Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра інженерії програмного забезпечення

ДИПЛОМНИЙ ПРОЕКТ

Інтелектуальна рекомендаційна система вибору автомобілів

Рівень вищої освіти Перший (бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДППІЗ.170102.01.02.ПЗ

Виконав студент IV курсу група ІПЗ-17-1 _____ Д.С. Григорчук
Підпис Ініціали, прізвище

Керівник канд. техн. наук, доцент _____ Ю.В.Форкун
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ Г. І. Радельчук
Підпис Ініціали, прізвище

До захисту допускаю:

Завідувач кафедри інженерії
програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

16 червня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Програмування та комп'ютерних і телекомунікаційних систем
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
05 02 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Григорчуку Данилі Сергійовичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Інтелектуальна рекомендаційна система вибору автомобілів ----------

Керівник проєкту (роботи) Форкун Юрій Вікторович, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 05.02.2021 р. № 11

2. Строк подання студентом проєкту (роботи) на кафедру 01.06.2021 р.

3. Вихідні дані до проєкту (роботи) Матеріали переддипломної практики

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація системи, тестування програмної системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів дипломного проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Радельчук Г. І., доцент кафедри ІПЗ		
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ		

7. Дата видачі завдання «_05_» __лютого_ 2021 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітк
1 Ознайомлення з тематикою дипломного проектування (ДП), визначення та узгодження індивідуальних там ДП	01.12 – 30.12.2020	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2021	
3 Проектування програмного забезпечення	01.02 – 28.02.2021	
4 Програмна реалізація	01.03 – 10.04.2021	
5 Тестування програмного забезпечення	11.04 – 30.04.2021	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2021	
7 Попередній захист ДП	Травень 2021 (згідно графіка)	
8 Перевірка ДП на плагіат, нормконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки)	26.05 – 30.05.2021	
9 Підготовка до захисту та захист ДП	з 01.06.2021	

Студент _____ Григорчук Д.С.
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ Гурман І.В.
Підпис Ініціали, прізвище

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А	ДПІПЗ.170115.01.02.ПЗ	Пояснювальна записка	72		
2	А		Завдання на дипломний	1		
			проект			
3	А		Анотація	1		
			Графічні документи			
4	А		Презентаційні матеріали			

Змн.	Арк.	№ докум.	Підпис	Дата	ДПІПЗ.170102.01.02.ВД		
Виконав		Григорчук Д.С.		5.06	Інтелектуальна рекомендаційна система вибору автомобілів **Відомість документів**	Літ.	Арк. 1 / Акрушів 1
Керівник		Форкун Ю.В.		5.06			
Рецензент						ХНУ, ІПЗ-17-1	
Н. Контр.		Радельчук Г.І.		5.06			
Зав. каф.		Бедратюк Л.П.		5.06			

АНОТАЦІЯ

Тема дипломного проекту: «Інтелектуальна рекомендаційна система вибору автомобілів».

Автор проекту: Григорчук Данило Сергійович.

Керівник проекту: Форкун Юрій Вікторович.

Пояснювальна записка: 72 с., 34 рис., 4 табл., 4 дод., 19 джерел.

Графічна частина: 10 презентаційних слайдів.

РЕКОМЕНДАЦІЙНА СИСТЕМА, API, MVC, C#, DOTNET, ENTITY FRAMEWORK, DATA BASE, SQL-СЕРВЕР.

Метою проекту є аналіз, реалізація методів та розробка програмного забезпечення з формування рекомендацій щодо вибору автомобілів користувачами веб-сайту з продажу автомобілів.

У дипломному проекті здійснено аналіз предметної області та видів, змісту і методів інформаційного забезпечення, визначені функціональні вимоги до програмної системи, розроблено архітектуру додатку, розроблена структура бази даних, та здійснено конструювання додатку з допомогою технології Model-View-Controler.

Для реалізації програмного забезпечення використано мову програмування C#, середовище розробки Visual Studio 2019, сервер баз даних MS SQL Server та фреймворк Entity FrameWork.

Результатом проектування стала програмна реалізація рекомендаційної довідково-інформаційної системи вибору автомобілів у вигляді веб-сайту.

16.06.2021

ПЕРЕЛІК СКОРОЧЕНЬ

ADO.NET	– набір бібліотек, що постачається з Microsoft .NET Framework;
Collaborative filtering	– колаборативна фільтрація;
Content-based filtering	– фільтрація на основі контенту;
DB	– база даних;
EF	– Entity Framework – набір технологій в ADO.NET;
Memory based	– фільтрація на основі сусідства;
Model based	– модель фільтрація заснована на моделі;
MVC	– архітектурний шаблон модель, подання , контролер;
MS SQL	– сервер баз даних Microsoft SQL Server;
PC	– рекомендаційна система.

ВСТУП

В теперішній час найбільш поширеним і найбільш доступним джерелом інформації виступає всесвітня мережа Інтернет, яка забезпечує можливість зручного пошуку та обробки інформації про вподобання користувачів онлайн-спільнот та різних веб-ресурсів.

Зокрема в Інтернет є велика кількість різних пошукових систем, каталогів, баз даних, файл-серверів тощо. Кількість контенту невпинно зростає, різні сайти та сервіси надають інформацію, яка в процесі існування породжує іншу інформацію. Головним, проте не єдиним завдання різноманітних рекомендаційних сервісів – це допомога споживачу такої інформації орієнтуватися у їх результатах.

Одним з таких сервісів є рекомендаційні системи - підклас систем фільтрування інформації, яка будує рейтинг певних об'єктів (речі, книги, статі, музика, фільми, новини тощо), яким користувачі можуть надавати певну перевагу. Для вирішення таких задач, використовується інформація з профілю самих користувачів, а також інформація про самі об’єкти. Рекомендаційні системи широко застосовуються в таких сферах, зокрема, як електронна комерція, соціальні мережі, веб-додатки тощо, де акцент робиться на різного роду інформації про дані користувача та інформацію про самі об’єкти.

Актуальність теми дипломного проекту полягає у розробці методів щодо допомоги користувачу веб-сервісів при виборі автомобілів та наданні йому рекомендацій згідно його вподобань, а також розробці спеціалізованого програмного забезпечення для вирішення даної задачі.

Метою даної роботи є аналіз та реалізація методів з формування рекомендацій та вибору автомобілів користувачами веб-сайту з продажу автомобілів, та розробка відповідного програмного забезпечення.

Об’єктом дослідження роботи виступають веб-сервіси, що спеціалізуються на продажі чи наданні доступу кінцевому користувачеві інформації про автомобілі, та мають певну частоту переглядів.

Предметом дослідження є алгоритми та методи надання рекомендацій, які використовуються для прогнозування уподобань користувачів, які проявляються у вигляді надання їм пропозицій згідно їхніх уподобань та побажань.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз та визначення видів, змісту та методів інформаційного забезпечення предметної області

Рекомендаційні системи – це підклас систем пошуку та пдбору інформації, яка будує рейтинг певних об'єктів (речі, продукти, товари, книги, автомобілі, статі, музика, фільми тощо), яким користувачі можуть надати певну перевагу на основі власних вподобань та рекомендацій і порад інших користувачів. За основу в таких системах використовується інформація з профілю користувача, історій його покупок, переглядів тощо. Рекомендаційні системи широко застосовуються в таких сферах, зокрема, як електронна комерція, соціальні мережі, веб-додатки тощо, де акцент робиться на різного роду інформації про дані користувача.

Рекомендації можуть бути однакові для усіх типів користувачів та поділені по групах за певною ознакою (віком, статтю, місцем проживання тощо), а також персоналізованими. Оцінка користувачем продукту може бути прямою (рейтинг, відгук, вподобання) чи непрямою (куплено, переглянуто).

Найчастіше таки системи використовуються у комерційних сервісах, таких як інтернет магазин, де вони виступають в ролі рекомендавців товарів та продуктів. А також використовуються надавачами контенту для платформ соціальних мереж, зокрема Instagram, Facebook, Twitter. Крім того, рекомендаційні системи використовують у інших типах веб-систем, де вони витупають у ролі генераторів списків відтворення для відеоконтенту та музичних сервісів, наприклад, YouTube та Netflix.

Рекомендаційні довідкові системи виступають альтернативою пошуковим сервісам, так як вони допомагають користувачам виявляти певні характеристики, які пошуковими системами можуть і не знайтись. Так, для інтернет-магазину це досить важливо. Рекомендації в даному випадку не виступають звичайною додатковою опцією, а забезпечують зручність користувачам у користуванні

послугами. Якщо, наприклад, інтернет-магазин містить декілька тисяч товарів, то зорієнтуватись користувачу буде досить складно, а в більших системах майже неможливо. Це в свою чергу і ставить задачу вирішення проблем формування груп товарів певного виду, за певними характеристиками, серед яких виділяють вподобання користувачів.

Існує велика кількість прикладів застосування рекомендаційних систем різних видів. Як наголошувалось вище, найчастіше вони знаходять своє застосування в он-лайн комерції, зокрема в інтернет-магазинах (рисунок 1.1).

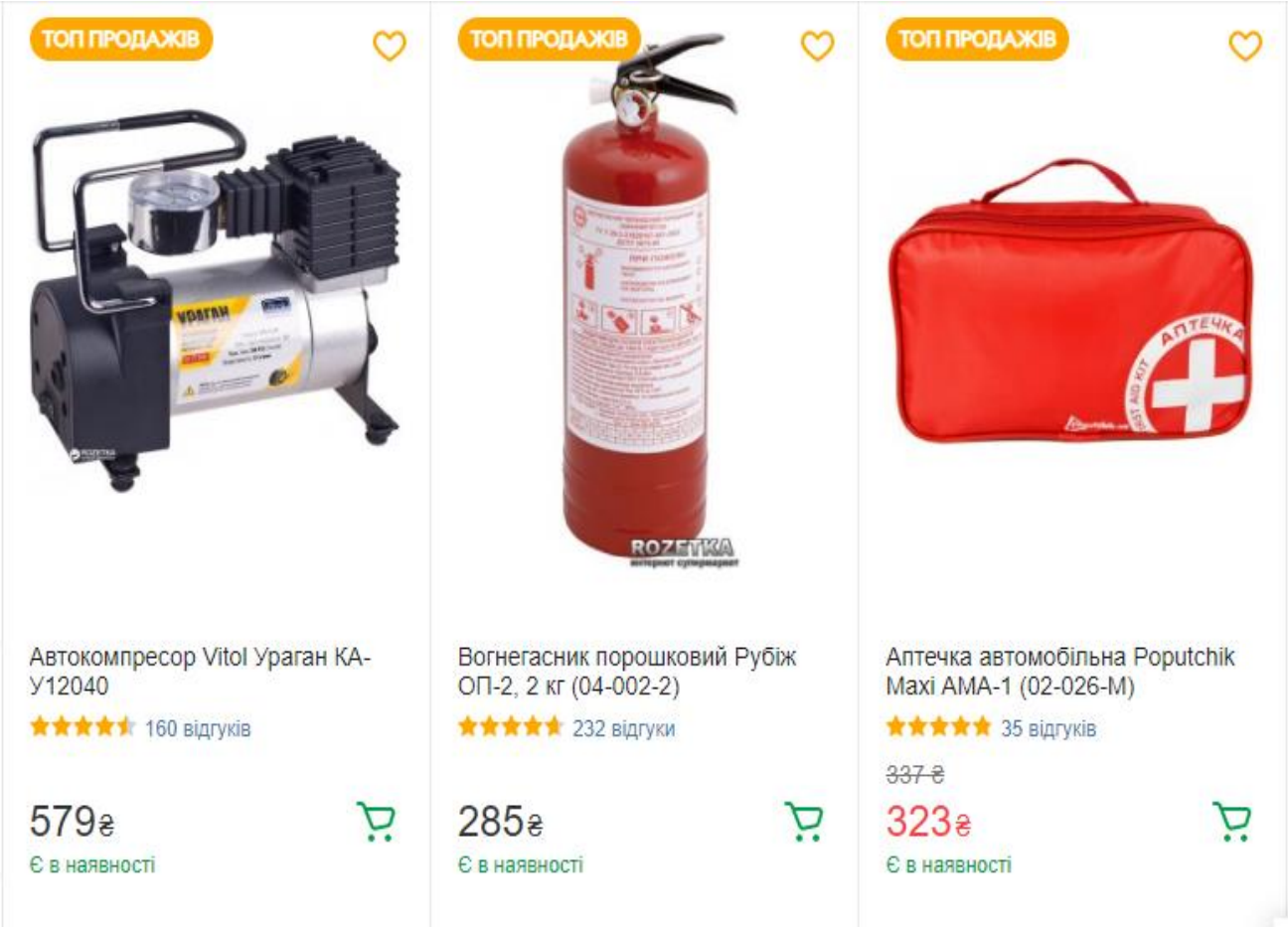


Рисунок 1.1 – Рекомендації товарів в інтернет-магазині.

Проте, не тільки онлайн-магазини використовують даний підхід, але також вони знаходять своє використання в соціальних мережах, наприклад Instagram, FaceBook, Twiter тощо (рисунок 1.2).

Також, подібне використання та застосування легко можна бачити на різних інтернет-сайтах, які присвячені музичному контенту, відофільмам, відеоіграм, книгам на новинних ресурсах тощо. Даний підхід досить часто застосовується та є відомим і популярним.

Загалом, на теперішній час, в виділених системах виділяють дві основні та найбільш використовувані стратегії створення рекомендаційних систем]:

– Content Based Filtering – контентна фільтрація (фільтрація на основі вмісту);

– Collaborative Filtration – колаборативна фільтрація.

