

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
РОЗПІЗНАВАННЯ ПОШКОДЖЕНЬ ДЛЯ АГРЕГАТОРА ПОШУКУ
ОГОЛОШЕНЬ ПРО ПРОДАЖ АВТОМОБІЛІВ
RESEARCH AND SOFTWARE IMPLEMENTATION OF DEFECT
RECOGNITION METHODS FOR CAR SALE AGGREGATOR

Виконав: студент(ка) 2 курсу, групи КНТ-222м
Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

ЛАКТИОНОВ В.В.

(ПРІЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРІЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти магістр
 Спеціальність 122 Комп'ютерні науки
 (код і найменування)
 Освітня програма (спеціалізація) Системи штучного інтелекту
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 « » 20 23 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ЛАКТИОНОВА Валерія Віталійовича
 (ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів розпізнавання пошкоджень для агрегатора пошуку оголошень про продаж автомобілів. Research and Software Implementation of Defect Recognition Methods for Car Sale Aggregator

керівник проєкту (роботи) к.т.н., доцент ЛЬОВКІН Валерій Миколайович,
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «14» листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 1 грудня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Проєктування програмного забезпечення. 4. Основні рішення щодо реалізації програмного забезпечення. 5. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання «4» вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Дослідження проблеми	3-5 тижні	Розділ 2
4	Розробка архітектури програми	6 тиждень	Розділ 3
5	Розробка програми	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програми	9-10 тижні	Розділ 5
7	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	11 тиждень	Додатки
8	Захист роботи	12 тиждень	

Студент(ка)

(підпис) Валерій ЛАКТИОНОВ
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Валерій ЛЬОВКІН
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
111 с., 42 рис., 7 табл., 3 дод., 45 джерел.

РОЗПІЗНАВАННЯ ПОШКОДЖЕНЬ, АГРЕГАТОР ПОШУКУ,
ПРОДАЖ АВТОМОБІЛІВ, МОДЕЛЬ YOLOV5.

Об'єкт дослідження – процес розпізнавання пошкоджень автомобілів.

Предмет дослідження – програмне забезпечення для пошуку оголошень про продаж автомобілів.

Мета роботи – розробка програмного забезпечення пошуку оголошень про продаж автомобілів для підвищення зручності користувачів через агрегацію даних та інтегрування розпізнавання пошкоджень автомобілів у процес пошуку.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, середовища розробки Visual Studio 2022 та Visual Studio Code, мови програмування C#, Python та TypeScript, базове апаратне забезпечення – ноутбук Asus X509h (процесор Intel Core I5, 20 Гб RAM, 256 Гб SSD, Nvidia MX250, під керуванням ОС Microsoft Windows 11 Pro).

Наукова новизна роботи полягає в запропонованому методі автоматичного виявлення пошкоджень автомобілів на основі агрегації даних, який виконує модифікацію архітектури моделі розпізнавання, що дозволяє виявляти пошкодження на фотографіях, прикріплених до оголошень, та класифікувати оголошення про продаж автомобілів.

Результати. Розроблено програмне забезпечення для пошуку оголошень про продаж автомобілів з функцією розпізнавання пошкоджень.

Висновки. Розроблене програмне забезпечення для пошуку оголошень про продаж автомобілів з функцією розпізнавання пошкоджень прискорює та полегшує пошук транспортного засобу.

Галузь використання – сфера вторинного ринку автомобілів.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 111 pages, 42 figures, 7 tables, 3 appendixes, 45 sources.

DAMAGE RECOGNITION, SEARCH AGGREGATOR, SALE OF CARS, YOLOV5 MODEL.

The object of research is the process of recognizing car damage.

The subject of the study is software for finding ads for the sale of cars.

The goal of the work is to develop software for searching for car ads for sale to improve user experience through data aggregation and integration of car damage recognition into the search process.

Materials, methods and technical means: object-oriented programming, Visual Studio 2022 and Visual Studio Code development environments, C#, Python and TypeScript programming languages, basic hardware – Asus X509h laptop (Intel Core I5 processor, 20 GB RAM, 256 GB SSD, Nvidia MX250, running Microsoft Windows 11 Pro).

The scientific novelty of the work consists in the proposed method of automatic detection of car damage based on data aggregation, which performs a modification of the recognition model architecture, which allows detecting damage in photos attached to ads and classifying car sales ads.

The results. Developed software for searching ads for sale of cars with the function of damage recognition.

Conclusions. The developed software for searching ads for sale of cars with the function of recognition of damage makes it faster and easier to find a vehicle.

The field of use is the field of the secondary car market.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	10
1.1 Огляд програмного забезпечення.....	10
1.2 Порівняння аналогів	13
1.3 Висновки за розділом	16
2 Матеріали і методи.....	18
2.1 Задача розпізнавання пошкоджень автомобілів	18
2.2 Нейронні мережі на основі архітектури CNN	19
2.3 Метод автоматичного виявлення пошкоджень автомобілів на основі агрегації даних.....	21
2.4 Висновки за розділом 2	27
3 Проектування програмного забезпечення	28
3.1 Вибір засобів розробки.....	28
3.2 Моделювання програмного забезпечення на основі функціональних вимог.....	38
3.3 Проектування структури бази даних.....	42
3.4 Висновки за розділом 3	44
4 Основні рішення щодо реалізації програмного забезпечення	45
4.1 Порядок роботи програмного забезпечення	45
4.2 Рішення з реалізації вебзастосунку	50
4.3 Висновки за розділом 4	52
5 Експлуатація, тестування та експериментальне дослідження програми	53
5.1 Опис програми.....	53
5.2 Експлуатація програми.....	53
5.3 Експериментальне дослідження та тестування.....	62
5.4 Висновки за розділом 5	65
Висновки	67

Перелік джерел посилання	68
Додаток А Технічне завдання	73
Додаток Б Текст програми	77
Додаток В Слайди презентації.....	103

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CNN	– Convolutional Neural Network;
DOM	– Document Object Model;
IDE	– Integrated Development Environment;
JSON	– JavaScript Object Notation;
ORM	– Object-Relational Mapping;
SQL	– Structured Query Language;
UI	– User Interface;
VIN	– Vehicle Identification Number
YOLO	– You Only Look Once;
БД	– база даних;
ПЗ	– програмне забезпечення.
СУБД	– система управління базами даних;
ШІ	– штучний інтелект.

ВСТУП

Сучасні технології активно проникають у різні сфери повсякденного життя, вносячи значні зміни та покращення. Однією з таких сфер є ринок уживаних автомобілів, де важливим аспектом є прозорість та достовірність інформації про транспортні засоби, що надається. У цьому контексті постає завдання створення інноваційних інструментів, здатних підвищити ефективність процесу вибору автомобілів.

Також існує багато майданчиків оголошень про продаж автомобілів, кожен з яких може надавати унікальні пропозиції та можливості. Однак для потенційного покупця це може означати необхідність постійного переходу між різними платформами у пошуку відповідного варіанту. З цього випливає розробка агрегатора оголошень, здатного об'єднувати інформацію з різних джерел в одному місці, що стає важливим кроком у спрощенні та покращенні користувацького досвіду.

Метою дипломної роботи є розробка програмного забезпечення пошуку оголошень про продаж автомобілів для підвищення зручності користувачів через агрегацію даних та інтегрування розпізнавання пошкоджень автомобілів у процес пошуку.

У світлі постійного зростання інтересу до онлайн-ринку вживаних автомобілів створення інструменту, здатного автоматично виявляти ушкодження, стає вкрай актуальним. Область використання програмної системи охоплює широкий спектр споживачів, сприяючи більш ефективному та надійному досвіду купівлі вживаних автомобілів.

Таким чином, актуальність та практична значимість роботи полягає у створенні комплексного рішення, яке сприяє більш прозорому та пришвидшеному процесу вибору автомобіля на вторинному ринку, завдяки зручному та інтуїтивно зрозумілому інтерфейсу.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд програмного забезпечення

У сучасному світі стрімко розвивається сфера комп'ютерного зору та машинного навчання через велику затребуваність у різних галузях. Кожного дня з'являються нові можливості для автоматизації та покращення різноманітних завдань. Одним із таких завдань є оптимізація пошуку оголошень з продажу автомобілів, шляхом розпізнавання пошкоджень транспортних засобів.

В Україні існує багато платформ оголошень про продаж автомобілів, кожна з яких надає унікальні можливості та пропозиції. Ця різноманітність ускладнює пошук та порівняння пропозицій для потенційних покупців, вимагаючи переходу між різними джерелами.

В основному, покупцями авто ставиться за мету знайти автомобіль без дефектів. При перегляді сотень оголошень, витрачається багато часу на відфільтровування автомобілів з пошкодженнями.

Досить часто в оголошеннях про продаж автомобілів існує нестача достовірної інформації про технічний стан транспортного засобу. Покупці можуть мати справу з невизначеністю щодо наявності пошкоджень, що ускладнює прийняття обґрунтованих рішень.

Основним завданням є створення єдиної платформи, здатної отримувати дані із різних майданчиків оголошень. Це дозволить користувачам проводити пошук та аналіз пропозицій у зручному та одноманітному середовищі.

Допомогти з невизначеністю щодо наявності пошкоджень може розробка та впровадження системи машинного зору для автоматичного визначення пошкоджень на фотографіях автомобілів. Модуль розпізнавання пошкоджень покликаний підвищити прозорість та достовірність інформації про стан автомобілів в оголошеннях.

Розробка інструменту пошуку, який забезпечує користувачів інформацією про стан автомобілів на основі даних із модуля розпізнавання пошкоджень, значно має вплинути на покращення довіри потенційних покупців до представлених оголошень, підвищуючи тим самим ефективність процесу вибору автомобіля.

Метою роботи є розробка програмного забезпечення пошуку оголошень про продаж автомобілів для підвищення зручності користувачів через агрегацію даних та інтегрування розпізнавання пошкоджень автомобілів у процес пошуку. Для досягнення поставленої мети потрібно виконати порівняння аналогів та провести дослідження існуючих методів автоматичного виявлення пошкоджень автомобілів, що буде підставою для вибору оптимальних алгоритмів для впровадження модулю розпізнавання пошкоджень та пошуку оголошень.

Як було згадано вище, в нашій країні є декілька онлайн-майданчиків, що надають послуги з пошуку та продажу автомобілів. Досить важливо обрати для роботи найпопулярніші сайти для охоплення найбільшої кількості оголошень та найбільш заохочених клієнтів.

Вибір найпопулярніших майданчиків було здійснено з урахуванням охоплення оголошень та активності користувачів. Для аналізу популярності використовувалися дані з Google Trends, що надає інформацію про частоту запитів у пошуковій системі Google.

Google Trends – це сервіс, що надає дані про те, як часто визначені терміни пошуку вводяться в пошукову систему Google у певний період часу. Шляхом порівняння популярності запитів, пов'язаних із різними автомобільними майданчиками, можна визначити їхню відносну активність.

У статті [1] дослідники показали, що Google Trends може бути використаний для оцінки популярності сайтів з достатньою точністю. У своєму дослідженні вони використовували дані Google Trends для порівняння популярності сайтів "Amazon", "Walmart" та "Target". Результати

дослідження показали, що Google Trends може передбачити відносну популярність цих сайтів з точністю до 90%.

У статті [2] також було підтверджено, що Google Trends може бути використаний для оцінки популярності сайтів. У цьому дослідженні дослідники порівнювали точність Google Trends з точністю інших методів оцінки популярності сайтів, таких як аналіз посилань та аналіз трафіку. Результати дослідження показали, що Google Trends є одним з найбільш точних методів оцінки популярності сайтів.

Таким чином, наукові дослідження підтверджують, що Google Trends можна використовувати для визначення того, який сайт є популярнішим.

Для порівняння було обрано наступні майданчики оголошень: AutoRia, OLX, RST, Bon та Avtobazar. Результати наведені на рис. 1.1.

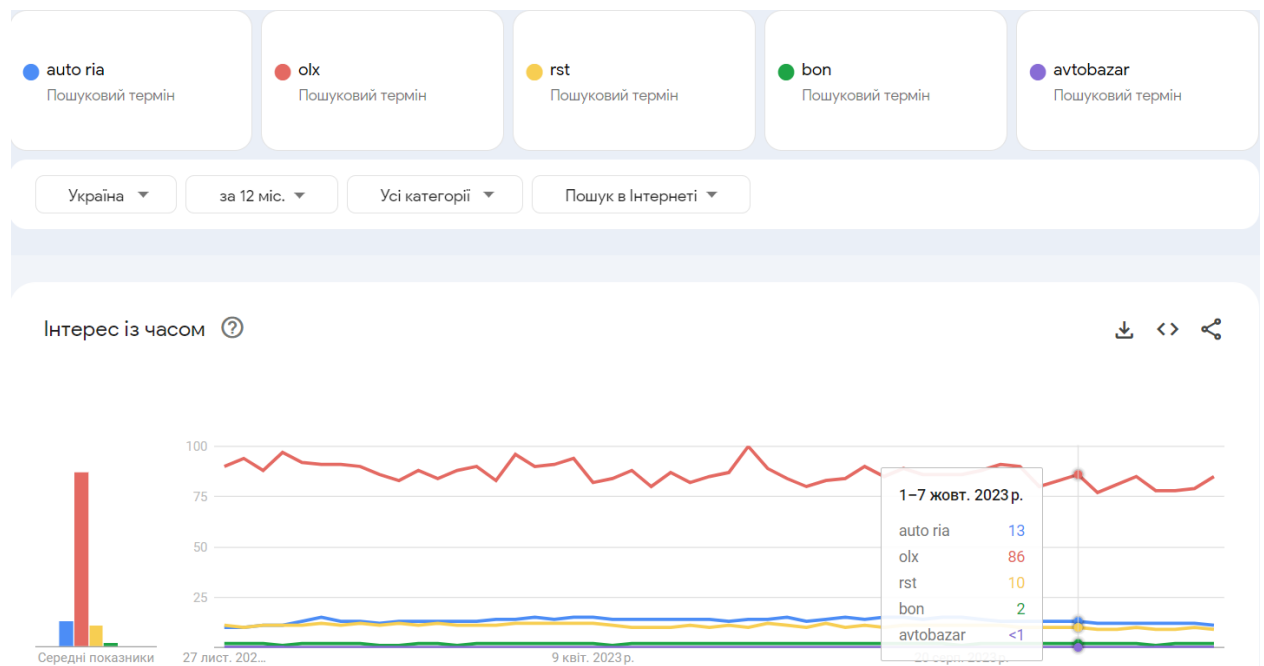


Рисунок 1.1 – Аналітика запитів у Google Trends

За результатами аналітики за останні 12 місяців видно, що майданчики OLX та AvtoRia займають лідируючі позиції. Порівняно з іншими, OLX має такий великий розрив через те, що цей майданчик багатoproфільний, а не призначений суто для автомобілів.

Вибір AutoRia та OLX обґрунтований їхньою високою популярністю та охопленням у сфері онлайн-продажів автомобілів в Україні. Використання даних Google Trends забезпечує кількісну основу для порівняння популярності цих майданчиків у різні періоди часу, що є важливим при аналізі динаміки змін на користь користувачів до певних майданчиків.

1.2 Порівняння аналогів

У сфері агрегаторів автомобільних оголошень існує декілька конкурентів, кожен з яких має свої особливості та переваги. Серед аналогів програмного продукту, що розробляється, виділяються два популярні агрегатори в Україні: АвтоПошук та AutoMoto.

1.2.1 Огляд сервісу АвтоПошук

АвтоПошук [3] надає користувачеві наступні можливості:

- пошук та фільтрація оголошень про продаж автомобілів;
- виявлення однакових оголошень;
- перегляд списку знайдених оголошень;
- перегляд детальної інформації по оголошенню;
- перехід на майданчики оголошень;
- збереження фільтру пошуку у збережені;
- збереження оголошень у збережені;
- керування збереженими фільтрами пошуку та оголошеннями.

Головна сторінка сервісу пошуку автомобілів АвтоПошук зображена на рис. 1.2.

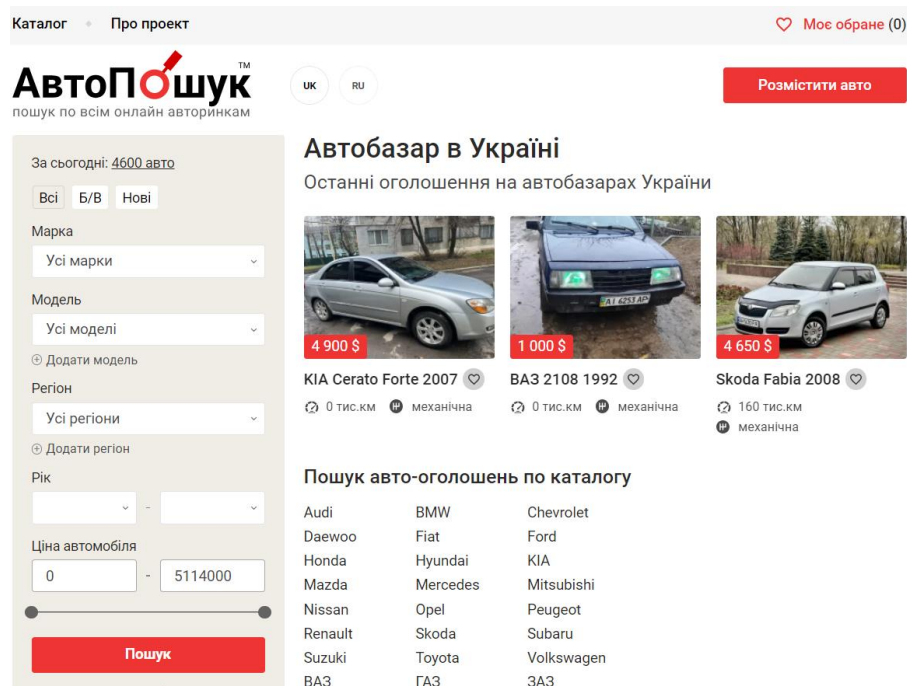


Рисунок 1.2 – Головна сторінка сервісу АвтоПошук

1.2.2 Огляд сервісу AutoMoto

AutoMoto [4] надає користувачеві наступні можливості:

- пошук та фільтрація оголошень про продаж автомобілів;
- виявлення однакових оголошень;
- перегляд списку знайдених оголошень;
- перегляд детальної інформації по оголошенню;
- перехід на майданчики оголошень;
- створення облікового запису;
- збереження оголошень у збережені;
- керування збереженими оголошеннями;
- перегляд відгуків та новин.

Головна сторінка сервісу пошуку автомобілів AutoMoto зображена на рис. 1.3.

The screenshot displays the AutoMoto.ua website. At the top, there is a navigation bar with the site logo, a search bar, and various filters. Below this is a secondary navigation bar with categories like 'Вживані авто', 'Нові авто', 'Автосалони', 'Каталоги Європи', 'Всі відгуки (321 617)', 'Новини', and 'Ще'. The main search area features a form with filters for vehicle type (All, Used, New), body type (Sedan, etc.), price range, and year range. Below the search form are two car listings, each with a '3 VIN-кодом' label and a photo of the vehicle.

Рисунок 1.3 – Головна сторінка сервісу AutoMoto

1.2.3 Порівняльний аналіз

На відміну від АвтоПошук та AutoMoto, розроблюване програмне забезпечення повинно мати інтегрований модуль розпізнавання пошкоджень на зображеннях автомобілів, що значно покращить інформативність оголошень і сприятиме довірі до представленої інформації.

Також програмна система буде мати функціональність підписки на повідомлення про появу нових оголошень за фільтрами пошуку через Telegram-бот. Це буде в нагоді тим, кому важливо автоматично дізнаватися про нові пропозиції на ринку для змоги якнайшвидшого початку комунікації з продавцем автомобіля.

У зв'язку з впровадженням технології розпізнавання пошкоджень і додатковою функціональністю, в поєднанні зі зручним та інтуїтивно зрозумілим інтерфейсом, програмне забезпечення, що розробляється, прагне залучити широку аудиторію користувачів, надаючи їм спрощений процес вибору автомобіля.

Порівняння ключових особливостей аналогів та програмного забезпечення CarFinder, що розробляється, наведено в табл. 1.1.

Таблиця 1.1 – Порівняння особливостей аналогів та розробки

Особливості	Аналоги		Розробка
	АвтоПошук	AutoMoto	CarFinder
Пошук та фільтрація оголошень	+	+	+
Виявлення однакових оголошень	+	+	+
Перегляд списку знайдених оголошень	+	+	+
Перегляд детальної інформації по оголошенню	+	+	+
Перехід на майданчики оголошень	+	+	+
Створення облікового запису	—	+	+
Збереження фільтру пошуку у збережені	+	—	+
Збереження оголошень у збережені	+	+	—
Керування збереженими	+	+	+
Перегляд відгуків та новин	—	+	—
Розпізнавання пошкоджень	—	—	+
Підписка на повідомлення про нові оголошення через Telegram-бот	—	—	+

1.3 Висновки за розділом

Можна виділити наступні завдання дослідження:

- проаналізувати існуючі вебсайти агрегаторів оголошень про продаж автомобілів;
- розробити метод автоматичного виявлення пошкоджень автомобілів на основі агрегації даних;
- виконати проєктування програмного забезпечення агрегатора пошуку оголошень про продаж автомобілів;

- реалізувати сервер та вебзастосунок агрегатора пошуку оголошень про продаж автомобілів;
- виконати експериментальне дослідження на основі проблеми автоматичного виявлення пошкоджень автомобілів.

Розглянувши сучасні тенденції на ринку старих автомобілів, було виявлено ключові проблеми, з якими стикаються потенційні покупці. Різноманітність майданчиків оголошень та невизначеність щодо стану автомобілів порушують питання необхідності інноваційних рішень. Поставлені завдання дослідження спрямовані на створення комплексного підходу до поліпшення досвіду користувача при виборі автомобіля, включаючи створення агрегатора оголошень і впровадження модуля розпізнавання пошкоджень з власним методом.

Вибір майданчиків оголошень AutoRia та OLX обґрунтований їх широким охопленням та популярністю серед користувачів в Україні. Використання Google Trends дозволяє об'єктивно порівняти їхню динаміку популярності.

Порівняльний аналіз існуючих аналогів, таких як АвтоПошук та AutoMoto, дозволяє виділити переваги продукту, що розробляється. Інтегрований модуль розпізнавання пошкоджень та повідомлення про нові оголошення ставить розроблюване програмне забезпечення в ряд інноваційних рішень. Ключові особливості, такі як широка функціональність і зручний інтерфейс, мають забезпечити оптимізований досвід користувача.

Отже, розробка інноваційного рішення, що поєднує агрегатор оголошень, модуль розпізнавання пошкоджень та сервіс повідомлень, направлена на те, щоб не тільки стати ефективним інструментом пошуку, а й надійним джерелом інформації для прийняття обґрунтованих рішень при виборі автомобіля.

2 МАТЕРІАЛИ І МЕТОДИ

У цьому розділі наведено матеріали та методи, використані для вирішення завдання розпізнавання пошкоджень автомобілів у агрегаторі пошуку автомобілів. Процес включав збір даних, модифікацію архітектури моделі розпізнавання зображень, навчання моделі та агрегацію даних з обраних джерел даних з розпізнаванням на основі навченої моделі. Метою було створення ефективної системи, здатної автоматично виявляти пошкодження автомобілів на основі агрегації даних.

2.1 Задача розпізнавання пошкоджень автомобілів

Автомобілі можуть постраждати від контакту з гострими предметами, дорожньо-транспортних пригод або будь-яких інших інцидентів, які пошкоджують кузов автомобіля, впливаючи на його початкову форму. Автоматизоване виявлення пошкоджень автомобіля стає все більш поширеним, і багато користувачів звертаються до цих складних систем. Ця технологія вдосконалюється, оскільки виявлення пошкоджень автомобіля за допомогою штучного інтелекту (ШІ) стає все більш поширеним [5].

Автоматизоване виявлення пошкоджень автомобілів є завданням класифікації, в рамках якої система аналізує зображення пошкоджень і привласнює їм відповідний клас – «Damage». Це завдання є частиною більш загального тренду у розвитку технологій, де ШІ використовується для автоматизації процесів, покращення точності та ефективності в галузі діагностики та оцінки стану автомобілів.

В основі цього завдання лежить концепція класифікації даних, де модель навчається на різноманітних зображеннях ушкоджень, виробляючи здатність визначення наявності пошкоджень. Це включає навчання моделі на різних варіантах ушкоджень, таких як подряпини, вм'ятини, тріщини та інші,

що забезпечує систему всебічною компетентністю в аналізі різноманітних автомобільних пошкоджень.

Таким чином, класифікація даних за класом «Damage» за допомогою методів штучного інтелекту демонструє високий потенціал у галузі автоматизованого виявлення пошкоджень.

