,'Фургон',2009);

INSERT INTO "Cars" VALUES (11,15000.0,'Люксовий компактний кросовер німецької компанії BMW. X3 першого покоління розроблений і випущений спільно з компанією Magna Steyr на заводі в Граце, Австрія. Друге покоління кросовера було представлено в липні 2010 року, виробництво автомобіля почалося 1 вересня 2010 року. За станом на липень 2010 року у всьому світі було продано близько 600 000 екземплярів BMW X3.','3',380,'BWM','X3','https://res.cloudinary.com/daclkkup2/image/upload/v1668025045/alcqvj7aakmiv9mitfex.webp','alcqvj7aakmiv9mitfex',0,'Кросовер',2014);

INSERT INTO "Cars" VALUES (12,11000.0,'Американський Ford Fusion — середньорозмірний автомобіль, виготовлений компанією Ford Motor Company для ринків Північної і, частково, Південної Америки з 2005 року.','2.5',175,'FORD','Fusion','https://res.cloudinary.com/daclkkup2/image/upload/v1668025482/hlalgaupb0uyf4lvjt1m.jpg','hlalgaupb0uyf4lvjt1m',0,'Седан',2015);

INSERT INTO "Cars" VALUES (13,1300.0,'Volkswagen Passat B7 — сьоме покоління автомобілів сімейства Volkswagen Passat, яке вперше було представлене 2 жовтня 2010 року на Паризькому автосалоні. В Китаї модель називається Volkswagen Magotan. ','1.8',180,'VOLKSWAGEN ','Passat','https://res.cloudinary.com/daclkkup2/image/upload/v1668025697/g2pgy5yydgnuob3iookh.webp','g2pgy5yydgnuob3iookh',0,'Седан',2014);

INSERT INTO "Cars" VALUES (14,15000.0,'Volkswagen e-Golf (Фольксваген е-Гольф) – передньопривідний хетчбек класу «С». Оновлена електрична версія моделі VW Golf. Презентація автомобіля пройшла на автосалоні в Лос-Анджелесе в листопаді 2016 року.','0',115,'VOLKSWAGEN ','e-

Golf','https://res.cloudinary.com/daclkkup2/image/upload/v1668025811/equ0vxcxtj34z3nnoapm.webp','equ0vxcxtj34z3nnoapm',0,'Хетчбек',2015);
INSERT INTO "Cars" VALUES (16,17000.0,'BMW 428i 2015 року представляє собою 2-дверний 4-місний автомобіль з кузовом купе, попередниками якого є седани 3 Серії. Основним конкурентом BMW 428 вважається Toyota Camry[1].

BMW 428i оснащується 2,0-літровим 4-циліндровим двигуном, потужністю 240 кінських сил. Час розгону до 100 км/год складає 5,7 секунд. Максимальна швидкість, яку здатний розвинути автомобіль, дорівнює 250 км/год. Двигун автомобіля працює в парі з 6-ступінчастою механічною або 8-ступінчастою автоматичною коробкою передач[2].','2',258,'BMW','428','https://res.cloudinary.com/daclkkup2/image/upload/v1668026165/yjktgyvdmdz6zwsmgqde.webp','yjktgyvdmdz6zwsmgqde',0,'Купе',2015);
INSERT INTO "Cars" VALUES (17,70000.0,'Виробництво Land Cruiser першого покоління почалося в 1951 році, автомобіль представляв собою подібність цивільного джипа. Land Cruiser випускався як кабріолет, хардтоп, універсал і вантажівка (пікап). Надежність і довговічність автомобіля завоювала його величезну популярність, особливо в Австралії, на Ближньому Сході та в Африці (старі серії) завдяки неіснуючій рамі та приводу на всіх чотирьох колесах. Toyota тестує Land Cruiser в австралійській глибині, вважаючи, що це одне з найскладніших умови експлуатації, завдяки місцевому рельєфу і температурі. Основними конкурентами вважаються автомобілі Range Rover, Land Rover Discovery, Jeep Wrangler, Mitsubishi Pajero Sport і Nissan Patrol. В Японії Land Cruiser продається тільки через дилерську сеть під назвою Toyota Store.','4.5',259,'TOYOTA ','Land Cruiser','https://res.cloudinary.com/daclkkup2/image/upload/v1668026440/my813fpibxwxywd6zdiq.webp','my813fpibxwxywd6zdiq',0,'Позашляховик',2017);
INSERT INTO "Cars" VALUES (18,11000.0,'У Європі Toyota позиціонувала Auris як заміну хетчбеку Corolla, тоді як версія седана продовжувала мати табличку Corolla. Починаючи з моделі E210, табличка Auris була припинена, а замість неї використовувалася табличка Corolla, за винятком Тайваню, де до липня 2020 року зберігалася табличка Auris для версії хетчбек.

Тільки для першого покоління розкішніший Auris в Японії отримав назву Toyota Blade. Auris прийшов на зміну Allex в Японії та Corolla RunX. Toyota Австралія та Toyota Нова Зеландія протистояли пропозиціям Toyota Japan прийняти нову європейську назву Auris для Corolla.','2',150,'TOYOTA ','Auris','https://res.cloudinary.com/daclkkup2/image/upload/v1668026552/gemzm84xb1niphymslcv.webp','gemzm84xb1niphymslcv',0,'Універсал',2014);
INSERT INTO "Cars" VALUES (19,1200.0,'ВАЗ-2109 «Супутник» — радянський та російський передньопривідний автомобіль II групи малого класу з кузовом типу хетчбек. Розроблений та серійно випускався на Волзькому автомобільному заводі у 1987—2004 роках. З 2004 до кінця 2011 варіант ВАЗ-21093 збирався з машинокомплектів в Україні на заводі «ЗАЗ». Є п"ятидверною модифікацією ВАЗ-2108 в сімействі моделей Лада «Супутник».','1.2',140,'LADA','9','https://res.cloudinary.com/daclkkup2/image/upload/v1668089256/ge88qnclijhrkco6ftl1.jpg','ge88qnclijhrkco6ftl1',0,'Хетчбек',2003);
CREATE INDEX IF NOT EXISTS "IX_AspNetRoleClaims_RoleId" ON "AspNetRoleClaims" (
    "RoleId"
);
CREATE UNIQUE INDEX IF NOT EXISTS "RoleNameIndex" ON "AspNetRoles" (
    "NormalizedName"
);
CREATE INDEX IF NOT EXISTS "IX_AspNetUserClaims_UserId" ON "AspNetUserClaims" (
    "UserId"
);
CREATE INDEX IF NOT EXISTS "IX_AspNetUserLogins_UserId" ON "AspNetUserLogins" (

```
    "UserId"
);
CREATE INDEX IF NOT EXISTS "IX_AspNetUserRoles_RoleId" ON "AspNetUserRoles" (
    "RoleId"
);
CREATE INDEX IF NOT EXISTS "EmailIndex" ON "AspNetUsers" (
    "NormalizedEmail"
);
CREATE UNIQUE INDEX IF NOT EXISTS "UserNameIndex" ON "AspNetUsers" (
    "NormalizedUserName"
);
CREATE INDEX IF NOT EXISTS "IX_FavouritesCars_CarId" ON "FavouritesCars" (
    "CarId"
);
CREATE INDEX IF NOT EXISTS "IX_FavouritesCars_UserId" ON "FavouritesCars" (
    "UserId"
);
CREATE INDEX IF NOT EXISTS "IX_Comments_CarId" ON "Comments" (
    "CarId"
);
CREATE INDEX IF NOT EXISTS "IX_Comments_UserId" ON "Comments" (
    "UserId"
);
COMMIT;
```