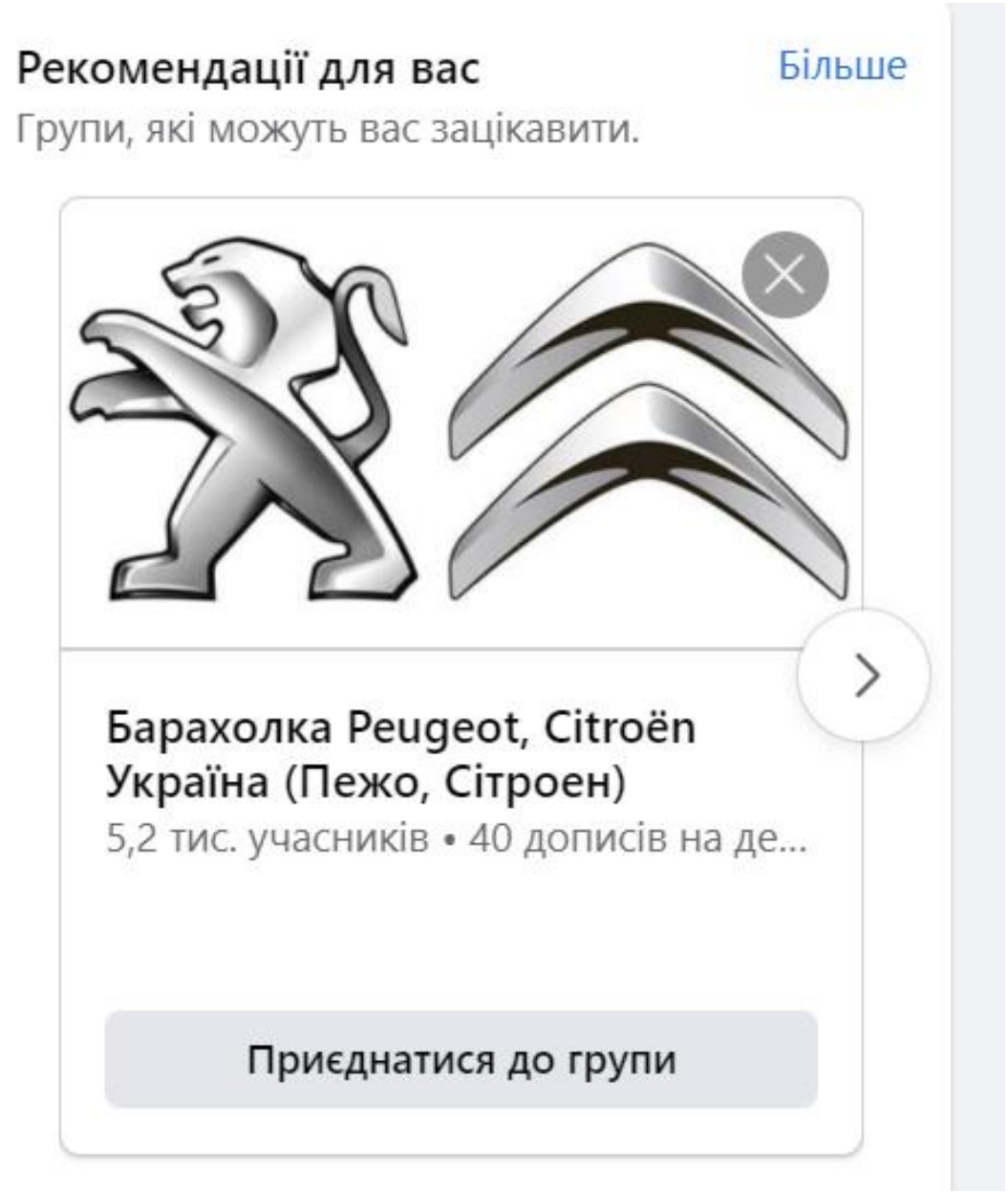


Рисунок 1.2 – Рекомендації приєднання до груп в соціальній мережі FaceBook.

На практиці доволі часто використовують гібридний підхід. Даний підхід використовує у змішаному вигляді обидві основні стратегії.

Основа алгоритму при колаборативній фільтрації (Collaborative Filteration) полягає в припущені того, що користувачі, які однаково оцінювали певні об'єкти в минулому, схильні також надавати схожі оцінки схожих об'єктів і в подальшому. Процес колаборативної фільтрації має три основних кроки, першим з них є збір користувацької інформації. Наступним кроком будується матриця для розрахунку певних асоціацій, на основі чого, надається певна рекомендація. За допомогою колаборативної фільтрації, наприклад, відео-сервіс, маючи деякий список переваг користувача (як симпатій так і антипатій) здатен прогнозувати, яке відео буде до вподоби користувачу, Прогнози надаються індивідуально кожному користувачеві, а інформація, яка для цього була відібрана, надходить від багатьох інших учасників.

В свою чергу, колаборативна фільтрація має два таких основних підходи:

– алгоритм колаборативної фільтрації, заснований на сусідському порівнянні (memory based);

– алгоритм колаборативної фільтрації заснований на моделі (model based).

Алгоритм колаборативної фільтрації, заснований на сусідському порівнянні (memory based), являє собою зважування вподобань по користувачах (user-based) та на основі елементів (item-based), що утворюють певну категорію.

Алгоритм колаборативної фільтрації, заснований на моделі (model based), надає рекомендації, вимірюючи параметри статистичних моделей для оцінок користувачів, побудованих за допомогою різних. Моделі розробляються з використанням інтелектуального аналізу даних, алгоритмів машинного навчання, для знаходження закономірності на наборі навчальних даних. Цей підхід є більш комплексним і дає більш точні прогнози, так як допомагає розкрити певні фактори, що пояснюють отримані прогнозовані оцінки.

Досить часто при розробці рекомендаційних систем використовують алгоритм колаборативної фільтрації заснований на сусідстві. Виділяють два методи підбору сусіді»: «User-User» та «Product-Product».

Для створення нових рекомендацій активному користувачеві метод «User-User» намагається знайти користувачів з найбільш подібним профілями дій. Таких користувачів вважають найближчими сусідами, для схожих товарів, які є найбільш цікавими для такого типу користувачів. Зокрема ідея, на якій ґрунтується такий метод – це пошук користувачів, чиї вподобання схожі на вподобання прогнозованого цільового користувача. Тобто, якщо раніше «користувач *m*» і «користувач *n*» проставили схожі оцінки для декількох продуктів, то вважається, що вподобання у них подібні, і за оцінками тих чи інших продуктів, проставленим користувачем *m*, можна спрогнозувати оцінки користувача *n*.

Для прикладу, якщо одному користувачеві сподобався автомобіль типу седан за ціною 5000 умовних одиниць, і йому також сподобався автомобіль типу універсал з такою ж ціною, а іншому користувачеві сподобався автомобіль типу седан за ціною 5000 умовних одиниць, то ми іншому користувачеві також повинні надати рекомендацію переглянути автомобіль типу універсал з приблизно такою ж ціною.

Уподобання користувачів заносять до матриці, яка має назву «користувачі-елементи», на основі якої відшукують найближчих сусідів (рисунок 1.3).

Рисунок 1.3 – Матриця уподобань користувачів

Метод «Product-Product» полягає у тому, що користувачеві рекомендуються товари з характеристиками схожі на ті, які йому подобались раніш. Тобто схожість розраховується між характеристиками товарів, а не характеристикою користувачів (рисунок 1.4). При цьому дві характеристики вважаються схожими, якщо більшість користувачів обирали схожі характеристики.

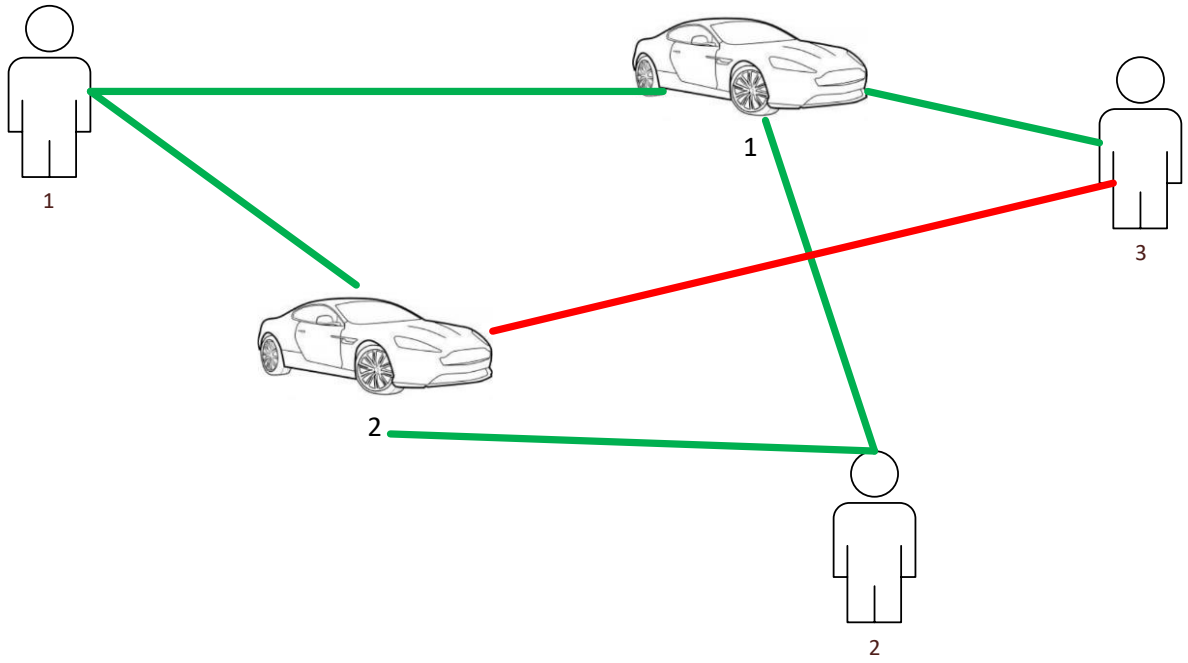


Рисунок 1.4 – Метод «Product-Product»

Даний підхід також використовує вище описану технологію. В ньому, розглядаються стовпці, які представляють собою вектор взаємодії певної характеристики з кожним користувачем. Для надання рекомендації окремому користувачу обираються характеристики, які йому найбільше сподобались. Вони являють собою вектори взаємодій між користувачами. Потім обчислюється міра схожості між цими характеристиками з іншими та обираються найбільш схожі, які потім становлять рекомендацію.

Метод «User-User» оснований на пошуку схожих користувачів на основі взаємодій з деякими об’єктами. В цьому випадку, якщо кожен користувач взаємодіє з невеликою кількістю об’єктів, то метод стає досить відчутливим до будь-яких спостережених взаємодій. Навпаки, якщо остаточна рекомендація

базується лише на спостережених взаємодій користувачів зі схожими вподобаннями, то він дозволяє отримати певний персональний результат.

Метод «Product-Product», на відміну від методу «User-User», оснований на пошуку схожих продуктів на базі взаємодій користувачів з ними. Загалом, якщо велика кількість користувачів взаємодіяли з продуктом то пошук сусідів є значно менш чутливим до одиничних взаємодій. Аналогічно, дії будь-яких користувачів розглядаються у рекомендації, перетворюючи даний метод до неперсоналізованого. Тобто, такий підхід є менш персональним ніж підхід «User-User», але він більш надійніший.

В результаті аналізу підходів колаборативної фільтрації можемо встановити її ключові недоліки:

– розрідженість даних;
– масштабованість;
– проблема «Холодного старту».

Розрідженість даних полягає у тому, що більшість комерційних рекомендаційних систем заснована на великій кількості даних про продукти, без врахування того, що більшість користувачів не ставить оцінки продуктам чи товарам. В результаті цього матриця оцінок «product-user» стає досить великою та розрідженою, що відповідно, приводить до проблем при обрахуванні рекомендацій та вподобань;

З збільшенням кількості користувачів системи з’являється проблема масштабованості.

Проблема «Холодного старту» виникає при застосуванні алгоритмів колаборативної фільтрації, які найкраще працюють для продуктів та користувачів з багатьма оцінками. Якщо продукт або користувач новий - система не може знайти для них подібних.

Для оцінки ефективності рекомендаційних систем і/або готових алгоритмів можна застосовувати різні набори метрик:

– метрики точності прогнозу (MAE, RMSE, MSE);

– метрики прийняття рішень (ROC, Precision/recall);

– метрики оцінки ранжування результатів (Spearman Rank Coefficient, Fraction of concordant pairs);

– бізнес-метрики (збільшення продаж, конверсії тощо).

Серед інструментів для розробки рекомендаційних систем можна виділити:

– Apache Mahout;

– LensKit;

– MyMediaLight;

– GraphLab;

– PredictionlO;

– EasyRec;

– Crab;

– Recommended ab.

В зв’язку з описаним вище та необхідними параметрами розробки рекомендаційних довідкових систем нами було обрано для використання гібридне рішення, яке включає в себе такі рішення, як колаборативна система в поєднанні із загальною статистикою та фільтрацією згідно контекту.

Дане поєднання методів добре підходить системі рекомендації автомобілів, оскільки вони дозволить користувачу знаходити дійсно автомобілі за своїми вподобаннями та здійснювати вибір вже серед запропонованих.

1.2 Постановка задачі

Задачу створення рекомендаційної системи будемо виконувати на прикладі вибору автомобілів. В даний час існує велика кількість веб-платформ, які здійснюють посередницькі послуги між продавцем та покупцем на ринку продажу, купівлі та оренди автомобілів.

Такі системи, як правило, вони надають такий набір послуг:

– купівля продаж автомобілів;
– продаж автомобілів
– оренда автомобілів;
– пошук продавців автомобілів;
– пошук покупців автомобілів.

При розробці таких систем прийнято використовувати веб-технології, та реалізації її у вигляді веб-додатку, за клієнт-серверною технологією. Тому, для реалізації нашої системи ми будемо створювати веб-додаток використовуючи сучасні технології проектування, зокрема спроектуємо наш додаток використовуючи концепцію MVC (Model View Controler: модель-вид-контролер).

Концепція MVC дуже часто використовується в світі веб-програмування в останні роки. Кожен, хто хоч якось пов'язаний з розробкою веб-додатків, так чи інакше стикався з даними акронімом.

MVC - це не шаблон проекту, це конструкційний шаблон, який описує спосіб побудови структури нашого застосування, сфери відповідальності та взаємодія кожної з частин в даній структурі.

Вперше вона була описана в 1979 році, звичайно ж, для іншого оточення. Тоді не існувало концепції веб-додатків. Tim Berners Lee (Тім Бернерс Лі) посіяв насіння World Wide Web (WWW) на початку дев'яностих і назавжди змінив світ. Шаблон, який ми використаємо в проекті, є адаптацією оригінального шаблону до веб розробки.

Шалена популярність даної структури в веб додатках склалася завдяки її включенню в два середовища розробки, які стали дуже популярними: Struts і Ruby on Rails. Ці дві середовища розробки намітили шляхи розвитку для сотень робочих середовищ, створених пізніше (рисунок 1.3).

У технології MVC, і у нашій проектованій системі зокрема, додаток розділяється на три основних компоненти, кожен з яких відповідає за різні завдання в проекті (рисунок 1.4).

Перший компонент Модель (Model) - це дані і правила, які використовуються для роботи з даними, які представляють концепцію управління

додатком. У будь-якому додатку вся структура моделюється як дані, які обробляються певним чином. Що таке користувач для додатка - повідомлення або книга? Тільки дані, які повинні бути оброблені відповідно до правил (дата не може вказувати в майбутнє, e-mail повинен бути в певному форматі, ім'я не може бути довшим X символів, і так далі).

Рисунок 1.3 – Приклади Фреймворків

Рисунок 1.4 – Модель MVC

Модель дає контролеру уявлення даних, які запросив користувач (повідомлення, сторінку книги, фотоальбом, тощо). Модель даних буде однаковою, незалежно від того, вона буде представлена користувачеві. Тому ми вибираємо будь-який доступний вид для відображення даних.

Модель містить найбільш важливу частину логіки нашого застосування, логіки, яка вирішує завдання, з якою ми маємо справу (форум, магазин, банк, підприємство тощо).

Другий компонент – View (Представлення) забезпечує різні способи представлення даних, які отримані з моделі. Він може бути шаблоном, який заповнюється даними. Може бути кілька різних представлень, і контролер вибирає, який підходить якнайкраще для поточної ситуації.

Третій компонент, це – Контролер(Controller), який керує запитами користувача (одержувані у вигляді запитів HTTP GET або POST, коли користувач натискає на елементи інтерфейсу для виконання різних дій). Його основна функція - викликати і координувати дію необхідних ресурсів і об'єктів, потрібних для виконання дій, що задаються користувачем. Зазвичай контролер викликає відповідну модель для задачі і вибирає відповідний вид. Контролер містить в основному організаційну логіку для самого додатка.

Веб-додаток зазвичай складається з набору контролерів, моделей і видів. Контролер може бути влаштований як основний, який отримує всі запити і викликає інші контролери для виконання дій в залежності від ситуації.

Реалізацію алгоритму системи вибору автомобілів відповідно буде реалізовано у компоненті Model.