2.2 Нейронні мережі на основі архітектури CNN

В даний час, згорткові нейронні мережі або Convolutional Neural Network (CNN) є одними з найефективніших методів комп'ютерного зору. Вони використовуються для таких завдань, як розпізнавання облич, розпізнавання об'єктів, класифікація зображень, сегментація зображень та багато іншого.

CNN є класом нейронних мереж, який спеціалізується на обробці даних, що мають сіткову топологію, наприклад, зображення. Цифрове зображення – це двійкове представлення візуальних даних. Воно містить серію пікселів, упорядкованих у вигляді сітки, яка містить значення пікселів, щоб позначити, наскільки яскравим і якого кольору повинен бути кожен піксель [6].

Архітектура CNN схожа на схему людського мозку. Так само, як мозок складається з мільярдів нейронів, CNN також мають нейрони, розташовані певним чином: таке розташування гарантує, що все поле зору охоплено, таким чином уникається проблема поетапної обробки зображень традиційних нейронних мереж, яким потрібно подавати зображення у вигляді фрагментів зі зниженою роздільною здатністю, а порівняно з класичними нейронними мережами CNN забезпечує кращу продуктивність із вхідними зображеннями [7].

Згорткові нейронні мережі відрізняються від інших форм штучних нейронних мереж тим, що замість зосередження на всій проблемній області використовують знання про конкретний тип вхідних даних. Це, у свою чергу,

дозволяє налаштувати набагато простішу мережеву архітектуру. Одна з ключових відмінностей полягає в тому, що нейрони на шарах CNN складаються з нейронів, організованих у три виміри, тобто просторову розмірність вхідних даних (висота та ширина) та глибина [8].

CNN може мати декілька рівнів, кожен з яких навчається виявляти різні характеристики вхідного зображення: фільтр або ядро застосовуються до кожного зображення, щоб створити результат, який стає все кращим і більш деталізованим після кожного шару, на нижніх рівнях фільтри можуть починатися як прості функції [7].

CNN отримала свою назву від математичної лінійної операції між матрицями, що називається згорткою, маючи в своїй структурі такі шари: згортковий, нелінійності, об'єднання та повнозв'язаний, при цьому згортковий і повнозв'язаний шари мають параметри, але шари об'єднання та нелінійності не мають параметрів [9].

З кожним шаром CNN зростає у своїй складності, ідентифікуючи більші частини зображення: попередні шари зосереджені на простих функціях, таких як кольори та краї, коли дані зображення просуваються між шарами CNN, вони починають розпізнавати більші елементи або форми об'єкта, поки нарешті не ідентифікують запланований об'єкт [10].

You Look Only Once (YOLO) – підхід до виявлення об'єктів, який використовує згорткові нейронні мережі для виявлення об'єктів, його головною перевагою є те, що він підтримує виявлення об'єктів у реальному часі, YOLO дивиться на все зображення лише один раз, і звідси назва «ви дивитесь лише один раз» [11].

YOLO використовує все зображення для прогнозування кожної обмежувальної рамки: прогноз об'єктів, що належать до кількох класів, виконується одночасно, в основному, зображення поділено на сітку $S \times S$, коли центр будь-якого об'єкта потрапляє в клітинку сітки, ця клітинка відповідає за виявлення об'єкта, будь-яку кількість обмежувальних прямокутників (bounding boxes) можна з упевненістю передбачити для

кожної клітинки сітки, обмежувальні рамки мають 5 прогнозованих значень, як-от (x, y, w, h, c) , які є координатами центру рамки відносно клітинки сітки, шириною та висотою прогнозованого відносно всього зображення та достовірністю відповідно, модель реалізована як CNN і з використанням набору даних PASCAL VOC [12].

YOLO має дві частини: частина, яка витягує характеристики із зображення – це згорткові шари, а друга частина – це повнозв'язні шари, які забезпечують вихідні ймовірності та координати, мережа складається з 24 згорткових шарів, за якими слідує 2 повнозв'язні шари, використовуються згорткові шари 1×1 , а потім 3×3 згорткові шари [12].

Отже, згорткові нейронні мережі стали важливим інструментом у сфері комп'ютерного зору, які використовуються для різноманітних завдань. Архітектура CNN, аналогічна організації людського мозку, дозволяє їй ефективно обробляти великі області вхідних даних, забезпечуючи кращу продуктивність порівняно з традиційними нейронними мережами. Подальший розвиток цих технологій, також інтегрованих у виявлення об'єктів в реальному часі, підтверджує їх значущість і перспективність в області ШІ. Згорткові нейронні мережі залишаються потужним інструментом для обробки візуальної інформації, відкриваючи нові можливості у сфері автоматичного аналізу та розпізнавання зображень.

2.3 Метод автоматичного виявлення пошкоджень автомобілів на основі агрегації даних

Можна виділити наступні етапи вирішення поставленої задачі розпізнавання пошкоджень автомобілів:

- збір даних;
- розмітка даних;
- аугментація даних;
- створення датасету;

- створення моделі розпізнавання;
- навчання моделі з модифікованою архітектурою;
- агрегація даних з обраних джерел даних з розпізнаванням на основі навченої моделі.

Для реалізації методу спочатку було зібрано набір даних, що містить зображення автомобілів з різними видами ушкоджень на платформі Roboflow [3]–[17]. Приклад зображень зі створеного датасету зображений на рис. 2.1.

На платформі Roboflow був створений власний проєкт із зібраних фотографій пошкоджених автомобілів. Далі набір даних було розмічено класом «Damage», шляхом виділення пошкоджених областей на фотографіях.

Важливим кроком у формуванні датасету є аугментація зображень. У статті [18] виділяють такі переваги використання аугментації:

- аугментація дозволяє збільшити розмір навчальної вибірки, що може призвести до покращення точності моделі;
- аугментація може допомогти зменшити перенавчання, оскільки вона дозволяє моделі бачити різні варіації одного й того самого об'єкта;
- аугментація може допомогти моделі краще узагальнювати нові дані, оскільки вона дозволяє моделі бачити різні умови, у яких можуть зустрічатися об'єкти.

Так як аугментація зображень є ефективним методом покращення точності та узагальнюючої здатності моделей машинного навчання, то було її застосовано, а саме: яскравість від -25 до +25 відсотків, експозицію від -20 до +20 відсотків. Таким чином, набір даних збільшився в 3 рази.

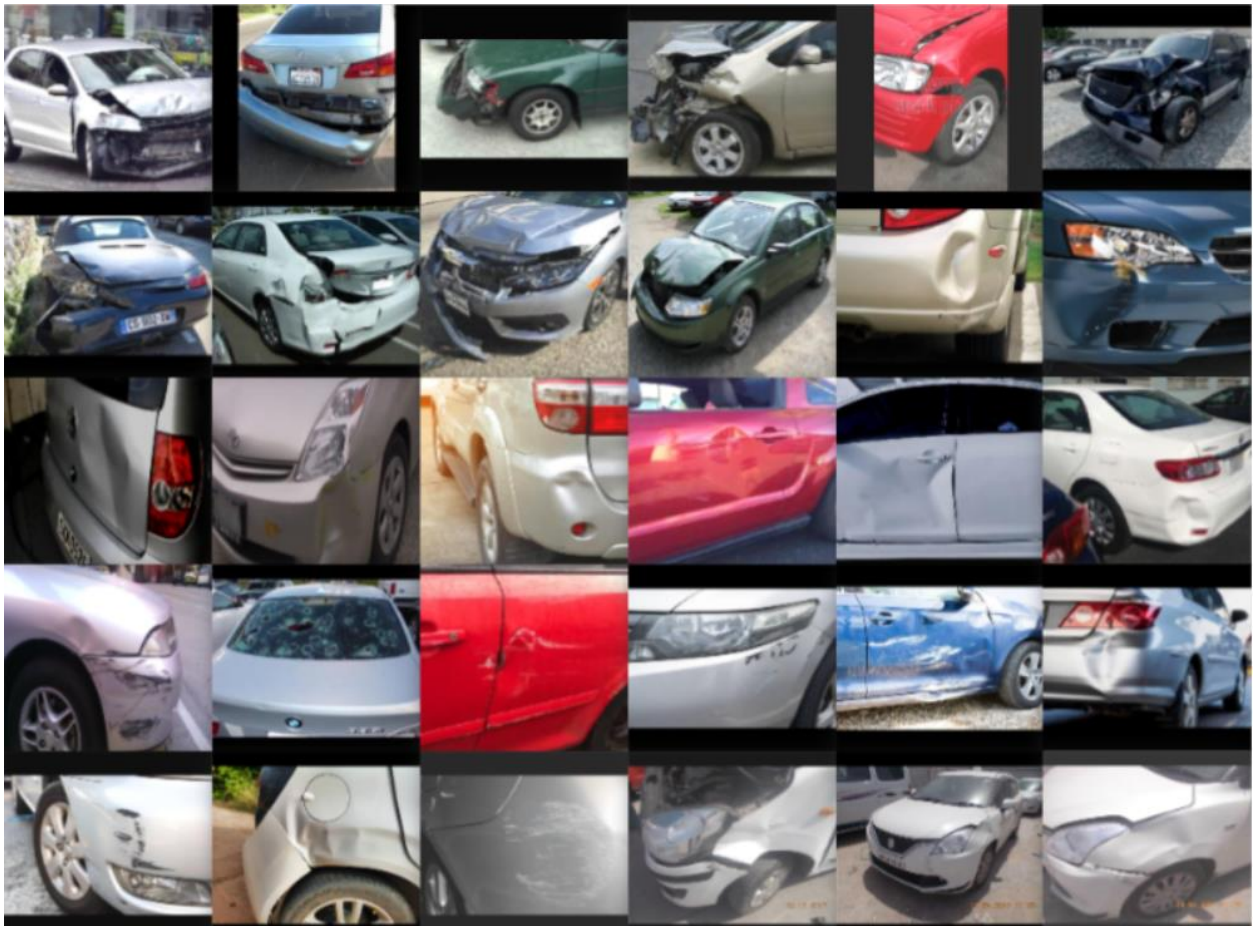


Рисунок 2.1 – Приклад зображень з датасету

Приклад розмічених зображень наведений на рис. 2.2.

Останнім кроком у формуванні датасету було його створення з наступними параметрами: використана функція автоорієнтації зображень та зміни розміру зображень до 640x640 зі збереженням пропорцій. В результаті чого було створено аугментований і, внаслідок цього, збільшений датасет. В подальшому його можна буде експортувати у потрібному форматі для навчання моделі.

Використання оптимізованої інтегрованої моделі для розпізнавання пошкоджень у агрегаторі автомобільних оголошень є вкрай критичним, адже система повинна мати змогу дуже швидко обробляти сотні фотографій за короткий час. Тому було прийнято рішення виконати оптимізацію архітектури моделі YOLO.



Рисунок 2.2 – Приклад розмічених зображень з датасету

У статті [19] доводиться, що використання моделі YOLO для розпізнавання пошкоджень автомобілів – це виправданий вибір серед моделей на основі CNN.

Мережа YOLO складається з трьох основних частин:

- backbone: згорткова нейронна мережа, яка об'єднує та формує характеристики зображення з різною деталізацією;
- neck: серія шарів для змішування та комбінування елементів зображення, щоб передати їх у передбачення;
- head: бере функції з neck та виконує кроки передбачення області та класу [20].

З усіх версій YOLO вибрано YOLO версії 5 (YOLOv5), оскільки вона надає збалансований набір характеристик: високу точність, відносно низьку кількість параметрів, високу швидкість навчання та розпізнавання, а також існує багато документації та матеріалів.

У статтях [21]–[24] описуються вдалі спроби модифікації архітектури YOLOv5. Автори застосовують різні зміни в частинах архітектури та отримують покращені моделі. Серед них – це збільшення точності, швидкості та зменшення моделі.

Перед початком власної реалізації по модифікації архітектури YOLOv5 було підготовлено середовище та датасет. У якості платформи для навчання було обрано Kaggle [25]. Підставою для вибору Kaggle, як середовища для навчання моделі, було те, що ця платформа надає зрозумілі часові рамки використання GPU, порівняно з Google Colab [26], адже потрібно буде робити багато експериментів та тестувань.

Щодо датасету, то він був експортований з платформи Roboflow у потрібному форматі для використання – «YOLO v5 PyTorch».

В якості основи було обрано варіацію YOLOv5 – YOLOv5m через доцільне співвідношення потрібної кількості навчальних даних до отримуваних характеристик моделі (точність та розмір).

Наступним кроком є навчання моделі на отриманому датасеті. Необхідні дії для тренування виконувались за прикладом, описаним у статті [27]. Для тренування необхідна наявність встановленої бібліотеки PyTorch та наявність GPU – це вже є у середовищі Kaggle. Далі було потрібно завантажити репозиторій yolov5, перейти в завантажену папку, встановити необхідні залежності, які описані у файлі requirements; у файлі data.yaml треба було прописати шляхи до папок з фото для тренування та перевірки, кількість класів та назву класу (в даному випадку це один клас «Damage»); далі у папку models треба було помістити конфігураційний файл архітектури моделі і тоді можна було приступати до навчання.

Спочатку було навчено модель з оригінальною архітектурою протягом 150 епох з такими налаштуваннями: img (розмір зображення) 640 та batch-size -1, що означає автоматичне його визначення та використання якомога більше пам'яті відеокарти. Модель навчилася за 10,8 годин.

Для покращення характеристик моделі було проведено серію експериментів, модифікуючи архітектуру моделі. Конфігурація моделі YOLOv5 відбувається у файлі *.yaml. Там описуються параметри, такі як кількість класів, глибина та ширина моделі; anchors, backbone та head [20].

Модифікована архітектура моделі YOLOv5 зазнала змін параметрів згорткових шарів усередині backbone для кращого отримання ознак на зображеннях автомобілів. Ці зміни включають зменшення ядер та кроку згортки в Conv шарах ([64, 6, 2, 2] => [64, 3, 2, 1]). Також додано заповнення (padding) у цих шарах для того, щоб розмір вихідного зображення був таким самим, як розмір вхідного зображення, щоб модель враховувала крайові пікселі, щоб уникнути урізання інформації ([128, 3, 2] => [128, 3, 2, 1]; [256, 3, 2] => [256, 3, 2, 1]; [512, 3, 2] => [512, 3, 2, 1]; [1024, 3, 2] => [1024, 3, 2, 1]).

Нова модель теж навчалася протягом 150 епох та з однаковими параметрами, як і перша модель, що зайняло 11,2 години.

В результаті, було отримано модифіковану архітектуру моделі YOLOv5. Модель з новою архітектурою отримала у своїх характеристиках збільшену точність та швидкість розпізнавання порівняно з моделлю оригінальної архітектури.

Таким чином, у моделі зменшилася кількість параметрів та обчислювальні витрати (у GFLOP), за рахунок чого збільшилася точність та зменшився час розпізнавання.

Також, увагу було приділено методу роботи моделі в поєднанні з програмним забезпеченням агрегатора оголошень. Для такої системи дуже критичним є швидкість розпізнавання пошкоджень на фотографіях, адже, система повинна буде одночасно мати справу з великою кількістю запитів і, відповідно, фотографій. Тому програмне забезпечення повинно мати не просто асинхронні методи, а і мати можливість паралельно виконувати певні функції. Таким чином, система має паралельно отримувати оголошення з майданчиків оголошень і потім паралельно передавати список посилань на фотографії з кожного оголошення в модуль розпізнавання, в якому має бути

паралельне завантаження відповідних фотографій та одночасне розпізнавання отриманих фотографій. І далі, по мірі отримання результатів розпізнавання, результати мають негайно передаватись у вебзастосунок для відображення користувачу.

2.4 Висновки за розділом 2

Матеріали та методи, представлені у цьому розділі, забезпечують базу для створення ефективного агрегатора оголошень з системою автоматичного виявлення пошкоджень автомобілів на основі агрегації даних. Комплексний підхід до завдання розпізнавання пошкоджень автомобілів, починаючи від збирання даних, до агрегації даних з обраних джерел даних з розпізнаванням на основі навченої моделі, дозволяють створити інноваційне рішення. Отримані результати є основою для інтеграції даної технології в автомобільний агрегатор з метою надання користувачам більш детальної та достовірної інформації про стан пропонованих автомобілів.

Наукова новизна роботи полягає в запропонованому методі автоматичного виявлення пошкоджень автомобілів на основі агрегації даних, який виконує модифікацію архітектури моделі розпізнавання, що дозволяє виявляти пошкодження на фотографіях, прикріплених до оголошень, та класифікувати оголошення про продаж автомобілів. Це дозволяє автоматизувати процес попередньої оцінки стану автомобіля і надавати користувачам більш інформативні дані про автомобілі, що продаються.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Створення ефективного та надійного агрегатора пошуку оголошень про продаж автомобілів потребує детального проєктування. Цей етап стане важливим фундаментом, надаючи не лише технічний каркас для реалізації, але й забезпечуючи оптимальну інтеграцію всіх модулів системи для ефективного функціонування та високої продуктивності розроблюваного агрегатора.

3.1 Вибір засобів розробки

Один із ключових етапів у забезпеченні успіху агрегатора пошуку оголошень про продаж автомобілів – це компетентний вибір засобів розробки. Детальний аналіз та обґрунтування вибору інструментів, мов програмування та технологічних рішень забезпечить оптимальну реалізацію завдань з розпізнавання пошкоджень та ефективне функціонування агрегатора в цілому.

Вибір засобів розробки впливає не тільки на процес створення програмного забезпечення, але і на його майбутню підтримку, масштабованість і здатність адаптуватися до змін ринку. Тому правильно підібрані засоби розробки не лише оптимізують процес створення програмного забезпечення, а й забезпечують стійкість та ефективність агрегатора у довгостроковій перспективі.

Щодо організації, то було вирішено використовувати систему контролю версій Git для зручності ведення історії розробки.

3.1.1 Вибір засобів для створення сервера

Серверна частина агрегатора, що є основною ланкою в обробці даних та взаємодії з клієнтськими запитами, відіграє вирішальну роль у

забезпеченні продуктивності та стабільності системи.

В даний час існує безліч мов програмування, які можна використовувати для створення серверних програм. До найпопулярніших з них належать:

- C# – об'єктно-орієнтована мова програмування з сильною типізацією, розроблена компанією Microsoft. C# є основною мовою програмування для платформи .NET і широко використовується для створення вебдодатків;

- Python – інтерпретована мова програмування загального призначення з динамічною типізацією. Python є однією з найшвидше зростаючих мов програмування у світі і широко використовується для створення вебдодатків, машинного навчання та штучного інтелекту;

- PHP – скриптова мова програмування з динамічною типізацією, спочатку розроблена для створення динамічних вебсторінок. PHP є однією з найпопулярніших мов програмування для веброзробки та має широку екосистему фреймворків та бібліотек;

- Java – об'єктно-орієнтована мова програмування з сильною типізацією, створена компанією Sun Microsystems. Java є однією з найпопулярніших мов програмування у світі та має широку екосистему інструментів та бібліотек [28].

C# має ряд переваг, які роблять його привабливим вибором для створення серверних додатків:

- є високопродуктивною мовою програмування, що робить її хорошим вибором для додатків, що вимагають високої продуктивності;

- є безпечною мовою програмування, яка підтримує безліч функцій безпеки, таких як типізація, управління пам'яттю та перевірки безпеки;

- має широку екосистему інструментів та бібліотек, які спрощують розробку та розгортання серверних додатків;

- надає високорівневі абстракції, що спрощує процес розробки, а також забезпечує надійність та ефективність у роботі з даними;

– підтримує кросплатформенну розробку, що забезпечує гнучкість у виборі операційної системи для розгортання [29].

Узагальнююча таблиця порівняння мов програмування сервера наведена в табл. 3.1.

Таблиця 3.1 – Узагальнююча таблиця порівняння мов програмування сервера

Критерії порівняння	Мова програмування сервера			
	C#	Python	Java	PHP
Статична типізація	+	-	+	-
Продуктивність	++	+	++	+
Зручність середовища розробки	++	+	+	+
Сприйняття коду	++	+	+	+

C# є потужною та універсальною мовою програмування, яка підходить для широкого спектру серверних програм. Тому для успішної реалізації сервера системи було вирішено використовувати мову програмування C# у поєднанні з ASP.NET Core Web API.

ASP.NET Core Web API – це фреймворк для створення вебслужб, що дозволяє створювати RESTful API. RESTful API – це вебслужби, які використовують HTTP-методи для доступу до даних [30].

ASP.NET Core Web API має ряд переваг, які роблять його привабливим вибором:

- є високопродуктивним фреймворком, який може обробляти велику кількість запитів;
- включає багато функцій безпеки, які допомагають захистити дані від несанкціонованого доступу;
- має широку екосистему інструментів та бібліотек, які спрощують розробку та розгортання вебслужб [31].

Вибір C# та ASP.NET Core Web API є фундаментальним кроком у забезпеченні ефективної та надійної серверної частини розроблюваного програмного забезпечення. Цей вибір забезпечує високу продуктивність, зручність розробки та широкі можливості інтеграції з іншими компонентами системи.

3.1.2 Вибір засобів для створення клієнта

На сьогоднішній день існує багато платформ для написання клієнтської програми. Найбільш популярними є:

- React – це бібліотека JavaScript, яка використовується для створення User Interface (UI). Вона заснована на концепції компонентів, що дозволяє створювати модульні та повторно використовувані елементи інтерфейсу. React також має вбудовану підтримку для керування станом, що полегшує оновлення інтерфейсу у відповідь на зміни даних;

- Vue.js – це ще одна бібліотека JavaScript для створення інтерфейсів користувача. Вона схожа на React, але має більш лаконічний синтаксис. Vue.js також має вбудовану підтримку для керування станом, але не так розвинена, як у React;

- Angular – це фреймворк JavaScript, який використовується для створення складних клієнтських програм. Він має більш багату функціональність, ніж React або Vue.js, але й складніше в освоєнні [32].

Основні переваги вибору React для написання клієнтської програми:

- має високу продуктивність;
- надає декларативний підхід до створення інтерфейсу користувача, що спрощує сприйняття та зміну структури компонентів. Це сприяє легкості у розробці, тестуванні та обслуговуванні клієнтського коду;
- має простий і зрозумілий синтаксис;
- підтримує компонентний підхід, дозволяючи створювати незалежні та перевикористовувані елементи інтерфейсу;

- використовує віртуальний Document Object Model (DOM) та реактивний підхід, що забезпечує ефективні оновлення інтерфейсу та підвищує продуктивність програми;

- має велику та активну спільноту розробників, що забезпечує підтримку та доступ до великої кількості ресурсів [33].

Узагальнююча таблиця порівняння засобів розробки вебклієнта наведена в табл. 3.2.

Таблиця 3.2 – Узагальнююча таблиця порівняння засобів розробки вебклієнта

Критерії порівняння	Засоби розробки вебклієнта		
	React	Vue.js	Angular
Галузь застосування	+	-	+
Продуктивність	++	+	++
Бібліотеки	++	+	+
Сприйняття коду	++	+	+

React – це бібліотека JavaScript, але React можна поєднати з Typescript. Typescript – це супермова, що розширює JavaScript статичною типізацією. Використання Typescript з React дозволяє покращити якість коду, підвищити його читабельність та зменшити кількість помилок.

Основні переваги використання Typescript:

- дозволяє явно вказувати типи даних для змінних, функцій та об'єктів. Це робить код зрозумілішим і полегшує його читання та супровід;
- може виявляти помилки в коді на етапі компіляції. Це дозволяє уникнути помилок, які можуть виникнути під час виконання програми;
- може оптимізувати код для більш ефективної роботи [34].

Узагальнююча таблиця порівняння мов програмування вебклієнта наведена в табл. 3.3.

Таблиця 3.3 – Узагальнююча таблиця порівняння мов програмування вебклієнта

Критерії порівняння	Мова програмування вебклієнта	
	TypeScript	JavaScript
Статична типізація	+	-
Продуктивність	++	+
Бібліотеки	+	+
Сприйняття коду	++	+

Таким чином, для забезпечення інтуїтивного та ефективного користувацького інтерфейсу агрегатора оголошень про продаж автомобілів було вирішено використовувати бібліотеку React у поєднанні з мовою TypeScript.

3.1.3 Вибір засобів для створення БД та взаємодії з нею

У розробці агрегатора оголошень про продаж автомобілів, ефективне зберігання та керування даними відіграє критичну роль. Існує багато варіантів організації бази даних (БД). Найбільш популярними є:

- Oracle – це комерційна база даних, яка пропонує широкий спектр функцій та можливостей. Однак вона є дорогою і складною в інсталяції та налаштуванні;

- PostgreSQL – це безкоштовна та відкрита база даних з об'єктно-реляційною моделлю даних. Вона є однією з найпопулярніших та технологічно передових баз даних;

- MySQL – це безкоштовна та відкрита база даних, однак вона не пропонує таку функціональність, як PostgreSQL, і може бути менш надійною;

- Microsoft SQL Server – це комерційна база даних, яка є гарним вибором для програм, що працюють у середовищі Microsoft. Однак вона може бути дорогою і складною в інсталяції та налаштуванні;

– MongoDB – це база даних NoSQL, яка не має схеми. Це робить її хорошим вибором для додатків, що працюють із неструктурованими даними. Однак вона може бути менш ефективною, ніж Structured Query Language (SQL) бази даних для додатків, що працюють із структурованими даними [35].