ДОДАТОК Б

Лістинг програмного коду

Лістинг коду клієнтської частини

```
import React from "react";
import "./index.css";
import App from "./App";
import { CssBaseline } from "@mui/material";
import { store } from "./app/store";
import { Provider } from "react-redux";
import {createRoot} from "react-dom/client";

const container = document.getElementById('root');
// @ts-ignore
const root = createRoot(container);

root.render(
  <React.StrictMode>
    <CssBaseline />
    <Provider store={store}>
      <App />
    </Provider>
  </React.StrictMode>
);

import { ThemeProvider } from "@emotion/react";
import { Box, createTheme, experimental_sx as sx } from "@mui/material";
import { useEffect, useState } from "react";
import Footer from "./components/Footer";
import Header from "./components/Header";
import HomePage from "./pages/HomePage";
import Vehicle from "./pages/Vehicle";
import { green, orange } from "@mui/material/colors";
import { BrowserRouter as Router, Routes, Route } from "react-router-dom";
import Account from "./pages/Profile";
import Favorite from "./pages/Favorite";
import SignIn from "./pages/SignIn";
import SignUp from "./pages/SignUp";
import Profile from "./pages/Profile";
import PrivateRoute from "./components/PrivateRoute";
import VehicleItems from "./pages/dashboard/admin/VehicleListPage";
import {useSelector} from "react-redux";
import StatisticsPage from "./pages/StatisticsPage";

function App() {
  const [darkMode, setDarkMode] = useState(false);
  const paletteType = darkMode ? "dark" : "light";

  const theme = createTheme({
    palette: {
      mode: paletteType,
      // primary: {
      //   main: "#dd6555",
      // },
    },
  });

  return (
    <ThemeProvider theme={theme}>
      <Router>
        <Box
          sx={{
            display: "flex",
            flexDirection: "column",
            minHeight: "100vh",
          }}
```

```
      >
        <Header mode={darkMode} setMode={setDarkMode} />
        <Routes>
          <Route path="/" element={<HomePage />} />
          <Route path="/statistics" element={<StatisticsPage />} />
          <Route path="/account" element={<PrivateRoute />}>
            <Route path="/account" element={<Profile />} />
          </Route>
          <Route path="/account/vehicleList" element={<PrivateRoute />}>
            <Route path="/account/vehicleList" element={<VehicleItems />} />
          </Route>
          <Route path="/vehicle/:vehicleId" element={<Vehicle />} />
          <Route path="/sign-in" element={<SignIn />} />
          <Route path="/sign-up" element={<SignUp />} />
        </Routes>
        <Footer />
      </Box>
    </Router>
  </ThemeProvider>
 );
}

export default App;

const HomePage = () => {
 return (
   <Container maxWidth="lg">
      <FiltersCard/>
      <CarsList />

   </Container>
 );
};

export default HomePage;


const theme = createTheme();

export default function SignIn() {
 const location = useLocation();
 const navigate = useNavigate();
 const dispatch = useDispatch();
 const {
   register,
   handleSubmit,
   formState: { isSubmitting, errors, isValid },
 } = useForm({ mode: "all" });

 const submitForm = async (data: FieldValues) => {
   await dispatch(login(data));
 };

 const auth = useSelector((state: RootState) => state.auth);
 const { isLoading, isError, user } = auth;

 useEffect(() => {
   if (user) {
     navigate("/account");
   }
 }, [navigate, user]);

 // const handleSubmit = (event: React.FormEvent<HTMLFormElement>) => {
 //   event.preventDefault();

 //   dispatch(login({ username, password }));
 // };

 return (
```

```
<ThemeProvider theme={theme}>
  <Container component="main" maxWidth="xs">
    <CssBaseline />
    <Box
      sx={{
        marginTop: 8,
        display: "flex",
        flexDirection: "column",
        alignItems: "center",
      }}
    >
      <Avatar sx={{ m: 1, bgcolor: "secondary.main" }}>
        <LockOutlinedIcon />
      </Avatar>
      <Typography component="h1" variant="h5">
        Sign in
      </Typography>
      <Box
        component="form"
        onSubmit={handleSubmit(submitForm)}
        noValidate
        sx={{ mt: 1 }}
      >
        <TextField
          margin="normal"
          // required
          fullWidth
          id="username"
          label="Емейл"
          autoComplete="username"
          // autoFocus
          {...register("username", { required: "Username is required." })}
          error={!!errors.username}
          helperText={errors?.username?.message}
        />
        <TextField
          margin="normal"
          // required
          fullWidth
          label="Пароль"
          type="password"
          id="password"
          autoComplete="current-password"
          {...register("password", { required: "Password is required." })}
          error={!!errors.username}
          helperText={errors?.username?.message}
        />
        <FormControlLabel
          control={<Checkbox value="remember" color="primary" />}
          label="Remember me"
        />
        <LoadingButton
          // color={theme.palette.primary.main}
          disabled={!isValid}
          loading={isSubmitting}
          type="submit"
          fullWidth
          variant="contained"
          sx={{ mt: 3, mb: 2 }}
        >
          Увійти
        </LoadingButton>
        <Grid container>
          {/* <Grid item xs>
            <Link href="#" variant="body2">
              Forgot password?
            </Link>
          </Grid> */}
          <Grid item>
```

```
            <Link
              to={"/sign-up"}
              // to={redirect ? `/register?redirect=${redirect}` : "/register"}
              // variant="body2"
            >
              {"Ще немає акаунти? Зареєструватися"}
            </Link>
          </Grid>
        </Grid>
      </Box>
    </Box>
  </Container>
 </ThemeProvider>
 );
}

const Vehiclea = () => {
  const {vehicleId} = useParams();
  const id = vehicleId ? parseInt(vehicleId) : 0;
  const dispatch = useDispatch();

  const classes = useStyles();

  useEffect(() => {
    dispatch(getCar(id));
  }, [dispatch, id]);

  const {car, isLoading, isSuccess} = useSelector(
    ( state: RootState ) => state.vehicleItems
  );

  const {
    manufacturer,
    model,
    description,
    yearOfProduction,
    averagePrice,
    type,
    quantityInCountry,
    engineDisplacement,
    engineHorsepower,
    pictureUrl
  } = car;

  useEffect(() => {
  }, []);

  return !isLoading ? (
    <Grid container className={classes.root} spacing={2}>
      <Grid container item xs={12} sm={7} direction={"column"}>
        {/* <Grid item xs> */}
        <Grid item mb={4}>
          {/* <Box mb={2}> */}
          <img className={classes.img} alt="product" src={pictureUrl || image}/>
          {/* </Box> */}
        </Grid>
        <Grid item container xs>
          <Paper className={classes.paper} elevation={1}>
            <Typography gutterBottom
                        variant={'h6'}
                        className={classes.padding}
            >
              {manufacturer} - {model}
            </Typography>
            <Divider/>


            <Typography
              variant="body1"
```

```
                gutterBottom
                className={classes.padding}
              >
                Тип кузова: {type}
              </Typography>
              <Divider/>
              <Typography
                variant="body1"
                gutterBottom
                className={classes.padding}
              >
                Рік виготовлення: {yearOfProduction}
              </Typography>
              <Divider/>
              <Typography
                variant="body1"
                gutterBottom
                className={classes.padding}
              >
                Середня ринкова вартість: {averagePrice} грн
              </Typography>
              <Divider/>
              <Typography
                variant="body1"
                gutterBottom
                className={classes.padding}
              >
                Виробник: {quantityInCountry}
              </Typography>
              <Divider/>
              <Typography
                variant="body1"
                gutterBottom
                className={classes.padding}
              >
                Об'єм двигуна: {engineDisplacement}
              </Typography>
              <Divider/>
              <Typography
                variant="body1"
                gutterBottom
                className={classes.padding}
              >
                Потужність: {engineHorsepower} к/с
              </Typography>
              {/*<Divider />*/}
            </Paper>
          </Grid>
        </Grid>
        {/* </Grid> */}
        <Grid item xs={12} sm={5}>
          <Paper className={classes.paper}>
            <Typography variant="h6" gutterBottom>
              {"Опис"}
            </Typography>
            <Typography variant="body2" gutterBottom>
              {description}
            </Typography>
          </Paper>

        </Grid>
      </Grid>

  ) : (
    <Grid container>
      {/* justify="center" alignItems="center"> */}
      <CircularProgress size={80}/>
    </Grid>
  );
```

```
};

export default Vehiclea;
const StatisticsPage = () => {
    const dispatch = useDispatch();
    // dispatch(getCarsStatistics());
    // @ts-ignore
    const vehicleItems = useSelector(state => state.vehicleItems)
    console.log(vehicleItems);

    useEffect(() => {
        dispatch(getCarsStatistics());
    }, [dispatch]);


    const data = {
        labels: vehicleItems.carsStatistics?.popularity?.map(( vehicle: any ) => vehicle.manufacturer),
        datasets: [
            {
                label: 'Популярність',
                data: vehicleItems.carsStatistics?.popularity?.map(( vehicle: any ) => vehicle.count),
                backgroundColor: [
                    'rgba(255, 99, 132, 0.2)',
                    'rgba(54, 162, 235, 0.2)',
                    'rgba(255, 206, 86, 0.2)',
                    'rgba(75, 192, 192, 0.2)',
                    'rgba(153, 102, 255, 0.2)',
                    'rgba(255, 159, 64, 0.2)',
                ],
                borderColor: [
                    'rgba(255, 99, 132, 1)',
                    'rgba(54, 162, 235, 1)',
                    'rgba(255, 206, 86, 1)',
                    'rgba(75, 192, 192, 1)',
                    'rgba(153, 102, 255, 1)',
                    'rgba(255, 159, 64, 1)',
                ],
                borderWidth: 1,
            },
        ],
    };



    return (
        <Container maxWidth="lg">

            {vehicleItems &&
            <Grid container justifyContent="center"
                  alignItems="center" spacing={2}>

                <Grid item xs={8} md={6} spacing={2} alignItems="center">
                    <Pie data={data}/>
                </Grid>

                {/*<Grid item xs={8} md={6}>*/}
                {/*    <Line options={options} data={data2}/>*/}
                {/*</Grid>*/}
            </Grid>
            }
        </Container>
    )

}

export default StatisticsPage;

export const validationSchema = yup.object({
    manufacturer: yup.string().required("Будь ласка, введіть марку."),
```

```
    model: yup.string().required(),
    description: yup.string().required("Будь ласка, добавте опис."),
    yearOfProduction: yup.number().required(),
    averagePrice: yup.number().required("Будь ласка, введіть середню ціну в $."),
    type: yup.string().required(),
    quantityInCountry: yup.number(),
    engineDisplacement: yup.number().required().moreThan(1),
    engineHorsepower: yup.number().required(),
    file: yup.mixed().when('pictureUrl', {
      is: (value: string) => !value,
      then: yup.mixed().required("Будь ласка, добавте фотографію.")
    })
})

const VehicleFormPage = ( {vehicle, cancelEdit}: Props ) => {
    const dispatch = useDispatch();
    const {control, reset, handleSubmit, watch, formState: {isDirty, isSubmitting, errors }} = useForm({
      mode: 'all',
      resolver: yupResolver<any>(validationSchema)
    });

    const watchFile = watch('file', null);

    useEffect(() => {
      if (vehicle && !watchFile && !isDirty) reset(vehicle);

      return () => {
        if (watchFile) URL.revokeObjectURL(watchFile.preview);
      }
    }, [vehicle, reset, watchFile, isDirty]);

    const handleSubmitData = ( data: FieldValues ) => {
      try {
        if (vehicle) {
          dispatch(updateCar(data));
        } else {dispatch(createCar(data));
        }
        cancelEdit();
      } catch (error) {
        console.log(error)
      }
    }

    return (
      <Box component={Paper} sx={{p: 4}}>
        <Typography variant="h4" gutterBottom sx={{mb: 4}}>
          Product Details
        </Typography>
        <form onSubmit={handleSubmit(handleSubmitData)}>
          <Grid container spacing={3}>
            <Grid item xs={12} sm={12}>
              <AppTextInput control={control} name='manufacturer' label='Марка'/>
            </Grid>
            <Grid item xs={12} sm={6}>
              <AppTextInput control={control} name='model' label='Модель'/>
            </Grid>
            <Grid item xs={12} sm={6}>
              <AppTextInput control={control} name='type' label='Кузов'/>
            </Grid>
            <Grid item xs={12} sm={6}>