Зупинимось більш детально на реалізації саме цієї частини нашого додатку, оскільки вона є основним моментом реалізації, та опишемо їх суть та алгоритм за яким ми будемо здійснювати систему вибору.

Аналіз основних стратегій по створенню рекомендаційних систем показав, що вони не можуть бути застосовані напряму для вирішення поставленої нам задачі, через необхідність використання та збереження історії рейтингів, що

сильно буде вливає на роботу системи загалом. В свою чергу, рекомендаційні системи засновані на знаннях, хоч і призначені для роботи саме з такою предметною областю, не будуть універсальними у зв'язку з необхідністю явного збору знань для нашої задачі та явній побудові різних правил асоціацій залежності та з залученням експертів в цій предметній області.

Отже, можна зробити висновок, що існує необхідність у розробці нової стратегії, методів та способів прогнозування вподобань користувачів, які б дали змогу опрацьовувати нашу предметну область та, при цьому, не потребували явного збору знань для них окремо. Тобто така стратегія має відповідати наступним таким вимогам:

– бути незалежною від історії рейтингів;

– логіки побудови прогнозів також має бути незалежною від обраної предметної області.

Певні пункти розглянутих методів можна застосувати, використати та об'єднати між собою для створення системи рекомендацій, яка б мала змогу здатна працювати з обраною предметною областю та мінімізувати проблемну задачу в способі отримання потрібних знань. Загалом, найкраще частково застосувати метод фільтрації заснований на контенті, при якому аналізується лише характеристики автомобілів, без побудови профілю користувача.

Загальний алгоритм полягає у наступному: користувачу будуть пропонуватися автомобілі, які максимально схожі на обраний ним автомобіль, при цьому «вподобаним» вважається автомобіль, який переглядається користувачем у конкретний момент часу. Тобто, за основу такої рекомендації виступає так званий «метод ідеальної точки». Ідеальною точкою при цьому і є автомобіль, якийсь обрав користувач. Даний алгоритм дозволить знаходити автомобілі, які не точно відповідають пошуковому запиту (фільтру) користувача, по те є близько схожими за певними характеристиками.

Для знаходження «однаковості» вподобаних автомобілів, будемо використовувати різноманітні метрики подібності, для порівняння значень

атрибутів характеристик авто. При цьому для кожного атрибута будемо враховувати його відносну важливість та вигляд функції, яка описує подібність двох значень даної характеристики.

Крім цього необхідно застосувати ідею діалогу інтелектуальних рекомендаційних систем, яка дасть змогу організувати зворотній зв'язок з користувачем. Тут вибір користувача буде формуватися в межах сеансу користувача, а рекомендації будуть формуватися шляхом аналізу характуристик елементів, які відносяться до його вподобання.

При такому алгоритмі створення рекомендацій буде являти двоступеневий процес. Спочатку користувачу пропонуються автомобілі, які найбільше мають схожих характеристик, до того який він наразі переглядає. Другим етапом процесу буде етап, коли користувач вже переглянув певну кількість автомобілів, і рекомендації будуть базуватись на групі автомобілів з певної кількості переглянутих ним пропозицій .

Група останніх переглянутих автомобілів формується так званим методом «ковзаючого вікна» - алгоритмі трансформації, що дозволяє сформувати з членів часового ряду набір даних, який може служити навчальною множиною для побудови моделі прогнозування. Під «вікном», у даному випадку, будемо розуміти деякий часовий інтервал, який містить набір характеристик, які будуть використані при формування прикладу для навчання. У процесі роботи алгоритму вікно послідовно зміщується на одиницю спостереження, при цьому «забуваючи» попередні спостереження. Такий підхід чудово реалізує ідею «діалогу», враховуючі поточні уподобання користувача та враховуючи їх зміну шляхом «забування» старих елементів.

Також, потрібно врахувати ідею обмежень для зменшення кількості характеристик, які будуть пропонуватись в якості рекомендацій, згідно до головних обмежень у пошукового запиту. Наприклад, не має змісту пропонувати користувачеві спортивні автомобілі, якщо при пошуку він вибрав сімейний автомобіль, навіть при повній відповідності іншим характеристикам.

Даний алгоритм має наступні ключові переваги над існуючими рішеннями, які полягають в наступному :

– відпадає необхідність створення профілю користувача, оскільки система буде базуватись лише на характеристиках автомобілів;

– вподобання користувача будуть проявлятись у реальному часі;

– універсальність, оскільки відпадає необхідність створення асоціативних правил пошуку для кожного автомобіля.

Однак, такий алгоритм має певні недоліки: оскільки система рекомендацій заснована на характеристиках автомобілів, то для вірного розрізнення характеристик, необхідно створити дуже детальний та змістовний опис кожного автомобіля, а також обов'язковою умовою є необхідність максимального заповнення характеристик автомобіля. Також такий підхід не буде універсальним у повній мірі, адже адміністратору системи необхідно буде задати критерії рекомендацій, відносну важливість характеристик автомобілів та спосіб їх порівняння. Також суттєвим обмеженням є те, що даний алгоритм передбачає, що рекомендації мають будуватись на множині однакових за характеристиками автомобілів, що характеризуються певним набором однакових параметрів. Тобто рекомендація буде працювати лише в межах обраних характеристик, рекомендуватися буде лише той самий тип автомобілів.

1.3 Математичний опис задачі

Описаний вище алгоритм встановлює, що певний автомобіль характеризується деяким вектором значень певних наперед визначених характеристик. Для кількісної оцінки схожості характеристик з обраним будемо використовувати метод зваженої суми критеріїв.

Кожен елемент, який виступає в ролі характеристики автомобіля є по суті вектором критеріїв:

$$E = \{e_1, .., e_n\}, n = 1, \propto \quad (2.1)$$

де e_i – кількісна оцінка схожості i-тої характеристики автомобіля з i-тою характеристикою обраного автомобіля.

Відносна значимість кожної з характеристик автомобіля описується вектором вагових коефіцієнтів:

$$V = \{v_1, .., v_n\}, n = 1, \propto, \quad (2.2)$$

де v_i – відносна важливість i-тої характеристики автомобіля;

$$\sum vi = 1, \quad (2.3)$$

Результуючу оцінку подібності автомобілів будемо розраховувати за наступною формулою:

$$O = \sum(ei \times vi)/n, i = 1..n \quad (2.4)$$

Для оцінки на основі групи характеристик оцінки O (2.4) кожної характеристики, що розраховуються у групі переглянутих, підсумовуються для отримання остаточного результату:

$$Op = \sum Oi, i = 1..n \quad (2.5)$$

де n – кількість врахованих характеристик.

Тут такий розрахунок схожості можна розглядати, як рангування переглянутих автомобілів за непереглянуті, та автомобіль, який має найбільші значення оцінок (2.4) відносно характеристик у певній групі, і, відповідно отримує найвищу оцінку.

Оцінка схожості автомобілів відбувається окремо за кожною характеристикою з подальшим формуванням вектора E. Для характеристик, що представлені у інтервальній оцінці схожості, характеристики елемента, що аналогічні характеристиці базового, або обраного автомобіля розраховується за

допомогою функцій розподілу. Функції розподілу здійснюють оцінку за деякою величиною, з допомогою функції затухання, в залежності від збільшення відстані між значенням цієї величини та певним заданим еталоном. Ця оцінка схожа на запит, щодо входження значення в заданий діапазон, з подальшим призначенням кількісної оцінки даному значенню, залежно від близкості до еталону.

В загальному прийнято використовувати три основні форми залежності розмивання або розпаду:

– розмивання за функцією Гауса;

– експоненціальний розпад;

– лінійний розпад.

В рекомендаційній системі, що проектується, будемо використовати Гаусівську залежність, оскільки лінійна залежність визначає тільки пропорціональне зменшення оцінки в процесі віддалення від початкової точки, а експоненціальний розподіл має сильну залежність поряд в центрі, та меншу на кінці, що означає різке зменшення значень оцінок в межах обраного діапазону, та більше за його межами. В даному випадку необхідна зворотня залежність, яку найкраще визначає розмивання за функцією Гауса.

Відповідно, затухання для деякої характеристики певної характеристики можна визначити таким чином:

$$Ei = exp\,(-(|zi - z| - v)\,2/\,(2 \times (-\,w2\,/(2 \times ln(s))\,)), \qquad (2.6)$$

де *Ei* – кількісна оцінка схожості *i*-ої характеристики елемента з *i*-ою характеристикою базового елемента, приймає значення в діапазоні від 0 до1:

z_i – значення і-тої характеристики автомобіля;

z – значення і-тої характеристики обраного автомобіля;

v – відстань, на якій значення характеристики прирівнюється обраній;

w – відстань від центру зі зміщенням, яка визначає діапазон прийнятної нами допустимої подібності;

s – значення подібності, з допомогою якого оцінюється значення характеристики на відстані *w* від *z*.

Для характеристик, наведених номінальною шкалою встановлюється лише подібність, або відмінність об'єктів. Таким чином, схожість значень таких характеристик буде визначатись таким правилом:

Ei=1, якщо значення відповідних характеристик однакові;

Ei= 0, якщо значення відповідних характеристик різняться.

Це означає, що при повному співпадінні значень буде оцінюватись максимальною оцінкою, або при відсутності співпадіння – нулевою оцінкою. Якщо значення деякої характеристики відсутнє, то оцінка схожості даної характеристики автомобіля аналогічній характеристиці обраного автомобіля буде дорівнювати нулю.

Отже, в даному розділі було проведено досліджено предметну область та здійснено її аналіз. Розкрито поняття рекомендаційних систем. Здійснено постановку завдання та описано математичну модель. Також у ході аналізу предметної області було розроблено технічне завдання (додаток А).

2 ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Проектування архітектури та структури системи

Програмне забезпечення таких довідково-інформаційна систем зазвичай являє собою веб-додаток, який складаються з клієнтської частини, що представлений веб-додатком, який обробляється сервером та відображається у браузері, який відповідає за отримання даних з сервера для подальшої представлення в зручному вигляді для користувача та відправки даних на сервер, який приймає запити від браузера, зберігає та оброблює дані.

Сервіси з пошуку автомобілів надають своїм користувачам наступні найбільш загальні можливості:

– реєстрація користувачів у системі;

– перегляд та пошук оголошень за відповідними критеріями;

– фільтрація автомобілів за різними критеріями;

– додавання/видалення оголошень за відповідними рубриками;

– можливість напису відгуків;

– надання користувачу рекомендацій згідно його вподобань.

Приклад такої системи зображено на рисунку 2.1

Розроблювана довідково-інформаційна система з пошуку автомобілів буде виконувати усі описані стандартні можливості та буде спеціалізуватися на подані інформації про автомобілі для продажу та надані рекомендацій вибору автомобілів згідно до вподобань користувача системи.

Рішення даного комплексу задач призначене для створення такої довідково-інформаційної системи, яка повинна задовольняти таким вимогам:

– відсутність необхідності використання історії рейтингів;

– відсутність проблеми «холодного старту»;

– відсутність необхідності створення асоціативної моделі експертоми у даній предметної області;

– мінімізація вузького місця набуття знань стосовно предметної області;

– відповідність обмеженням, що встановлює користувач при пошуку;

– гнучкість прогнозувань, яка залежить від зміни вектору уподобань користувача під час перегляду характеристик автомобілів.

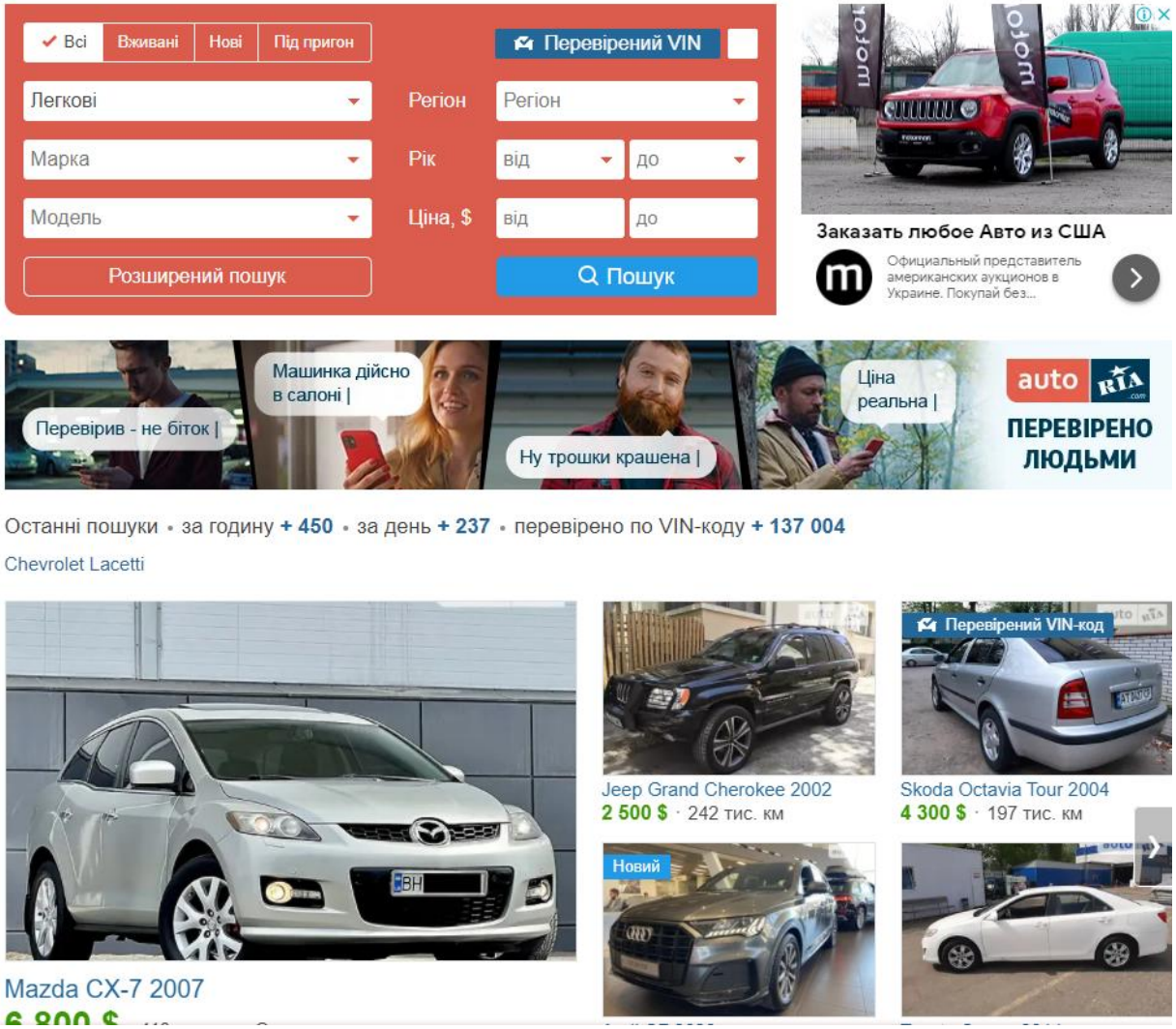


Рисунок 2.1 – Сторінка пошуку у сервісі з продажу автомобілів «Auto Ria»

Проектована інтелектуальна довідково-інформаційна система система, як описувалось раніше, буде однією з спроектованих задач нашого програмного забезпечення. Вона повністю інтегрована в додаток та спеціалізується на пропонуванні користувачу автомобілів, які мають схожі характеристики, згідно вподобань користувача. Програмне забезпечення призначено для створення,

редагування та керування оголошеннями про купівлю, продаж або оренду автомобілів.