Було вирішено використовувати базу даних PostgreSQL, тому що вона є популярною, потужною, надійною та повністю безкоштовною реляційною базою даних.

Для взаємодії з базою даних було обрано фреймворк Entity Framework Core [36]. Він надає зручний та ефективний спосіб взаємодії з базою даних із програми. Використання EF Core спрощує роботу з даними, надаючи Object-Relational Mapping (ORM) та інструменти для виконання запитів до бази даних. Також, стає можливим використання Language Integrated Query (LINQ) [37]. Це дозволяє писати запити до даних на C# у стилі мови запитів, що робить код більш зрозумілим та зручним.

Існує два підходи у створенні БД: Data Base First (база даних та таблиці створюються у програмі керування БД і потім генеруються C# класи та контекст) та Code First (база даних, таблиці та контекст створюються у C# коді та потім генерується БД). Було обрано підхід Data Base First, тому що було зручніше створювати базу даних у системі керування БД для зменшення вірогідності допущення помилок та пришвидшення процесу створення бази даних. Також цей підхід забезпечує гнучкість у зміні схеми БД та дозволяє ефективно керувати її розвитком [38].

Вибір PostgreSQL та EF Core з підходом Data Base First забезпечує потужні інструменти для ефективної роботи з даними, управління змінами у структурі БД та забезпечує високу продуктивність системи.

3.1.4 Вибір засобів для модулю розпізнавання

Є декілька мов програмування, які підходять для реалізації модулю розпізнавання зображень. Найпопулярнішим є Python, але потрібно розглянути й інші варіанти, щоб бути впевненим у доцільності вибору. Існують наступні мови для полегшеного написання програм для взаємодії з моделями розпізнавання:

- Java є популярною мовою для розпізнавання зображень. Вона має схожі переваги з Python, включаючи популярність, підтримку спільноти та потужні бібліотеки для обробки зображень. Однак Java є більш складною мовою для використання, ніж Python;

- C++ є більш продуктивною мовою, ніж Python або Java. Однак, як і Java, C++ є більш складною мовою для використання [39].

Мова програмування Python є актуальним вибором для написання модуля розпізнавання зображень через ряд наступних переваг, які роблять її особливо придатним для цього завдання:

- є однією з найпопулярніших мов програмування у світі, що означає, що існує велика спільнота розробників та вдосталь документації;

- є відносно простою мовою для використання. Мова проста і лаконічна, що сприяє швидкій та зрозумілій реалізації алгоритмів;

- має широкий набір бібліотек, таких як TensorFlow, PyTorch, scikit-learn та інші, які надають потужні інструменти для реалізації алгоритмів машинного навчання [40].

Узагальнююча таблиця порівняння мов програмування модулю розпізнавання наведена в табл. 3.4.

Таблиця 3.4 – Узагальнююча таблиця порівняння мов програмування модулю розпізнавання

Критерії порівняння	Мова програмування модулю розпізнавання		
	Python	Java	C++
Продуктивність	+	++	++
Бібліотеки для машинного навчання	++	+	+
Автоматичне управління пам'яттю	++	++	-
Швидкість написання коду	++	+	-

Мова програмування Python є відмінним вибором для написання модуля розпізнавання зображень у програмі. Вона забезпечує не тільки доступ до передових інструментів машинного навчання, але й зручність у розробці та інтеграції до загальної архітектури агрегатора.

3.1.5 Вибір середовища розробки

Для розробки ASP.NET програм існує два основних середовища розробки:

– Visual Studio – це інтегроване середовище розробки або Integrated Development Environment (IDE) від Microsoft, яке є найбільш повним та функціональним середовищем для розробки ASP.Net Core додатків. Воно надає широкий спектр інструментів та функцій, які спрощують та прискорюють процес розробки;

– JetBrains Rider – це платна IDE від JetBrains, яка є альтернативою Visual Studio для розробки ASP.Net Core додатків. Вона пропонує широкий набір функцій та інструментів, які конкурують із Visual Studio [41].

Наступні переваги Visual Studio 2022 роблять її кращим середовищем для розробки ASP.Net Core додатків:

- надає повнофункціональне інтегроване середовище розробки для мови C# та технології ASP.NET Core;
- надає інтелектуальне автодоповнення коду, яке допомагає швидко та легко писати код;
- надає потужні інструменти налагодження, які допомагають виявляти та усувати помилки в коді;
- інтегрована з іншими інструментами та сервісами, наприклад GitHub.

Також існує багато середовищ розробки, які можна використовувати для створення програм на React. Деякі з найпопулярніших включають:

- Visual Studio Code – це безкоштовне та відкрите середовище розробки від Microsoft, яке підтримує багато мов програмування. VS Code пропонує широкий спектр функцій, які роблять його добрим вибором для розробки React додатків;
- Atom – це безкоштовний та відкритий текстовий редактор, який також підтримує розробку React-додатків. Atom має простий інтерфейс, що налаштовується, а також широкий спектр плагінів, які можна використовувати для розширення його функціональності;
- Reactide – це платне середовище розробки, яке спеціально призначене для розробки React-додатків. Reactide пропонує багато функцій, які спрощують та прискорюють розробку [42].

У підсумку Visual Studio Code було обрано як найкращий вибір для розробки React-додатків з таких причин:

- підтримка Hot Reload. Hot Reload – це функція, яка дозволяє вносити зміни до коду програми без необхідності перезапустити програму;
- має великий каталог плагінів, які можна використовувати для розширення функціональності, наприклад, ESLint, Prettier та React DevTools;
- має велику та активну спільноту розробників, яка створює та підтримує плагіни, розширення та інші ресурси [43].

Також Visual Studio Code добре підходить для написання коду мовою програмування Python, що стане у нагоді, при розробці модулю розпізнавання пошкоджень.

Таким чином, для забезпечення ефективної та комфортної розробки серверної, клієнтської частин та модуля розпізнавання в рамках агрегатора оголошень про продаж автомобілів, було вирішено використовувати середовища розробки Visual Studio 2022 для серверної частини та Visual Studio Code для клієнтської частини та модуля розпізнавання.

3.2 Моделювання програмного забезпечення на основі функціональних вимог

Процес розробки програмного забезпечення починається з моделювання, яке включає аналіз і проєктування на основі функціональних вимог. Функціональні вимоги визначають, які функції та завдання мають виконуватися програмою, і є основою для побудови моделей.

До розроблюваного програмного забезпечення поставлені наступні функціональні вимоги:

- пошук авто за фільтрами пошуку: користувач повинен мати можливість вказати різноманітні критерії для пошуку оголошень, а саме: бренд, модель, рік випуску від та до, тип коробки передач, тип кузова, пробіг від та до, об'єм двигуна від та до, ціна від та до, тип палива, область, населений пункт та сортування;
- виявлення однакових оголошень: програма повинна бути спроможна виявляти однакові оголошення, розміщені на майданчиках оголошень;
- розпізнавання зображень для виявлення пошкоджень: програма повинна мати модуль, який буде завантажувати та аналізувати зображення по кожному знайденому оголошенню. Результати виявлення пошкоджень повинні бути відображені користувачу;

- перегляд списку знайдених оголошень: користувач повинен мати змогу переглядати список оголошень автомобілів з різних майданчиків за вказаними фільтрами пошуку;
- перегляд оголошення: користувач повинен мати можливість переглядати детальну інформацію по оголошенню, таку як фотографії, опис, параметри та додаткові параметри, вказані продавцем на відповідному майданчику оголошень;
- перехід на майданчик оголошення: користувач повинен мати можливість перейти на відповідні майданчики оголошення, на яких було знайдено певний автомобіль;
- реєстрація та авторизація: користувач повинен мати можливість виконати реєстрацію облікового запису та авторизацію для отримання додаткової функціональності;
- збереження фільтру пошуку у збережені: користувач повинен мати змогу зберігати часто використовувані фільтри пошуку у збережені для швидкого заповнення фільтрів;
- підв'язка до Telegram-боту: користувач повинен мати змогу підв'язати свій обліковий запис до Telegram-боту для отримання додаткової функціональності;
- підписка на повідомлення про нові оголошення через Telegram-бот: у разі прив'язки облікового запису до Telegram-боту користувач повинен мати можливість зберегти фільтр пошуку до підписок для отримання повідомлень через Telegram-бот про появу нових оголошень;
- управління збереженими фільтрами та підписками: користувач повинен мати змогу видаляти непотрібні збережені фільтри пошуку та підписки за фільтрами пошуку.

На рис. 3.1 зображено узагальнену діаграму прецедентів розроблюваної програмної системи. Ця діаграма відображає всі основні функціональні вимоги, які були сформульовані для майбутнього програмного забезпечення.

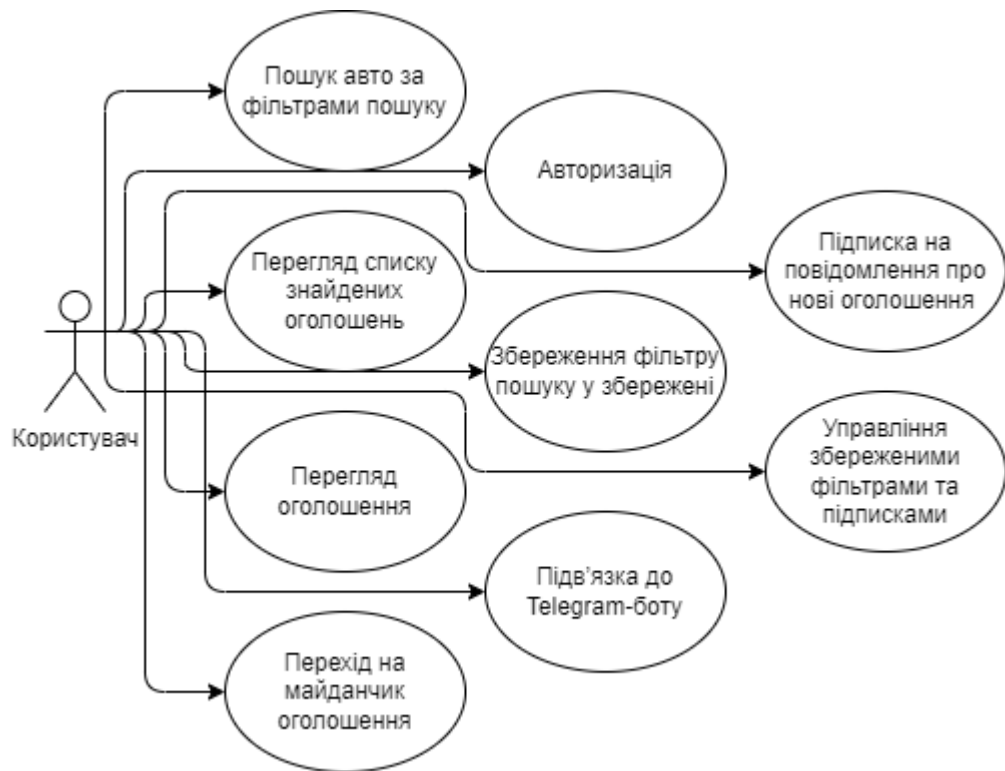


Рисунок 3.1 – Узагальнена діаграма прецедентів

На основі поставлених функціональних вимог було вирішено розроблювати клієнт-серверне програмне забезпечення. Клієнт-серверна архітектура є моделлю взаємодії між комп'ютерами, в якій сервер надає ресурси або послуги, а клієнти використовують ці ресурси. Ця модель широко застосовується у розробці програмного забезпечення, включаючи вебзастосунки, та забезпечує ефективний розподіл завдань між сервером та клієнтами [44].

До основних переваг клієнт-серверної архітектури можна віднести те, що сервер надає загальні ресурси та послуги для множини клієнтів, що дозволяє ефективніше використовувати обчислювальні та мережеві ресурси.

Використання такого підходу дає можливим використовувати в якості клієнтського програмного забезпечення вебзастосунк. Таким чином, нівелюється необхідність встановлення програми на кожному пристрої клієнта, адже можна отримати доступ до функціональності на сайті, відкритому в браузері.

Структурна схема розроблюваного програмного забезпечення зображена на рис. 3.2.

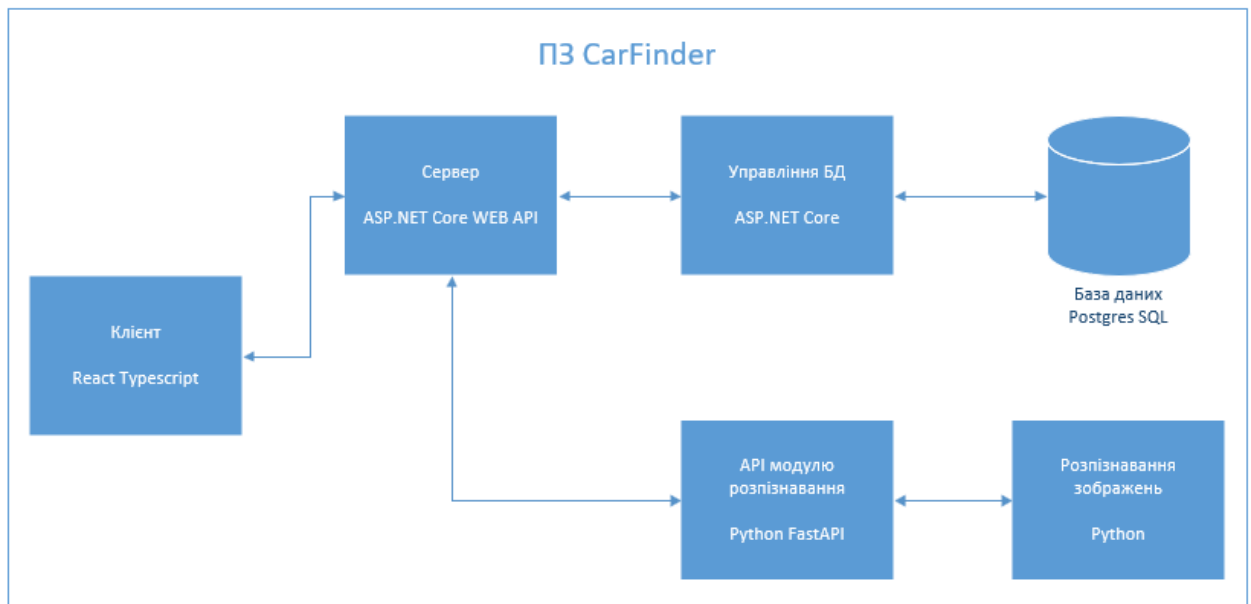


Рисунок 3.2 – Структурна схема програмного забезпечення

На основі виконаного вибору засобів для розробки, представленого у розділі 3.1, випливає, що сервер має розроблятися мовою програмування C# та з використанням технології ASP.NET Core WEB API, клієнт має реалізовуватися мовою Typescript в поєднанні з бібліотекою React, модуль розпізнавання пошкоджень – мовою Python, а в якості сховища даних буде використано базу даних PostgreSQL.

Для забезпечення функціональності розпізнавання пошкоджень було вирішено створити окремий модуль мовою програмування Python, тому що цією мовою можна зручно реалізовувати взаємодію з моделями розпізнавання зображень, на відміну від C#.

Модуль розпізнавання має надавати API за допомогою бібліотеки FastAPI. Таким чином сервер зможе комунікувати з ним.

Взаємодія з базою даних має відбуватися через проміжний проєкт. В ньому буде знаходитися контекст бази даних та моделі, відповідно до структури бази даних.

3.3 Проєктування структури бази даних

Проєктування бази даних є процесом створення та визначення структури бази даних, яка зберігатиме, управлятиме і забезпечуватиме доступ до даних в інформаційній системі. Цей процес включає визначення сутностей, їх атрибутів, а також відносин між ними, щоб забезпечити ефективне і надійне зберігання даних.

Проєктування схеми бази даних відбувалося на ресурсі, який дозволяє створювати ER діаграму бази даних з подальшим експортуванням у SQL скрипт, який буде використаний для створення бази даних у системі управління базами даних (СУБД) PostgreSQL. Це значно спрощує процес проєктування та створення бази даних.

Структурна схема бази даних зображена на рис. 3.3. Було виділено наступні сутності:

- користувач – містить інформацію про користувача, таку як email, пароль, id ролі та ідентифікатор Telegram чату;
- роль – містить інформацію роль у системі;
- вподобаний пошук – містить інформацію про збережені фільтри пошуку: ідентифікатор користувача та фільтри пошуку у форматі JavaScript Object Notation (JSON);
- підписка на повідомлення – містить інформацію про збережену підписку на повідомлення через Telegram-бот про нові оголошення, містить ідентифікатор користувача, фільтри пошуку у форматі JSON та дату останнього повідомленого оголошення.

Для наведених сутностей можна виділити наступні атрибути:

- користувач – Id, Email, Password, RoleId та TelegramChatId;
- роль – Id та Name;
- вподобаний пошук – Id, UserId та SearchJson;
- підписка на повідомлення – Id, UserId, SearchJson та LastAdDate.

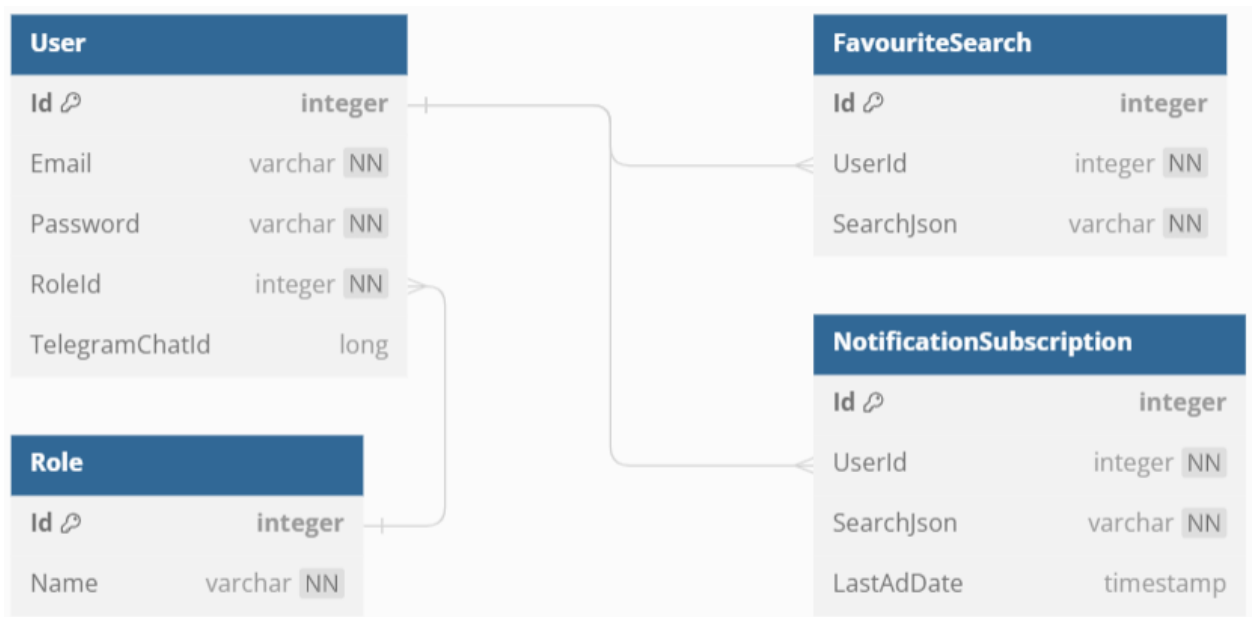


Рисунок 3.3 – Структурна схема бази даних

Сутність «користувач» відноситься до сутностей «вподобаний пошук» та «підписка на повідомлення» як один до багатьох, так само, як і сутність «роль» до сутності «користувач».

Так як робота агрегатора пошуку оголошень заснована не на постійному моніторингу та збереженню оголошень, а на пошуку в реальному часі, при запиті клієнта, то немає необхідності у зберіганні величезних обсягів даних про оголошення у базі даних.

На рис. 3.4 наведена структурна схема основних моделей, які будуть основними в роботі агрегатора. На ній можна побачити модель **CarSearchRequest**, яка являє собою фільтри пошуку. Модель **CarAdShortResponse** повинна містити основну інформацію по оголошенню і буде використовуватися при отриманні даних з майданчиків оголошень та показу списку оголошень. Модель **CarAdResponse** наслідує всі поля з моделі **CarAdShortResponse** та містить додаткові поля, щоб містити всю необхідну інформацію, отриману з майданчиків оголошень та показу на сторінці деталей по оголошенню. Інші моделі являються допоміжними, які будуть містити ідентифікатори оголошень, посилання на оголошення, фотографії, параметри та розширені параметри.

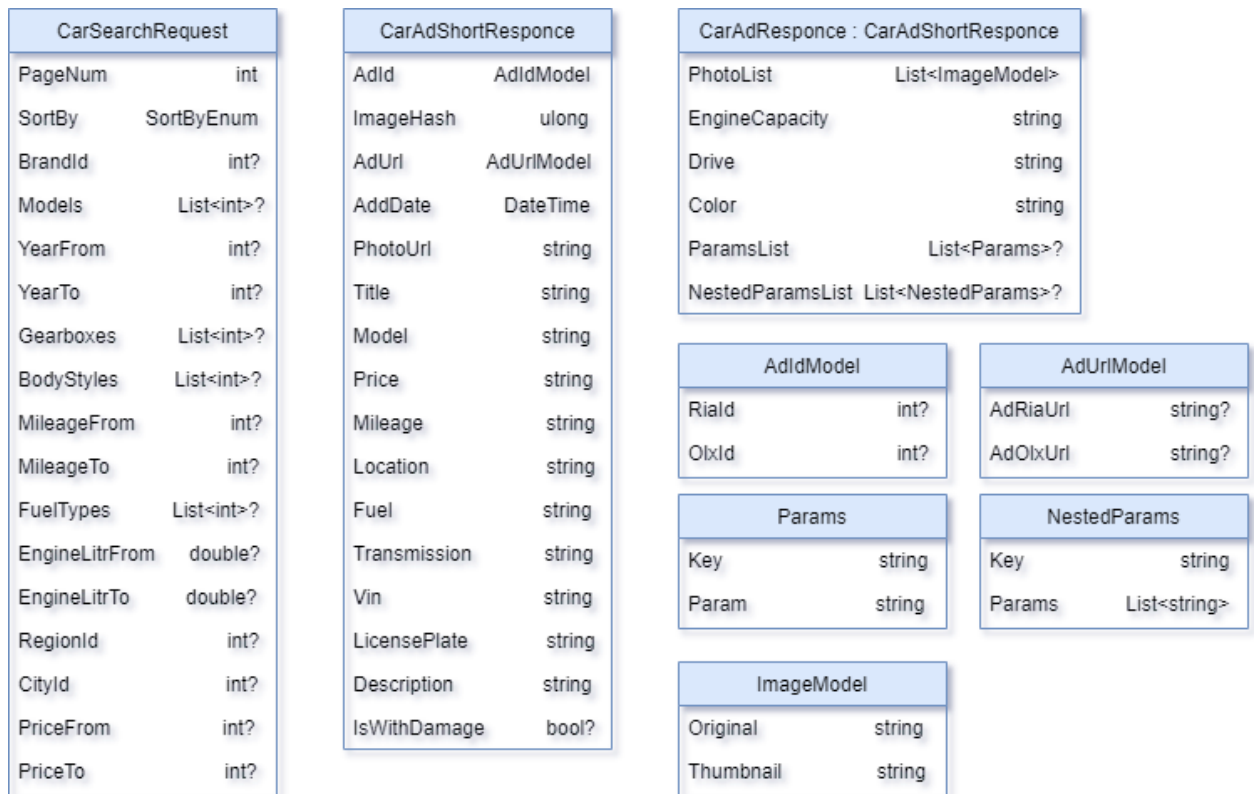


Рисунок 3.4 – Структурна схема основних моделей

3.4 Висновки за розділом 3

У ході проєктування програмного забезпечення для агрегатора оголошень про продаж автомобілів було проведено ретельний аналіз та ухвалено ключові рішення, що охоплюють різні аспекти розробки. Процес прийняття рішень був заснований на стратегічному підході до вибору технологій та інструментів, спрямованому на забезпеченні високої продуктивності, надійності та зручності використання кінцевих користувачів.