              <AppTextInput type='number' control={control} name='yearOfProduction' label='Рік виробництва'/>
            </Grid>
            <Grid item xs={12} sm={6}>
              <AppTextInput type='number' control={control} name='averagePrice' label='Середня вартість'/>
            </Grid>
            <Grid item xs={12} sm={6}>
              <AppTextInput type='number' control={control} name='engineDisplacement' label="Об'єм"/>
            </Grid>
            <Grid item xs={12} sm={6}>
```

```
                <AppTextInput type='number' control={control} name='engineHorsepower' label="Потужність"/>
              </Grid>
              <Grid item xs={12}>
                <AppTextInput multiline={true} rows={4} control={control} name='description' label='Опис'/>
              </Grid>
              <Grid item xs={12}>
                <Box display='flex' justifyContent='space-between' alignItems='center'>
                  <AppDropzone control={control} name='file'/>
                  {watchFile ? (
                    <img src={watchFile.preview} alt='preview' style={{maxHeight: 200}}/>
                  ) : (
                    <img src={vehicle?.pictureUrl} alt={vehicle?.manufacturer} style={{maxHeight: 200}}/>
                  )}
                </Box>
              </Grid>
            </Grid>
            <Box display='flex' justifyContent='space-between' sx={{mt: 3}}>
              <Button onClick={cancelEdit} variant='contained' color='inherit'>Cancel</Button>
              <LoadingButton loading={isSubmitting} type='submit' variant='contained'
                      color='success'>Submit</LoadingButton>
            </Box>
          </form>
        </Box>
    );
};

export default VehicleFormPage;

export default function StickyHeadTable() {
    const dispatch = useDispatch();
    const navigate = useNavigate();
    const {cars, carUpdateLoading, isSuccess,vehicleParams, metaData } = useSelector(
        (state: any) => state.vehicleItems
    );

    const [editMode, setEditMode] = useState(false);
    const [selectedVehicle, setSelectedVehicle] = useState<Vehicle | undefined>(undefined);
    const [target, setTarget] = useState(0);

    useEffect(() => {
        dispatch(getCars());

    }, [dispatch, isSuccess, carUpdateLoading, vehicleParams]);

    const deleteClickHandler = async (id: number)=>{
        setTarget(id);
        dispatch(deleteCar(id));
    }

    const editClickHandler = (vehicle: Vehicle)=>{
        setSelectedVehicle(vehicle);
        setEditMode(true);
    }
    const [page, setPage] = React.useState(0);
    const [rowsPerPage, setRowsPerPage] = React.useState(10);


    const handleChangePage = (event: unknown, newPage: number) => {
        setPage(newPage);
    };

    const handleChangeRowsPerPage = (event: React.ChangeEvent<HTMLInputElement>) => {
        setRowsPerPage(+event.target.value);
        setPage(0);
    };

    const cancelEdit =()=>{
        if(selectedVehicle) setSelectedVehicle(undefined);
        setEditMode(false);
```

```
  }

  if(editMode) return <VehicleFormPage vehicle={selectedVehicle} cancelEdit={cancelEdit}/>

  if (carUpdateLoading) return (
    <Box sx={{ display: 'flex' }}>
      <CircularProgress />
    </Box>
  );

  return (
    <Paper sx={{ width: '100%', overflow: 'hidden' }}>
      <TableContainer  sx={{ maxHeight: '50%' }}>
        <Table stickyHeader aria-label="sticky table">
          <TableHead>
            <TableRow>
              {columns.map((column) => (
                <TableCell
                  key={column.name}
                  align={column.align}
                  style={{ minWidth: column.minWidth }}
                >
                  {column.label}
                </TableCell>
              ))}
            </TableRow>
          </TableHead>
          <TableBody>
            {cars
              .slice(page * rowsPerPage, page * rowsPerPage + rowsPerPage)
              .map((car:any) => {
                return (
                  <TableRow hover role="checkbox" tabIndex={-1}>
                    {columns.map((column) => {
                      // @ts-ignore
                      const value = car[column.name];
                      return (
                        <TableCell key={column.name} align={column.align}>
                              {column.format && typeof value === 'number'
                              ? column.format(value)
                              : value
                              }
                            {column.name == "delete" &&
                              <LoadingButton
                              loading={carUpdateLoading && target === car.id}
                              startIcon={<DeleteOutlineIcon />} color='error'
                              onClick={() => deleteClickHandler(car.id)}
                              />
                            }
                            {column.name == "edit" && <IconButton onClick={()=>editClickHandler(car)}><EditIcon
/></IconButton>}
                        </TableCell>
                      );
                    })}
                  </TableRow>
                );
              })}
          </TableBody>
        </Table>
      </TableContainer>
      <Grid container justifyContent={"space-between"} alignItems={"center"}>
        <Grid item>
          <IconButton onClick={() => setEditMode(true)}>
            <AddIcon/>
          </IconButton>
        </Grid>
        {metaData && <AppPagination metaData={metaData} /> }
      </Grid>
    </Paper>
```

```
    );
}

export const useAuthStatus = () => {
  const [loggedIn, setLoggedIn] = useState(false);
  const [checkingStatus, setCheckingStatus] = useState(true);

  const {user} = useSelector((state: RootState) => state.auth);

  useEffect(() => {
    if (user) {
      setLoggedIn(true);
    } else {
      setLoggedIn(false);
    }
    setCheckingStatus(false);
  }, [user]);

  return { loggedIn, checkingStatus };
};

export interface VehicleState {
    cars: Vehicle[];
    car: Vehicle | {};
    manufacturers: string[];
    isError: boolean;
    isSuccess: boolean;
    isLoading: boolean;
    isLoadingFilters: boolean;
    vehiclesLoaded: boolean;
    message: string;
    error: {};
    vehicleParams: VehicleParams
    metaData: MetaData | null
    carUpdateLoading: boolean
    carsStatistics: [],
}

const initParams = () => ({
    pageNumber: 1,
    pageSize: 6,
    orderBy: "name",
    manufacturers: [],
    isLoading: true,
    searchTerm: "",
})

const vehicleAdapter = createEntityAdapter<Vehicle>();

const getAppParams = ( vehicleParams: VehicleParams ) => {
    const params = new URLSearchParams();
    params.append('pageNumber', vehicleParams.pageNumber.toString());
    params.append('pageSize', vehicleParams.pageSize.toString());
    params.append('orderBy', vehicleParams.orderBy);
    if (vehicleParams.searchTerm) params.append('searchTerm', vehicleParams.searchTerm);
    if (vehicleParams.manufacturers.length > 0) params.append('manufacturers', vehicleParams.manufacturers.toString());
    return params;
}

export const getCars = createAsyncThunk<Vehicle[], void, { state: RootState }>("cars/getAll",
    async ( _, thunkAPI ) => {
        try {
            // @ts-ignore
            const params = getAppParams(thunkAPI.getState().vehicleItems.vehicleParams);
            // const token = thunkAPI.getState().auth.user.token;
            const response = await vehicleService.getCars(params);
            thunkAPI.dispatch(setMetaData(response.metaData))
            return response.items
        } catch (error) {
```

```
            // const message =
            //   (error.response && error.response.data && error.response.data.message) ||
            //   error.message ||
            //   error.toString();
            // return thunkAPI.rejectWithValue(message);
        }
    });

export const getCarsStatistics = createAsyncThunk("cars/getCarsStatistics",
    async ( _, thunkAPI ) => {
        try {
            const response = await vehicleService.getCarsStatistics();
            console.log(response);
            return response
        } catch (error) {
            // @ts-ignore
            return thunkAPI.rejectWithValue(error.message);
        }
    });

export const getCar = createAsyncThunk(
    "cars/get",
    async ( vehicleId: number, thunkAPI: {} ) => {
        try {
            // const token = thunkAPI.getState().auth.user.token;
            return await vehicleService.getCar(vehicleId);
        } catch (error) {
            // const message =
            //   (error.response &&
            //     error.response.data &&
            //     error.response.data.message) ||
            //   error.message ||
            //   error.toString();
            // return thunkAPI.rejectWithValue(message);
        }
    }
);

export const getFilters = createAsyncThunk(
    "cars/fetchFilters",
    async ( _, thunkAPI ) => {
        try {
            return vehicleService.getFilters();
        } catch (error: any) {
            return thunkAPI.rejectWithValue({error: error.data});
        }
    }
);

function createFormData( item: any ) {
    let formData = new FormData();
    for (const key in item) {
        formData.append(key, item[key])
    }
    return formData;
}

export const createCar = createAsyncThunk(
    "cars/createCar",
    async ( vehicle: FieldValues, thunkAPI ) => {
        try {
            // @ts-ignore
            const token = thunkAPI.getState().auth.user.token;

            return await vehicleService.createCar(token, createFormData(vehicle))
        } catch (error: any) {
            return thunkAPI.rejectWithValue({error: error.data});
        }
    })
```

```
export const updateCar = createAsyncThunk(
  "cars/updateCar",
  async ( vehicle: FieldValues, thunkAPI ) => {
    try {
      // @ts-ignore
      const token = thunkAPI.getState().auth.user.token;
      return await vehicleService.updateCar(token, createFormData(vehicle))
    } catch (error: any) {
      return thunkAPI.rejectWithValue({error: error.data});
    }
  })

export const deleteCar = createAsyncThunk(
  "cars/deleteCar",
  async ( vehicleId: number, thunkAPI ) => {
    try {
      // @ts-ignore
      const token = thunkAPI.getState().auth.user.token;
      return await vehicleService.deleteCar(token, vehicleId);
    } catch (error: any) {
      return thunkAPI.rejectWithValue({error: error.data});
    }
  })
export const vehicleSlice = createSlice({
  name: "vehicle",
  initialState: vehicleAdapter.getInitialState<VehicleState>({
    cars: [],
    car: {},
    carUpdateLoading: false,
    manufacturers: [],
    isError: false,
    isSuccess: false,
    isLoading: false,
    isLoadingFilters: false,
    vehiclesLoaded: false,
    message: "",
    error: {},
    vehicleParams: initParams(),
    metaData: null,
    carsStatistics: [],
  }),
  reducers: {
    setAppParams: ( state, action ) => {
      state.vehiclesLoaded = false;
      state.vehicleParams = {...state.vehicleParams, ...action.payload, pageNumber: 1};
    },
    setPageNumber: ( state, action ) => {
      state.vehiclesLoaded = false;
      state.vehicleParams = {...state.vehicleParams, ...action.payload};
    },
    setMetaData: ( state, action ) => {
      state.metaData = action.payload;
    },
    resetProductParams: ( state ) => {
      state.vehicleParams = initParams();
    },
    // setProduct: (state, action) => {
    //   productsAdapter.upsertOne(state, action.payload);
    //   state.vehiclesLoaded = false;
    // },
    // removeProduct: (state, action) => {
    //   productsAdapter.removeOne(state, action.payload);
    //   state.vehiclesLoaded = false;
    // }
  },
  extraReducers: ( builder ) => {
    builder
      .addCase(getCars.pending, ( state ) => {
```

```
      state.isLoading = true;
    })
    .addCase(getCars.fulfilled, ( state, action ) => {
      state.isLoading = false;
      state.isSuccess = true;
      state.cars = action.payload;
    })
    .addCase(getCars.rejected, ( state, action: { payload: any } ) => {
      state.isLoading = false;
      state.isError = true;
      state.message = action.payload;
    });
  builder
    .addCase(getCar.pending, ( state ) => {
      state.isLoading = true;
    })
    .addCase(getCar.fulfilled, ( state, action ) => {
      state.isLoading = false;
      state.isSuccess = true;
      state.car = action.payload;
    })
    .addCase(getCar.rejected, ( state, action: { payload: any } ) => {
      state.isLoading = false;
      state.isError = true;
      state.message = action.payload;
    });
  builder
    .addCase(getFilters.pending, ( state ) => {
      state.isLoadingFilters = true;
    })
    .addCase(getFilters.fulfilled, ( state, action ) => {
      state.isLoadingFilters = false;
      state.isSuccess = true;
      state.manufacturers = action.payload.manufacturers;
    })
    .addCase(getFilters.rejected, ( state, action: { payload: any } ) => {
      state.isLoadingFilters = false;
      state.isError = true;
      state.message = action.payload;
    });
  builder
    .addCase(createCar.pending, ( state ) => {
      state.carUpdateLoading = true;
    })
    .addCase(createCar.fulfilled, ( state, action ) => {
      state.carUpdateLoading = false;
      state.isSuccess = true;
    })
    .addCase(createCar.rejected, ( state, action: { payload: any } ) => {
      state.carUpdateLoading = false;