Виходячи з мети та цілей нашої довідково-інформаційна системи, до неї висуваються певні системні вимоги. Система має складатися з клієнтської частини, яка має відповідати за отримання даних з сервера для відображення в зручному вигляді та відправки даних на сервер, який має приймати запити від браузера клієнта, обробляти їх та зберігати інформацію про користувача та прогнозувати уподобання користувачів за допомогою алгоритму надання рекомендацій. Оскільки довідково-інформаційна системи отримують велику кількість запитів, то вона також має відмовостійкою. Нам необхідно розробити довідково-інформаційна системе так, щоб в ході її експлуатації і можна було б масштабувати застосовуючи найменшу кількість дій. Проектований додаток не має бути прив’язаний до певної платформи, та повинен бути розроблений програмний інтерфейс системи реалізований у вигляді веб-додатку.

Для визначення функціональних вимог до системи була розроблена її функціональна модель з використанням стандарту IDEF0. У рамках цієї методології був створений набір діаграм, що описують її, визначають певні процедури та правила (рисунок 2.2 - 2.4)

У наведеній діаграмі декомпозиції функції рекомендаційної системи пошуку автомобілів на початковому етапі необхідно отримати набір характеристик, які описують заданий автомобіль та значення абсолютної важливості, характеристику базового автомобіля, та характеристики усіх автомобілів, для яких будемо знаходити найкращі наближення вподобань.

Таким чином, проведене функціональне моделювання та аналіз отриманих діаграм встановлює такі функціональні вимоги до розроблюваної системи:

– реєстрація нових користувачів;

– перевірка введених даних;

– створення нових автомобілів за певними характеристиками;

– редагування характеристик автомобілів;

– написання відгуків про автомобілі оголошенні;

– редагування профілю користувача;

– здійснення рейтинг користувача;

– відображення характеристик автомобіля;

– відправка повідомлення на електронну пошту користувачеві;

– сортування та відбір автомобілів за різними критеріями;

– надання рекомендацій щодо вподобань користувача;

– відображення повідомлень при некоректних даних.

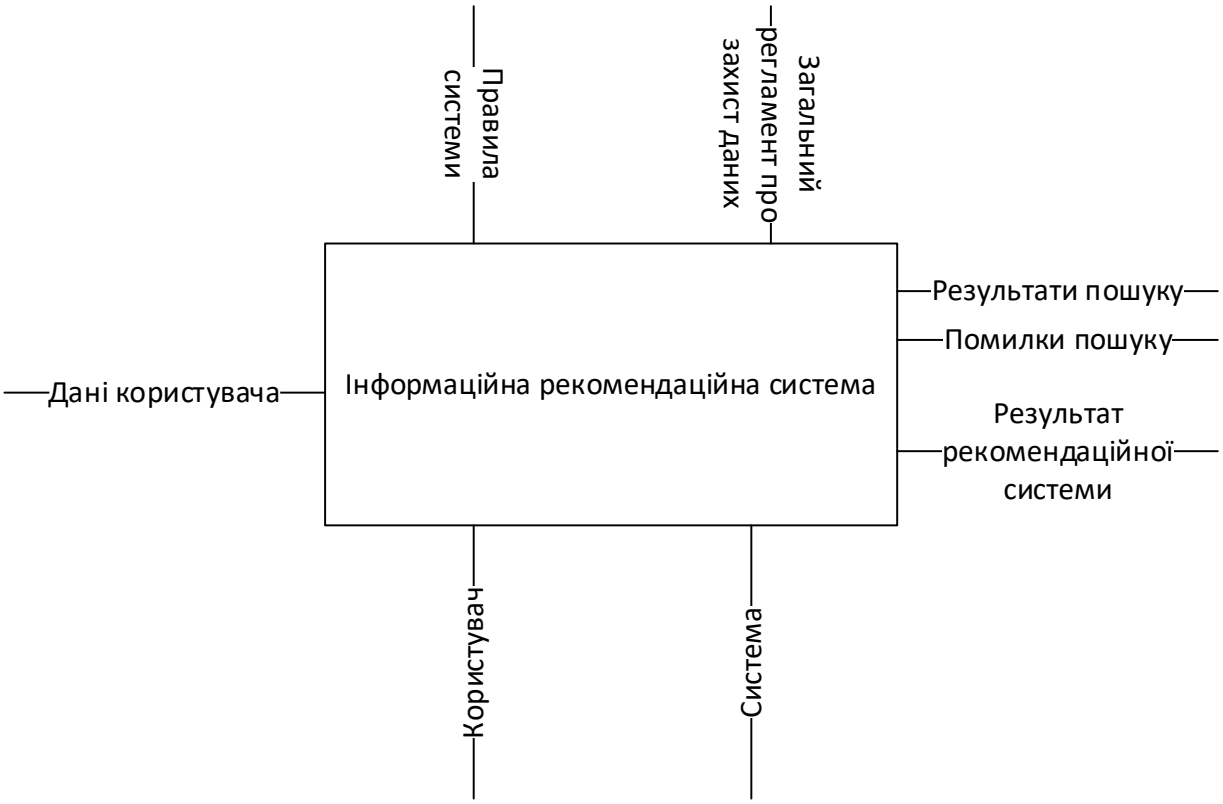


Рисунок 2.2 – Концептуальна діаграма інформаційної системи

Відповідно до встановлених системних і функціональних вимог для розроблюваної системи була обрана клієнт-серверна архітектура, при якій взаємодія модулів відбувається за протоколом HTTP шляхом передачі запитів від клієнта до сервера. У якості архітектурного шаблону сервісу було обрано шаблон проектування Model-View-Controler. Даний шаблон проектування передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої

логіки на три окремих компоненти: модель, представлення та контролер – які є незалежними і програмуються окремо один від одного.

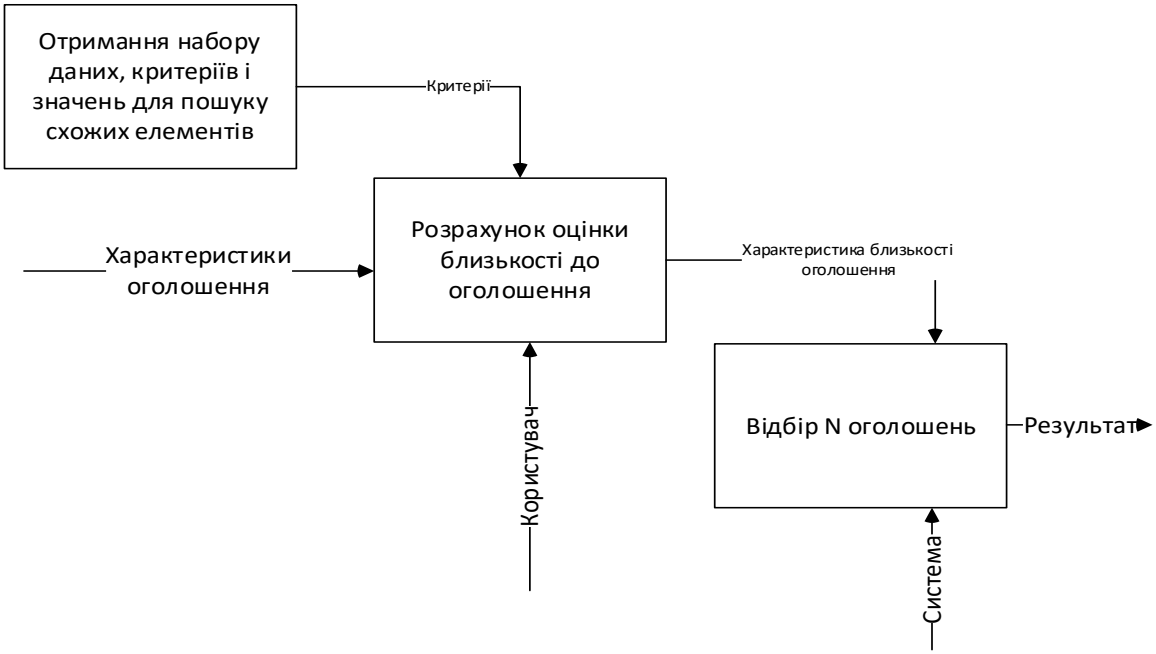


Рисунок 2.3 – Діаграма функції рекомендаційної системи пошуку автомобілів

Під моделлю, як зазначалось вище, будем розуміти частину, яка містить в собі логіку програмного забезпечення обробки даних . Модель в нашому випадку повністю незалежна від інших частин програмного забезпечення. Рівень компонента Model відокремлений від представлення і не вирішує ніяких задач, щодо передачі та відображення даних В обов'язки рівня View входить відображення даних отриманих від рівня View. Рівень View загалом отримує доступ до даних тільки на читання. У представлені View реалізується відображення даних, які він отримує від моделі від моделі певним способом. Прикладами представлення можуть бути: HTML-сторінка, форма Windows Prezentation Fundation, WindowsForm тощо. Контролер відповідно інтерпретує дії користувача, сповіщаючи модель про необхідність змін. Контролер здійснює, обробку запитів користувачів та виклик і передача відповідних ресурсів між додатком та сервером та між моделями в додатку. Крім того, він відправляє дані від користувача до програмної системи та навпаки.

2.2 Аналіз та вибір технологій і методів реалізації бази даних. Вибір СКБД

На сьогодні існує дуже багато різноманітних систем керувань базами даних. Їх класифікують за різними характеристиками, зокрема, такими як: технології фізичного зберігання даних, моделі даних, їх вміст та технології доступу. Серед сучаних СКБД доцільно розглянути декілька реляційних СКБД, тому що саме ця модель даних буде в подальшому використовуватись для створення, збереження та обробки даних в нашій бази даних.

PostgreSQL — об'єктно-реляційна система керування базами даних. Вона є альтернативою для таких СКБД, як Oracle Database, Microsoft SQL Server, IBM DB2, MySQL, MariaDB, SQLite тощо. На сьогоднішній день достатньо потужна система керування базою даних яку часто використовується для створення сучасних проектів, зокрема для розробки сайтів.

Microsoft SQL Server — система керування реляційними базами даних, корпорації Microsoft. Основна використовувана мова запитів — Transact-SQL. Microsoft SQL Server є одним із найкращих варіантів для розміщення СКБД на Windows-серверах.

СКБД Oracle —СКБД, яка дозволяє створювати бази даних та застосунки практично будь-якої складності. Дана СКБД може зберігати та обробляти майже нескінченну кількість даних. Вона найчастіше використовується для масштабних проектів, завдяки своїй високій продуктивності та масштабованості у порівнянні з іншими популярними СКБД.

MySQL — вільно розповсюджувана система керування базами даних. Дана система керування базами даних з відкритим кодом. MySQL є досить зручним рішенням для малих і середніх застосунків та веб-сайтів, особливо для тих, які потребують наявність на сервері бази даних. Із-за своєї доступності та легкості в обслуговуванні вона є найкращим варіантом для різного типу комерційних веб-сайтів та різних проектів, а також для рішень серверного додатку.

MySQL-сервер може працювати, як локально, так і на виділених окремих серверах та хостингах. Обробка інфомації виконується за допомогою підключення до бази даних та відправлення відповідних SQL запитів. Вона дуже зручна для невеликих проектів та має достатньо багато реалізованих засобів з'єднання під різні мови програмування, що дозволяє розробляти програмний продукт однаково зручно на всіх сучасних мовах програмування. Гнучкість MySQL забезпечується підтримкою великої кількості типів таблиць. Зокрема розробники можуть обирати серед таблиць типу MyISAM, які найкраще підтримують повнотекстовий пошук, так і серед таблиць типу InnoDB, які підтримують виконання транзакцій на рівні окремих записів. Дана СКБД надає досить багато різних інструментів, як для розробки та супроводження самої бази даних так і програмного забезпечення.

Основні позитивні сторони та можливості СКБД MySQL:

– простота у встановленні та налаштуванні;

– простота в супроводжені та підтримці;

– підтримка необмеженої кількості користувачів, які одночасно працюють із базою даних;

– масштабованість – MySQL працює з різними обсягами даних та легко масштабується та підтримується;

– підтримка великої кількості рядків у таблицях може досягати декілька десятків мільйонів;

– наявність простої та ефективної системи безпеки доступу до даних;

– швидкість обробки даних – спрощення деяких стандартів дозволяє MySQL значно збільшити продуктивність у порівнянні з іншими, які при простих операціях читання може значно уповільнити сервер.

Для реалізації нашого проекту була обрана СКБД MS SQL Server 2019. MS SQL Server є однією з найбільш популярних систем керування базами даних в світі. Дана СКБД, як і MySQL підходить для самих різних проектів: від невеликих застосунків до великих високонавантажених проектів.

SQL Server був створений компанією Microsoft. Перша версія вийшла в 1987 році. А поточною версією є версія 2019, яка вийшла в 2019 році.

SQL Server довгий час був винятково системою керування базами даних для операційних систем Mirosoft Windows, проте, починаючи з версії 2016 дана система доступна для операційних систем сімейства Linux.

SQL Server характеризується такими особливостями як:

– продуктивність – SQL Server працює дуже швидко;

– надійність і безпека – SQL Server надає шифрування даних;

– простота – з даної СКБД відносно легко працювати і вести адміністрування.

Для організації баз даних MS SQL Server використовує реляційну модель. Ця модель баз даних була розроблена ще в 1970 році Едгаром Коддом. А на сьогоднішній день вона фактично є стандартом для організації баз даних.

MS SQL Server доступний в різних варіаціях. Перш за все, це MS SQL Server Enterprise - повний випуск, націлений на використання в реальних проектах. Саме він використовується на різних хостингах і серверах баз даних. Однак він доступний тільки в платній версії (не рахуючи тріального періоду) і вартує досить пристойних грошей.

Для простих додатків також може вистачити і випуску Express: він безкоштовний. До того ж у нього є перевага - його можна ставити в якості реального сервера і використовувати в реальних задачах, однак він має урізаний функціонал в порівнянні з повною версією.

І також є MS SQL Server Developer Edition. Це повнофункціональний випуск, який містить весь функціонал, що і повна версія MS SQL Server Enterprise, тільки націлена для потреб розробки. У той же час ця версія не може бути використана для розгортання в якості реального сервера на реальних проектах. Однак для використання всієї механіки MS SQL Server ця версія являє оптимальний варіант, тому саме цю версію ми і будемо використовувати при розробці нашої інформаційної системи.

2.3 Проектування логічної структури бази даних

Прийнято виділяти всі сутності в додатку в окремі моделі. Залежно від поставленого завдання і складності застосування можна визначити різну кількість моделей. Так, в нашому додатку використовуються дві моделі - клас для автомобіля і клас для пошуку автомобіля.

Моделі є простими класами і знаходяться в проекті в каталозі Models. Моделі описують логіку даних. Наприклад, модель, що представляє автомобіль і переглядає її.

Модель складається тільки з властивостей, крім того, вона має конструктор та деякі допоміжні методи. Головне не перевантажувати клас моделі і пам'ятати, що її призначення - описати дані. Маніпулювання даними та бізнес-логіка – це більше сфера контролерів.

Дані моделі зберігаються в базі даних. Для роботи з базою даних використовується фреймворк Entity Framework, який дозволяє абстрагуватися від запису SQL-запитів, побудови бази даних та фокусування на логіці програми.