Також було проведено проєктування структури бази даних, що є важливим етапом у розробці агрегатора оголошень про продаж автомобілів, який визначає ефективне управління та взаємодію з даними.

Підхід до проєктування програмного забезпечення базується на балансі між сучасними технологіями та потребами проєкту. Представлені рішення сприяють створенню високопродуктивної, гнучкої та інноваційної системи агрегатора оголошень про продаж автомобілів.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Порядок роботи програмного забезпечення

Фундаментальним завданням сервера програмного забезпечення є пошук оголошень про продаж автомобілів на майданчиках оголошень AutoRia та OLX. Так як платформа OLX не має публічного API, а AutoRia має, але безкоштовно надає обмежену кількість запитів, було прийнято рішення використовувати парсинг відповідних сайтів. Для цього потрібно формувати рядок пошуку з відповідними параметрами за фільтрами пошуку та парсити відповідь сайту на HTTP запит.

Для початку потрібно було створити та уніфікувати конфігурації для фільтрів пошуку, таких як списки брендів, моделей авто, типів кузова, палива, коробок передач; областей та населених пунктів. За основу було взято конфігурації з API AutoRia [45], які було збережено у форматі json. Вони мають наступну структуру: Id (для запиту) та Name (для відображення користувачу). Дочірні конфігурації такі як, наприклад, моделі за брендами автомобілів мають таку структуру: ParentId (id батьківського компонента) та список Items зі звичайними компонентами.

Далі за допомогою парсингу сайту OLX визначались конфігурації, і була створена програма, яка зіставленням згенерувала відповідні конфігурації. Для запитів для отримання оголошень на OLX використовуються не ідентифікатори параметрів, а строкові назви цих параметрів. Тому конфігурації мають таку ж структуру, як і для AutoRia, але у полях Id та ParentId містяться ідентифікатори параметрів з AutoRia, а у полі Name міститься рядок для пошуку.

Так як конфігурації фільтрів пошуку будуть використовуватися досить часто, то для пришвидшення швидкодії сервера, конфігурації завантажуються у відповідний кеш, щоб кожного разу не звертатися до файлів.

Для пошуку оголошень було розроблено відповідний сервіс. При надходженні запиту на пошук оголошень викликається метод з цього сервісу, в якому здійснюється паралельний пошук на майданчиках оголошень списку оголошень за фільтрами пошуку від користувача. Після отримання знайдених оголошень відбувається паралельне створення хешу для головного фото по кожному оголошенню за допомогою сервісу для хешування зображень та використання алгоритму `PerceptualHash` з бібліотеки `CoenM`. Потім відбувається порівняння списку оголошень на наявність дублікатів, таким чином залишаються унікальні оголошення, а для деяких встановлюється по два посилання на відповідні майданчики оголошень. Передостаннім кроком у процесі пошуку є запуск процесу розпізнавання пошкоджень. А останнім – сортування списку оголошень за вподобанням користувача та повернення йому списку для відображення на сайті.

Запущений процес розпізнавання відбувається у відповідному сервісі. В ньому паралельно визивається API модулю розпізнавання пошкоджень для кожного оголошення. Після отримання запиту на розпізнавання модуль паралельно завантажує всі фотографії по оголошенню та потім одночасно запускає розпізнавання всіх завантажених фотографій по оголошенню.

По мірі надходження результатів розпізнавання пошкоджень сервіс розпізнавання передає результат клієнта за допомогою технології `SignalR`.

Описаний процес пошуку оголошень відображає функціональна схема, зображена на рис. 4.1.

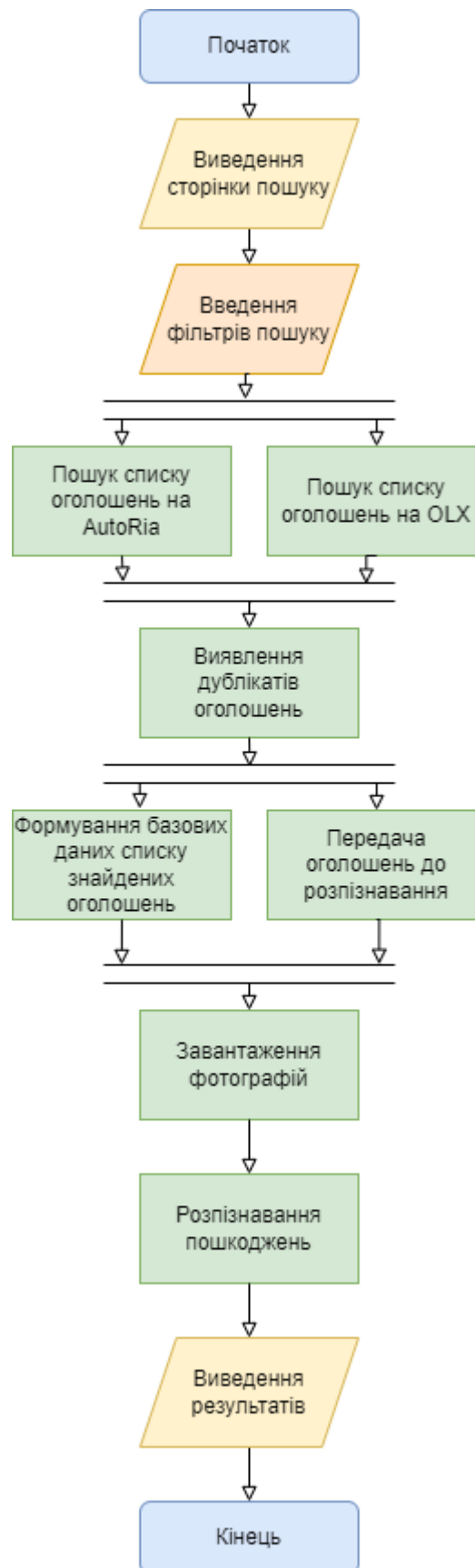


Рисунок 4.1 – Функціональна схема процесу пошуку оголошень

Відповідно до функціональних вимог, користувач може отримати можливість отримувати повідомлення про нові автомобілі через Telegram-бот після реєстрації. Для забезпечення безпеки програмного забезпечення, при створенні користувача його пароль не записується до бази даних у відкритому вигляді, а створюється хеш паролю за допомогою функції для хешування паролів, яка використовує бібліотеку BCrypt та метод хешування SHA384. Таким чином, ні адміністратор системи, ні зловмисник не зможе отримати оригінальний пароль. А завдяки принципам хешування, навіть для однакових паролів, створюється унікальний хеш, що унеможливорює частотний аналіз і значно підвищує безпеку програмного забезпечення.

Для підтримання функціональності повідомлень через Telegram-бот, було створено відповідний сервіс. В ньому, при запуску сервера, запускається сам Telegram-бот та очікує повідомлення про підв'язку нового користувача та надає функціональність для відправлення повідомлень користувачам. Підв'язка до боту відбувається наступним чином: користувачу пропонується пройти цей процес, він, за спеціальним посиланням, переходить до боту, натискає кнопку «Старт» і боту відправляється спеціальна команда, яка містить ідентифікатор акаунту користувача у системі. Сервіс боту приймає повідомлення, шукає користувача за відповідним ідентифікатором, якщо користувач дійсно зареєстрований, то береться ідентифікатор з Telegram користувача та зв'язується з акаунтом користувача у системі. Тепер, знаючи отриманий ідентифікатор з Telegram, можна буде відправляти користувачу повідомлення.

Для того щоб було що відправляти, було створено сервіс для повідомлень, який через задані проміжки часу виконує пошук оголошень по підпискам користувачів і, якщо є оголошення, свіжіші за дату останнього повідомленого оголошення, то формується повідомлення та відправляються користувачам.

Багато уваги було приділено різним загальним аспектам. Було створені middleware для глобального логування запитів та відповідей, а

також для глобального оброблення виключних ситуацій. Такий підхід значно зменшує дублювання коду та робить його чистішим. Також, було реалізовано можливість гнучких налаштувань сервера в файлі appsettings. Наприклад, можна налаштовувати рядок підключення до бази даних, змінити інтервал для сервісу повідомлень та багато іншого.

На рис. 4.2 зображена діаграма основних класів сервера програмного забезпечення.



Рисунок 4.2 – Діаграма основних класів сервера

Структура проєкту сервера програмного забезпечення наведена на рисунку 4.3.

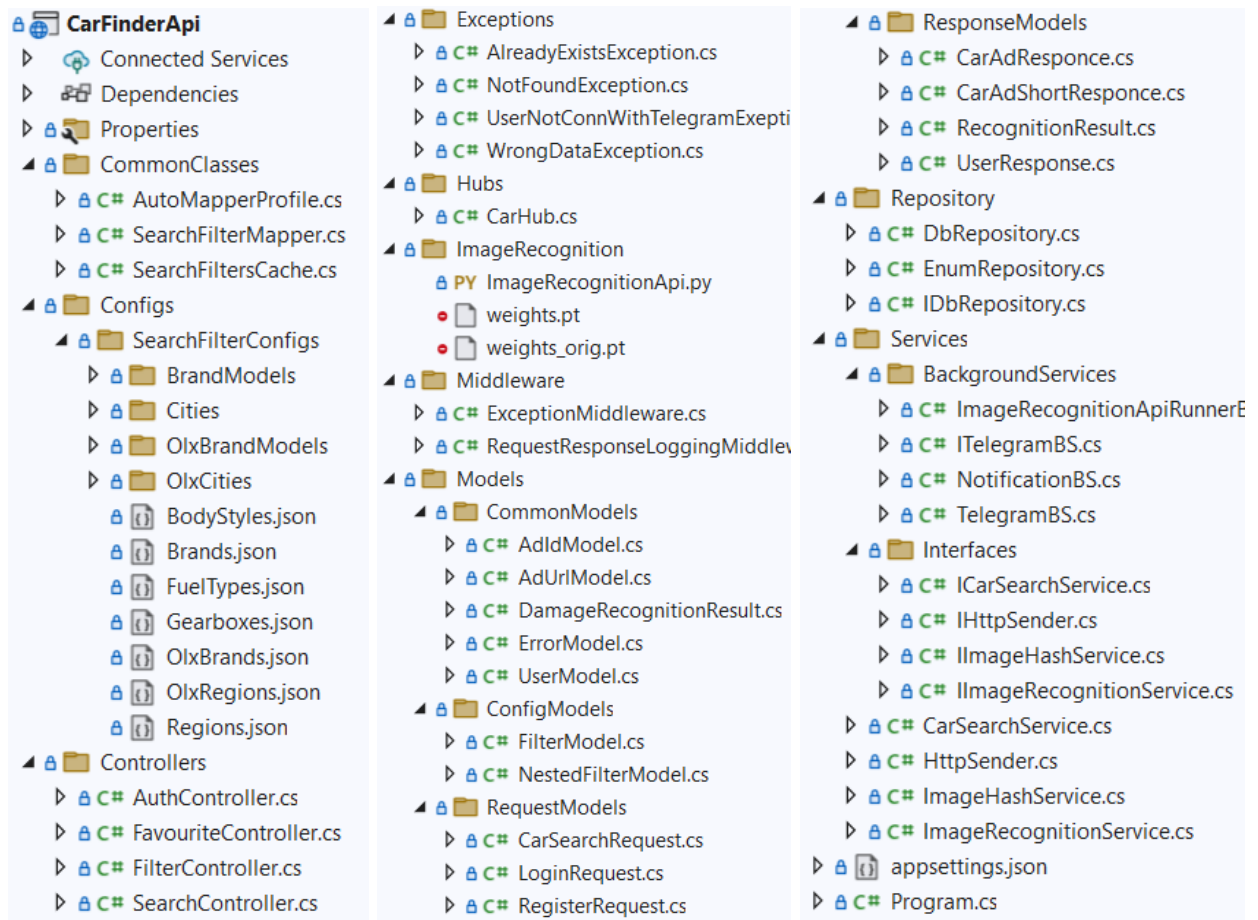


Рисунок 4.3 – Структура проєкту сервера

4.2 Рішення з реалізації вебзастосунку

Розробка вебзастосунку для агрегатора оголошень про продаж автомобілів вимагала прийняття цілого ряду рішень, спрямованих на забезпечення зручності використання, високої продуктивності та безпеки. Нижче приведені ключові аспекти реалізації, використані в процесі створення вебдодатку.

Для створення вебзастосунку було обрано стек React і TypeScript. Використання TypeScript забезпечує статичну типізацію, що сприяє

підвищенню надійності коду, а React забезпечує динамічний і чуйний інтерфейс користувача.

Для створення сучасного та стильного інтерфейсу користувача була обрана бібліотека MaterialUI. Вона надає готові компоненти та стилі, спрощуючи процес розробки та забезпечує однаковість дизайну.

Використання JSON Web Token (JWT) для автентифікації забезпечує безпеку передачі даних між клієнтом та сервером. JWT-токен зберігається в куках сайту, забезпечуючи безпечне та ефективне керування сесіями користувачів.

Застосування динамічного завантаження сторінок, під час прокручування оптимізує продуктивність вебпрограми, дозволяючи завантажувати лише необхідні дані за необхідності. Це скорочує час завантаження і забезпечує кращий користувацький досвід.

Використання SignalR для передачі результатів розпізнавання пошкоджень забезпечує миттєве сповіщення користувачів про результати. Це створює інтерактивну та оперативну взаємодію з сервісом розпізнавання пошкоджень.

Для забезпечення персоналізованого досвіду використання, було прийняте рішення зберігати фільтри пошуку в локальному сховищі браузера, що дозволяє зберігати уподобання користувачів між сесіями.

Ефективне поєднання цих рішень забезпечує створення сучасного, функціонального та безпечного вебзастосунку для агрегатора оголошень про продаж автомобілів. Кожне рішення спрямоване на поліпшення досвіду користувача, оптимізацію продуктивності та забезпечення надійності системи.

Структура проєкту вебзастосунку наведена на рис. 4.4.

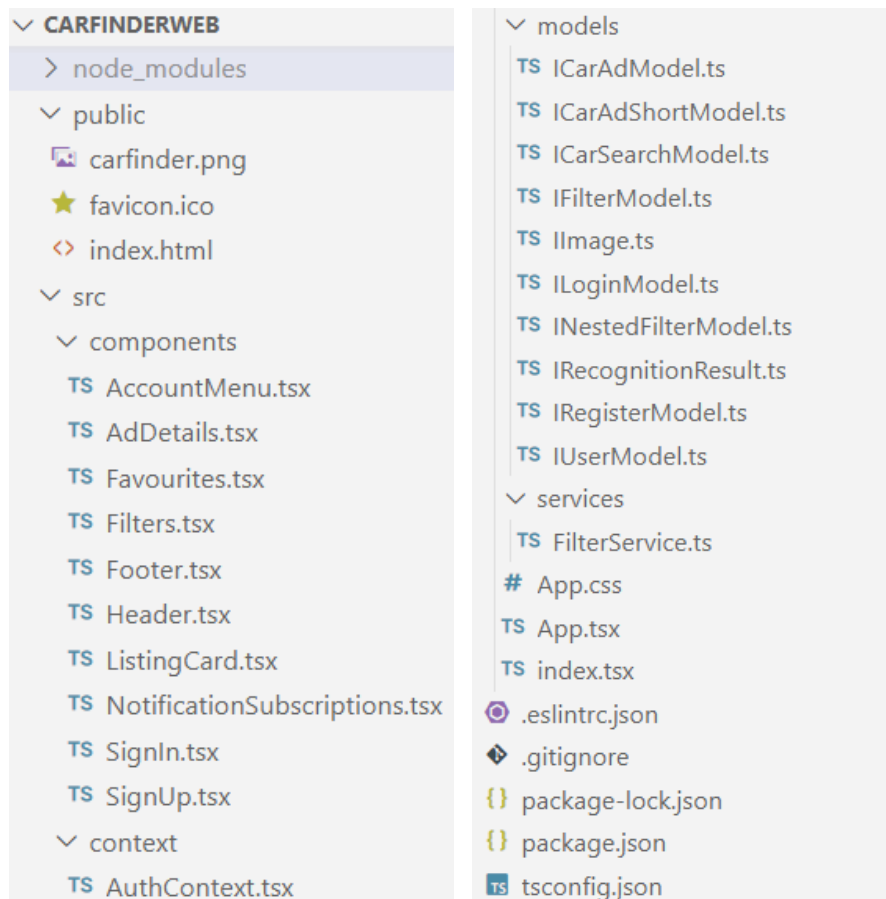


Рисунок 4.4 – Структура проекту вебзастосунку

4.3 Висновки за розділом 4

Прийняті рішення в ході реалізації програмного забезпечення для агрегатора оголошень про продаж автомобілів, забезпечили цілеспрямований та інноваційний підхід до створення системи. У розділі було описано ключові аспекти, які спільно забезпечують ефективність, функціональність, безпеку та привабливість програмного забезпечення. Такий інтегрований підхід до розробки дозволив створити сучасну та інноваційну систему, що відповідає функціональним вимогам та надає ефективні рішення у сфері агрегації оголошень про продаж автомобілів з функцією виявлення пошкоджень на зображеннях автомобілів.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис програми

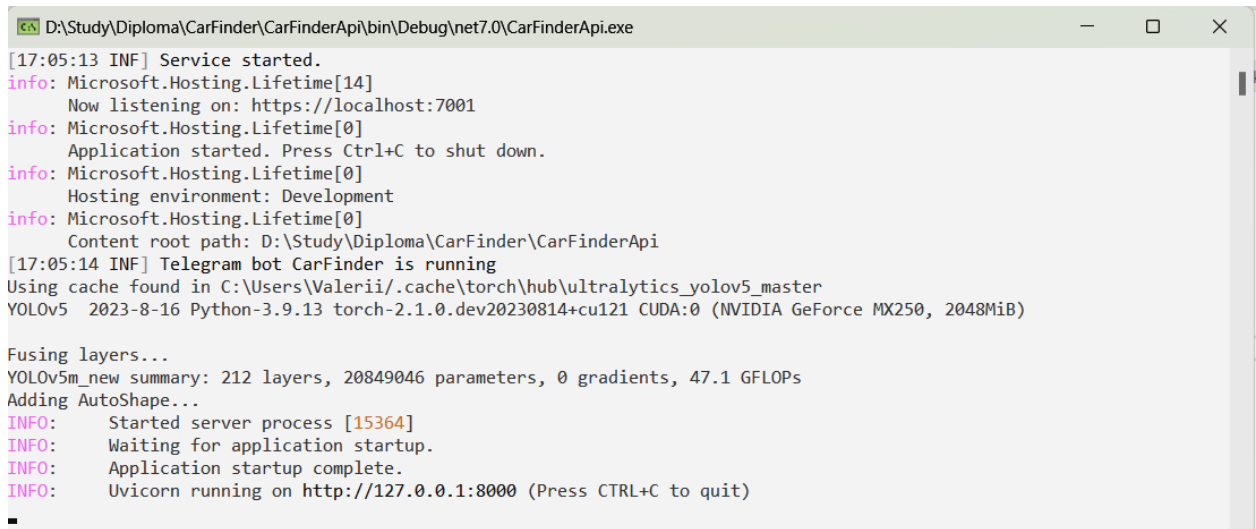
Розроблене програмне забезпечення створено для пошуку оголошень про продаж транспортних засобів на майданчиках оголошень, таких як AvtoRia та OLX. На головній сторінці сервісу можна обрати фільтри пошуку, за якими буде виведено список доступних оголошень. При кліку на певне оголошення, можна подивитися детальну інформацію про авто. Агрегатор має функцію автоматичного розпізнавання наявності пошкоджень на фотографіях автомобілів, які надані продавцями. На кожній плашці з оголошенням виводиться інформація, чи виявила система пошкодження на фото автомобіля. Після реєстрації стає можливим зберігання улюблених фільтрів пошуку та підписка на повідомлення про нові оголошення.

5.2 Експлуатація програми

Програма має коректно працювати за умов:

- наявності встановленого та налаштованого середовища ASP.NET Core, Python та npm;
- наявності налаштованої бази даних PostgreSQL;
- наявності відеокарти Nvidia та налаштованого середовища CUDA;
- наявності доступу до мережі Інтернет;
- наявності та запуску останньої версії сервера та клієнта.

Для роботи у програмному забезпеченні необхідно, запустити сервер програмного забезпечення (ПЗ) та React-клієнт на серверному комп'ютері. Під час запуску сервера, автоматично напускається Python скрипт модулю розпізнавання. Вигляд консолі сервера програми CarFinder після запуску зображений на рис. 5.1, а консолі клієнта – на рис. 5.2.



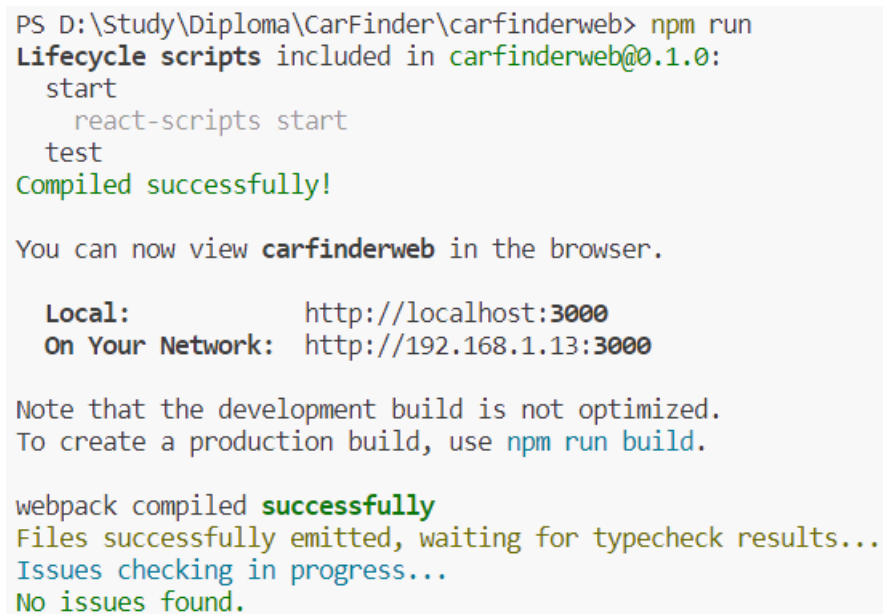
```

D:\Study\Diploma\CarFinder\CarFinderApi\bin\Debug\net7.0\CarFinderApi.exe
[17:05:13 INF] Service started.
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: https://localhost:7001
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: D:\Study\Diploma\CarFinder\CarFinderApi
[17:05:14 INF] Telegram bot CarFinder is running
Using cache found in C:\Users\Valerii\.cache\torch\hub\ultralytics_yolov5_master
YOLOv5 2023-8-16 Python-3.9.13 torch-2.1.0.dev20230814+cu121 CUDA:0 (NVIDIA GeForce MX250, 2048MiB)

Fusing layers...
YOLOv5m_new summary: 212 layers, 20849046 parameters, 0 gradients, 47.1 GFLOPs
Adding AutoShape...
INFO: Started server process [15364]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)

```

Рисунок 5.1 – Консоль сервера програми CarFinder



```

PS D:\Study\Diploma\CarFinder\carfinderweb> npm run
Lifecycle scripts included in carfinderweb@0.1.0:
  start
    react-scripts start
  test
    test
Compiled successfully!

You can now view carfinderweb in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.1.13:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
Files successfully emitted, waiting for typecheck results...
Issues checking in progress...
No issues found.

```

Рисунок 5.2 – Консоль клієнта програми CarFinder

Після того, як ПЗ було запущено, користувачеві стає доступний сайт для взаємодії з системою. Головна сторінка сайту зображена на рис 5.3 та 5.4. На ній користувач має можливість використовувати фільтри пошуку, переглядати список знайдених оголошень, управляти обліковим записом, додавати в обране фільтри пошуку та підписуватися на повідомлення про нові оголошення за фільтрами пошуку.