      state.isError = true;
      state.message = action.payload;
    });
  builder
    .addCase(updateCar.pending, ( state ) => {
      state.carUpdateLoading = true;
    })
    .addCase(updateCar.fulfilled, ( state, action ) => {
      state.carUpdateLoading = false;
      state.isSuccess = true;
    })
    .addCase(updateCar.rejected, ( state, action: { payload: any } ) => {
      state.carUpdateLoading = false;
      state.isError = true;
      state.message = action.payload;
    });
  builder
    .addCase(deleteCar.pending, ( state ) => {
      state.carUpdateLoading = true;
```

```
        })
        .addCase(deleteCar.fulfilled, ( state, action: any ) => {
          state.cars = state.cars.filter(car => car.id !== action.payload.vehicleId);
          state.isSuccess = true;
          state.carUpdateLoading = false;
        })
        .addCase(deleteCar.rejected, ( state, action: { payload: any } ) => {
          state.carUpdateLoading = false;
          state.isError = true;
          state.message = action.payload;
        });
      builder
        .addCase(getCarsStatistics.pending, ( state ) => {
          state.carUpdateLoading = true;
        })
        .addCase(getCarsStatistics.fulfilled, ( state, action: any ) => {
          console.log("action:", action);
          state.carsStatistics = action.payload;
          state.isSuccess = true;
        })
        .addCase(getCarsStatistics.rejected, ( state, action: { payload: any } ) => {
          state.carUpdateLoading = false;
          state.isError = true;
          state.message = action.payload;
        });
  },
});

// Action creators are generated for each case reducer function
export const {setAppParams, setPageNumber, resetProductParams, setMetaData} = vehicleSlice.actions;

export default vehicleSlice.reducer;

const API_URL = "http://localhost:5000/api/Cars";

axios.interceptors.response.use(async response => {
  const pagination = response.headers['pagination'];
  if (pagination) {
    response.data = new PaginatedResponse(response.data, JSON.parse(pagination));
        return response;
  }

  return response;
})

// axios.interceptors.response.use(async response=>{
//   const pagination = response.headers['pagination'];
//
//   console.log(pagination);
//
//   return response;
// })

// Get cars
const getCars = async ( params: URLSearchParams) => {
  // const config = {
  //   headers: {
  //     Authorization: `Bearer ${token}`,
  //   },
  // };
  const response = await axios.get(API_URL, {params});

  return response.data;
};

const getCarsStatistics = async () => {
  const response = await axios.get(API_URL+ "/statistics");

  console.log(response.data);
```

```
  return response.data;
};

const getCar = async (carId: number) => {
  // const config = {
  //   headers: {
  //     Authorization: `Bearer ${token}`,
  //   },
  // };

  const response = await axios.get(API_URL + `/${carId}`);

  return response.data;
};

const getFilters = async () => {
  const response = await axios.get(API_URL + "/filters");

  return response.data;
};

const createCar = async (token: string, data: FormData) => {
  const response = await axios.post(API_URL, data,{
    headers: {
      Authorization: `Bearer ${token}`,
      'Content-type': 'multipart/form-data'}
  });

  console.log(data);

  return response;
}

const updateCar = async (token: string, data: FormData) =>{
  const response = axios.put(API_URL, data, {
    headers: {
      Authorization: `Bearer ${token}`,
      'Content-type': 'multipart/form-data'}
  })

  return response;
}

const deleteCar = async (token: string, vehicleId: number) =>{
  const response = axios.delete(
      API_URL+ `/${vehicleId}`, {
        headers:{ Authorization: `Bearer ${token}`}
      });

  return {response, vehicleId};
}


const vehicleService = {
  getCars,
  getCar,
  getFilters,
  updateCar,
  createCar,
  deleteCar,
  getCarsStatistics
};

export default vehicleService;

export interface Vehicle{
  id: number;
  manufacturer: string;
```

```
  model: string;
  description: string;
  yearOfProduction: number;
  averagePrice: number;
  type: string;
  quantityInCountry: number | null;
  engineDisplacement: number;
  engineHorsepower: number;
  pictureUrl: string
}

export interface VehicleParams {
  orderBy: string;
  searchTerm?: string;
  manufacturers: string[];
  pageNumber: number;
  pageSize: number;
  isLoading: boolean;
}

export interface User {
    email: string;
    token: string;
    // basket?: Basket;
    roles?: string[];
}

export interface MetaData {
    currentPage: number;
    totalPages: number;
    pageSize: number;
    totalCount: number;
}

export class PaginatedResponse<T> {
    items: T;
    metaData: MetaData;

    constructor(items: T, metaData: MetaData) {
        this.items = items;
        this.metaData = metaData;
    }
}

import axios, { AxiosError, AxiosResponse } from "axios";
import { toast } from "react-toastify";

const responseBody = (response: AxiosResponse) => response.data;

const requests = {
  get: (url: string, params?: URLSearchParams) => axios.get(url, {params}).then(responseBody),
  post: (url: string, body: {}) => axios.post(url, body).then(responseBody),
  put: (url: string, body: {}) => axios.put(url, body).then(responseBody),
  delete: (url: string) => axios.delete(url).then(responseBody),
  postForm: (url: string, data: FormData) => axios.post(url, data, {
      headers: {'Content-type': 'multipart/form-data'}
  }).then(responseBody),
  putForm: (url: string, data: FormData) => axios.put(url, data, {
      headers: {'Content-type': 'multipart/form-data'}
  }).then(responseBody)
}

const Account = {
  login: (values: any) => requests.post('account/login', values),
  register: (values: any) => requests.post('account/register', values),
  currentUser: () => requests.get('account/currentUser'),
  fetchAddress: () => requests.get('account/savedAddress')
}
```

```
const TestErrors = {
  get400Error: () => requests.get('buggy/bad-request'),
  get401Error: () => requests.get('buggy/unauthorised'),
  get404Error: () => requests.get('buggy/not-found'),
  get500Error: () => requests.get('buggy/server-error'),
  getValidationError: () => requests.get('buggy/validation-error'),
}

const agent = {
  TestErrors,
  Account,
  }

export default agent;

import { UploadFile } from '@mui/icons-material';
import { FormControl, FormHelperText, Typography } from '@mui/material';
import { useCallback } from 'react'
import { useDropzone } from 'react-dropzone'
import { useController, UseControllerProps } from 'react-hook-form'

interface Props extends UseControllerProps { }

export default function AppDropzone(props: Props) {
    const { fieldState, field } = useController({ ...props, defaultValue: null });

    const dzStyles = {
        display: 'flex',
        border: 'dashed 3px #eee',
        borderColor: '#eee',
        borderRadius: '5px',
        paddingTop: '30px',
        alignItems: 'center',
        height: 200,
        width: 500
    }

    const dzActive = {
        borderColor: 'green'
    }

    const onDrop = useCallback(acceptedFiles => {
        acceptedFiles[0] = Object.assign(acceptedFiles[0],
            {preview: URL.createObjectURL(acceptedFiles[0])});
        field.onChange(acceptedFiles[0]);
    }, [field])
    const { getRootProps, getInputProps, isDragActive } = useDropzone({ onDrop })

    return (
        <div {...getRootProps()}>
            <FormControl error={!!fieldState.error} style={isDragActive ? {...dzStyles, ...dzActive} : dzStyles}>
                <input {...getInputProps()} />
                <UploadFile sx={{fontSize: '100px'}} />
                <Typography variant='h4'>Drop image here</Typography>
                <FormHelperText>{fieldState.error?.message}</FormHelperText>
            </FormControl>
        </div>
    )
}
import { TextField } from "@mui/material";
import { useController, UseControllerProps } from "react-hook-form";

interface Props extends UseControllerProps {
    label: string;
    multiline?: boolean;
    rows?: number;
    type?: string;
}
```

```
export default function AppTextInput(props: Props) {
    const {fieldState, field} = useController({...props, defaultValue: ''})
    return (
        <TextField
            {...props}
            {...field}
            multiline={props.multiline}
            rows={props.rows}
            type={props.type}
            fullWidth
            variant='outlined'
            error={!!fieldState.error}
            helperText={fieldState.error?.message}
        />
    )
}
import { FormControl, InputLabel, Select, MenuItem, FormHelperText } from "@mui/material";
import { useController, UseControllerProps } from "react-hook-form";

interface Props extends UseControllerProps {
    label: string;
    items: string[];
}

export default function AppSelectList(props: Props) {
    const { fieldState, field } = useController({ ...props, defaultValue: '' });
    return (
        <FormControl fullWidth error={!!fieldState.error}>
            <InputLabel>{props.label}</InputLabel>
            <Select
                value={field.value}
                label={props.label}
                onChange={field.onChange}
            >
                {props.items.map((item, index) => (
                    <MenuItem key={index} value={item}>{item}</MenuItem>
                ))}
            </Select>
            <FormHelperText>{fieldState.error?.message}</FormHelperText>
        </FormControl>
    )
}
import { Typography, Pagination } from "@mui/material";
import { Box } from "@mui/system";
import { MetaData } from "../app/models/pagination";
import {useDispatch} from "react-redux";
import {setPageNumber} from "../features/vehicles/vehicleSlice";

interface Props {
    metaData: MetaData;
    // onPageChange: (page: number) => void;
}

export default function AppPagination({metaData}: Props) {
    const {currentPage, totalCount, totalPages, pageSize} = metaData;
    const dispatch = useDispatch();;

    function handlePageChange( number: number) {
        dispatch(setPageNumber({pageNumber: number}))
    }

    return (
        <Box display='flex' justifyContent='space-between' alignItems='center' p={2}>
            <Typography>
                Показано {(currentPage-1)*pageSize+1}-
                {currentPage*pageSize > totalCount
                    ? totalCount
                    : currentPage*pageSize} з {totalCount}
            </Typography>
```

```
          <Pagination
            color='secondary'
            size='large'
            count={totalPages}
            page={currentPage}
            onChange={(e, page) => handlePageChange(page)}
          />
        </Box>
    )
}
import { useDispatch, useSelector } from "react-redux";
import {Vehicle, VehicleParams} from "../app/models/vehicleModel";
import ItemCard from "./ItemCard";
import CircularProgress from "@mui/material/CircularProgress";
import { useEffect } from "react";
import { getCars, getFilters} from "../features/vehicles/vehicleSlice";
import {MetaData} from "../app/models/pagination";
import AppPagination from "./AppPagination";
import {setPageNumber} from "../features/vehicles/vehicleSlice";


interface RootState {
  vehicleItems: {
    cars: Vehicle[];
    isLoading: boolean;
    isSuccess: boolean;
    vehicleParams: VehicleParams;
    metaData: MetaData
  };
}
// interface Props {
//   cars: Car[];
// }

// const CarsList = ({ cars }: Props) => {
const CarsList = () => {
  const dispatch = useDispatch();

 const { cars, isLoading, isSuccess,vehicleParams, metaData } = useSelector(
    (state: RootState) => state.vehicleItems
  );

  useEffect(() => {
    dispatch(getCars());
    dispatch(getFilters());
  }, [dispatch, vehicleParams]);

    return (
    <>
        <>
      {isLoading ? (
        <CircularProgress />
      ) : (
        cars.map((car) => <ItemCard key={car.id} item={car} />)
      )}
        </>

        {metaData && <AppPagination metaData={metaData} /> }
    </>
  );
};

export default CarsList;

import { FormGroup, FormControlLabel, Checkbox } from "@mui/material";
import { useState } from "react";

interface Props {
    items: string[];
```

```
    checked?: string[];
    onChange: (items: string[]) => void;
}

export default function CheckboxButtons({items, checked, onChange}: Props) {
    const [checkedItems, setCheckedItems] = useState(checked || []);

    function handleChecked(value: string) {
        const currentIndex = checkedItems.findIndex(item => item === value);
        let newChecked: string[] = [];
        if (currentIndex === -1) newChecked = [...checkedItems, value];
        else newChecked = checkedItems.filter(item => item !== value);
        setCheckedItems(newChecked);
        onChange(newChecked);
    }

    return (
        <FormGroup>
            {items.map(item => (
                <FormControlLabel
                    control={<Checkbox
                        checked={checkedItems.indexOf(item) !== -1}
                        onClick={() => handleChecked(item)}
                    />}
                    label={item}
                    key={item}
                />
            ))}
        </FormGroup>
    )
}

import {Grid, Paper, TextField, Typography} from '@mui/material';
import React from 'react';
import RadioButtonGroup from "./RadioButtonGroup";
import {types} from "util";