Під час створення проекту MVC 5 ми обрали модель з автентифікацією, для цього підключили фреймворк EntityFramework через менеджер пакетів NuGet, як подано в третьому розділі.

Крім того, ми використовувати консоль менеджера пакунків як альтернативу NuGet. Для цьогоо консоль менеджера пакетів відкриється внизу пакета Microsoft Visual Studio наступною командою:

```
PM> Install-Package EntityFramework -Version 5.0.2
```

Після введення команди пакет Entity Framework звантажено і встановлено. Іноді краще використовувати цю консоль при установці пакетів, ніж менеджер NuGet, оскільки менеджер NuGet може трохи запізнитися на випуск останніх

пакетів. Або навпаки, за необхідності встановлюються пакети більш ранньої версії, і тут NuGet запропоновує тільки поточну версію.

Структура сутностей підтримує підхід "Code First", який передбачає зберігання або вилучення інформації з бази даних на сервері SQL без створення схеми бази даних або використання конструктора в Visual Studo. Навпаки, ми створюємо регулярні класи, а з допомогою Entity Framework вже визначаємо, як і де зберігати об'єкти цих класів.

Для підключення до бази даних через Entity Framework нам потрібен посередник - контекст даних. Контекст даних - це клас, отриманий з класу DbContext. Контекст даних містить одну або кілька властивостей, таких як DbSet<T>, де T представляє тип об'єкта, що зберігається в базі даних.

З автомобілями та рекомендаціями ми маємо доступ до даних відповідних моделей, які зберігаються в базі даних.

Для збереження даних про автомобілі, їх характеристики, дані користувача та іншу інформацію, що використовується в програмному забезпечені, визначимо сутності бази даних (таблиці 2.1)

Пояснення зв’язків між сутностями запропонованої моделі даних наведені в таблиці 2.2.

Таблиця 2.1 – Сутності бази даних та їх характеристики

№	Ім’я сутності	Характеристики
1	2	3
1	AspNetUsers	Інформація про користувача
2	AspNetRoles	Інформація про роль користувача
3	AspNetUserClaim	Інформація про діяльність користувача
4	AspNetUserTokens	Маркер перевірки достовірності для користувача
5	AspNetUserLogin	Прив’язка користувача з іменем входу
6	AspNetRoleClaims	Оточення яке надається користувачеві в системі

Кінець таблиці 2.1

1	2	3
7	AspNetUserRole	Сутність JOIN, яка пов’язує користувачів та ролі
8	Havto	Характеристики автомобілів
9	Recomendation	Результати розрахунків оцінок близькості для автомобілів
10	PriceAvto	Вартість автомобіля
11	TextFields	Інформація про оголошення автомобіля
12	ServicesItems	Інформація результатів порівнянь розрахунків оцінок близькості для автомобілів

Таблиця 2.2 – Типи зв’язків даталогічної моделі бази даних

№	Сутність	Пов’язана сутність	Тип зв’язку
1	AspNetUsers	AspNetUserLogins	1:M
2	AspNetUsers	AspNetUserRolse	1:M
3	AspNetUsers	AspNetRoleClaims	1:M
4	AspNetUsers	AspNetUserTokens	1:M
5	AspNetRoles	AspNetRoleClaims	1:M
10	AspNetUserRoles	AspNetRoles	1:M
11	Havto	ConfirmRegistration	1:M
12	Havto	Tavto	1:M
13	PriceAvto	Havto	1:M
14	TextField	ServicesItems	1:M

Реалізація бази даних засобами MS SQL Server та MS SQL Server описана в розділі 3.1 «Реалізація бази даних».

Даталогічна модель бази даних наведена у Додатку Б, Рисунок Б.1.

2.4 Аналіз та вибір технологій і методів реалізації системи

Оскільки наш проект має клієнт-серверну архітектуру, то оперувати базою даних буде сервер в залежності від запитів клієнта. В результаті потрібно два застосунки, зокрема сам серверний додаток та клієнтський застосунок, який буде з ним взаємодіяти.

Для розробки цього проекту було обрано середовище розробки Visual Studio 2019 та мову програмування C#. Серверну частину пропонується розробляти на основі технології ASP.NET Core, на основі шаблону WEB API. Клієнтську частину розробляти для платформи .NET Framework, як веб-додаток. Проекти на ASP.NET Core достатньо зручно будувати за допомогою технології MVC (Model-View-Controller), хоч у WEB API не так важливий компонет View, оскільки, інформація автоматично повертається з контролерів в форматі json .

Для роботи з базою даних C#, інструментарій Microsoft Visual Studio 2019 та Entity Framework мають багато можливостей для зручної роботи з різними типами СКБД. В даному проекті, для підключення до бази даних та маніпуляцій з нею, доцільно використовувати бібліотеку засоби Microsoft Visual Studio для роботи з СКБД Microsoft SQL Server, а також інструментарій Microsoft SQL Server Management Studio with Azure Data Studio (рисунок 2.4). Загалом C# це об’єктно-орієнтована мова програмування, тому обробляти складну інформацію у вигляді об’єктів достатньо з допомогою неї досить зручно. C# використовується, як для розробки програм для настільних ПК, так і для розробки веб-сервісів, сайтів та інших застосунків. Дана мова є універсальною. На ній можна розробляти, як і прості проекти, так і досить потужні додатки. Також платформа .NET Core підтримує не лише операційну систему MS Windows, але й операційні системи сімейства Linux та MacOs.

Для створення веб-додатку використовується WEBAPI шаблон проекту ASP.NET Core. Вся основна логіка для роботи з базою даних знаходиться у веб-

додатку. Для проекту також була розроблена адміністративна веб-сторінка для управління контентом проектованого програмного забезпечення.

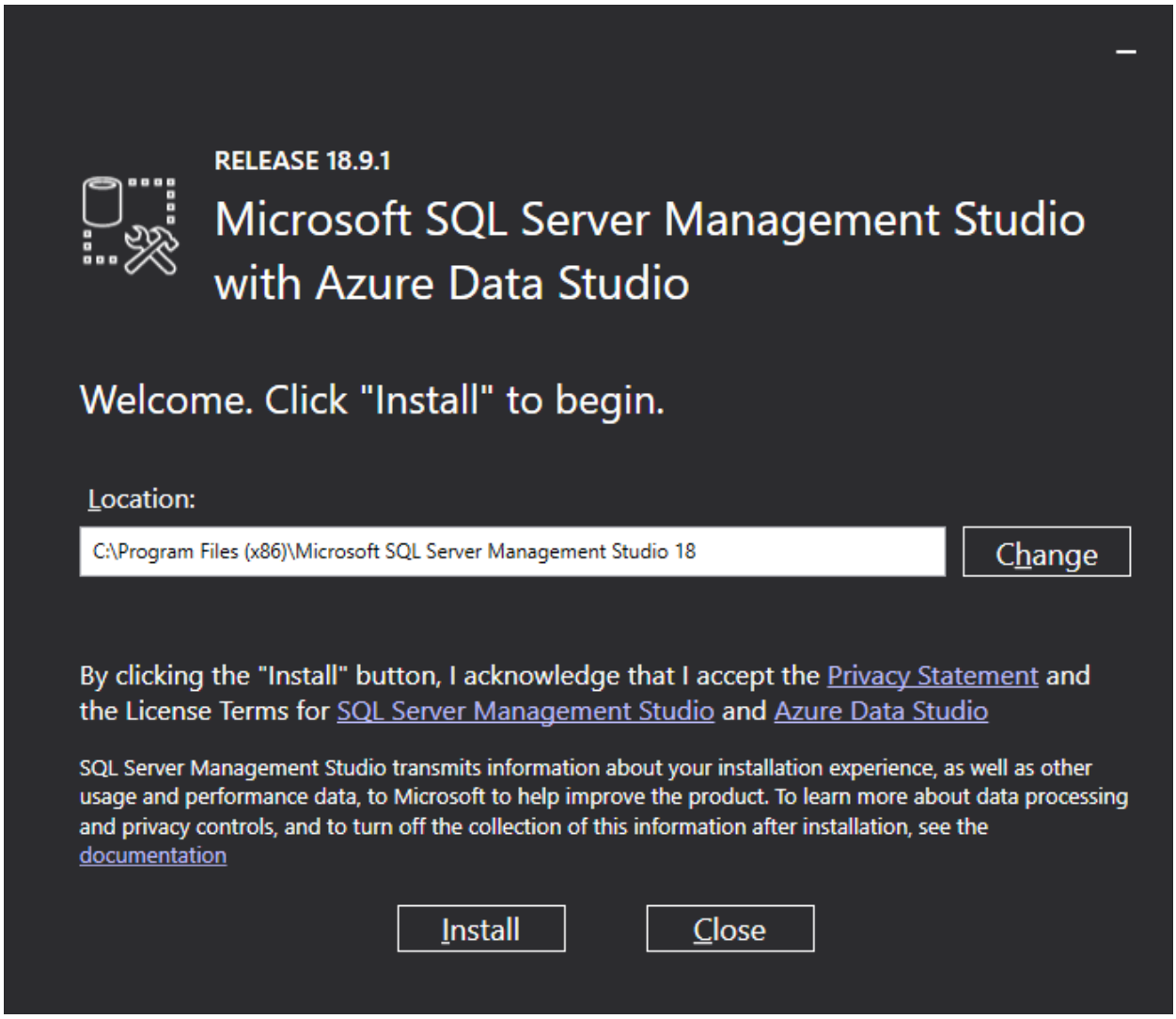


Рисунок 2.4 - Інструментарій Microsoft SQL Server Management Studio with Azure Data Studio

Для створення в базі даних системних таблиць, та допоміжних таблиць, щодо ведення бази даних та проекту, доцільно скористатись міграцією бази даних за допомогою інструменту dotnet ef. Міграція бази даних програмного коду в базу даних MS SQL Server наведено нижче:

```
C:\Users\Grygorchuk\source\repos\WebApplication1\WebApplication1>dotnet ef migrations add _initial
```

```
Build started...
Build succeeded.
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
Entity Framework Core 3.1.1 initialized 'AppDbContext' using provider
'Microsoft.EntityFrameworkCore.SqlServer' with options: None
Done. To undo this action, use 'ef migrations remove' .
```

Отже в розділі здійснено Проектування архітектури та структури системи. Запропоновано вибір технології проектвання, зокрема у якості архітектурного шаблону сервісу була обрана «MVC» модель. Здійснено аналіз та вибір СКБД та здійснено проектування логічної структури бази даних.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Детальне проектування модулів та реалізація бази даних

Наведемо опис основних алгоритмів роботи розроблюваної нами довідково-інформаційної системи.

На рисунку 3.1 зображено повний цикл для подання автомобіля з його характеристика. Для того, щоб подати інформацію про автомобіль, користувач повинен бути зареєстрованими в системі – мати свій обліковий запис. Тобто, для роботи з системою він повинен авторизуватись. Якщо ж він не зареєстрований в системі то йому буде запропоновано створити свій обліковий запис. Після авторизації він зможе обирати різні оголошення та створювати свої власні.

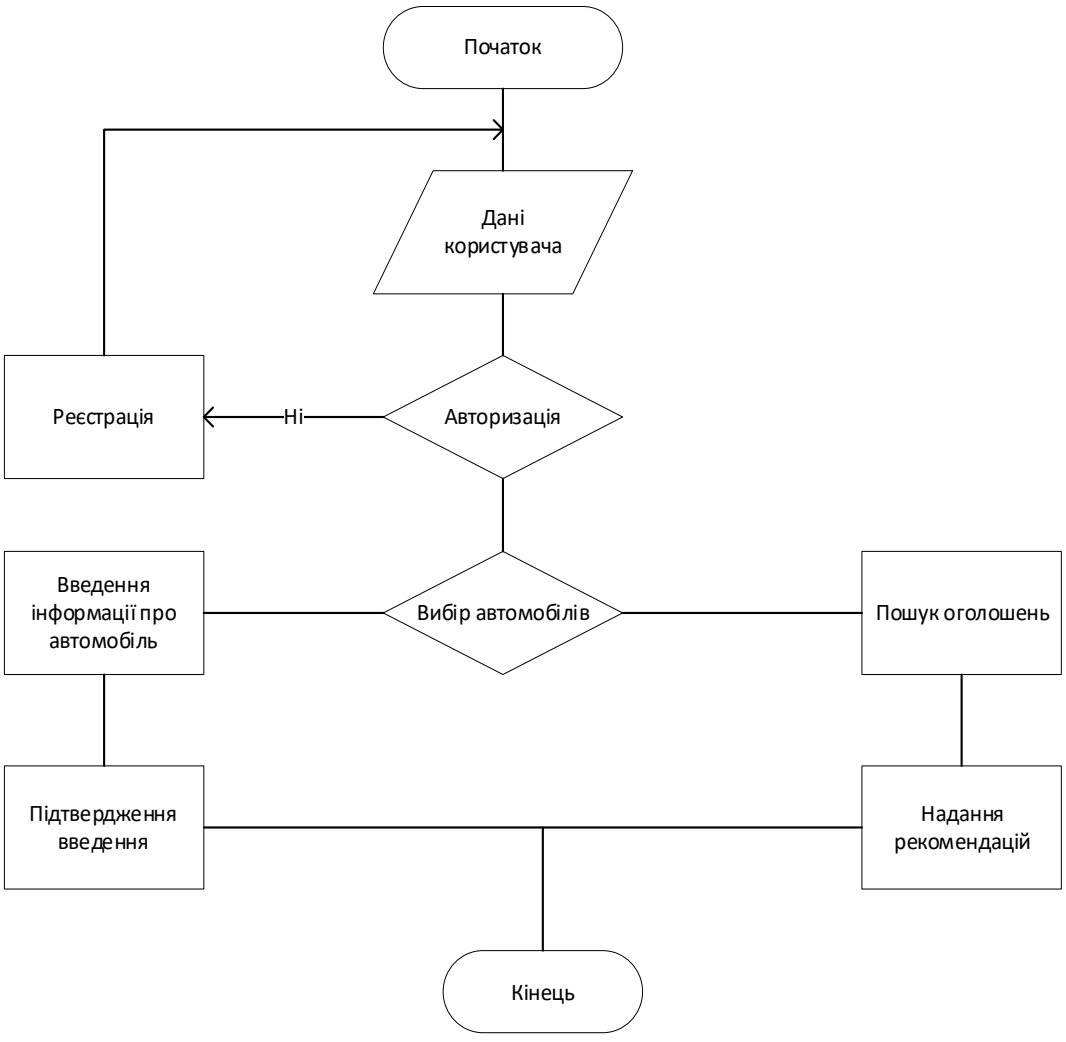


Рисунок 3.1 – Схема алгоритму пошук автомобілів та подання оголошень

Для знаходження необхідного автомобіля та його характеристик, користувач має обрати із запропонованого списку необхідний йому автомобіль, задати критерії пошуку вподобаних характеристик та серед запропонованих автомобілів обрати необхідний.