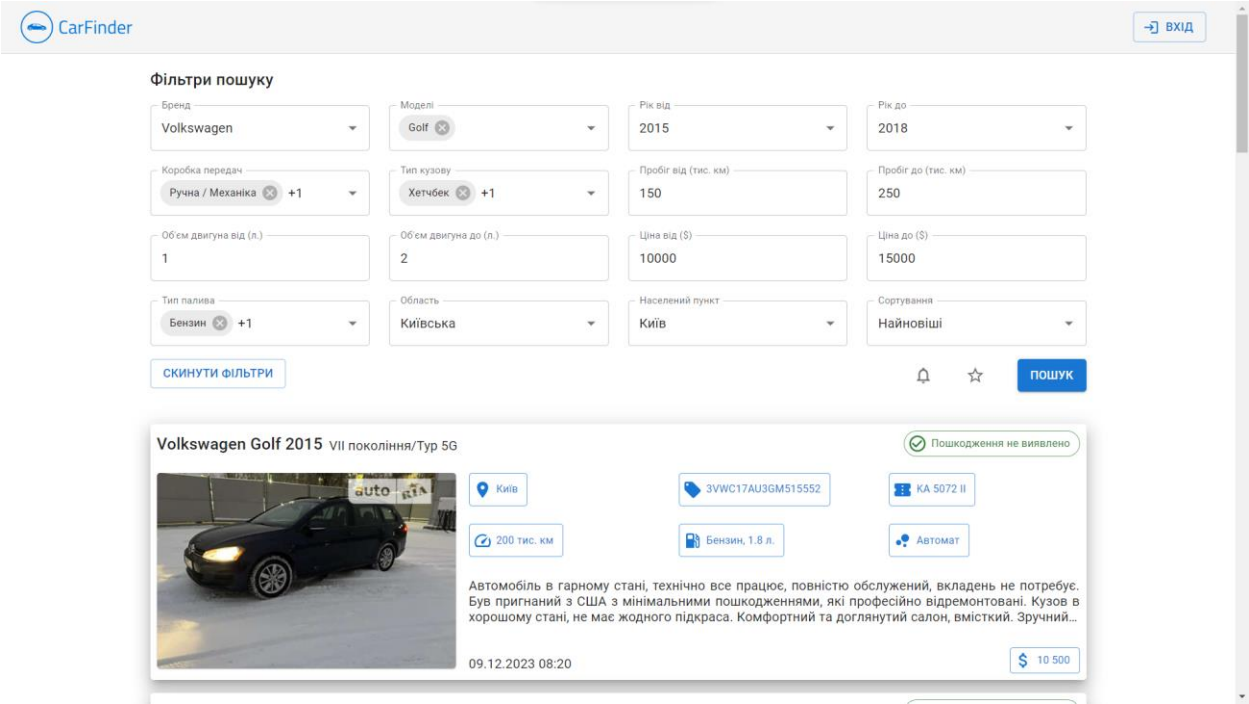


Рисунок 5.3 – Головна сторінка вебсайту (повна версія вебсайту)

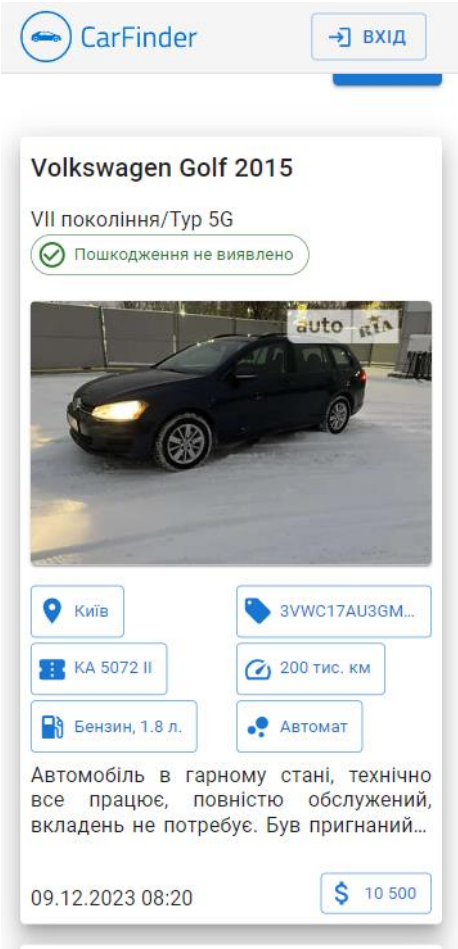


Рисунок 5.4 – Головна сторінка вебсайту (мобільна версія вебсайту)

Не кожен користувач системи буде використовувати комп'ютер для користування сервісом. Тому багато уваги було приділено адаптації сайту для перегляду на мобільних пристроях.

Фільтрами пошуку можна задати наступні параметри: бренд, модель, межі року випуску, пробігу, об'єм двигуна, ціни; тип коробки передач, кузову, палива, область, населений пункт та тип сортування. За замовчуванням, фільтри пошуку порожні, крім типу сортування – обрано від найновіших.

На картці оголошення наявна наступна основна інформація, отримана з майданчику оголошення: фотографія, назва авто, рік випуску, модель, локація, Vehicle Identification Number (VIN), реєстраційний номер, пробіг, тип палива, тип коробки передач, опис, дата додання та ціна.

Також у верхньому правому кутку картки оголошення наявний бейдж, який відповідає за показ інформації користувачу стосовно розпізнаних пошкоджень автомобіля. На початку повідомляється про пошук пошкоджень на фотографіях оголошення модулем розпізнавання, надписом «Пошук пошкоджень». Потім бейдж змінює колір та текст, відповідно до результату розпізнавання: «Виявлено пошкодження», чи «Пошкодження не виявлено». Приклад карточки оголошення з виявленими пошкодженнями наведений на рис. 5.5.

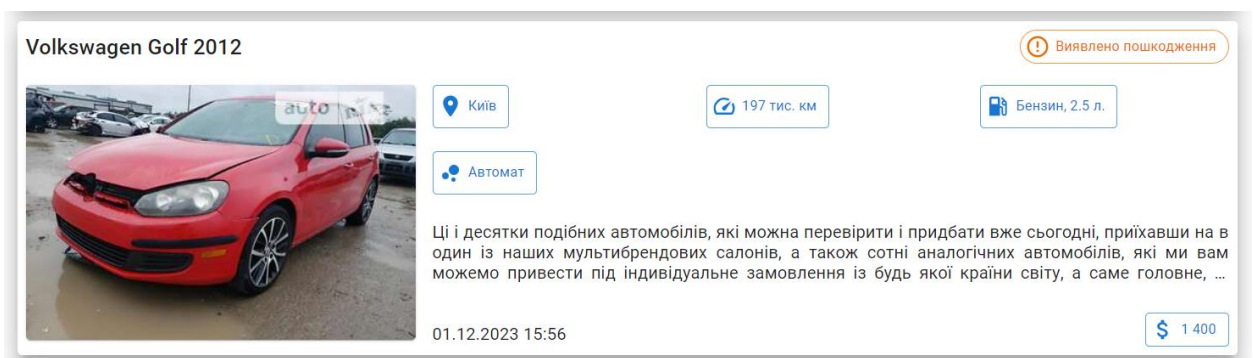


Рисунок 5.5 – Карточка оголошення з виявленим пошкодженням

Для зручності використання системи було реалізовано функцію збереження введених фільтрів пошуку до локального сховища сайту у браузері. За допомогою цього, при відкритті браузера, чи нової вкладки сайту, останні введені фільтри пошуку автоматично підставляються. Також, після входу в аккаунт, введені фільтри пошуку неавторизованим користувачем не збиваються. Це значно покращує користувацький досвід.

При натисканні на заголовок картки оголошення відкривається нова вкладка в браузері зі сторінкою детальної інформації по оголошенню (рис. 5.6 та 5.7). На цій сторінці можна переглянути всю необхідну інформацію, взятую з майданчику оголошення: фотографії, назву авто, рік випуску, модель, ціну, локацію, VIN номер, реєстраційний номер, пробіг, тип палива, тип коробки передач, об'єм двигуна, тип приводу, колір, опис та додаткові параметри.

Також, біля ціни є кнопка для переходу на майданчик оголошення. В ситуації, коли система виявляє однакові оголошення на різних майданчиках, буде показано дві кнопки для переходу на AutoRia та OLX відповідно.

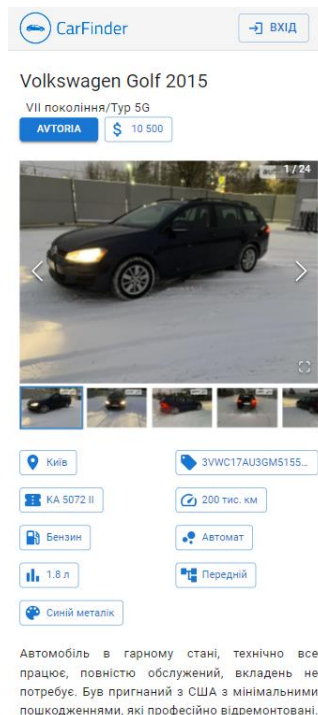


Рисунок 5.6 – Сторінка детальної інформації по оголошенню (мобільна версія вебсайту)

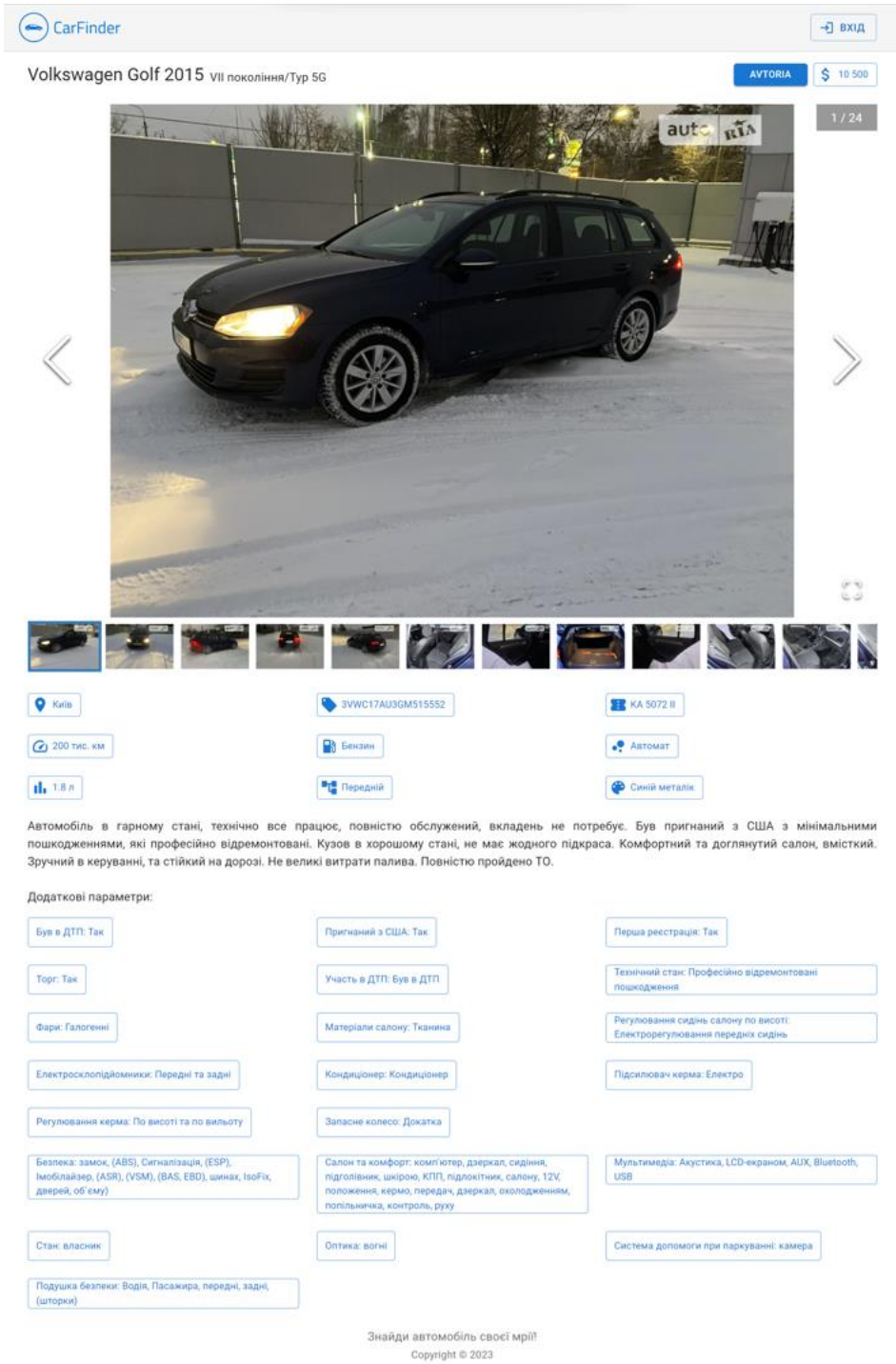


Рисунок 5.7 – Сторінка детальної інформації по оголошенню (повна версія вебсайту)

Для отримання додаткової функціональності на сайті потрібно пройти процес реєстрації, або, якщо є аккаунт, то увійти в нього. Сторінка реєстрації та авторизації зображені на рис. 5.8. Для покращення користувацького досвіду було спрощено процес реєстрації нового облікового запису. Для

реєстрації потрібно лише ввести адресу електронної пошти та пароль. Відповідно, для входу потрібно використовувати ці ж дані.

Якщо користувач збереже облікові дані до браузеру, то при вході система автоматично заповнить поля для входу.

The image displays two side-by-side web forms for the CarFinder website. Both forms have a header with the CarFinder logo and a 'ВХІД' (Login) button. The left form is titled 'Реєстрація' (Registration) and features input fields for 'Email адреса *' and 'Пароль *', followed by a blue 'ЗАРЕЄСТРУВАТИСЯ' (Register) button. Below the button is a link: 'Вже є акаунт? Ввійти' (Already have an account? Login). The right form is titled 'Вхід' (Login) and features input fields for 'Email адреса *' (containing 'wyknait@gmail.com') and 'Пароль *' (masked with dots), followed by a blue 'ВВІЙТИ' (Login) button. Below the button is a link: 'Немає акаунту? Зареєструватися' (No account? Register). Both forms have a footer with the text 'Знайди автомобіль своєї мрії! Copyright © 2023'.

Рисунок 5.8 – Сторінка реєстрації та входу на вебсайт

Створення облікового запису обумовлено отриманням таких додаткових функцій, як збереження фільтрів пошуку у список вподобаних та підписка на повідомлення через Telegram-бот про нові оголошення за фільтрами пошуку. Для цього потрібно написнути на відповідну кнопку внизу фільтрів пошуку – на зірочку або дзвіночок (рис. 5.9). При повторному натисненні відбувається протилежна дія – видалення зі збереження чи підписок.



Рисунок 5.9 – Кнопки для додання пошуку в збережені та підписки на повідомлення

При першій підписці на оголошення користувачеві буде показано повідомлення (рис. 5.10), в якому запропоновано підв'язати свій обліковий запис Telegram до боту.

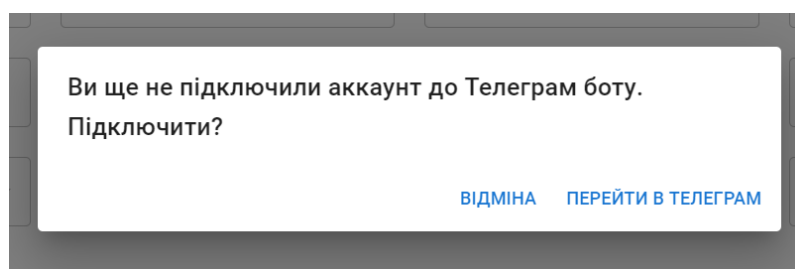


Рисунок 5.10 – Повідомлення про необхідність підключення до Telegram-боту

Після переходу на сторінку Telegram-боту, потрібно натиснути кнопку «Старт». В результаті, відправляється відповідна команда та користувач отримує вітання (рис. 5.11).

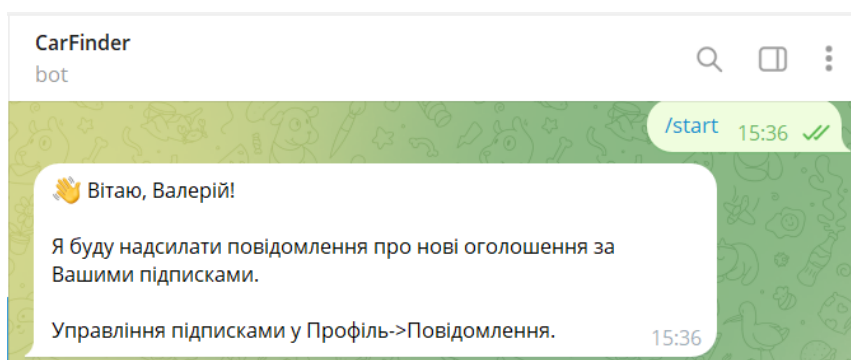


Рисунок 5.11 – Вікно Telegram-боту під час підключення

Приклад роботи Telegram-боту CarFinder можна побачити на рис. 5.12. В повідомленні може бути список з оголошень або одне оголошення. Та приводиться найнеобхідніша інформація: назва, модель, рік випуску, пробіг,

розташування, тип палива, коробки передач та ціна, а також посилання на майданчик оголошення, або у разі виявлення дублікатів, то посилання на два майданчики оголошення.

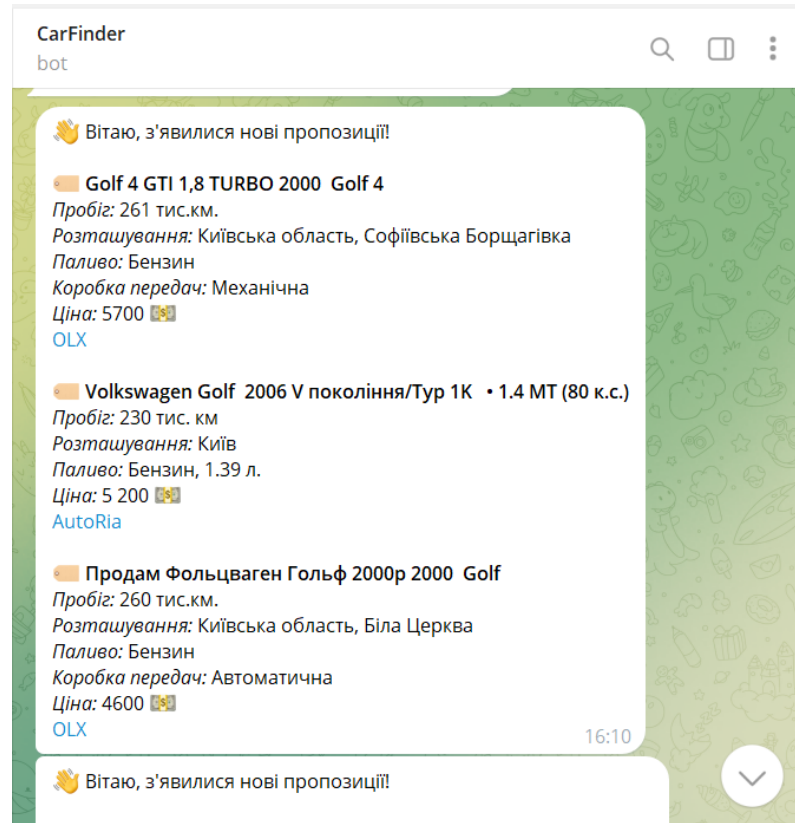


Рисунок 5.12 – Вікно Telegram-боту з повідомленнями про нові оголошення

При натисканні на значок профілю, відкривається список опцій (рис. 5.13), а саме: кнопка переходу на сторінку збережених пошуків та на сторінку підписок на повідомлення. Також є кнопка для виходу з акаунту.

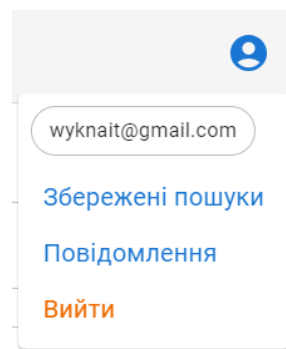


Рисунок 5.13 – Опції керування акаунтом

На рис. 5.14 зображені скріншоти сторінки збережених фільтрів пошуку та сторінки підписок на повідомлення через Telegram-бот за фільтрами пошуку. На цих сторінках можна видаляти збережені, а також переходити на головну сторінку пошуку за відповідними фільтрами пошуку.

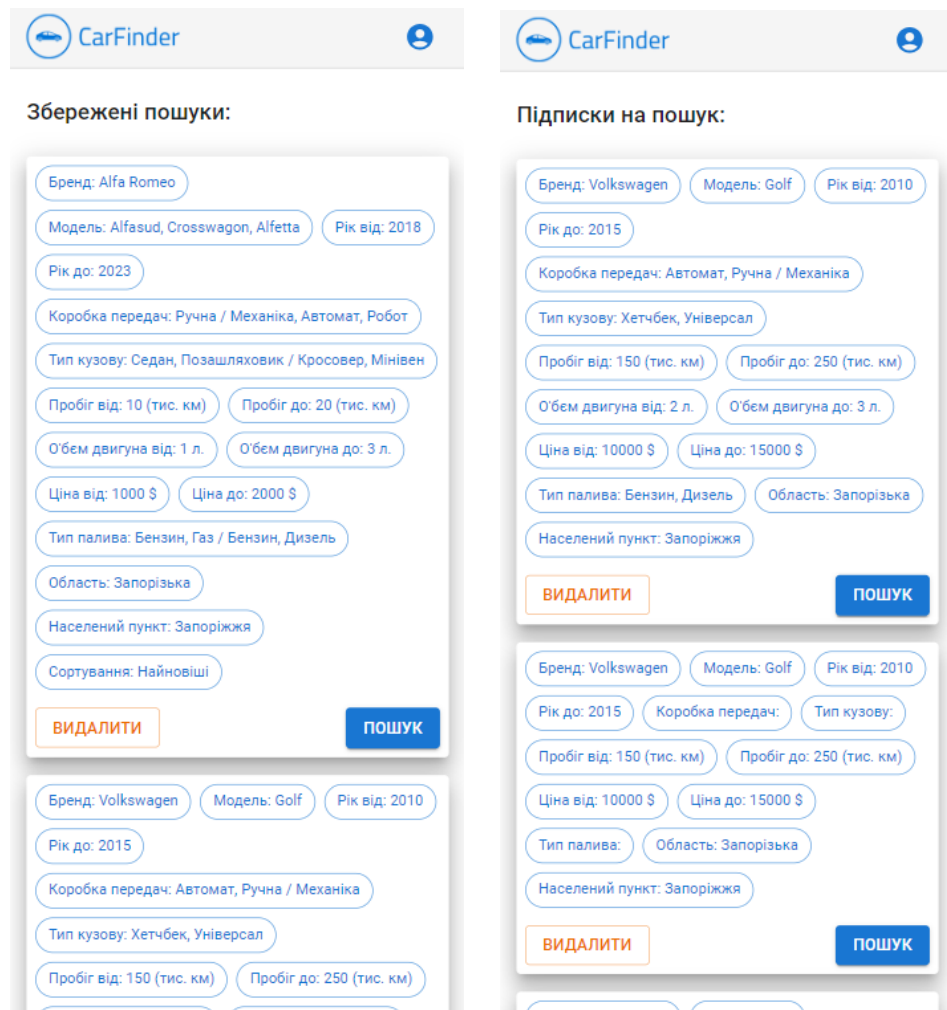


Рисунок 5.14 – Сторінки керування збереженими пошуками та підписками

5.3 Експериментальне дослідження та тестування

Експериментальне дослідження та тестування різноманітних компонентів системи агрегатора оголошень про продаж автомобілів є важливим етапом. Метою даного етапу є переконання у працездатності, ефективності, надійності та продуктивності програмного забезпечення

агрегатора оголошень про продаж автомобілів. У процесі розробки було застосовано мануальне тестування для перевірки коректності всіх функцій, а також проведено порівняння двох моделей розпізнавання пошкоджень – зі стандартною архітектурою та з модифікованою архітектурою моделей.

5.3.1 Експериментальне дослідження

Було проведено експериментальне дослідження, яке порівнювало модель розпізнавання пошкоджень з оригінальною архітектурою із модифікованою моделлю. Результати тестування моделі на тестовій вибірці даних з оригінальною та модифікованою архітектурою наведені у табл. 5.1.

Таблиця 5.1 – Результати тестування моделей

Показник	Модель з оригінальною архітектурою	Модель та модифікованою архітектурою
GFLOPs	47,9	47,1
P	0,745	0,754
R	0,605	0,609
mAP50	0,615	0,629
mAP50-95	0,307	0,315
Speed: pre-process	1,1 мс	1,1 мс
Speed: inference	83,4 мс	66,1 мс

Основні метрики оцінки моделі включають точність та швидкість розпізнавання зображень. Результати експерименту показали, що модифікована модель розпізнавання пошкоджень продемонструвала зменшений час роботи на 20 %, що є важливим показником для забезпечення чуйності системи. Крім того, було відзначено більш високу точність, порівняно зі стандартною моделлю (більше на 2,2 %). Це засвідчує ефективність розробленої модифікації архітектури моделі.

Ці результати свідчать, що використання модифікованої моделі розпізнавання пошкоджень і загалом відповідного методу, який призвів до отримання цієї моделі, в розробленому агрегаторі, є ефективним з погляду якості та продуктивності. Експеримент підтверджує прийняті рішення у виборі технологій та підходів при проектуванні та реалізації агрегатора оголошень про продаж автомобілів.