import CheckboxButtons from './CheckboxButtons';
import AppPagination from './AppPagination';
import {useDispatch, useSelector} from "react-redux";
import {Vehicle, VehicleParams} from "../app/models/vehicleModel";
import {setAppParams} from "../features/vehicles/vehicleSlice";

const sortOptions = [
    {value: "name", label: "За назвою"},
    {value: "priceDesc", label: "Від доступніших"},
    {value: "priceInc", label: "Від дорожчих"},
    {value: "yearDesc", label: "За роком виготовлення"},
]

interface RootState {
    vehicleItems: VehicleParams,
}

const FiltersCard = () => {
    const dispatch = useDispatch();

    let {
        orderBy,
        pageSize,
        pageNumber,
        searchTerm,
        manufacturers,
        isLoading
    } = useSelector(( state: RootState ) => state.vehicleItems)
    return (
        <Grid container spacing={2}>
            <Grid item xs={6}>
                <Typography variant={'h6'}>
                    Сортування
```

```
            </Typography>
            <Paper sx={{mb: 2, p: 2}}>
              <RadioButtonGroup
                selectedValue={orderBy}
                options={sortOptions}
                onChange={( e ) => dispatch(setAppParams({orderBy: e.target.value}))}
              />
            </Paper>
          </Grid>

          <Grid item xs={6}>
            <Typography variant={'h6'}>
              Виробник
            </Typography>
            <Paper sx={{mb: 2, p: 2}}>
              <CheckboxButtons
                items={manufacturers}
                checked={manufacturers}
                onChange={( items: string[] ) => dispatch(setAppParams({manufacturers: items}))}
              />
            </Paper>
          </Grid>
        </Grid>
    );
};

export default FiltersCard;

import {
  Box,
  Container,
  ContainerProps,
  styled,
  experimental_sx as sx,
  Typography,
} from "@mui/material";
import { Link } from "react-router-dom";
import React from "react";

interface Props {
  mode: boolean;
  setMode: (mode: boolean) => void;
}

function Copyright(props: any) {
  return (
    <Typography
      variant="body2"
      color="text.secondary"
      align="center"
      {...props}
    >
      {"Copyright © "}

      <Link color="inherit" to="/">
        Course project
      </Link>
      {" - "}
      {new Date().getFullYear()}
      {"."}
    </Typography>
  );
}

const StyledFooter = styled("footer")(({ theme }) =>
  sx({
    color: () =>
      theme.palette.mode === "dark"
        ? theme.palette.text.primary
```

```
      : theme.palette.primary.contrastText,
    backgroundColor: () =>
      theme.palette.mode === "dark" ? "#272727" : theme.palette.primary.main,
    padding: theme.spacing(2.5),
    // position: "fixed",
    left: 0,
    bottom: 0,
    right: 0,
    marginTop: "auto",
  })
);

const StyledContainer = styled(Container)<ContainerProps>({
  display: "flex",
  justifyContent: "center",
});

const Footer = () => {
  return (
    <StyledFooter>
      <StyledContainer>
        <Copyright />
      </StyledContainer>
    </StyledFooter>
  );
};

export default Footer;

import * as React from "react";
import AppBar from "@mui/material/AppBar";
import Box from "@mui/material/Box";
import Toolbar from "@mui/material/Toolbar";
import IconButton from "@mui/material/IconButton";
import Typography from "@mui/material/Typography";
import Menu from "@mui/material/Menu";
import MenuIcon from "@mui/icons-material/Menu";
import Container from "@mui/material/Container";
import Avatar from "@mui/material/Avatar";
import Button from "@mui/material/Button";
import Tooltip from "@mui/material/Tooltip";
import MenuItem from "@mui/material/MenuItem";
import DirectionsCarIcon from '@mui/icons-material/DirectionsCar';
import LightModeIcon from "@mui/icons-material/LightMode";
import DarkModeIcon from "@mui/icons-material/DarkMode";
import {useNavigate} from "react-router-dom";
import {useDispatch, useSelector} from "react-redux";
import {RootState} from "../app/store";
import {logout} from "../features/auth/authSlice";
import ProductSearch from "./ProductSearch";

interface Props {
    mode: boolean;
    setMode: ( mode: boolean ) => void;
}

const Header = ( {mode, setMode}: Props ) => {
    const navigate = useNavigate();
    const dispatch = useDispatch();

    const {user: email} = useSelector(( state: RootState ) => state.auth);

    const mainPageHandler = () => navigate("/");
    const controlPanelHandler = () => navigate("/account/vehicleList");
    const signInHandler = () => navigate("/sign-in");
    const signUpHandler = () => navigate("/sign-up");
    const profileHandler = () => navigate("/account");
    const statisticsHandler = () => navigate("/statistics");
    const productsHandler = () => navigate("/edit/products");
```

```
const logoutHandler = () => dispatch(logout());

const settings = [
  {
    id: 1,
    name: "Profile",
    handleClick: profileHandler,
  },
  {
    id: 2,
    name: "Dashboard",
    handleClick: profileHandler,
  },
  {
    id: 3,
    name: "Logout",
    handleClick: logoutHandler,
  },
];
const settings2 = [
  {
    id: 1,
    name: "Увійти",
    handleClick: signInHandler,
  },
  {
    id: 2,
    name: "Зареєструватися",
    handleClick: signUpHandler,
  },
];

const [anchorElNav, setAnchorElNav] = React.useState<null | HTMLElement>(
  null
);
const [anchorElUser, setAnchorElUser] = React.useState<null | HTMLElement>(
  null
);

const handleOpenNavMenu = ( event: React.MouseEvent<HTMLElement> ) => {
  setAnchorElNav(event.currentTarget);
};
const handleOpenUserMenu = ( event: React.MouseEvent<HTMLElement> ) => {
  setAnchorElUser(event.currentTarget);
};

const handleCloseNavMenu = () => {
  setAnchorElNav(null);
};

const handleCloseUserMenu = () => {
  setAnchorElUser(null);
};

return (
  <AppBar
    position="static"
    sx={{
      marginBottom: 2,
    }}
  >
    <Container maxWidth="xl">
      <Toolbar disableGutters>
        <Typography
          variant="h6"
          noWrap
          component="div"
          sx={{mr: 2, display: {xs: "none", md: "flex"}}}
        >
```

```
    <DirectionsCarIcon/>
  </Typography>

  <Box sx={{flexGrow: 1, display: {xs: "flex", md: "none"}}}>
    <IconButton
      size="large"
      aria-label="account of current user"
      aria-controls="menu-appbar"
      aria-haspopup="true"
      onClick={handleOpenNavMenu}
      color="inherit"
    >
      <MenuIcon/>
    </IconButton>
    <Menu
      id="menu-appbar"
      anchorEl={anchorElNav}
      anchorOrigin={{
        vertical: "bottom",
        horizontal: "left",
      }}
      keepMounted
      transformOrigin={{
        vertical: "top",
        horizontal: "left",
      }}
      open={Boolean(anchorElNav)}
      onClose={handleCloseNavMenu}
      sx={{
        display: {xs: "block", md: "none"},
      }}
    >
      <MenuItem onClick={mainPageHandler}>
        <Typography textAlign="center">All cars</Typography>
      </MenuItem>
      <MenuItem onClick={statisticsHandler}>
        <Typography textAlign="center">Favorite</Typography>
      </MenuItem>
      <MenuItem onClick={controlPanelHandler}>
        <Typography textAlign="center">Панель керування</Typography>
      </MenuItem>
    </Menu>
  </Box>
  <Typography
    variant="h6"
    noWrap
    component="div"
    sx={{flexGrow: 1, display: {xs: "flex", md: "none"}}}
  >
    <DirectionsCarIcon/>
  </Typography>
  <Box sx={{
    flexGrow: 1,
    display: {xs: "none", md: "flex"}
  }}>
    <Button
      // key={1}
      onClick={mainPageHandler}
      sx={{my: 2, color: "white", display: "block"}}
    >
      Список автомобілів
    </Button>
    <Button
      // key={2}
      onClick={statisticsHandler}
      sx={{my: 2, color: "white", display: "block"}}
    >
      Статистика
    </Button>
```

```
                {email.email === "admin@test.com" &&
                  <Button
                    // key={3}
                    onClick={controlPanelHandler}
                    sx={{my: 2, color: "white", display: "block"}}
                  >
                    Панель керування
                  </Button>
                }
              </Box>

              <ProductSearch/>

              <Box sx={{flexGrow: 0}}>
                <IconButton onClick={() => setMode(!mode)}>
                  {mode ? <DarkModeIcon/> : <LightModeIcon/>}
                </IconButton>

                <Tooltip title="Open settings">
                  <IconButton onClick={handleOpenUserMenu} sx={{p: 0}}>
                    <Avatar alt="Remy Sharp" src="/static/images/avatar/2.jpg"/>
                  </IconButton>
                </Tooltip>
                <Menu
                  sx={{mt: "45px"}}
                  id="menu-appbar"
                  anchorEl={anchorElUser}
                  anchorOrigin={{
                    vertical: "top",
                    horizontal: "right",
                  }}
                  keepMounted
                  transformOrigin={{
                    vertical: "top",
                    horizontal: "right",
                  }}
                  open={Boolean(anchorElUser)}
                  onClose={handleCloseUserMenu}
                >
                  {email
                    ? settings.map(( setting ) => (
                      <MenuItem key={setting.id} onClick={setting.handleClick}>
                        <Typography textAlign="center">{setting.name}</Typography>
                      </MenuItem>
                    ))
                    : settings2.map(( setting ) => (
                      <MenuItem key={setting.id} onClick={setting.handleClick}>
                        <Typography textAlign="center">{setting.name}</Typography>
                      </MenuItem>
                    ))}
                </Menu>
              </Box>
            </Toolbar>
          </Container>
        </AppBar>
    );
};
export default Header;



import { Box, Paper, Grid, Typography, Button } from "@mui/material";
import React from "react";
import { useNavigate } from "react-router-dom";
// @ts-ignore
import BMWimg from "../imgs/BMWM3.png";
import { Vehicle } from "../app/models/vehicleModel";

interface Props {
```

```
  item: Vehicle;
}

const ItemCard = ({ item }: Props) => {
  const {
    id,
    manufacturer,
    model,
    description,
    yearOfProduction,
    averagePrice,
    type,
    quantityInCountry,
    engineDisplacement,
    engineHorsepower,
    pictureUrl
  } = item;
  let navigate = useNavigate();

  return (
    <Box mb={2}>
      <Paper
        elevation={1}
        sx={{
          backgroundColor: (theme) =>
            theme.palette.mode === "dark" ? "#1A2027" : "#fff",
          overflow: "hidden",
        }}
      >
        <Grid container>
          <Grid item xs={4}>
            <Box
              component={"img"}
              sx={{
                maxWidth: "sm",
                width: "100%",
                height: "100%",
              }}
              src={pictureUrl || BMWimg}
            />
          </Grid>

          <Grid item container xs={8}>
            <Grid
              item
              container
              xs={12}
              p={3}
              direction={"column"}
              justifyContent={"space-between"}
            >
              <Typography variant="h6">
                {manufacturer} - {model}
              </Typography>

              <Typography
                variant="body1"
                paragraph
                gutterBottom
                align="left"
                sx={{
                  display: "block",
                }}
              >
                {description}
              </Typography>

              <Grid item>
                <Button
```

```
              variant="outlined"
              onClick={() => navigate(`/vehicle/${id}`)}
            >
              Детальніше
            </Button>
          </Grid>
        </Grid>
      </Grid>
    </Paper>
  </Box>
 );
};

export default ItemCard;


import React from "react";
import { Navigate, Outlet } from "react-router-dom";
import { useAuthStatus } from "../hooks/useAuthStatus";
import CircularProgress from "@mui/material/CircularProgress";
import { Box } from "@mui/material";

const PrivateRoute = () => {
 const { loggedIn, checkingStatus } = useAuthStatus();

 if (checkingStatus) {
  return (
   <Box sx={{ display: "flex" }}>
    <CircularProgress />
   </Box>
  );
 }

 return loggedIn ? <Outlet /> : <Navigate to={"/sign-in"} />;
};

export default PrivateRoute;


import {debounce, InputBase, TextField} from "@mui/material";
import { useState } from "react";
import {useDispatch, useSelector} from "react-redux";
import {setAppParams} from "../features/vehicles/vehicleSlice";

export default function ProductSearch() {
  // @ts-ignore
  const {vehicleParams} = useSelector((state)=> state.vehicleItems);
  const [searchTerm, setSearchTerm] = useState(vehicleParams.searchTerm);
  const dispatch = useDispatch();

  const debouncedSearch = debounce((event: any) => {
    dispatch(setAppParams({searchTerm: event.target.value}))
  }, 1000)

  return (
    <TextField
      sx={{
        marginLeft: 'auto',
        marginRight: 0,
      }}
      label='Search products'
      variant='outlined'

      value={searchTerm || ''}
      onChange={(event: any) => {
        setSearchTerm(event.target.value);
        debouncedSearch(event);
      }}
```