Для проектування даної системи ми використали об'єктно-орієнтований підхід, було розроблено ряд діаграм послідовностей для варіантів використання. Зокрема, на рисунку 3.2 подано діаграму послідовності модуля «Аналіз запитів-файлів»

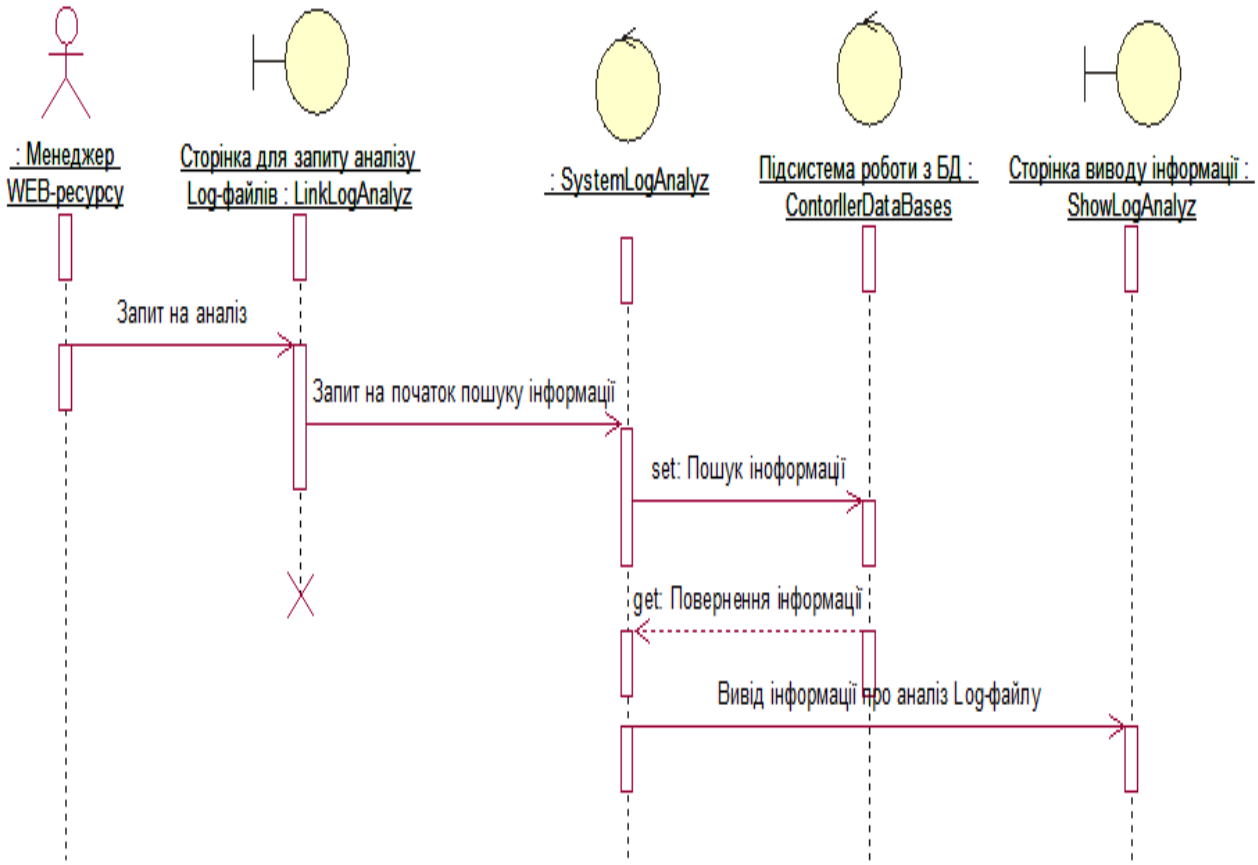


Рисунок 3.2 – Діаграма послідовності модуля аналізу запитів

Наступним кроком є побудова діаграми кооперацій, яка надає представлення про відношення класів, підсистем та інтерфейсів. На рисунку 3.3 подано діаграму кооперацій модуля аналізу запитів.

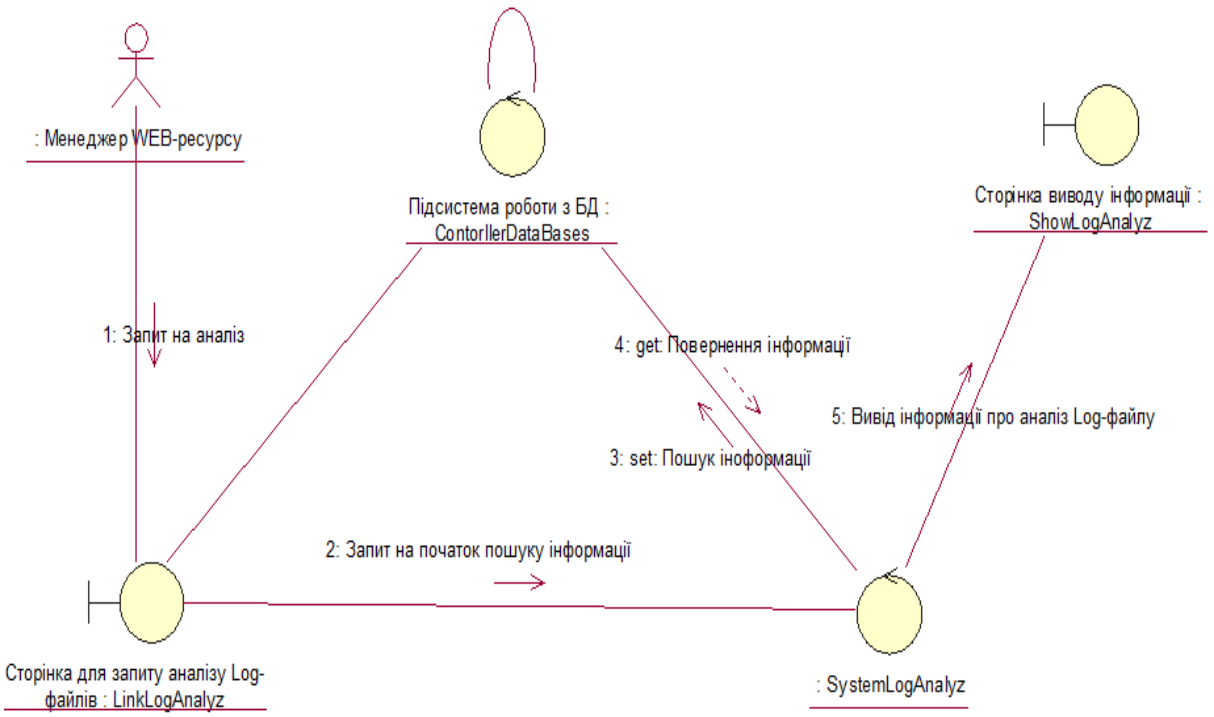


Рисунок 3.2 – Діаграма кооперацій модуля аналізу запитів

Аналогічним чином створюється діаграма модуля для статистичних сервісів (рисунок 3.3).

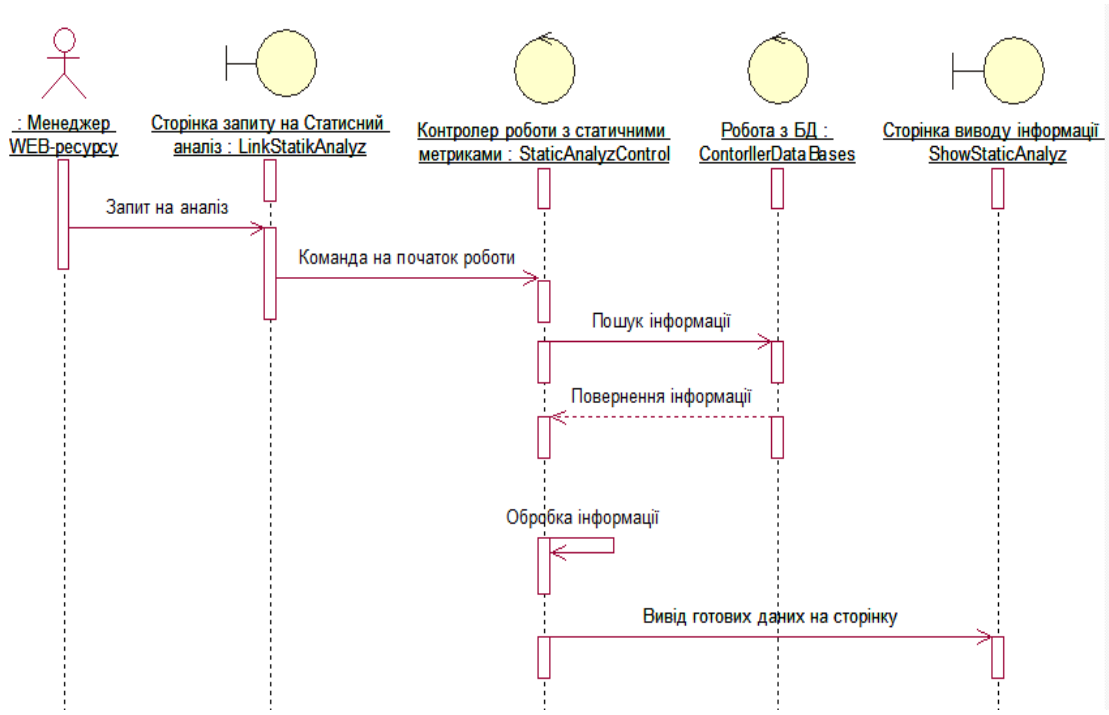


Рисунок 3.3 – Діаграма послідовності модуля «Статистичні сервіси»

На рисунку 3.4 подано діаграму кооперацій для модуля статистичних сервісів.

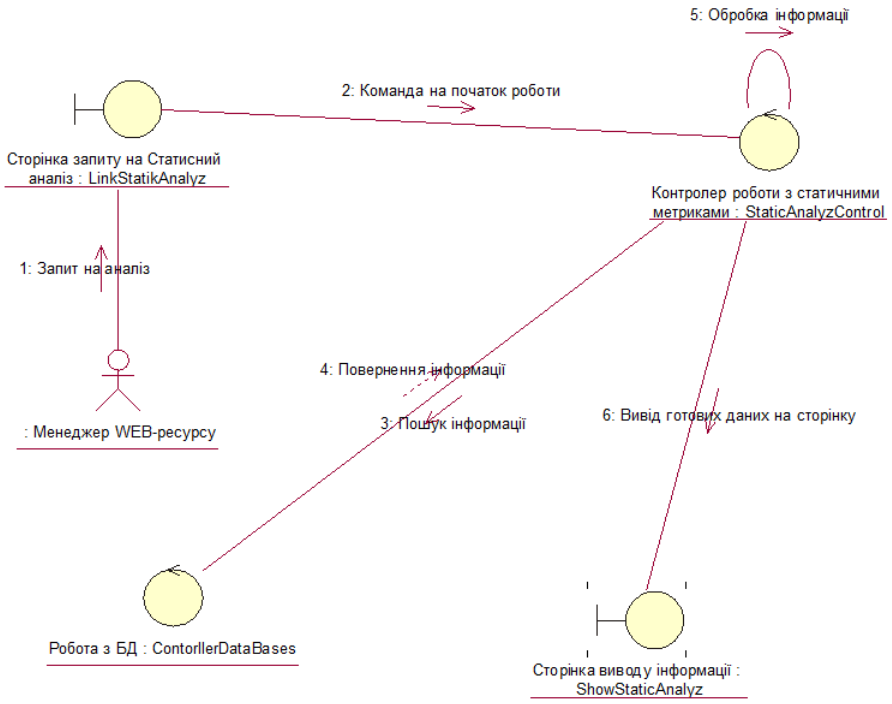


Рисунок 3.4 – Діаграма кооперацій модуля «Статистичні сервіси»

Наступник кроком опишемо відношення спроектованих класів для модуля наших сервісів (рисунок 3.5).

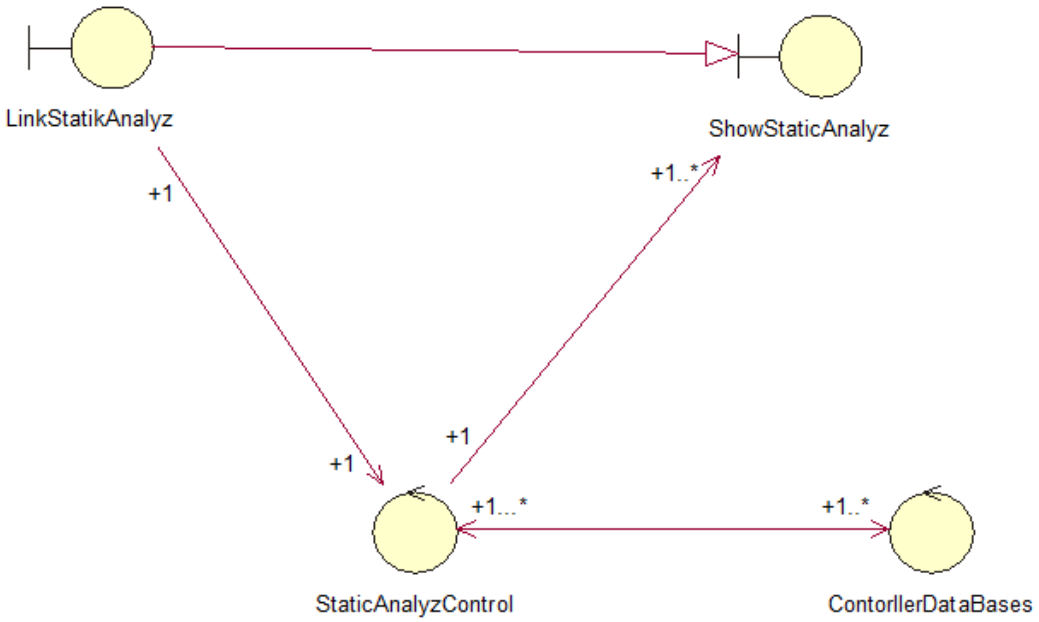


Рисунок 3.5 – Діаграма класів для модуля статистичних сервісів

Для реалізації додатку було обрано веб-додаток, який будемо здійснювати на основі архітектурного патерну MVC (Model-View-Controller) на основі opensource-фреймворку ASP.NET Core.

ASP.NET Core може працювати поверх крос-платформного середовища .NET Core, яка може бути розгорнута на основних популярних операційних системах: Windows, Mac OS, Linux. І таким чином, за допомогою ASP.NET Core можна створювати крос-платформні додатки. Тобто ми можемо запускати веб-додатки не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-додатки можна використовувати традиційний IIS, або крос-платформний веб-сервер Kestrel.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget. Крім того, на відміну від попередніх версій платформи немає необхідності використовувати бібліотеку System.Web.dll.

ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версії платформи дані технології реалізувалися окремо і тому містили багато дублюючої функціональності. Зараз же вони об'єднані в одну програмну модель ASP.NET Core MVC. А Web Forms повністю пішли в минуле.

Крім об'єднання вищезазначених технологій в одну модель в MVC був доданий ряд додаткових функцій.

Однією з таких функцій є тег-хелпери (tag helper), які дозволяють більш органічно поєднувати синтаксис html з кодом C #.

ASP.NET Core характеризується розширюваністю. Фреймворк побудований з набору щодо незалежних компонентів. І ми можемо або використовувати вбудовану реалізацію цих компонентів, або розширити їх за допомогою механізму спадкування, або зовсім створити і застосовувати свої компоненти зі своїм функціоналом.

Також було спрощено управління залежностями і конфігурація проекту. Фреймворк тепер має свій легкий контейнер для впровадження залежностей, і більше немає необхідності застосовувати сторонні контейнери, такі як Autofac, Ninject. Хоча при бажанні їх також можна продовжувати використовувати.

В якості інструментарію розробки ми можемо використовувати останні випуски Visual Studio 2019. Крім того, ми можемо створювати додатки в середовищі Visual Studio Code, яка є крос-платформної і може працювати як на Windows, так і на Mac OS X і Linux.