5.3.2 Тестування

Під час розробки програмного забезпечення кожна додана функція тестувалася в різних сценаріях. Там, де виникали помилки, код модифікувався для усунення помилок.

За допомогою техніки ручного (мануального) тестування було здійснено перевірку всіх функціональних блоків агрегатора, включаючи серверну частину, клієнтський інтерфейс і модуль розпізнавання пошкоджень автомобілів. Багато разів, за різними сценаріями використовувалися всі функції системи. Основною метою було переконатися у правильній взаємодії компонентів, коректному відображенні даних на клієнтській стороні та стабільній роботі сервера.

Завдяки тестуванню розробленого програмного забезпечення, було досягнуто стабільність роботи та отримано впевненість у коректній роботі всіх програмних модулів. Таким чином, тестування дозволило виявити та усунути потенційні помилки та недоліки у функціональності.

Чек-лист тестування наведений у табл. 5.2.

Таблиця 5.2 – Результати тестування ПЗ

Перевірка	Результат
Завантаження у кеш конфігурацій фільтрів пошуку	Пройдено
Запуск модулю розпізнавання з-під сервера	Пройдено

Продовження таблиці 5.2

Перевірка	Результат
Зв'язок з базою даних	Пройдено
Робота сервісу пошуку оголошень	Пройдено
Робота сервісу хешування зображень	Пройдено
Робота сервісу розпізнавання	Пройдено
Робота сервісу Telegram-боту	Пройдено
Робота сервісу повідомлень	Пройдено
Отримання запитів	Пройдено
Парсинг даних	Пройдено
Обробка даних	Пройдено
Збереження нових та модифікованих даних	Пройдено
Відповіді на запити	Пройдено
Процес пошуку оголошень	Пройдено
Процес перегляду оголошень	Пройдено
Процес переходу на майданчик оголошення	Пройдено
Процес реєстрації	Пройдено
Процес авторизації	Пройдено
Процес керування збереженими пошуками	Пройдено
Процес прив'язки аккаунту до Telegram-боту	Пройдено
Процес керування підписками на повідомлення	Пройдено
Логування	Пройдено
Безперервна робота	Пройдено

5.4 Висновки за розділом 5

Даний розділ є ключовим етапом у життєвому циклі розробки агрегатора оголошень про продаж автомобілів. У ході тестування було повністю перевірено функціональність програмного забезпечення, що дозволило виявити та усунути можливі помилки та невідповідності вимогам.

Результати експериментального дослідження підтверджують успішну модифікацію архітектури моделі розпізнавання пошкоджень.

Розроблене програмне забезпечення готове до ефективної експлуатації, забезпечуючи високу продуктивність, надійність і точність у вирішенні поставлених завдань.

ВИСНОВКИ

Дослідження та програмна реалізація методів розпізнавання пошкоджень для агрегатора пошуку оголошень про продаж автомобілів є комплексним підходом до створення інноваційної системи, сфокусованої на підвищенні ефективності та функціональності у сфері автомобільних оголошень.

Було створено програмне забезпечення для пошуку оголошень автомобілів на різних майданчиках оголошень, яке здатне автоматично розпізнавати наявність пошкоджень на фотографіях автомобілів, що дозволить прискорити та спростити процес пошуку відповідного автомобіля серед сотень оголошень.

Наукова новизна роботи полягає в запропонованому методі автоматичного виявлення пошкоджень автомобілів на основі агрегації даних, який виконує модифікацію архітектури моделі розпізнавання, що дозволяє виявляти пошкодження на фотографіях, прикріплених до оголошень, та класифікувати оголошення про продаж автомобілів. Це автоматизує процес попередньої оцінки стану автомобіля та надає користувачам більше корисних даних про автомобілі, виставлених на продаж.

Реалізована система є значним кроком у сфері агрегації оголошень про продаж автомобілів, інтегруючи технології машинного навчання із сучасними технологіями. При цьому залишаються перспективи для подальшого вдосконалення системи, розширення функціональності і адаптації до потреб ринку, що змінюються. Все це робить цю роботу важливим етапом у розвитку сучасних технологій в автомобільній індустрії та цифровій сфері.

Проведене тестування та експериментальне дослідження підтвердило працездатність розробленої програмної системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Arslan, S. Using Google Search Trends to Estimate Global Patterns in Learning [Text] / S. Arslan, M. Tiwari and C. Piech // Proceedings of the Seventh ACM Conference on Learning @ Scale. ACM, Aug. 12, 2020. – 2020. – Pp.185-195. – Doi : 10.1145/3386527.3405913.
2. Conciliating TOP-DOWN and BOTTOM-UP approaches in Websites Quality Evaluation [Electronic resource]. – Access mode : https://openportal.isti.cnr.it/data/2007/160817/2007_160817.pdf
3. Електронна система Автопошук [Електрон. ресурс]. – Режим доступу : <https://avtopoisk.ua/about/>.
4. Електронна система AutoMoto [Електрон. ресурс]. – Режим доступу : <https://automoto.ua/uk/about>.
5. Car Damage Detection using AI [Electronic resource]. – Access mode : <https://inspektlabs.com/blog/car-damage-detection-using-ai-methodology-and-approach-for-high-accuracy/>.
6. Convolutional Neural Networks [Electronic resource]. – Access mode : <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939>.
7. Convolutional neural network (CNN) [Electronic resource]. – Access mode : <https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network>.
8. O'Shea, K. An introduction to convolutional neural networks [Text] / K. O'Shea and R. Nash // arXiv preprint arXiv : 1511.08458, Dec, 2, 2015. – 2015. – Pp. 1-11.
9. Understanding of a convolutional neural network [Text] / S. Albawi, O. Bayat, S. Al-Azawi and O. Ucan // 2017 international conference on engineering and technology (ICET), Oct, 8, 2018. – 2018. – Pp. 1-6. – Doi : 10.1155/2018/6973103.
10. What are convolutional neural networks? [Electronic resource]. –

Access mode : <https://www.ibm.com/topics/convolutional-neural-networks>.

11. Deepika, H. An Overview of You Only Look Once : Unified, Real-Time Object Detection [Text] / H. Deepika, V. Raghuvver and G. Srinivasan // International Journal for Research in Applied Science and Engineering Technology, 8(6), IJRASET, Jun, 30, 2020. – 2020. – Pp. 607-609. – Doi : 10.22214/ijraset.2020.6098.

12. Redmon, J. You only look once : Unified, real-time object detection. [Text] / J. Redmon // Proceedings of the IEEE conference on computer vision and pattern recognition, 2016. – 2016. – Pp. 779-788.

13. Car Damage Detection using AI : Methodology and Approach for High Accuracy [Electronic resource]. – Access mode : <https://universe.roboflow.com/romil-nisar/car-damage-detection-rzcmb>.

14. Final Dataset Annotations Computer Vision Project [Electronic resource]. – Access mode : <https://universe.roboflow.com/be-project-damage-detection/final-dataset-annotations>.

15. Car_Dents_Scratches Computer Vision Project [Electronic resource]. – Access mode : https://universe.roboflow.com/damage-detection-jbyko/car_dents_scratches-fyvkkx.

16. Car damage Computer Vision Project [Electronic resource]. – Access mode : <https://universe.roboflow.com/student-obb8s/final-dam>.

17. Cars_damage Computer Vision Project [Electronic resource]. – Access mode : <https://universe.roboflow.com/student-obb8s/final-dam>.

18. Image data augmentation for deep learning : A survey [Text] / S. Yang, W. Xiao, M. Zhang, S. Guo, J. Zhao and F. Shen // arXiv preprint arXiv : 2204.08610, Nov, 5, 2023. – 2020.

19. Mallikarjuna, B. Vehicle Damage Detection and Classification Using Image Processing [Text] / B. Mallikarjuna, K. Arun and L. Kumar // International Journal of Advanced Research in Science, Communication and Technology. Naksh Solutions, Jun, 30, 2022. – 2022. – Pp. 568–574. – Doi : 10.48175/ijarsct-5414.

20. What is YOLOv5? A Guide for Beginners. [Electronic resource]. –

Access mode : <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>.

21. A Real-Time Apple Targets Detection Method for Picking Robot Based on Improved YOLOv5 [Text] / B. Yan, P. Fan, X. Lei, Z. Liu and F. Yang // Remote Sensing, 13(9) MDPI AG, Apr, 21, 2021. – 2021. – Pp. 1619. – Doi : 10.3390/rs13091619.

22. A Two-Stage Industrial Defect Detection Framework Based on Improved YOLOv5 and Optimized-Inception-ResnetV2 Models [Text] / Z. Li, X. Tian, X. Liu, Y. Liu and X. Shi // Applied Sciences, 12(2) MDPI AG, Jan, 14, 2022. – 2022. – Pp. 834. – Doi : 10.3390/app12020834.

23. Automatic Extraction of Damaged Houses by Earthquake Based on Improved YOLOv5 : A Case Study in Yangbi [Text] / Y. Jing, Y. Ren, Y. Liu, D. Wang and L. Yu // Remote Sensing, 14(2) MDPI AG, Jan, 14, 2022. – 2022. – Pp. 382. – Doi : 10.3390/rs14020382.

24. Arifando, R. Improved YOLOv5-Based Lightweight Object Detection Algorithm for People with Visual Impairment to Detect Buses [Text] / R. Arifando, S. Eto and C. Wada // Applied Sciences, 13(9), 5802, May, 8, 2023 – 2023. – Pp. 5802 – Doi : 10.3390/app13095802.

25. Kaggle electronic system [Electronic resource]. – Access mode : <https://www.kaggle.com/docs/notebooks>.

26. Google Collaboratory electronic system [Electronic resource]. – Access mode : <https://research.google.com/colaboratory/faq.html>.

27. Thuan, D. Evolution of Yolo algorithm and Yolov5 : The State-of-the-Art object detection algorithm [Text] / D. Thuan // Oulu University of Applied Sciences, 2021. – 2021. – Pp. 36-37.

28. .NET Vs. PHP Vs. Java Vs. Python – Which Language Is Better For Web Development? [Electronic resource]. – Access mode : <https://www.valuecoders.com/blog/app-development/net-vs-php-vs-java-vs-python-which-language-is-better/>.

29. Advantages of C# [Electronic resource]. – Access mode : <https://www.codeguru.com/csharp/c-sharp-advantages/>.

30. Tutorial : Create a web API with ASP.NET Core [Electronic resource]. – Access mode : <https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-8.0&tabs=visual-studio>.
31. Comprehend The Benefits Of Using ASP.NET Web APIs To The Core [Electronic resource]. – Access mode : <https://www.brainvire.com/comprehend-the-benefits-of-using-asp-net-web-apis-to-the-core/>.
32. Angular vs React vs Vue : Core Differences [Electronic resource]. – Access mode : <https://www.browserstack.com/guide/angular-vs-react-vs-vue>.
33. Pros and Cons of ReactJS [Electronic resource]. – Access mode : <https://www.javatpoint.com/pros-and-cons-of-react>.
34. The Good and the Bad of TypeScript [Electronic resource]. – Access mode : <https://www.altexsoft.com/blog/typescript-pros-and-cons/>.
35. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others [Electronic resource]. – Access mode : <https://www.altexsoft.com/blog/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others/>.
36. Entity Framework Core [Electronic resource]. – Access mode : <https://learn.microsoft.com/en-us/ef/core/>.
37. Language Integrated Query (LINQ) [Electronic resource]. – Access mode : <https://learn.microsoft.com/en-us/dotnet/csharp/linq/>.
38. Code First Approach vs. Database First Approach [Electronic resource]. – Access mode : <https://medium.com/codex/code-first-approach-vs-database-first-approach-a3830c0cc9b6>.
39. Java vs Python and C++ [Electronic resource]. – Access mode : <https://www.codingninjas.com/studio/library/java-vs-python-and-c>.
40. The Pros and Cons of Python Programming [Electronic resource]. – Access mode : <https://www.linode.com/docs/guides/pros-and-cons-of-python/>.
41. Compare Rider Vs Visual Studio : A Detailed Look [Electronic resource]. – Access mode : <https://marketsplash.com/tutorials/visual-studio/rider->

vs-visual-studio/.

42. Best IDE for React [Electronic resource]. – Access mode :
<https://www.etatvasoft.com/blog/best-ide-for-react-js/> .

43. Visual Studio Code [Electronic resource]. – Access mode :
<https://www.educba.com/what-is-visual-studio-code/>.

44. What is Client Server Architecture? [Electronic resource]. – Access
mode : <https://intellipaat.com/blog/what-is-client-server-architecture/>.

45. Client-Server Model [Electronic resource]. – Access mode :
<https://www.geeksforgeeks.org/client-server-model/>.

ДОДАТОК А
Технічне завдання

Вступ

У роботі розглядається галузь розробки програмного забезпечення, пов'язана з дослідженням та програмною реалізацією методів розпізнавання пошкоджень для агрегатора пошуку оголошень про продаж автомобілів.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломний проєкт на тему «Дослідження та програмна реалізація методів розпізнавання пошкоджень для агрегатора пошуку оголошень про продаж автомобілів», затверджене наказом по національному університету «Запорізька політехніка», протокол № 437 від 14 листопада 2023 р.

A.2 Призначення розробки

Програмне забезпечення CarFinder призначене для пошуку оголошень про продаж автомобілів на майданчиках оголошень AutoRia та OLX. Користувачі повинні мати можливість заповнити певні критерії у фільтрі пошуку та отримати список відфільтрованих оголошень, зібраних з відповідних сайтів. У режимі реального часу повинен відбуватися пошук пошкоджень на фотографіях по кожному оголошенню та показуватися результат користувачу.

A.3 Основні вимоги до програмного забезпечення, що розробляється

A.3.1 Вимоги до функціональних характеристик

Забезпечення функціональних характеристик повинно вкладатися в наступний набір функцій:

- пошук авто за фільтрами пошуку;
- виявлення однакових оголошень;
- розпізнавання зображень для виявлення пошкоджень;
- перегляд списку знайдених оголошень;
- перегляд оголошення;
- перехід на майданчик оголошення;
- реєстрація та авторизація;
- збереження фільтру пошуку у збережені;
- підв'язка до Telegram-боту;
- підписка на повідомлення про нові оголошення через Telegram-бот;
- управління збереженими фільтрами та підписками.

А.3.2 Вимоги до надійності

Програма повинна забезпечувати:

- стабільну роботу агрегатора та модуля розпізнавання пошкоджень;
- безпеку даних користувачів;
- обробку виключних ситуацій.

А.3.3 Умови експлуатації

Використання програмного продукту виконується згідно з експлуатаційною документацією для створеного програмного забезпечення, яка відповідає стандартам і включає необхідну інформацію для оволодіння та використання даного програмного продукту.

А.3.4 Вимоги до складу і параметрів технічних засобів

Для нормальної роботи сервера розробленої програми потрібен комп'ютер під управлінням операційної системи Microsoft Windows 10 та

вище з такими характеристиками: процесор з тактовою частотою не нижче 1.5 ГГц; оперативний запам'ятовуючий пристрій – 8 ГБ; 10000 МБ вільного місця на жорсткому диску та відеокарта з підтримкою програмно-апаратною архітектурою паралельних обчислень CUDA не нижче Nvidia 250MX. Також необхідна встановлена система управління базами даних PostgreSQL. Для використання клієнтського сайту потрібен пристрій із встановленим браузером та доступом до мережі.

A.3.5 Вимоги до програмної документації

Система має бути поставлена разом із технічним завданням та інструкцією з експлуатації, наданою у відповідному розділі пояснювальної записки.

A.4 Порядок контролю та приймання

Строки виконання дипломної кваліфікаційної роботи магістра визначені в завданні на роботу безпосередньо, включаючи строк подання роботи, календарний план виконання роботи з зазначеними етапами, результатами та строками.

Захист роботи відбувається за оголошеним завчасно графіком захисту дипломних кваліфікаційних робіт і відповідно графіком роботи екзаменаційної комісії.

ДОДАТОК Б
Текст програми

Б.1 Текст файла CarSearchService.cs

```

public class CarSearchService : ICarSearchService
{
    private IConfiguration _configuration;
    private IHttpSender _httpSender;
    private IImageHashService _imageHashService;
    private IImageRecognitionService _imageRecognitionService;
    private IServiceScopeFactory _scopeFactory;
    private string _riaPath;
    private string _olxPath;
    private bool _isCompareByImageHashEnabled = false;
    private bool _isDamageRecognitionEnabled = false;
    public CarSearchService(
        IConfiguration configuration, IHttpSender httpSender,
        IImageHashService imageHashMaker, IImageRecognitionService imageRecognition,
        IServiceScopeFactory scopeFactory)
    {
        _configuration = configuration;
        _httpSender = httpSender;
        _imageHashService = imageHashMaker;
        _imageRecognitionService = imageRecognition;
        _scopeFactory = scopeFactory;

        _riaPath = _configuration["CarSearch:RiaPath"]!;
        _olxPath = _configuration["CarSearch:OlxPath"]!;
        _isCompareByImageHashEnabled =
bool.Parse(_configuration["CarSearch:IsCompareByImageHashEnabled"]!);
        _isDamageRecognitionEnabled =
bool.Parse(_configuration["CarSearch:IsDamageRecognitionEnabled"]!);
    }
    public async Task<List<CarAdShortResponse>> GetCarAdList(CarSearchRequest request)
    {
        var carAdList = new List<CarAdShortResponse>();
        var riaTask = GetCarAdListRia(request);
        var olxTask = GetCarAdListOlx(request);
        await Task.WhenAll(riaTask, olxTask);
        carAdList.AddRange(riaTask.Result);
        carAdList.AddRange(olxTask.Result);
        await CompareAdsByImageHash(carAdList);
        RecognizeDamage(carAdList);
        switch (request.SortBy)
        {
            case SortByEnum.PriceAsc:
                return carAdList
                    .OrderBy(x => int.Parse(x.Price.Replace(" ", "")))

```

```

        .ThenByDescending(x => x.AddDate)
        .ToList();
    case SortByEnum.PriceDesc:
        return carAdList
            .OrderByDescending(x => int.Parse(x.Price.Replace(" ", "")))
            .ThenByDescending(x => x.AddDate)
            .ToList();
    case SortByEnum.CreatedDesc:
        return carAdList
            .OrderByDescending(x => x.AddDate)
            .ToList();
    default:
        return carAdList
            .OrderByDescending(x => x.AddDate)
            .ToList();
    }
}

public async Task<CarAdResponce> GetCarAdInfo(AdUrlModel adUrl)
{
    var carAd = new CarAdResponce();
    if (!string.IsNullOrEmpty(adUrl.AdRiaUrl))
    {
        carAd = await GetCarAdInfoRia(adUrl.AdRiaUrl.Replace("auto.ria.com",
"auto.ria.com/uk"));
        carAd.AdUrl.AdOlxUrl = adUrl.AdOlxUrl;
    }
    else
    {
        carAd = await GetCarAdInfoOlx(adUrl.AdOlxUrl);
    }
    return carAd;
}

public async Task SetLastSubscriptionAdId(int subscriptionId, CarSearchRequest
request)
{
    var carAdList = await GetCarAdListForSubscription(request);
    var lastAdDate = carAdList?.FirstOrDefault()?.AddDate;
    using var scope = _scopeFactory.CreateScope();
    var dbRepository = scope.ServiceProvider.GetRequiredService<IDbRepository>();
    dbRepository.SetLastSubscriptionAdDate(subscriptionId, lastAdDate);
}

public async Task<List<CarAdShortResponce>>
GetCarAdListForSubscription(CarSearchRequest request)
{
    var carAdList = new List<CarAdShortResponce>();
    var riaTask = GetCarAdListRia(request);
    var olxTask = GetCarAdListOlx(request);

```

```

        await Task.WhenAll(riaTask, olxTask);
        carAdList.AddRange(riaTask.Result);
        carAdList.AddRange(olxTask.Result);
        await CompareAdsByImageHash(carAdList);
        return carAdList
            .OrderByDescending(x => x.AddDate)
            .ToList();
    }
    private async Task CompareAdsByImageHash(List<CarAdShortResponce> carAdList)
    {
        if (!_isCompareByImageHashEnabled)
        {
            return;
        }
        var calculateHashTasks = new List<Task>();
        foreach (var carAd in carAdList)
        {
            var task = Task.Run(async () =>
            {
                carAd.ImageHash = await
                _imageHashService.CalculateHashAsync(carAd.PhotoUrl);
            });
            calculateHashTasks.Add(task);
        }
        await Task.WhenAll(calculateHashTasks);
        try
        {
            for (int i = 0; i < carAdList.Count - 1; i++)
            {
                for (int j = i + 1; j < carAdList.Count; j++)
                {
                    bool isSimilar =
                    _imageHashService.IsImagesSimilar(carAdList[i].ImageHash, carAdList[j].ImageHash);

                    if (isSimilar)
                    {
                        if (carAdList[i].AdId.RiaId != null)
                        {
                            Log.Information($"CarSearchService.GetCarAdList " +
                                $"ria [{carAdList[i].AdUrl.AdRiaUrl}] == olx
                                [{carAdList[j].AdUrl.AdOlxUrl}]");
                            carAdList[i].AdId.OlxId = carAdList[j].AdId.OlxId;
                            carAdList[i].AdUrl.AdOlxUrl = carAdList[j].AdUrl.AdOlxUrl;
                            carAdList.RemoveAt(j);
                            j--;
                        }
                        else if (carAdList[j].AdId.RiaId != null)

```

```

        {
            Log.Information($"CarSearchService.GetCarAdList " +
                $"ria [{carAdList[j].AdUrl.AdRiaUrl}] == olx
[{carAdList[i].AdUrl.AdOlxUrl}]");
            carAdList[j].AdId.OlxId = carAdList[i].AdId.OlxId;
            carAdList[j].AdUrl.AdOlxUrl = carAdList[i].AdUrl.AdOlxUrl;
            carAdList.RemoveAt(i);
            i--;
            break;
        }
    }
}

catch (Exception ex)
{
    Log.Error($"CarSearchService.GetCarAdList {ex.GetType()} Error when
checking similarity" +
        $"{\nMessage: {ex.Message}\nErrorTrace: {ex.StackTrace}" +
        (ex.InnerException != null ? $"{\nInnerException: {ex.InnerException}"
: ""));
}

}

private async Task RecognizeDamage(List<CarAdShortResponse> carAdList)
{
    if (!_isDamageRecognitionEnabled)
    {
        return;
    }
    await PrepareAdsForRecognition(carAdList);
    _imageRecognitionService.RecognizeDamage(carAdList);
}

private async Task PrepareAdsForRecognition(List<CarAdShortResponse> carAdList)
{
    await Task.WhenAll(carAdList.Select(async carAd =>
    {
        if (!string.IsNullOrEmpty(carAd.AdUrl.AdRiaUrl))
            carAd.AllPhotos = await GetAllPhotosRia(carAd.AdUrl.AdRiaUrl);
    }));
}

private async Task<List<string>> GetAllPhotosRia(string riaUrl)
{
    var allPhotos = new List<string>();
    try
    {
        if (string.IsNullOrEmpty(riaUrl))
            throw new NotFoundException(nameof(riaUrl));
    }
}

```

```

var response = await _httpClient.SendGetAsync(riaUrl);
if (response.IsSuccessStatusCode)
{
    var content = await response.Content.ReadAsStringAsync();
    var config = Configuration.Default;
    var context = BrowsingContext.New(config);
    var document = await context.OpenAsync(req => req.Content(content));
    allPhotos =
document.QuerySelector(".wrapper[photocontainer='photo']")?
    .QuerySelectorAll("a.photo-74x56")?
    .Select(x =>
x?.QuerySelector("img")?.GetAttribute("src")?.Replace("s.jpg", "hd.jpg"))?
    .ToList();
    allPhotos?.RemoveAll(x => x.Contains("no-photo"));
}
else
{
    Log.Information("CarSearchService.GetAllPhotosRia Request error");
}
}
catch (Exception ex)
{
    Log.Error($"CarSearchService.GetAllPhotosRia" +
        $"\nMessage: {ex.Message}\nErrorTrace: {ex.StackTrace}" +
        (ex.InnerException != null ? $"\nInnerException: {ex.InnerException}"
: ""));
}
return allPhotos;
}

private async Task<List<CarAdShortResponse>> GetCarAdListRia(CarSearchRequest
request)
{
    var adList = new List<CarAdShortResponse>();
    try
    {
        request.PageNum--;
        var requestPath = MakeRiaRequestPath(request);
        if (string.IsNullOrEmpty(requestPath))
            throw new NotFoundException(nameof(requestPath));
        var response = await _httpClient.SendGetAsync(requestPath);
        if (response.IsSuccessStatusCode)
        {
            var content = await response.Content.ReadAsStringAsync();
            var config = Configuration.Default;
            var context = BrowsingContext.New(config);
            var document = await context.OpenAsync(req => req.Content(content));