Лістинг коду серверної частини

Startup.cs

```
using CourceProject.Entities;
using CourceProject.Middleware;
using CourceProject.Services;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.IdentityModel.Tokens;
using Microsoft.OpenApi.Models;
using System.Collections.Generic;
using System.Text;
using AutoMapper;
using CourceProject.Models;
using CourceProject.RequestHelpers;

namespace CourceProject
{
   public class Startup
   {
      public Startup(IConfiguration configuration)
      {
         Configuration = configuration;
      }

      public IConfiguration Configuration { get; }

      // This method gets called by the runtime. Use this method to add services to the container.
      public void ConfigureServices(IServiceCollection services)
      {

         services.AddControllers();
         services.AddAutoMapper(typeof(MappingProfiles).Assembly);
         services.AddHttpClient();
         services.AddSwaggerGen(c =>
         {
            c.SwaggerDoc("v1", new OpenApiInfo { Title = "WebAPIv5", Version = "v1" });
            c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
            {
               Description = "Jwt auth header",
               Name = "Authorization",
               In = ParameterLocation.Header,
               Type = SecuritySchemeType.ApiKey,
               Scheme = "Bearer"
            });
            c.AddSecurityRequirement(new OpenApiSecurityRequirement
            {
               {
                  new OpenApiSecurityScheme
                  {
                     Reference = new OpenApiReference
                     {
                        Type = ReferenceType.SecurityScheme,
                        Id = "Bearer"
                     },
                     Scheme = "oauth2",
                     Name = "Bearer",
                     In = ParameterLocation.Header
                  },
                  new List<string>()
               }
            });
         });
```

```
            services.AddDbContext<CarsContext>(opt =>
            {
                opt.UseSqlite(Configuration.GetConnectionString("Default"));
            });
            services.AddCors();
            services.AddIdentityCore<User>(opt =>
            {
                opt.User.RequireUniqueEmail = true;
            })
                .AddRoles<IdentityRole>()
                .AddEntityFrameworkStores<CarsContext>();
                services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                .AddJwtBearer(opt =>
                {
                    opt.TokenValidationParameters = new TokenValidationParameters
                    {
                        ValidateIssuer = false,
                        ValidateAudience = false,
                        ValidateLifetime = true,
                        ValidateIssuerSigningKey = true,
                        IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8
                            .GetBytes(Configuration["JWTSettings:TokenKey"]))
                    };
                });
            services.AddAuthorization();
            services.AddScoped<TokenService>();
            services.AddScoped<ImageService>();
        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            //app.UseMiddleware<ExceptionMiddleware>();


            if (env.IsDevelopment())
            {
                app.UseSwagger();
                app.UseSwaggerUI(c => c.SwaggerEndpoint("/swagger/v1/swagger.json", "WebAPIv5 v1"));
            }

            //app.UseHttpsRedirection();

            app.UseRouting();
            app.UseCors(opt =>
            {
                opt.AllowAnyHeader().AllowAnyMethod().WithOrigins("http://localhost:3000");
            });

            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```

Program.cs