Для обробки запитів тепер використовується новий конвеєр HTTP, який заснований на компонентах Katana і специфікації OWIN. А його модульність дозволяє легко додати свої власні компоненти.

Якщо підсумувати, то можна виділити наступні ключові відмінності ASP.NET Core від попередніх версій ASP.NET:

– новий легкий і модульний конвеєр HTTP-запитів;

– можливість розгортати додаток як на IIS, так і в рамках свого власного процесу;

– використання платформи .NET Core і її функціональності;

– поширення пакетів платформи через NuGet;

– інтегрована підтримка для створення та використання пакетів NuGet;

– єдиний стек веб-розробки, що поєднує Web UI і Web API;

– конфігурація для спрощеного використання в хмарі;

– вбудована підтримка для впровадження залежностей;

– можливість розширення;

– кросплатформеність: можливість розробки і розгортання додатків ASP.NET на Windows, Mac і Linux;

– розвиток як open source, відкритість до змін.

Ці та інші особливості і можливості стали основою для нової моделі програмування.

3.2 Реалізація модулів системи

Для створення додатку ASP.NET Core необхідно встановити необхідні пакети з допомого описаного вище сервісу NuGet. створимо директорії та файли нашого довідково-інформаційної системи.

Зокрема, необхідно встановити пакет Microsoft.VisualStudio/CodeGeneration Design, який призначений для створення заготовок класів контролерів та представлень (View) Code Generation tool for ASP.NET Core (рисунок 3.6)

Рисунок 3.6 - Пакет Microsoft.VisualStudio/CodeGeneration Design.

Встановити пакет Microsoft.Extensions.Loggging.Debug, який дозволяє виводити логи в процесі налагодження (рисунок 3.7)

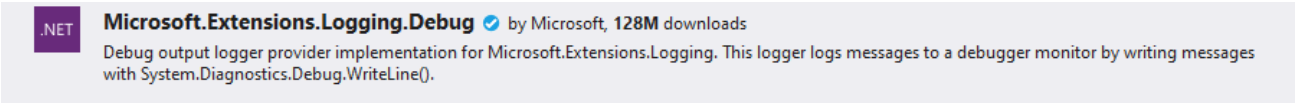


Рисунок 3.7 – Пакет Microsoft.Extensions.Loggging.Debug

Набір пакетів для роботи з Entity Framework (Рисунок 3.8)

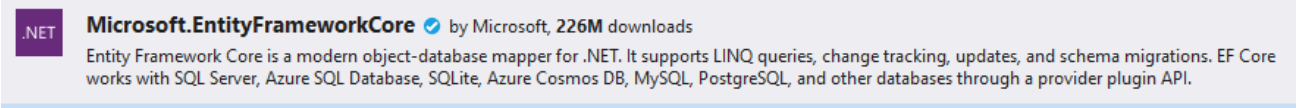


Рисунок 3.8– Пакет Microsoft.EntityFrameworkCore

Пакет для роботи з MS SQL Server - Провайдер для роботи з базами даних з фреймворком Entity Framework (Рисунок 3.10)

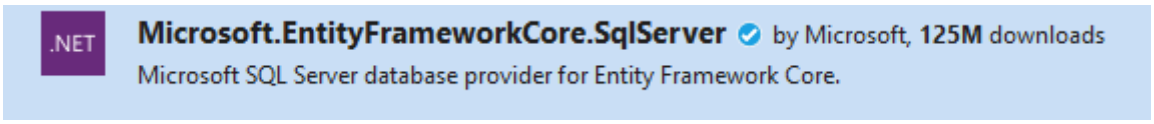


Рисунок 3.9 – Пакет Microsoft.EntityFarmeworkCore.SqlServer

Пакет для створення доменної моделі Microsoft.EntityFarmeworkCore.Design (рисунок 3.10)

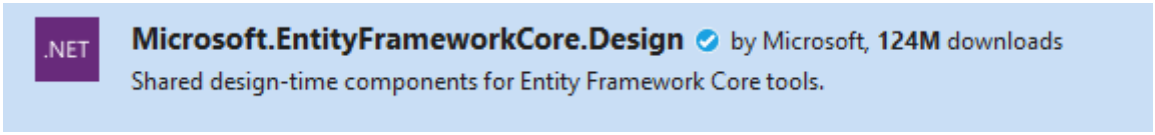


Рисунок 3.10 – Пакет Microsoft.EntityFarmeworkCore.Design

Пакет для роботи з користувачами, паролями, логінами та їх контролем Microsoft.AspNetCore.IdentityFrameWorkCore (Рисунок 3.11)

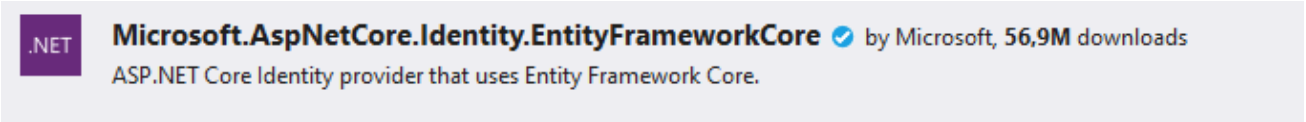


Рисунок 3.11 – Microsoft.AspNetCore.IdentityFrameWorkCore

Далі, для проектування додатку ASP.NET Core створимо необхідні директорії та файли нашої довідково-інформаційної системи.

Першою папкою, виступає папка wwwroot – стандартна папка ASP.NET Core, де будуть зберігатись статичні файли: стилі, скрипти, зображення тощо. Зокрема, у ній містяться області:

– css – для файлів стилів;
– images – для збереження зображень, фото, іконок та картинок;
– js – файли скриптів;

– sass – стилі метамови, які автоматично інтерпритуються в каскадні таблиці стилів (css).

Наступна директорія Areas.

Areas – область для всього додатку ASP.NET Core , в якій визначені окремий набір контролерів, представлень – окремою функціональною логікою. В цій директорії будуть міститись папкAdmin:

Admin – в цій області будуть всі операції по адмініструванню сайту , яка в свою чергу містяться папки Controllers (контролери) та Views (представлення).

Кореневі директорії:

– Controllers – контролери для основної частини додатку;

– Views/Shared – представлення для основної частини додатку;

– Layout.cshtml – майстер сторінка;

– Domain - для збереження доменної моделі, так звані бізнес-сутності (контекст підключення до бази даних, визначенні поля - тощо ;

– Models – моделі на рівні додатку, яка в свою чергу буде містити підпапку ViewComponents – компоненти представлення;

– Service - бізнес-логіка для обслуговування додатку, для роботи з функціоналос самого додатку.

У файлі кореневої директорії розміщуємо Файл appsettings.json – всі налаштування додатку та рядок підключення до бази даних:

```
"Project": {
    "ConnectionString": "Data Source=(local)\\SQLSERVER; Database=DAvto; Persist
Security Info=falce; User ID='sa'; Password='sa'; MultipleActiveResultSets=True;
Trusted_Connection=False;",
    "CompanyName": "Пошук автомобілів",
    "CompanyPhone": "+38 (097) 638-78-65",
    "CompanyPhoneShort": "+380976387865",
    "CompanyEmail": Mr.LonelyW1zard@gmail.com
```

Окрім того в кореневому директорії розміщуємо файли Program.cs та startup.cs:

Program.cs – головний файл, в якому визначено одноіменний клас Program і з якого, по суті, починається виконання додатку.

Файл startup.cs – це також точка входу, і він викликається після виконання файлів Program.cs на рівні додатків. Крім того він звертається до конвеєра запитів. Клас Statup запускається в момент запуску додатків.

На рисунку 3.13 наведено структуру папок нашого проектованого додатку в середовищі Microsoft Visual Studio 2019.

Опишемо далі створення модулів системи. Одним із головних модулів виступає модуль Startup.sh.

Створимо клас startup.

В Методі ConfigureServices() зареєструємо сервіси із колекції IServiceCollection – додамо підтримку контролерів та представлень (MVC) і опція сумісності з версіями AspNetCoreMVC:

```
public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews()

      .SetCompatibilityVersion(Microsoft.AspNetCore.Mvc.CompatibilityVersion.Version_3_0
      ).AddSessionStateTempDataProvider();
        }
```

В методі app.UseStaticFile() підключаємо підтримку статичних файлів css, js scss тощо.

В методі app.UseEndpoints() налаштовуєм маршрути по замовчуванню:

```
endpoints.MapControllerRoute("default","{controller=Home}/{ Action =
Index}/{id?}"
```

Домена модель сайту знаходиться у каталозі Entities, каталогу Domain, який містить базовий абстрактний клас для всіх сутностей EntityBase

Даний клас створений для сайту та не застосовується в доменній моделі, але відповідає за її підтримку:

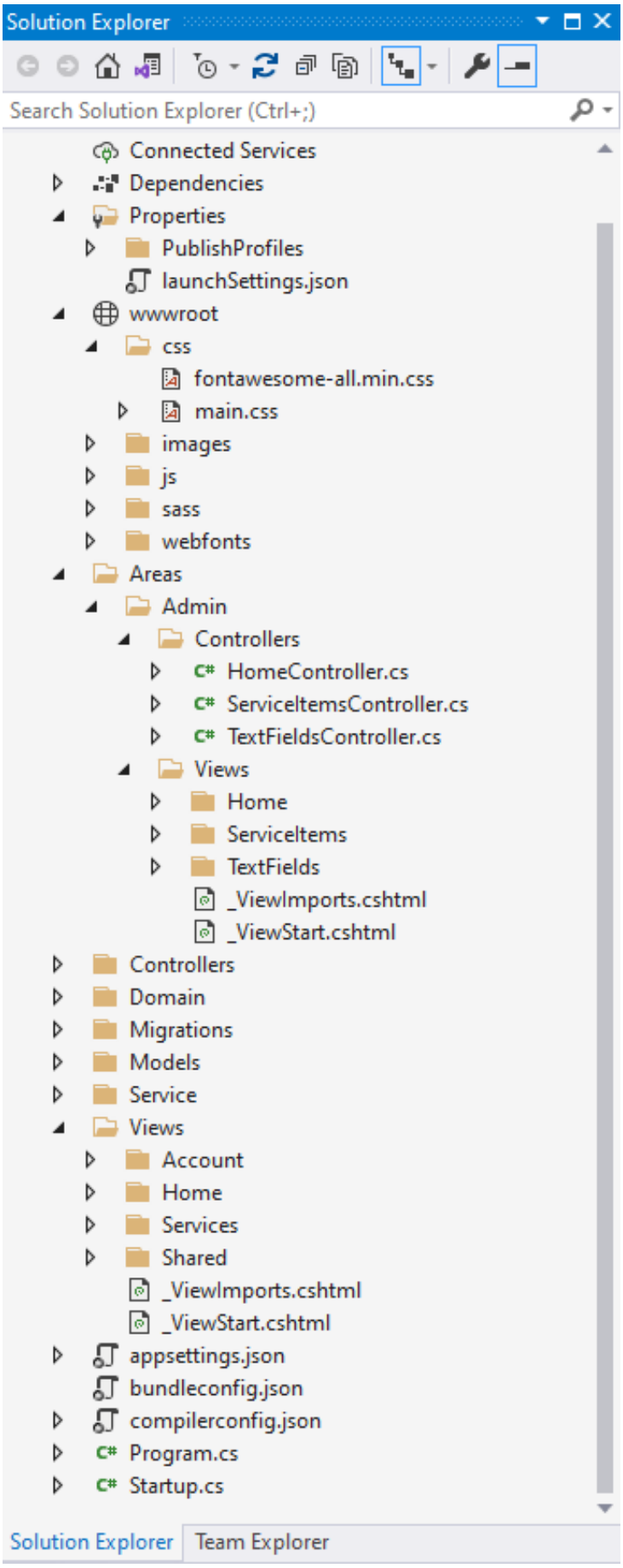


Рисунок 3.12- Структура модулів сайту

```
public abstract class EntityBase
        {
         protected EntityBase() => DateAdded = DateTime.UtcNow;
         [Required]
         public Guid Id { get; set; }
         [Display(Name = "Назва (заголовок)")]
         public virtual string Title { get; set; }
         [Display(Name = "Короткий опис")]
         public virtual string SubTitle { get; set; }
         [Display(Name = "Повний опис)")]
         public virtual string Text { get; set; }
         [Display(Name = "Титульна картинка")]
         public virtual string TitleImagePath { get; set; }
         [Display(Name = "SEO метатег Title")]
         public string MetaTitle { get; set; }
         [Display(Name = "SEO метатеu Description")]
         public virtual string MetaDescription { get; set; }
         [Display(Name = "SEO метатег keywords")]
         public virtual string MetaKeywords { get; set; }
         [DataType (DataType.Time)]
         public DataType DateAdded { get; set; }
```

Дочірній клас TextField класу EntityBase містить інформацію про сторінку з контактами і головної сторінки:

```
public class TextField : EntityBase
    {
         [Required]
         public string CodeWord { get; set; }
         [Display(Name = "Назва сторінки (заголовок)")]
         public override string Title { get; set; } = "Інформаційна сторінка";
         [Display(Name = "Вміст сторінки")]
         public override string Title { get; set; } = "Вміст заповнюється
адміністратором";
           }
```

Дочірній клас ServiceItem класу EntityBase, клас з обробкою даних послуг, які надаються нашим додатком:

```
public class ServiceItem:EntityBase
    {
         [Required(ErrorMessage ="Заповніть назву послуги")]
         [Display(Name = "Назва послуги")]
         public override string Title { get; set; }

         [Display(Name = "Короткий опис послуги")]
         public override string SubTitle { get; set; }

         [Display(Name = "Повний опис послуги")]
         public override string SubTitle { get; set; }
           }
```

Пов’яжемо дані домені об’єкти з базою даних, тобто спроектуємо їх на базу даних. Для цього створимо контекст бази даних для нашого додатку, в якому буде логіка роботи додатку з базою даних. Для цього створимо клас AppDbContext:

```
public class AppDbContext : IdentityDbContext<IdentityUser>
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }

        public DbSet<TextField> TextFields { get; set; }
        public DbSet<ServiceItem> ServiceItems { get; set; }

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

            modelBuilder.Entity<IdentityRole>().HasData(new IdentityRole
            {
                Id = "64546e06-8711-4a68-b88a-f271ae9p6eab",
                Name = "admin",
                NormalizedName = "ADMIN"
            });

            modelBuilder.Entity<IdentityUser>().HasData(new IdentityUser
            {
                Id = "3b62472e-4pp6-49pa-a25f-e7685b94577d8",
                UserName = "admin",
                NormalizedUserName = "ADMIN",
                Email = "my@email.com",
                NormalizedEmail = "MY@EMAIL.COM",
                EmailConfirmed = true,
                PasswordHash = new PasswordHasher<IdentityUser>().HashPassword(null,
"superpassword"),
                SecurityStamp = string.Empty
            });

            modelBuilder.Entity<IdentityUserRole<string>>().HasData(new
IdentityUserRole<string>
            {
                RoleId = "47546e06-8719-4ad7-b88a-f271ae7d6eab",
                UserId = "3332472e-4f36-49fa-a20f-e7685b9265d8"
            });

            modelBuilder.Entity<TextField>().HasData(new TextField
            {
                Id = new Guid("62dc2fa6-07ae-4391-8916-e053f71239ce"),
                CodeWord = "PageIndex",
                Title = "Головна"
            });
            modelBuilder.Entity<TextField>().HasData(new TextField
            {
                Id = new Guid("20bf145a-700a-4123-94c0-e83fce0a244f"),
                CodeWord = "PageServices",
                Title = "Наші послуги"
            });
            modelBuilder.Entity<TextField>().HasData(new TextField
            {
                Id = new Guid("2aa26a4c-c59d-229a-84c1-06e4484a137a"),
```

```
            CodeWord = "PageContacts",
            Title = "Контакти"
        });
    }
```

Системний об’єкт IdetityUser містить найбільш вживані атрибути, як ми розширюємо своїми, додаючи потрібні властивості. Серед них зокрема:

Id, UserName, PhoneNumber, Email, EmailConfirmed, TwoFactorEnable,PasswordHash, SecurityStamp, AccesFailedStamp тощо.