```

```

var carListings = document.querySelectorAll(".ticket-item");
foreach (var listing in carListings)
{
    try
    {
        var adId = listing.QuerySelector(".hide").GetAttribute("data-
id");

        var adUrl =
listing.QuerySelector(".hide").GetAttribute("data-link-to-view");
        var date = listing.QuerySelector(".footer_ticket
span").GetAttribute("data-add-date");
        var photoUrl = listing.QuerySelector(".outline.m-
auto").GetAttribute("src");
        var title =
listing.QuerySelector(".address").TextContent?.Trim();
        var model =
listing.QuerySelector(".generation").TextContent?.Trim();
        var price = listing.QuerySelector(".price-ticket
.bold.size22.green").TextContent?.Trim();
        var mileage = listing.QuerySelector(".item-char.js-
race").TextContent?.Trim();
        var location = listing.QuerySelector(".item-char .icon-
location").NextSibling?.TextContent?.Trim();
        var fuel = listing.QuerySelector(".item-char .icon-
fuel").NextSibling?.TextContent?.Trim();
        var transmission = listing.QuerySelector(".item-char .icon-
akp").NextSibling?.TextContent?.Trim();
        var vin = listing.QuerySelector(".label-vin
span").TextContent?.Trim();
        var licensePlate = listing.QuerySelector("span.state-
num.ua").TextContent?.Trim()?.Split(" ")[0];
        var description = listing.QuerySelector(".descriptions-ticket
span").TextContent?.Trim();
        adList.Add(new CarAdShortResponse
        {
            AdId = new AdIdModel { RiaId = int.Parse(adId) },
            AdUrl = new AdUrlModel { AdRiaUrl =
$"https://auto.ria.com{adUrl}" },
            AddDate = DateTime.Parse(date),
            PhotoUrl = photoUrl,
            Title = title,
            Model = model,
            Price = price,
            Mileage = mileage,
            Location = location,
            Fuel = fuel,

```

```

        Transmission = transmission,
        Vin = vin,
        LicensePlate = licensePlate,
        Description = description
    });
}
catch (Exception ex)
{
    Log.Error($"CarSearchService.GetCarAdListRia foreach" +
        $"\nMessage: {ex.Message}\nErrorTrace: {ex.StackTrace}" +
        (ex.InnerException != null ? $"\nInnerException:
{ex.InnerException}" : ""));
}
}
else
{
    Log.Information("CarSearchService.GetCarAdListRia Request error");
}
}
catch (Exception ex)
{
    Log.Error($"CarSearchService.GetCarAdListRia" +
        $"\nMessage: {ex.Message}\nErrorTrace: {ex.StackTrace}" +
        (ex.InnerException != null ? $"\nInnerException: {ex.InnerException}"
: ""));
}
    Log.Information($"CarSearchService.GetCarAdListRia      adList      count      =
{adList.Count}");
    return adList;
}

private async Task<List<CarAdShortResponse>> GetCarAdListOlx(CarSearchRequest
request)
{
    var adList = new List<CarAdShortResponse>();
    request.PageNum++; // increment after ria decrement
    try
    {
        string requestPath = MakeOlxRequestPath(request);
        if (string.IsNullOrEmpty(requestPath))
            throw new NotFoundException(nameof(requestPath));
        var userAgent = "Mozilla/5.0 (Linux; Android 10; SM-G975F)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.141 Mobile Safari/537.36";
        HttpResponseMessage response = await _httpClient.SendGetAsync(requestPath,
userAgent);
        if (response.IsSuccessStatusCode)
        {

```



```

        {
            title += $" {motorYear}";
        }

        string description =
(string)adToken["description"].ToString()?.Replace("<br />", "");

        string model =
$" {(string)adToken["params"].FirstOrDefault(p => (string)p["key"]
"model")?["value"]} " + $" {(string)adToken["params"].FirstOrDefault(p
(string)p["key"] == "modification")?["value"]}";

        string mileage =
(string)adToken["params"].FirstOrDefault(p => (string)p["key"]
"motor_mileage_thou")?["value"];

        string transmission =
(string)adToken["params"].FirstOrDefault(p => (string)p["key"]
"transmission_type")?["value"];

        string fuel =
(string)adToken["params"].FirstOrDefault(p => (string)p["key"]
"fuel_type")?["value"];

        string vin =
(string)adToken["params"].FirstOrDefault(p => (string)p["key"]
"vin_number")?["value"];

        string licensePlate =
(string)adToken["params"].FirstOrDefault(p => (string)p["key"]
"license_plate")?["value"];

        string price =
(string)adToken["price"]["regularPrice"]["value"];

        string location =
(string)adToken["location"]["pathName"];

        List<string> photoUrlList =
JsonConvert.DeserializeObject<List<string>>(adToken["photos"].ToString());
        for (int j = 0; j < photoUrlList?.Count; j++)
        {
            photoUrlList[j] =
GetOlxPhotoLink(photoUrlList[j], 640);
        }

        string photoUrl =
(string)adToken["photos"].FirstOrDefault();
        photoUrl = GetOlxPhotoLink(photoUrl, 250);
        adList.Add(new CarAdShortResponse
        {
            AdId = new AdIdModel { OlxId = adId },
            AdUrl = new AdUrlModel { AdOlxUrl = adUrl

},

            AddDate = date,
            PhotoUrl = photoUrl,
            Title = title,

```

```

        Model = model,
        Price = price,
        Mileage = mileage,
        Location = location,
        Fuel = fuel,
        Transmission = transmission,
        Vin = vin,
        LicensePlate = licensePlate,
        Description = description,
        IsWithDamage = photoUrlList.Count == 0 ?

false : null,

        AllPhotos = photoUrlList
    });
}
catch (Exception ex)
{
    Log.Error($"CarSearchService.GetCarAdListOlx
foreach" +
        $"\\nMessage:      {ex.Message}\\nErrorTrace:
{ex.StackTrace}" +
        (ex.InnerException != null ?
$"\\nInnerException: {ex.InnerException}" : ""));
}
}
}
}
}
}
}
}
}
else
{
    Log.Information("CarSearchService.GetCarAdListOlx Request error");
}
}
catch (Exception ex)
{
    Log.Error($"CarSearchService.GetCarAdListOlx" +
        $"\\nMessage: {ex.Message}\\nErrorTrace: {ex.StackTrace}" +
        (ex.InnerException != null ? $"\\nInnerException: {ex.InnerException}"
: ""));
}
    Log.Information($"CarSearchService.GetCarAdListOlx      adList      count      =
{adList.Count}");
    return adList;
}
private static string GetOlxPhotoLink(string photoUrl, int width)
{

```

```

        if (!string.IsNullOrEmpty(photoUrl))
        {
            string pattern = @"\"/([a-zA-Z0-9+)-UA\"/";
            Match match = Regex.Match(photoUrl, pattern);
            if (match.Success)
            {
                string parameterValue = match.Groups[1].Value;
                photoUrl =
@$"https://ireland.apollo.olxcn.com/v1/files/{parameterValue}-UA/image;s={width}x0";
            }
        }
        return photoUrl;
    }

    private async Task<CarAdResponse> GetCarAdInfoRia(string riaUrl)
    {
        var ad = new CarAdResponse();
        try
        {
            if (string.IsNullOrEmpty(riaUrl))
                throw new NotFoundException(nameof(riaUrl));
            var response = await _httpClient.SendGetAsync(riaUrl);
            if (response.IsSuccessStatusCode)
            {
                var content = await response.Content.ReadAsStringAsync();
                var config = Configuration.Default;
                var context = BrowsingContext.New(config);
                var document = await context.OpenAsync(req => req.Content(content));
                ad.AdId = new AdIdModel()
                {
                    RiaId = int.Parse(
                        document.QuerySelector("li.item:contains('ID
авто')")?.QuerySelector("span.bold")?.TextContent?.Trim());
                };
                ad.AdUrl = new AdUrlModel() { AdRiaUrl = riaUrl };
                ad.Title =
                    document.QuerySelector(".auto-
content_title")?.TextContent?.Trim();
                ad.Model = document.QuerySelector(".under-head")?.TextContent?.Trim();
                ad.Price =
                    document.QuerySelector(".price_value")?.TextContent?.Trim()?.Split('$')?.FirstOrDefault()
                    ?.Trim();
                ad.Mileage =
                    document.QuerySelector(".bold.dhide")?.TextContent?.Trim()?.Split('
')?.FirstOrDefault()?.Trim();
                ad.Location =
                    document.QuerySelector(".bold.dhide .i-
block")?.TextContent?.Trim();
                ad.Vin = document.QuerySelector(".label-vin")?.TextContent?.Trim();
            }
        }
    }

```

```

        ad.LicensePlate = document.QuerySelector(".state-
num.ua")?.TextContent?.Trim()?.Split(" Ми")?[0];
        ad.Description = document.QuerySelector(".full-
description")?.TextContent?.Trim();
        if (!string.IsNullOrEmpty(ad.Mileage))
            ad.Mileage += " тис. км";
        var technicalInfo = document.QuerySelector("#details");
        var ddElements = technicalInfo.QuerySelectorAll("dd");
        var engine = ddElements.FirstOrDefault(x => x.TextContent.StartsWith("
Двигун"))?.TextContent
            .Split('•')
            .Select(e => e.Trim())
            .Where(e => !string.IsNullOrEmpty(e))
            .ToList();
        ad.EngineCapacity = engine?.FirstOrDefault()?.Replace("Двигун ", "");
        ad.Fuel = engine?.LastOrDefault();
        ad.Transmission = ddElements.FirstOrDefault(x =>
x.TextContent.StartsWith(" Коробка передач"))?.TextContent?.Trim()?
            .Replace("Коробка передач ", "");
        ad.Drive = ddElements.FirstOrDefault(x => x.TextContent.StartsWith("
Привід"))?.TextContent?.Trim()?.Replace("Привід ", "");
        ad.Color = ddElements.FirstOrDefault(x => x.TextContent.StartsWith("
Колір"))?.TextContent?.Trim()?.Replace("Колір ", "");
        var photoList =
document.QuerySelector(".wrapper[photocontainer='photo']")?
            .QuerySelectorAll("a.photo-74x56")?
            .Select(x =>
x?.QuerySelector("img")?.GetAttribute("src"))?.ToList();
        photoList?.RemoveAll(x => x.Contains("no-photo"));
        ad.PhotoList = new List<ImageModel>();
        foreach (var item in photoList)
        {
            ad.PhotoList.Add(new ImageModel
            {
                Original = item.Replace("s.jpg", "hd.jpg"),
                Thumbnail = item
            });
        }
        ad.PhotoUrl = ad.PhotoList?.FirstOrDefault()?.Original;
        ad.ParamsList = new List<Params>();
        ad.ParamsList.AddRange(document.QuerySelectorAll(".label-param
.item")?
            .Select(item => new Params { Key = item.TextContent.Trim()
        }).ToList());
        var carNestedParams = ddElements?
            .Where(x => x.TextContent.Contains("Безпека") ||
x.TextContent.Contains("Салон та комфорт") ||

```

```

        x.TextContent.Contains("Мультимедіа") ||
x.TextContent.Contains("Оптика") ||
        x.TextContent.Contains("Система допомоги при
паркуванні") || x.TextContent.Contains("Подушка безпеки") ||
        x.TextContent.Contains("Стан"));
        var carParams = ddElements?
            .Where(x => !x.TextContent.Contains("Пробір") &&
!x.TextContent.Contains("Двигун") &&
!x.TextContent.Contains("Коробка передач") &&
!x.TextContent.Contains("Привід") &&
!x.TextContent.Contains("Колір") &&
!x.TextContent.Contains("Опис") &&
!x.TextContent.Contains("Безпека") &&
!x.TextContent.Contains("Салон та комфорт") &&
!x.TextContent.Contains("Мультимедіа") &&
!x.TextContent.Contains("Оптика") &&
!x.TextContent.Contains("Система допомоги при
паркуванні") && !x.TextContent.Contains("Подушка безпеки") &&
!x.TextContent.Contains("Стан"));
        foreach (var item in carParams)
        {
            ad.ParamsList.Add(new Params
            {
                Key = item.QuerySelector(".label")?.TextContent?.Trim(),
                Param = item.QuerySelector(".argument")?.TextContent?.Trim()
            });
        }
        ad.ParamsList?.RemoveAll(x => string.IsNullOrEmpty(x.Key));
        var paramsToFix = ad.ParamsList?.Where(x => x.Param == null || x.Param
== "null").ToList();
        foreach (var item in paramsToFix)
        {
            item.Param = "Так";
        }
        ad.NestedParamsList = new List<NestedParams>();
        foreach (var item in carNestedParams)
        {
            ad.NestedParamsList.Add(new NestedParams
            {
                Key = item.QuerySelector(".label")?.TextContent?.Trim(),
                Params = item.QuerySelectorAll(".argument .i-block")?
                    .Select(x => x.TextContent?.Trim()?.Replace("•",
""))?.ToList()
            });
        }
        ad.NestedParamsList?.RemoveAll(x => string.IsNullOrEmpty(x.Key) ||
x.Key.Contains("Опис"));

```

```

    }
    else
    {
        Log.Information("CarSearchService.GetCarAdInfoRia Request error");
    }
}
catch (Exception ex)
{
    Log.Error($"CarSearchService.GetCarAdInfoRia" +
        $"\nMessage: {ex.Message}\nErrorTrace: {ex.StackTrace}" +
        (ex.InnerException != null ? $"\nInnerException: {ex.InnerException}"
: ""));
}
Log.Information($"CarSearchService.GetCarAdInfoRia success");
return ad;
}
private async Task<CarAdResponse> GetCarAdInfoOlx(string olxUrl)
{
    var ad = new CarAdResponse();
    try
    {
        if (string.IsNullOrEmpty(olxUrl))
            throw new NotFoundException(nameof(olxUrl));
        HttpResponseMessage response = await _httpClient.SendGetAsync(olxUrl);
        if (response.IsSuccessStatusCode)
        {
            var htmlContent = await response.Content.ReadAsStringAsync();
            var parser = new HtmlParser();
            var document = parser.ParseDocument(htmlContent);
            var scriptElements = document.QuerySelectorAll("script[type='text/javascript']");
            if (scriptElements.Length >= 2)
            {
                var secondScriptElement = scriptElements[1];
                var scriptContent = secondScriptElement.TextContent;
                var startIndex = scriptContent.IndexOf("window.__PRERENDERED_STATE__ = \"");
                if (startIndex >= 0)
                {
                    startIndex += "window.__PRERENDERED_STATE__ = ".Length;
                    int endIndex = scriptContent.IndexOf("};\"", startIndex);
                    if (endIndex >= 0)
                    {
                        string jsonData = scriptContent.Substring(startIndex,
                            endIndex - startIndex + 1);
                        if (!string.IsNullOrEmpty(jsonData))
                        {

```

```

        string decodedJson = Regex.Unescape(jsonData);
        JObject data = JObject.Parse(decodedJson);
        JObject adObject = (JObject)data["ad"]["ad"];

        ad.AdId = new AdIdModel()
        {
            OlxId = (int)adObject["id"]
        };
        ad.AdUrl = new AdUrlModel() { AdOlxUrl = olxUrl };
        ad.Title = (string)adObject["title"];
        ad.Model =
        $"{(string)adObject["params"].FirstOrDefault(p => (string)p["key"] ==
        "model")?["value"]} " +
            $"{(string)adObject["params"].FirstOrDefault(p =>
        (string)p["key"] == "modification")?["value"]}";
        ad.Price =
        (string)adObject["price"]["regularPrice"]["value"];
        ad.Mileage =
        (string)adObject["params"].FirstOrDefault(p => (string)p["key"] ==
        "motor_mileage_thou")?["value"];
        ad.Location =
        (string)adObject["location"]["pathName"];
        ad.Vin = (string)adObject["params"].FirstOrDefault(p
        => (string)p["key"] == "vin_number")?["value"];
        ad.LicensePlate =
        (string)adObject["params"].FirstOrDefault(p => (string)p["key"] ==
        "license_plate")?["value"];
        ad.Description = (string)adObject["description"];
        ad.EngineCapacity =
        (string)adObject["params"].FirstOrDefault(p => (string)p["key"] ==
        "motor_engine_size_litre")?["value"];
        ad.Fuel = (string)adObject["params"].FirstOrDefault(p
        => (string)p["key"] == "fuel_type")?["value"];
        ad.Transmission =
        (string)adObject["params"].FirstOrDefault(p => (string)p["key"] ==
        "transmission_type")?["value"];
        ad.Drive = (string)adObject["params"].FirstOrDefault(p
        => (string)p["key"] == "drive_type")?["value"];
        ad.Color = (string)adObject["params"].FirstOrDefault(p
        => (string)p["key"] == "color")?["value"];
        ad.PhotoList = new List<ImageModel>();
        ad.PhotoUrl =
        (string)adObject["photos"].FirstOrDefault();
        foreach (var photo in adObject["photos"])
        {
            var imageModel = new ImageModel();
            string photoUrl = (string)photo;

```

```

        if (!string.IsNullOrEmpty(photoUrl))
        {
            imageModel.Original = photoUrl;
            string pattern = @"\"/([a-zA-Z0-9+)-UA\"/";
            Match match = Regex.Match(photoUrl, pattern);
            if (match.Success)
            {
                string parameterValue =
match.Groups[1].Value;

                photoUrl =
@$"https://ireland.apollo.olxcdn.com/v1/files/{parameterValue}-UA/image;s=250x0";
                imageModel.Thumbnail = photoUrl;
            }
            ad.PhotoList.Add(imageModel);
        }
    }

    ad.ParamsList = new List<Params>();
    ad.NestedParamsList = new List<NestedParams>();
    var carNestedParams = adObject["params"]?.Where(x =>
        x["name"].ToString().Contains("Безпека") ||
x["name"].ToString().Contains("Комфорт") ||
        x["name"].ToString().Contains("Мультимедіа") ||
x["name"].ToString().Contains("Оптика") ||
        x["name"].ToString().Contains("Система допомоги
при паркуванні") || x["name"].ToString().Contains("Подушка безпеки") ||
        x["name"].ToString().Contains("Стан") ||
x["name"].ToString().Contains("Технічний стан") ||
        x["name"].ToString().Contains("Інше"));
    var carParams = adObject["params"]?.Where(x =>
        !x["name"].ToString().Contains("Модель") &&
!x["name"].ToString().Contains("Модифікація") &&
        !x["name"].ToString().Contains("Рік випуску") &&
!x["name"].ToString().Contains("Пробіг") &&
        !x["name"].ToString().Contains("Колір") &&
!x["name"].ToString().Contains("Коробка передач") &&
        !x["name"].ToString().Contains("Тип приводу") &&
!x["name"].ToString().Contains("Вид палива") &&
        !x["name"].ToString().Contains("Об'єм двигуна") &&
!x["name"].ToString().Contains("Технічний стан") &&
        !x["name"].ToString().Contains("Безпека") &&
!x["name"].ToString().Contains("Комфорт") &&
        !x["name"].ToString().Contains("Мультимедіа") &&
!x["name"].ToString().Contains("Оптика") &&
        !x["name"].ToString().Contains("Система допомоги
при паркуванні") && !x["name"].ToString().Contains("Подушка безпеки") &&
        !x["name"].ToString().Contains("Стан") &&
!x["name"].ToString().Contains("Інше"));

```

```

        foreach (var item in carParams)
        {
            ad.ParamsList.Add(new Params
            {
                Key = (string)item["name"],
                Param = (string)item["value"]
            });
        }
        ad.ParamsList?.RemoveAll(x
string.IsNullOrEmpty(x.Key)); =>
        foreach (var item in carNestedParams)
        {
            ad.NestedParamsList.Add(new NestedParams
            {
                Key = (string)item["name"],
                Params = item["value"].ToString().Split(",
").ToList()

            });
        }
        ad.NestedParamsList?.RemoveAll(x
string.IsNullOrEmpty(x.Key)); =>
    }
}
}
}
}
else
{
    Log.Information("CarSearchService.GetCarAdInfoOlx Request error");
}
}
catch (Exception ex)
{
    Log.Error($"CarSearchService.GetCarAdInfoOlx" +
        $"\nMessage: {ex.Message}\nErrorTrace: {ex.StackTrace}" +
        (ex.InnerException != null ? $"\nInnerException: {ex.InnerException}"
: ""));
}
Log.Information($"CarSearchService.GetCarAdInfoOlx success");
return ad;
}
private string MakeRiaRequestPath(CarSearchRequest request)
{
    var uri = "";
    try
    {
        var uriBuilder = new UriBuilder(_riaPath);

```

```

var parameters =
System.Web.HttpUtility.ParseQueryString(uriBuilder.Query);
var cultureInfo = new CultureInfo("en-US");
var sortBy = "";
switch (request.SortBy)
{
    case SortByEnum.PriceAsc:
        sortBy = "price.asc";
        break;
    case SortByEnum.PriceDesc:
        sortBy = "price.desc";
        break;
    case SortByEnum.CreatedDesc:
        sortBy = "dates.created.desc";
        break;
    default:
        sortBy = "dates.created.desc";
        break;
}
parameters["sort[0].order"] = sortBy;
parameters["page"] = request.PageNum.ToString();
if (request.YearFrom != null)
    parameters["year[0].gte"] = request.YearFrom.ToString();
if (request.YearTo != null)
    parameters["year[0].lte"] = request.YearTo.ToString();
if (request.MileageFrom != null)
    parameters["mileage.gte"] = request.MileageFrom.ToString();
if (request.MileageTo != null)
    parameters["mileage.lte"] = request.MileageTo.ToString();
if (request.EngineLitrFrom != null)
    parameters["engine.gte"] = string.Format(cultureInfo, "{0}",
request.EngineLitrFrom);
if (request.EngineLitrTo != null)
    parameters["engine.lte"] = string.Format(cultureInfo, "{0}",
request.EngineLitrTo);
if (request.RegionId != null)
    parameters["region.id[0]"] = request.RegionId.ToString();
if (request.CityId != null)
    parameters["city.id[0]"] = request.CityId.ToString();
if (request.PriceFrom != null)
    parameters["price.USD.gte"] = request.PriceFrom.ToString();
if (request.PriceTo != null)
    parameters["price.USD.lte"] = request.PriceTo.ToString();
if (request.BrandId != null)
    parameters["$brand.id[0]"] = request.BrandId.ToString();
if (request.BrandId != null)
{

```

```

        for (int i = 0; i < request.Models?.Count; i++)
        {
            parameters[$"model.id[{i}]"] = request.Models[i].ToString();
        }
    }
    for (int i = 0; i < request.Gearboxes?.Count; i++)
    {
        parameters[$"gearbox.id[{i}]"] = request.Gearboxes[i].ToString();
    }
    for (int i = 0; i < request.BodyStyles?.Count; i++)
    {
        parameters[$"body.id[{i}]"] = request.BodyStyles[i].ToString();
    }
    for (int i = 0; i < request.FuelTypes?.Count; i++)
    {
        parameters[$"fuel.id[{i}]"] = request.FuelTypes[i].ToString();
    }

    uriBuilder.Query = parameters.ToString();
    uri = uriBuilder.Uri.ToString();
}
catch (Exception ex)
{
    Log.Error($"CarSearchService.MakeRiaRequestPath" +
        $"{Environment.NewLine}Message: {ex.Message}\nErrorTrace: {ex.StackTrace}" +
        (ex.InnerException != null ? $"{Environment.NewLine}InnerException: {ex.InnerException}"
        : ""));
}

Log.Information($"CarSearchService.MakeRiaRequestPath uri {uri}");
return uri;
}

private string MakeOlxRequestPath(CarSearchRequest request)
{
    var uri = "";
    try
    {
        var uriBuilder = new UriBuilder(_olxPath);
        var parameters =
System.Web.HttpUtility.ParseQueryString(uriBuilder.Query);
        var cultureInfo = new CultureInfo("en-US");
        if (request.PageNum > 1)
            parameters["page"] = request.PageNum.ToString();
        if (request.BrandId != null)
        {
            var olxBrandId = SearchFilterMapper.GetOlxBrand(request.BrandId ?? -
1);