```
using CourceProject.Data;
using CourceProject.Entities;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
```

```
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Threading.Tasks;
using CourceProject.Models;

namespace CourceProject
{
    public class Program
    {
        public static async Task Main(string[] args)
        {
            var host = CreateHostBuilder(args).Build();
            using var scope = host.Services.CreateScope();
            var context = scope.ServiceProvider.GetRequiredService<CarsContext>();
            var userManager = scope.ServiceProvider.GetRequiredService<UserManager<User>>();
            var logger = scope.ServiceProvider.GetRequiredService<ILogger<Program>>();
            try
            {
                await context.Database.MigrateAsync();
                await DBinitializer.Initialize(context, userManager);
            }
            catch (Exception ex)
            {
                logger.LogError(ex, "Problem migratig data.");
            }

            await host.RunAsync();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

TokenService.cs

```
using CourceProject.Entities;
using Microsoft.AspNetCore.Identity;
using Microsoft.Extensions.Configuration;
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using System.Threading.Tasks;
using CourceProject.Models;

namespace CourceProject.Services
{
    public class TokenService
    {
        private readonly UserManager<User> _userManager;
        private readonly IConfiguration _config;

        public TokenService(UserManager<User> userManager, IConfiguration config)
        {
            _userManager = userManager;
            _config = config;
        }

        public async Task<string> GenerateToken(User user)
        {
            var claims = new List<Claim>()
            {
                new Claim(ClaimTypes.Email, user.Email),
```

```
            new Claim(ClaimTypes.Name, user.UserName),
        };

        var roles = await _userManager.GetRolesAsync(user);

        foreach (var role in roles)
        {
            claims.Add(new Claim(ClaimTypes.Role, role));
        }

        var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["JWTSettings:TokenKey"]));
        var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha512);

        var tokenOptions = new JwtSecurityToken(
            issuer: null,
            audience: null,
            claims: claims,
            expires: DateTime.Now.AddDays(7),
            signingCredentials: creds
            );
        return new JwtSecurityTokenHandler().WriteToken(tokenOptions);
    }
  }
}
```

ImageService.cs

```
using System.Threading.Tasks;
using CloudinaryDotNet;
using CloudinaryDotNet.Actions;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Configuration;

namespace CourceProject.Services;

public class ImageService
{
    private readonly Cloudinary _cloudinary;

    public ImageService(IConfiguration config)
    {
        var acc = new Account(
            config["Cloudinary:CloudName"],
            config["Cloudinary:ApiKey"],
            config["Cloudinary:ApiSecret"]
        );

        _cloudinary = new Cloudinary(acc);
    }

    public async Task<ImageUploadResult> AddImageAsync(IFormFile file)
    {
        var uploadResult = new ImageUploadResult();

        if (file.Length > 0)
        {
            using var stream = file.OpenReadStream();
            var uploadParams = new ImageUploadParams
            {
                File = new FileDescription(file.FileName, stream)
            };
            uploadResult = await _cloudinary.UploadAsync(uploadParams);

        }

        return uploadResult;
    }

    public async Task<DeletionResult> DeleteImageAsync(string publicId)
    {
        var deleteParams = new DeletionParams(publicId);
```

```
        var result = await _cloudinary.DestroyAsync(deleteParams);

        return result;
    }
}
```

HttpExtensions.cs

```
using System.Text.Json;
using Microsoft.AspNetCore.Http;

namespace CourceProject.RequestHelpers;

public static class HttpExtensions
{
    public static void AddPaginationHeader(this HttpResponse response, MetaData metaData)
    {
        var options = new JsonSerializerOptions{PropertyNamingPolicy= JsonNamingPolicy.CamelCase};

        response.Headers.Add("Pagination", JsonSerializer.Serialize(metaData, options));
        response.Headers.Add("Access-Control-Expose-Headers", "Pagination");
    }
}
```

MappingProfiles.cs

```
using AutoMapper;
using CourceProject.DTOs;
using CourceProject.Entities;

namespace CourceProject.RequestHelpers;

public class MappingProfiles : Profile
{
    public MappingProfiles()
    {
        CreateMap<CreateCarDto, Car>();
        CreateMap<UpdateCarDto, Car>();
    }
}
```

MetaData.cs

```
namespace CourceProject.RequestHelpers;

public class MetaData
{
    public int CurrentPage { get; set; }
    public int TotalPages { get; set; }
    public int PageSize { get; set; }
    public int TotalCount { get; set; }
}
```

PagedList.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace CourceProject.RequestHelpers;

public class PagedList<T> :List<T>
{
    public PagedList(List<T>items, int count, int pageNumber, int pageSize)
    {
        MetaData = new MetaData
        {
            TotalCount = count,
            PageSize = pageSize,
            CurrentPage = pageNumber,
            TotalPages = (int) Math.Ceiling(count / (double) pageSize)
        };
        AddRange(items);
    }
```

```
    public MetaData MetaData { get; set; }

    public static async Task<PagedList<T>> ToPagedList(IQueryable<T> query,
        int pageNumber, int pageSize)
    {
        var count = await query.CountAsync();
        var items = await query.Skip((pageNumber - 1) * pageSize).Take(pageSize).ToListAsync();

        return new PagedList<T>(items, count, pageNumber, pageSize);
    }
}
```

PaginationParams.cs

```
namespace CourceProject.RequestHelpers;

public class PaginationParams
{
    private const int MaxPageSize = 50;
    public int PageNumber { get; set; } = 1;
    private int _pageSize { get; set; } = 6;
    public int PageSize
    {
        get => _pageSize;
        set=> _pageSize=value > MaxPageSize ? MaxPageSize: value;
        }
}
```

VehicleParams.cs

```
namespace CourceProject.RequestHelpers;

public class VehicleParams : PaginationParams
{
    public string OrderBy { get; set; }
    public string SearchTerm { get; set; }
    public string Manufacturers { get; set; }

}
```

User.cs

```
using Microsoft.AspNetCore.Identity;

namespace CourceProject.Models
{
    public class User : IdentityUser
    {

    }
}
```

Car.cs

```
namespace CourceProject.Entities
{
    public class Car
    {
        public int Id { get; set; }
        public string Manufacturer { get; set; }
        public string Model { get; set; }
        public string Description { get; set; }
        public int YearOfProduction { get; set; }
        public double AveragePrice { get; set; } // double
        public string Type { get; set; }
        public int QuantityInCountry { get; set; }
        public double EngineDisplacement { get; set; } // double
        public int EngineHorsepower { get; set; }
        public string PictureUrl { get; set; }
        public string PublicId { get; set; }
    }
}
```

VehicleExtension.cs

```
using System.Collections.Generic;
using System.Linq;
using CourceProject.Entities;
```

```
namespace CourceProject.Extensions;

public static class VehicleExtensions
{
    public static IQueryable<Car> Sort(this IQueryable<Car> query, string orderBy)
    {
        if (string.IsNullOrWhiteSpace(orderBy))
            return query.OrderBy(i => i.Manufacturer);

        query = orderBy switch
        {
            "price" => query.OrderBy(i => i.AveragePrice),
            "priceDesc" => query.OrderByDescending(i => i.AveragePrice),
            _ => query.OrderBy(i => i.Manufacturer)
        };

        return query;
    }

    public static IQueryable<Car> Search(this IQueryable<Car> query, string searTerm)
    {
        if (string.IsNullOrWhiteSpace(searTerm)) return query;

        var lowerCaseSearchTerm = searTerm.Trim().ToLower();

        var searchResult = query.Where(i => i.Manufacturer.ToLower().Contains(lowerCaseSearchTerm));

        if (searchResult.Any() == false)
            searchResult = query.Where(i => i.Model.ToLower().Contains(lowerCaseSearchTerm));

        return searchResult;
    }

    public static IQueryable<Car> Filter(this IQueryable<Car> query, string manufacturers)
    {
        var manufacturerList = new List<string>();
        // var typeList = new List<string>();

        if (!string.IsNullOrEmpty(manufacturers))
            manufacturerList.AddRange(manufacturers.ToLower().Split(",").ToList());

        query = query.Where(i => manufacturerList.Count == 0 || manufacturerList.Contains(i.Manufacturer.ToLower()));

        return query;
    }
}
```

UserDto.cs

```
namespace CourceProject.DTOs
{
    public class UserDto
    {
        public string Email { get; set; }
        public string Token { get; set; }
    }
}
```

UpdateCar.cs

```
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Http;

namespace CourceProject.DTOs;

public class UpdateCarDto
{
    public int Id { get; set; }
    [Required]
    public string Manufacturer { get; set; }
    [Required]
    public string Model { get; set; }
```

```
    [Required]
    public string Description { get; set; }
    [Required]
    public int YearOfProduction { get; set; }
    [Required]
    [Range(100, double.PositiveInfinity)]
    public double AveragePrice { get; set; } // double
    [Required]
    public string Type { get; set; }
    //[Required]
    [Range(0,double.PositiveInfinity)]
    public int QuantityInCountry { get; set; }
    [Required]
    public double EngineDisplacement { get; set; } // double
    [Required]
    public int EngineHorsepower { get; set; }

    public IFormFile File { get; set; }
}
```

RegisterDto.cs

```
namespace CourceProject.DTOs
{
    public class RegisterDto : LoginDto
    {
        public string Email { get; set; }
    }
}
```

LoginDto.cs

```
namespace CourceProject.DTOs
{
    public class LoginDto
    {
        public string Username { get; set; }
        public string Password { get; set; }
    }
}
```

CreateCarDto.cs