```

```

        if (olxBrandId == "")
            throw new NotFoundException($"{nameof(olxBrandId)} by ria
{request.BrandId}");
        uriBuilder.Path += $"{olxBrandId}/";
        if (request.Models?.Count > 0)
        {
            var olxModels = new List<string>();
            foreach (var model in request.Models)
            {
                olxModels.Add(SearchFilterMapper.GetOlxBrandModel(request.Br
andId ?? -1, model));
            }
            if (olxModels.Count < 1)
                throw new NotFoundException($"{nameof(olxModels)} " +
                    $"by ria brandId {request.BrandId}, models
[{string.Join(";", request.Models)}]");
            for (int i = 0; i < olxModels.Count; i++)
            {
                parameters["$search%5Bfilter_enum_model%5D%5B{i}%5D"] =
olxModels[i];
            }
        }
        if (request.CityId != null)
        {
            var olxCityId = SearchFilterMapper.GetOlxCity(request.RegionId ?? -1,
request.CityId ?? -1);
            if (olxCityId == "")
                throw new NotFoundException($"{nameof(olxCityId)} by ria regionId
{request.RegionId}, cityId {request.CityId}");
            uriBuilder.Path += $"{olxCityId}/";
        }
        else if (request.RegionId != null)
        {
            var olxRegionId = SearchFilterMapper.GetOlxRegion(request.RegionId ??
-1);
            if (olxRegionId == "")
                throw new NotFoundException($"{nameof(olxRegionId)} by ria
{request.RegionId}");
            uriBuilder.Path += $"{SearchFilterMapper.GetOlxRegion(request.RegionId
?? -1)}/";
        }
        var sortBy = "";
        switch (request.SortBy)
        {
            case SortByEnum.PriceAsc:
                sortBy = "filter_float_price:asc";

```

```

        break;
    case SortByEnum.PriceDesc:
        sortBy = "filter_float_price:desc";
        break;
    case SortByEnum.CreatedDesc:
        sortBy = "created_at:desc";
        break;
    default:
        sortBy = "created_at:desc";
        break;
    }
    parameters["search%5Border%5D"] = sortBy;
    if (request.YearFrom != null)
        parameters["search%5Bfilter_float_motor_year%3Afrom%5D"] =
request.YearFrom.ToString();
    if (request.YearTo != null)
        parameters["search%5Bfilter_float_motor_year%3Ato%5D"] =
request.YearTo.ToString();
    if (request.MileageFrom != null)
        parameters["search%5Bfilter_float_motor_mileage_thou%3Afrom%5D"] =
request.MileageFrom.ToString();
    if (request.MileageTo != null)
        parameters["search%5Bfilter_float_motor_mileage_thou%3Ato%5D"] =
request.MileageTo.ToString();
    if (request.EngineLitrFrom != null)
        parameters["search%5Bfilter_float_motor_engine_size_litre%3Afrom%5D"]
= string.Format(cultureInfo, "{0}", request.EngineLitrFrom);
    if (request.EngineLitrTo != null)
        parameters["search%5Bfilter_float_motor_engine_size_litre%3Ato%5D"] =
string.Format(cultureInfo, "{0}", request.EngineLitrTo);
    if (request.PriceFrom != null)
        parameters["search%5Bfilter_float_price%3Afrom%5D"] =
request.PriceFrom.ToString();
    if (request.PriceTo != null)
        parameters["search%5Bfilter_float_price%3Ato%5D"] =
request.PriceTo.ToString();
    for (int i = 0; i < request.Gearboxes?.Count; i++)
    {
        parameters["$search%5Bfilter_enum_transmission_type%5D%5B{i}%5D"] =
SearchFilterMapper.GetOlxGearbox(request.Gearboxes[i]);
    }
    for (int i = 0; i < request.BodyStyles?.Count; i++)
    {
        parameters["$search%5Bfilter_enum_car_body%5D%5B{i}%5D"] =
SearchFilterMapper.GetOlxBodyStyle(request.BodyStyles[i]);
    }
    for (int i = 0; i < request.FuelTypes?.Count; i++)

```

```

        {
            parameters["$search%5Bfilter_enum_fuel_type%5D%5B{i}%5D"]
            = SearchFilterMapper.GetOlxFuelType(request.FuelTypes[i]);
        }
        uriBuilder.Query = parameters.ToString();
        uri = uriBuilder.Uri.ToString();
    }
    catch (Exception ex)
    {
        Log.Error($"CarSearchService.MakeOlxRequestPath" +
            $"\nMessage: {ex.Message}\nErrorTrace: {ex.StackTrace}" +
            (ex.InnerException != null ? $"\nInnerException: {ex.InnerException}"
            : ""));
    }
    Log.Information($"CarSearchService.MakeOlxRequestPath uri {uri}");
    return uri;
}
}

```

Б.2 Текст файла ImageRecognitionService.cs

```

public class ImageRecognitionService : IImageRecognitionService
{
    private IConfiguration _configuration;
    private IHttpSender _httpSender;
    private readonly IHubContext<CarHub> _hubContext;
    private string _apiPath;

    public ImageRecognitionService(IConfiguration configuration, IHttpSender
httpSender, IHubContext<CarHub> hubContext)
    {
        _configuration = configuration;
        _httpSender = httpSender;
        _hubContext = hubContext;
        _apiPath = _configuration["ImageRecognition:ApiUrl"]!;
    }

    public async Task<bool> CheckHealth()
    {
        var response = await _httpSender.SendGetAsync(_apiPath + "checkHealth");
        return response.IsSuccessStatusCode;
    }

    public async Task RecognizeDamage(List<CarAdShortResponse> carAdList)
    {
        await Task.WhenAll(carAdList.Select(async ad =>
        {
            try

```

```

    {
        using (HttpClient client = new HttpClient())
        {
            string jsonRequest = JsonConvert.SerializeObject(ad.AllPhotos);
            string apiUrl = _apiPath + "detect";
            HttpResponseMessage response = await client.PostAsync(apiUrl, new
StringContent(jsonRequest, Encoding.UTF8, "application/json"));
            if (response.IsSuccessStatusCode)
            {
                string jsonResponse = await
response.Content.ReadAsStringAsync();
                var isCarDamaged =
JsonConvert.DeserializeObject<DamageRecognitionResult>(jsonResponse)?.IsCarDamaged;

                ad.IsWithDamage = isCarDamaged ?? false;
            }
            else
            {
                ad.IsWithDamage = false;
            }
        }
    }
    catch (Exception ex)
    {
        Log.Error($"ImageRecognitionService.RecognizeDamage {ex.GetType()}
occurred" +
        $"{\nMessage: {ex.Message}\nErrorTrace: {ex.StackTrace}" +
        (ex.InnerException != null ? $"{\nInnerException: {ex.InnerException}"
: ""));
    }
    await _hubContext.Clients.All.SendAsync("OnNewDentDetect", new
RecognitionResult
    {
        AdId = ad.AdId,
        IsWithDamage = ad.IsWithDamage ?? false
    });
});
});
}
}

```

Б.3 Текст файла ImageRecognitionApi.py

```

app = FastAPI()
currentDirectory = os.path.dirname(os.path.abspath(__file__))
weightsName = 'weights.pt'
weightsPath = os.path.join(currentDirectory, weightsName)

```

```

model = torch.hub.load('ultralytics/yolov5', 'custom', path=weightsPath)

def resize_image(image):
    width, height = image.size
    max_size = 640
    if width > height:
        new_width = max_size
        new_height = int(max_size * height / width)
    else:
        new_height = max_size
        new_width = int(max_size * width / height)
    resized_img = image.resize((new_width, new_height))
    new_img = Image.new("RGB", (max_size, max_size), (0, 0, 0))
    x_offset = (max_size - new_width) // 2
    y_offset = (max_size - new_height) // 2
    new_img.paste(resized_img, (x_offset, y_offset))
    return new_img

def isCarHasDamage(data):
    for entry in data:
        for item in entry:
            if "confidence" in item and item["confidence"] > 0.8:
                return True
    return False

@app.get("/checkHealth")
def checkHealth():
    return JSONResponse(content={}, status_code=200)

@app.post("/detect")
async def detect(image_urls: List[str]):
    try:
        images = []
        async with aiohttp.ClientSession() as session:
            async def fetch_image(url):
                async with session.get(url) as response:
                    if response.status == 200:
                        image_bytes = await response.read()
                        image = Image.open(BytesIO(image_bytes))
                        resized_image = resize_image(image)
                        images.append(resized_image)
            tasks = [fetch_image(url) for url in image_urls]
            await asyncio.gather(*tasks)
        results = model(images, size=640)
        detectResult = []
        for res in results.pandas().xyxy:
            if not res.empty:
                detectResult.append(res.to_json(orient="records"))
        isCarDamaged = isCarHasDamage([json.loads(item) for item in detectResult])

```

```
        return JsonResponse(content={"isCarDamaged": isCarDamaged})
    except Exception as e:
        return JsonResponse(content={"error": str(e.with_traceback())})
if __name__ == "__main__":
    uvicorn.run(app, host="127.0.0.1", port=8000)
```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
кафедра програмних засобів

Дипломна кваліфікаційна робота магістра
на тему

ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ РОЗПІЗНАВАННЯ
ПОШКОДЖЕНЬ ДЛЯ АГРЕГАТОРА ПОШУКУ ОГолошень ПРО ПРОДАЖ
АВТОМОБІЛІВ
RESEARCH AND SOFTWARE IMPLEMENTATION OF DEFECT RECOGNITION
METHODS FOR CAR SALE AGGREGATOR

Виконав:
студент групи КНТ-222м

Валерій ЛАКТИОНОВ

Керівник:
к.т.н., доцент

Валерій ЛЬОВКІН

2023

Рисунок В.1 – Титульний слайд

Мета, об'єкт і предмет дослідження

- **Мета роботи** – розробка програмного забезпечення пошуку оголошень про продаж автомобілів для підвищення зручності користувачів через агрегацію даних та інтегрування розпізнавання пошкоджень автомобілів у процес пошуку.
- **Об'єкт дослідження** – процес розпізнавання пошкоджень автомобілів.
- **Предмет дослідження** – програмне забезпечення для пошуку оголошень про продаж автомобілів.

Рисунок В.2 – Мета, об'єкт і предмет дослідження

Завдання дослідження

- проаналізувати існуючі вебсайти агрегаторів оголошень про продаж автомобілів;
- розробити метод автоматичного виявлення пошкоджень автомобілів на основі агрегації даних;
- виконати проектування програмного забезпечення агрегатора пошуку оголошень про продаж автомобілів;
- реалізувати сервер та вебзастосунок агрегатора пошуку оголошень про продаж автомобілів;
- виконати експериментальне дослідження проблеми автоматичного виявлення пошкоджень автомобілів.

3

Рисунок В.3 – Завдання дослідження

Аналіз предметної області

Особливості	Аналоги		Розробка
	АвтоПошук	AutoMoto	CarFinder
Пошук та фільтрація оголошень	+	+	+
Виявлення однакових оголошень	+	+	+
Перегляд списку знайдених оголошень	+	+	+
Перегляд детальної інформації по оголошенню	+	+	+
Перехід на майданчики оголошень	+	+	+
Створення облікового запису	–	+	+
Збереження фільтру пошуку у збережені	+	–	+
Збереження оголошень у збережені	+	+	–
Керування збереженими	+	+	+
Перегляд відгуків та новин	–	+	–
Розпізнавання пошкоджень	–	–	+
Підписка на повідомлення про нові оголошення через Telegram бот	–	–	+

4

Рисунок В.4 – Аналіз предметної області

Метод автоматичного виявлення пошкоджень автомобілів на основі агрегації даних

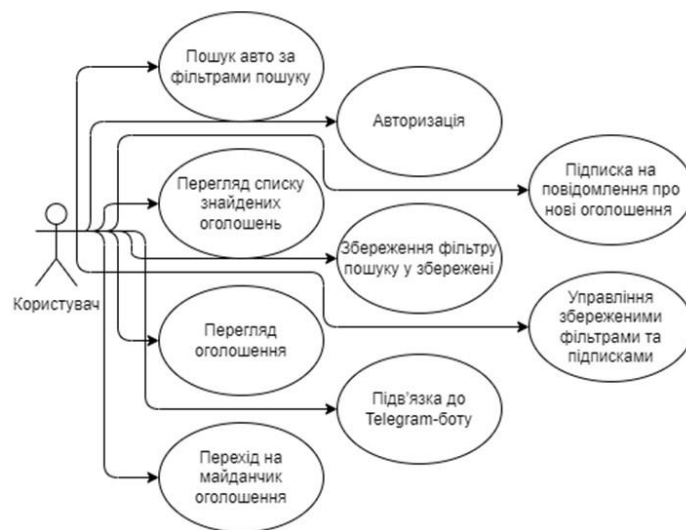
Етапи вирішення поставленої задачі розпізнавання пошкоджень автомобілів:

- збір даних;
- розмітка даних;
- аугментація даних;
- створення датасету;
- створення моделі розпізнавання;
- навчання моделі з модифікованою архітектурою;
- агрегація даних з обраних джерел даних з розпізнаванням на основі навченої моделі.

5

Рисунок В.5 – Метод автоматичного виявлення пошкоджень автомобілів на основі агрегації даних

Узагальнена діаграма прецедентів



6

Рисунок В.6 – Узагальнена діаграма прецедентів

Вибір засобів розробки

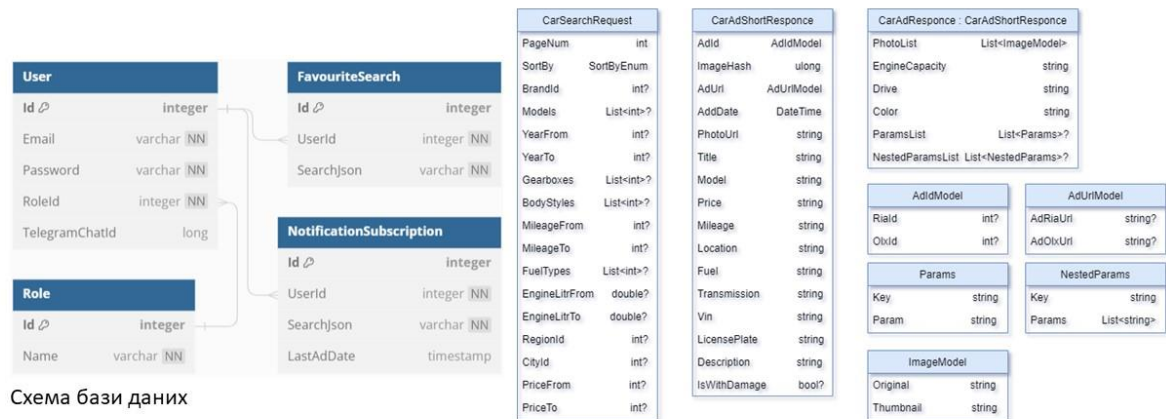
Критерії порівняння	Мова програмування сервера			Критерії порівняння	Засоби розробки вебклієнта		
	C#	Python	Java		React	Vue.js	Angular
Статична типізація	+	-	+	Галузь застосування	++	+	++
Продуктивність	++	+	++	Продуктивність	+	+	+
Зручність середовища розробки	++	+	+	Бібліотеки	++	+	++
Сприйняття коду	++	+	+	Сприйняття коду	++	+	+

Критерії порівняння	Мова програмування модулю розпізнавання			Критерії порівняння	Мова програмування вебклієнта	
	Python	Java	C++		TypeScript	JavaScript
Продуктивність	+	++	++	Статична типізація	+	-
Бібліотеки для машинного навчання	++	+	+	Продуктивність	++	+
Автоматичне управління пам'яттю	++	++	-	Бібліотеки	+	+
Швидкість напис. коду	++	+	-	Сприйняття коду	++	+

7

Рисунок В.7 – Вибір засобів розробки

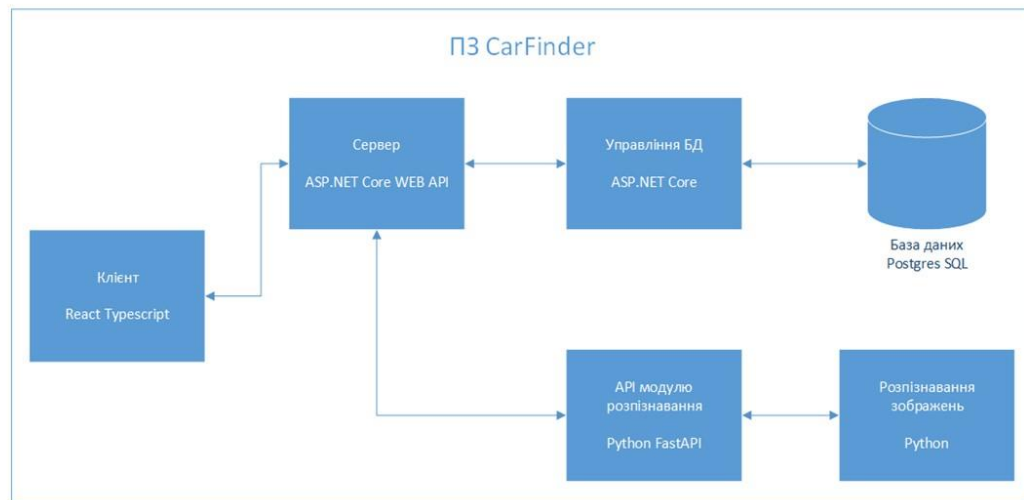
Схема даних



8

Рисунок В.8 – Схема даних

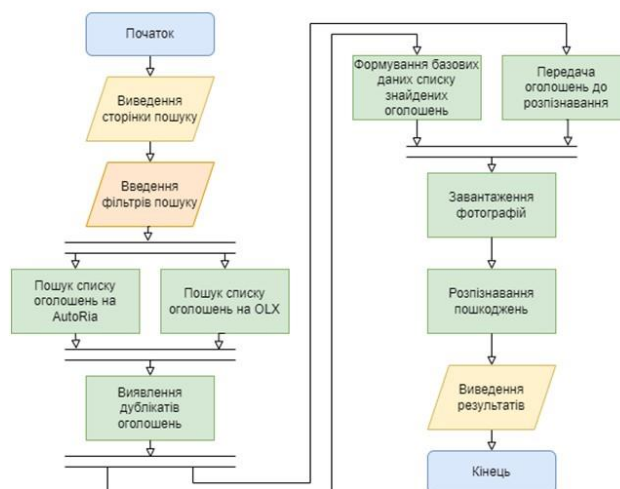
Схема структури програми



9

Рисунок В.9 – Схема структури програми

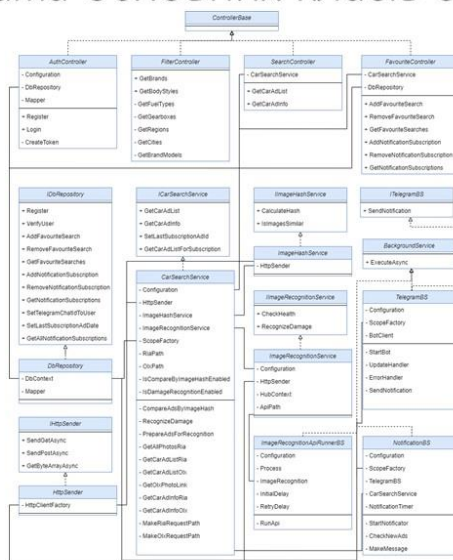
Функціональна схема процесу пошуку оголошень



10

Рисунок В.10 – Функціональна схема процесу пошуку оголошень

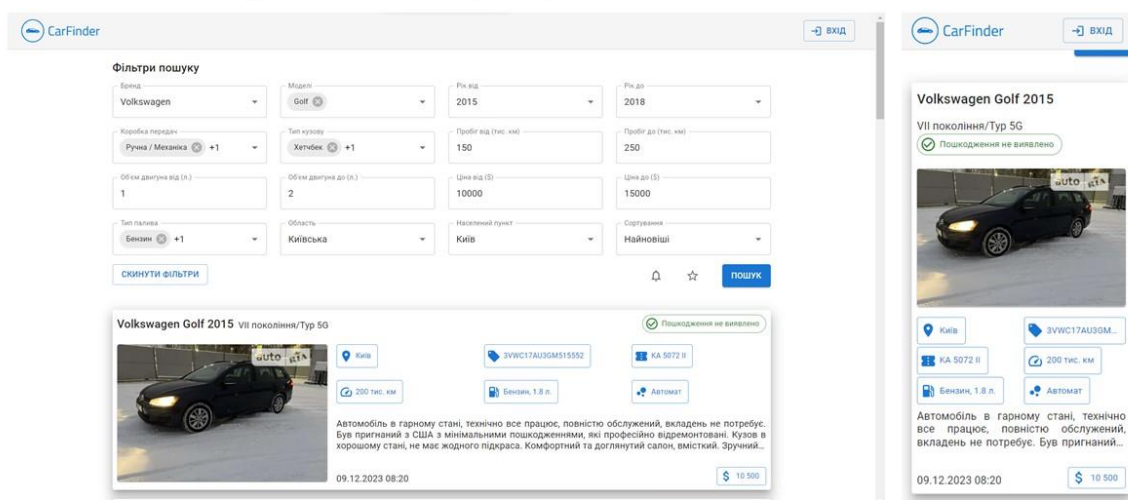
Діаграма основних класів сервера



11

Рисунок В.11 – Діаграма основних класів сервера

Розроблене програмне забезпечення

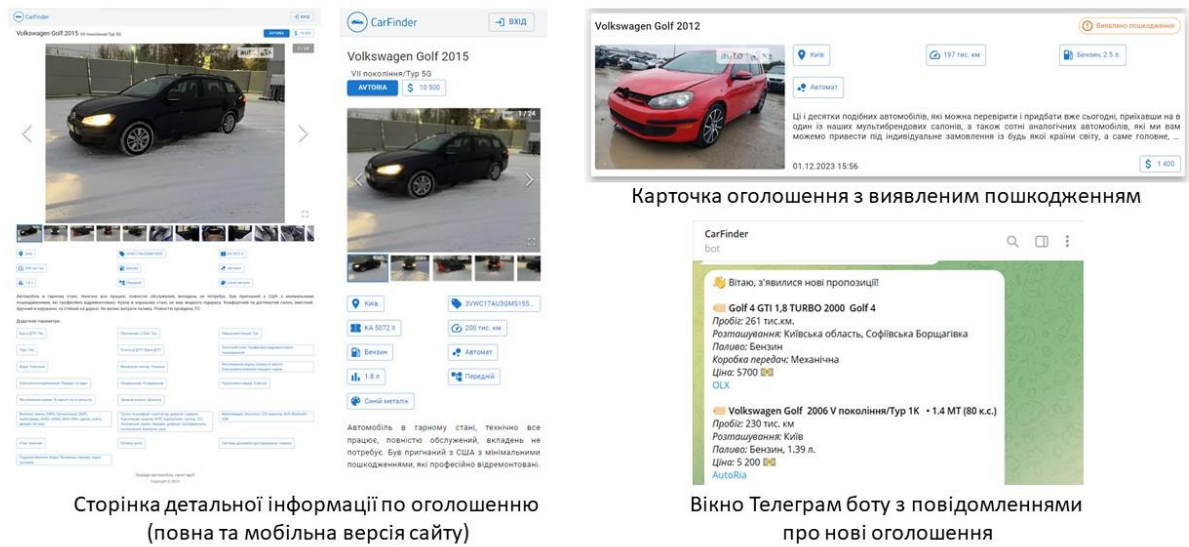


Головна сторінка сайту (повна та мобільна версія сайту)

12

Рисунок В.12 – Розроблене програмне забезпечення

Розроблене програмне забезпечення



13

Рисунок В.13 – Розроблене програмне забезпечення

Результати експериментального дослідження

Показник	Модель з оригінальною архітектурою	Модель з модифікованою архітектурою
GFLOPs	47,9	47,1
P	0,745	0,754
R	0,605	0,609
mAP50	0,615	0,629
mAP50-95	0,307	0,315
Speed: pre-process	1,1 мс	1,1 мс
Speed: inference	83,4 мс	66,1 мс

14

Рисунок В.14 – Результати експериментального дослідження

Висновки

Дослідження та програмна реалізація методів розпізнавання пошкоджень для агрегатора пошуку оголошень про продаж автомобілів є комплексним підходом до створення інноваційної системи, сфокусованої на підвищенні ефективності та функціональності у сфері автомобільних оголошень.

Було створено програмне забезпечення для пошуку оголошень автомобілів на різних майданчиках оголошень, яке здатне автоматично розпізнавати наявність пошкоджень на фотографіях автомобілів, що дозволить прискорити та спростити процес пошуку відповідного автомобіля серед сотень оголошень.

Наукова новизна роботи полягає в запропонованому методі автоматичного виявлення пошкоджень автомобілів на основі агрегації даних, який виконує модифікацію архітектури моделі розпізнавання, що дозволяє виявляти пошкодження на фотографіях, прикріплених до оголошень, та класифікувати оголошення про продаж автомобілів. Це автоматизує процес попередньої оцінки стану автомобіля та надає користувачам більше корисних даних про автомобілі, виставлених на продаж.

Реалізована система є значним кроком у сфері агрегації оголошень про продаж автомобілів, інтегруючи технології машинного навчання із сучасними технологіями. При цьому залишаються перспективи для подальшого вдосконалення системи, розширення функціональності і адаптації до потреб ринку, що змінюються. Все це робить цю роботу важливим етапом у розвитку сучасних технологій в автомобільній індустрії та цифровій сфері.

Проведене тестування та експериментальне дослідження підтвердило працездатність розробленої програмної системи.