```
using System.ComponentModel.DataAnnotations;
using Microsoft.AspNetCore.Http;

namespace CourceProject.DTOs;

public class CreateCarDto
{
        [Required]
        public string Manufacturer { get; set; }
        [Required]
        public string Model { get; set; }
        [Required]
        public string Description { get; set; }
        [Required]
        public int YearOfProduction { get; set; }
        [Required]
        [Range(100, double.PositiveInfinity)]
        public double AveragePrice { get; set; } // double
        [Required]
        public string Type { get; set; }
        [Required]
        [Range(0,double.PositiveInfinity)]
        public int QuantityInCountry { get; set; }
        [Required]
        public double EngineDisplacement { get; set; } // double
        [Required]
        public int EngineHorsepower { get; set; }
        [Required]
        public IFormFile File { get; set; }
}
```

DBinitializer.cs

```
using CourceProject.Entities;
using Microsoft.AspNetCore.Identity;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using CourceProject.Models;

namespace CourceProject.Data
{
   public static class DBinitializer
   {
      public static async Task Initialize(CarsContext context, UserManager<User> userManager)
      {
         if (!userManager.Users.Any())
         {
            var user = new User
            {
               UserName ="bob",
               Email="bob@test.com"
            };

            await userManager.CreateAsync(user, "Pa$$w0rd");
            await userManager.AddToRoleAsync(user, "Member");


            var admin = new User
            {
               UserName ="admin",
               Email="admin@test.com"
            };

            await userManager.CreateAsync(admin, "Pa$$w0rd");
            await userManager.AddToRolesAsync(admin, new[] { "Admin", "Member" });
         }

         if (context.Cars.Any()) return;

         var cars = new List<Car>
         {
            new Car()
            {
               Manufacturer = "BMW",
               Model = "i395",
               Description = "The most beautiful car",
               YearOfProduction = 1995,
               AveragePrice = 45000,
               Type = "Sedan",
               QuantityInCountry = 999,
               EngineDisplacement = 2,
               EngineHorsepower = 250
            },
            new Car()
            {
               Manufacturer = "Audi",
               Model = "A6",
               Description = "The most beautiful car",
               YearOfProduction = 1995,
               AveragePrice = 45000,
               Type = "Sedan",
               QuantityInCountry = 999,
               EngineDisplacement = 2,
               EngineHorsepower = 250
            },
            new Car()
            {
               Manufacturer = "Mercedes",
               Model = "i395",
               Description = "The most beautiful car",
```

```
                YearOfProduction = 1995,
                AveragePrice = 45000,
                Type = "Sedan",
                QuantityInCountry = 999,
                EngineDisplacement = 2,
                EngineHorsepower = 250
            },
            new Car()
            {
                Manufacturer = "Ford",
                Model = "i395",
                Description = "The most beautiful car",
                YearOfProduction = 1995,
                AveragePrice = 45000,
                Type = "Sedan",
                QuantityInCountry = 999,
                EngineDisplacement = 2,
                EngineHorsepower = 250
            },

        };

        foreach (var car in cars)
        {
            context.Cars.Add(car);
        }

        context.SaveChanges();
    }
  }
}
```

CarsContext.cs

```
using CourceProject.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace CourceProject.Entities
{
  public class CarsContext : IdentityDbContext<User>
  {
    public CarsContext(DbContextOptions options) : base(options)
    {

    }

    public DbSet<Car> Cars { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
      base.OnModelCreating(builder);

      builder.Entity<IdentityRole>()
       .HasData(
          new IdentityRole { Name = "Member", NormalizedName = "MEMBER" },
          new IdentityRole { Name = "Admin", NormalizedName = "ADMIN" }
       );
    }
  }
}
```

CarsController.cs

```
using CourceProject.Entities;
using CourceProject.Extensions;
using CourceProject.RequestHelpers;
using System.Collections.Generic;
using System.Linq;
using System.Text.Json;
using System.Threading.Tasks;
using AutoMapper;
```

```
using CourceProject.DTOs;
using CourceProject.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.EntityFrameworkCore;

namespace CourceProject.Controllers
{
    public class CarsController : BaseApiController
    {
        private readonly CarsContext _context;
        private readonly IMapper _mapper;
        private readonly ImageService _imageService;

        public CarsController(CarsContext context, IMapper mapper, ImageService imageService)
        {
            _context = context;
            _mapper = mapper;
            _imageService = imageService;
        }

        [HttpGet]
        public async Task<ActionResult<PagedList<Car>>> GetCars([FromQuery] VehicleParams vehicleParams)
        {
            var query = _context.Cars
                .Sort(vehicleParams.OrderBy)
                .Search(vehicleParams.SearchTerm)
                .Filter(vehicleParams.Manufacturers)
                .AsQueryable();

            var vehicle = await PagedList<Car>.ToPagedList(query, vehicleParams.PageNumber,
                vehicleParams.PageSize);

            Response.AddPaginationHeader(vehicle.MetaData);

            return vehicle;
        }

        [HttpGet("statistics")]
        public async Task<IActionResult> GetCarsStatistics()
        {
            var popularity = await _context.Cars
                .GroupBy(vehicle => vehicle.Manufacturer.ToLower())
                .Select(g => new
                {
                    Manufacturer = g.Key,
                    Count = g.Count()
                }).ToArrayAsync();

            return Ok(new{popularity});
        }

        [HttpGet("{id}", Name = "GetCar")]
        public async Task<ActionResult<Car>> GetCar(int id)
        {
            var car = await _context.Cars.FindAsync(id);

            if (car == null)
            {
                return NotFound();
            }

            return Ok(car);
        }

        [HttpGet("filters")]
        public async Task<IActionResult> GetFilters()
        {
```

```
    var manufacturers = await _context.Cars.Select(i => i.Manufacturer).Distinct().ToArrayAsync();

    return Ok(new {manufacturers});
}

[Authorize(Roles = "Admin")]
[HttpPost]
public async Task<ActionResult<Car>> CreateCar([FromForm]CreateCarDto carDto)
{
    var car = _mapper.Map<Car>(carDto);

    if (carDto.File != null)
    {
        var imageResults = await _imageService.AddImageAsync(carDto.File);

        if (imageResults.Error != null)
            return BadRequest(new ProblemDetails {Title = imageResults.Error.Message});

        car.PictureUrl = imageResults.SecureUrl.ToString();
        car.PublicId = imageResults.PublicId;
    }

    _context.Cars.Add(car);

    var result = await _context.SaveChangesAsync() > 0;

    if (result)
        return CreatedAtRoute("GetCar", new {Id = car.Id}, car);

    return BadRequest(new ProblemDetails {Title = "Problem creating new car."});
}

[Authorize(Roles = "Admin")]
[HttpPut]
public async Task<ActionResult> UpdateCar([FromForm]UpdateCarDto carDto)
{
    var car = await _context.Cars.FindAsync(carDto.Id);
    if (carDto.File != null)
    {
        var imageResult = await _imageService.AddImageAsync(carDto.File);

        if (imageResult.Error != null)
            return BadRequest(new ProblemDetails{Title = imageResult.Error.Message});

        if (!string.IsNullOrEmpty(car.PublicId))
            await _imageService.DeleteImageAsync(car.PublicId);

        car.PictureUrl = imageResult.SecureUrl.ToString();
        car.PublicId = imageResult.PublicId;
    }

    if (car == null) return NotFound();

    _mapper.Map(carDto, car);

    var result = await _context.SaveChangesAsync() > 0;
    if (result) return Ok();

    return BadRequest(new ProblemDetails {Title = "Problem updating car."});
}

[Authorize(Roles = "Admin")]
[HttpDelete("{id}")]
public async Task<ActionResult> DeleteProduct(int id)
{
    var car = await _context.Cars.FindAsync(id);
    if (car == null) return NotFound();

    if (!string.IsNullOrEmpty(car.PublicId))
```

```
            await _imageService.DeleteImageAsync(car.PublicId);

        _context.Cars.Remove(car);

        var result = await _context.SaveChangesAsync() > 0;
        if (result) return Ok();

        return BadRequest(new ProblemDetails {Title = "Problem deleting the car."});
      }
   }
}
```

BaseApiController.cs

```
using Microsoft.AspNetCore.Mvc;

namespace CourceProject.Controllers
{
   [ApiController]
   [Route("api/[controller]")]
   public class BaseApiController : ControllerBase
   {

   }
}
```

AccountController.cs

```
using CourceProject.DTOs;
using CourceProject.Entities;
using CourceProject.Services;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;
using CourceProject.Models;

namespace CourceProject.Controllers
{
   public class AccountController : BaseApiController
   {
      private readonly UserManager<User> _userManager;
      private readonly TokenService _tokenService;

      public AccountController(UserManager<User> userManager, TokenService tokenService)
      {
         _userManager=userManager;
         _tokenService=tokenService;
      }

      [HttpPost("login")]
      public async Task<ActionResult<UserDto>> Login(LoginDto loginDto)
      {
         var user = await _userManager.FindByNameAsync(loginDto.Username);

         if (user == null || !await _userManager.CheckPasswordAsync(user, loginDto.Password))
            return Unauthorized();

         return new UserDto
         {
            Email = user.Email,
            Token = await _tokenService.GenerateToken(user)
         };
      }

      [HttpPost("register")]
      public async Task<ActionResult> Register(RegisterDto registerDto)
      {
         var user = new User { UserName = registerDto.Username, Email = registerDto.Email };
         var result = await _userManager.CreateAsync(user, registerDto.Password);

         if (!result.Succeeded)
         {
```

```
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError(error.Code, error.Description);
            }
            return ValidationProblem();
        }

        await _userManager.AddToRoleAsync(user, "Member");
        return StatusCode(201);
    }

    [Authorize]
    [HttpGet("currentUser")]
    public async Task<ActionResult<UserDto>> GetCurrentUser()
    {
        var user = await _userManager.FindByNameAsync(User.Identity.Name);

        return new UserDto
        {
            Email = user.Email,
            Token = await _tokenService.GenerateToken(user)
        };
    }
  }
}
```