Тут пов’язується користувач з відповідними таблицями бази даних, в яких надаються послуги щодо вибору автомобілів та рекомендацій:

```
public DbSet<TextField> TextFields { get; set; }
public DbSet<ServiceItem> ServiceItems { get; set; }
```

Далі програмуємо функціонал, який дозволить проводити операції над даними класами – створювати, обновляти та видаляти інформацію, з допомогою фреймворку, через sql-запити в базу даних. Для цього в папці Domain, створимо папку простора імен Repositories для маніпуляцій над об’єктами, де будуть інтерфейси і відповідні їм класи. Зокрема в папці простру імен Abstract ми визначили інтерфейси для наших доменів об’єктів, які реалізують потрібну нам поведінку. Наприклад, інтерфейс ITextFieldsRepository містить методи: зробити вибірку всіх текстових полів, вибрати текстове поле по ідентифікатору, вибрати по ключовому слову, зберегти зміни в базу даних і видалити текстове поле:

```
public interface ITextFieldRepository
    {
      IQueryable<TextField> GetTextFields();
      TextField GetTextFieldById(Guid id);
      TextField GetTextFieldByCodeWord(string codeWord);
      void SaveTextField(TextField entity);
      void DeleteTextField(Guid id);}
```

Аналогічний код створюємо для наших послуг в інтерфейсі IServiceItemsRepository: вибати послугу, оновити або створти, а також видалити деяку певну послугу.

В папці простору імен EntityFramework, створюємо конкретну реалізацію даних методів у класах EFServiceItemsRepository та EFTextFieldsRepository :

Далі створимо обслуговуючий клас Datamager в якому буде централізовано зберігатись управління нашим репозиторієм:

```
public class DataManager
    {
        public ITextFieldsRepository TextFields { get; set; }
        public IServiceItemsRepository ServiceItems { get; set; }

        public DataManager(ITextFieldsRepository textFieldsRepository,
IServiceItemsRepository serviceItemsRepository)
        {
            TextFields = textFieldsRepository;
            ServiceItems = serviceItemsRepository;
        }
      }
```

За функціонування та роботу всієї системи відповідає файл проекту Startup.cs, налаштування в якому здійснимо наступним алгоритмом:

– підключаємо функціонал додатку в якості сервісів;

– підключаємо контекст бази даних;

– налаштовуємо identity систему - налаштовуємо мінімальну довжину пароля, встановлюємо використання цифр обов’язково в паролях тощо. Тобто визнаємо мінімальні вимоги до системи безпеки;

– налаштовуємо authentication cookie;

– підключаємо в методі Configure авторизацію і автентифікацію.

3.3 Розгортання та встановлення системи

Розглянемо другу форму фізичного представлення програмної системи - діаграму розгортання (розміщення) .Діаграма розгортання - діаграма, на якій представлені вузли виконання як програмних компонентів реального часу, а також процеси та об'єкти.

Діаграма розгортання програм призначена для представлення загальної конфігурації та топології розподіленої програмної системи та містить зображення розміщення компонентів за окремими модулями. Крім того, діаграма розгортання показує наявність фізичних з'єднань - маршрути передачі інформації між програмними компонентами, що реалізуються в проектованій системі.

Оскільки наш додаток розроблений за технологією клієнт-сервер, то в найбільш загальному вигляді її можна представити наступним чином (рисунок 3.13).

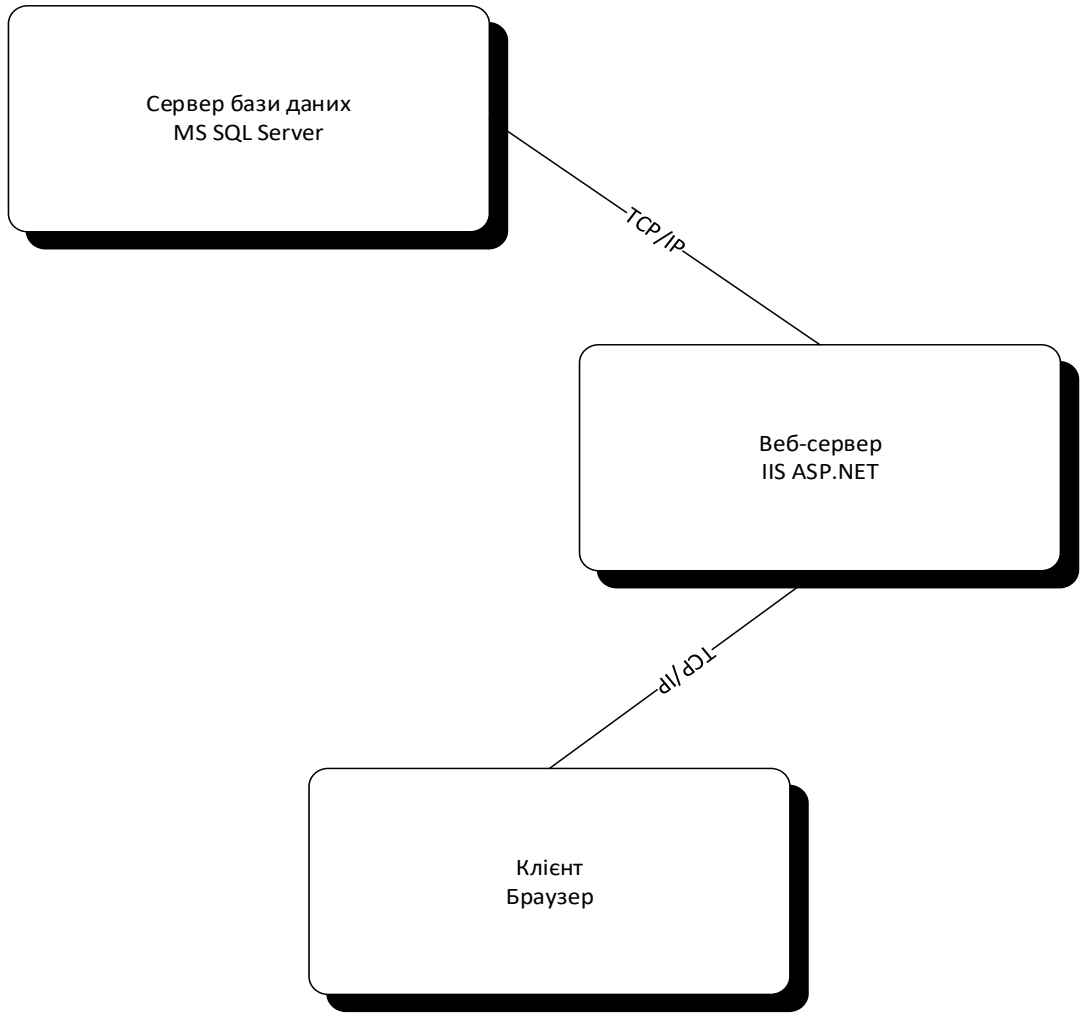


Рисунок 3.13 – Діаграма розгортання

Більш детальне представлення діаграми розгортання саме нашої довідково -інформаційної системи наведено на рисунку 3.14

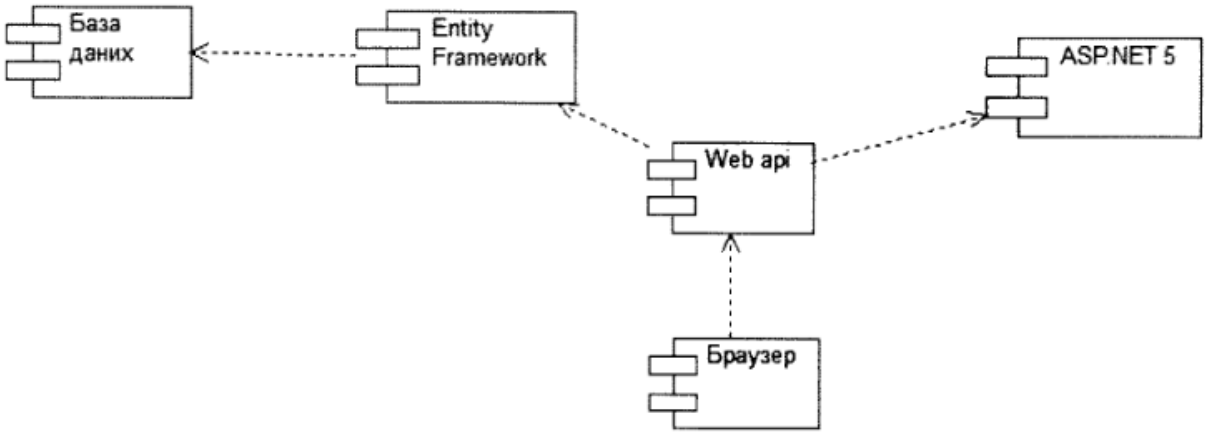


Рисунок 3.14 – Компоненти ситеми

Для встановлення нашої довідково -інформаційної системи системи достатньо розмістити файли проекту в кореневому каталозі веб-сервера та налаштувати систему керування базами даних.

В нашому випадку ключовим моментом розгортання системи є налагодження СКБД. Як зазначалось раніше, якості СКБД ми обрали MS SQL Server 2019. Отже, встановимо MS SQL Server 2019 Developer Edition. Для цього перейдемо за адресою https://www.microsoft.com/en-us/sql-server/sql-server-downloads. Знайдемо на сторінці пункт "Developer" і натиснемо на кнопку завантаження дистрибутива.

Після завантаження програми установки запустимо її.На початку нам буде запропоновано два варіанти установки: базова або з налаштуваннями окремих компонентів. Для потреб даного проекту цілком вистачить і базової установки.

На початковому етапі вкажемо в якості типу установки тип "Базова":

Після прийняття ліцензійної угоди нам буде запропоновано встановити шлях для установки:

Залишимо всі налаштування за замовчуванням і натиснемо на кнопку "Встановити".

Після завершення установки на фінальному екрані ми побачимо ряд даних, які нам можуть знадобиться в подальшому при роботі з MS SQL Server.

Далі переходимо до вибору компонентів для установки MS SQL Server. Перш за все це ім'я екземпляра - за замовчуванням MSSQLSERVER . З цього імені ми зможемо знайти запущений сервер в панелі служб або можна це ім'я використовувати при зверненні до сервера з інших програм.

Також варто відзначити рядок підключення, яка за умовчанням "Server=localhost;Database=master;Trusted_Connection=True;". Через цей рядок підключення ми зможемо підключатися до сервера з інших програм, запущених на цьому ж комп'ютері. Отже, ми встановили SQL Server 2019, у якого ім'я екземпляра - "MSSQLSERVER" (рисунок 3.15). Слід зазначити, що перед підключенням до нього, треба переконатися, що він запущений. Для цього можна відкрити вікно служб. Якщо він не запущений, там же в панелі служб ми його можемо запустити, і після цього працювати створивши пароль адміністратора (рисунок 3.16– 3.17).

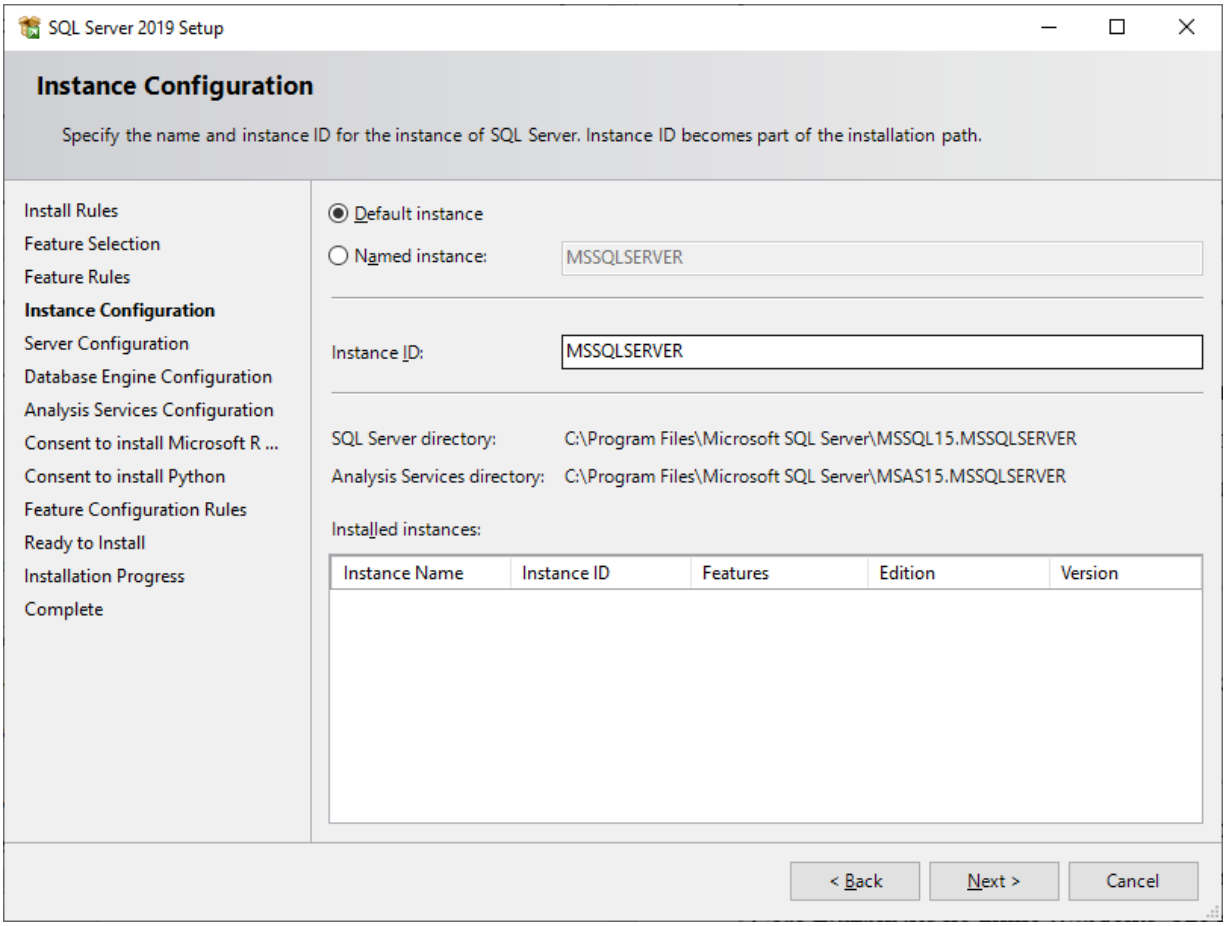


Рисунок 3.15 – Встановлення та налаштування сервера СКБД

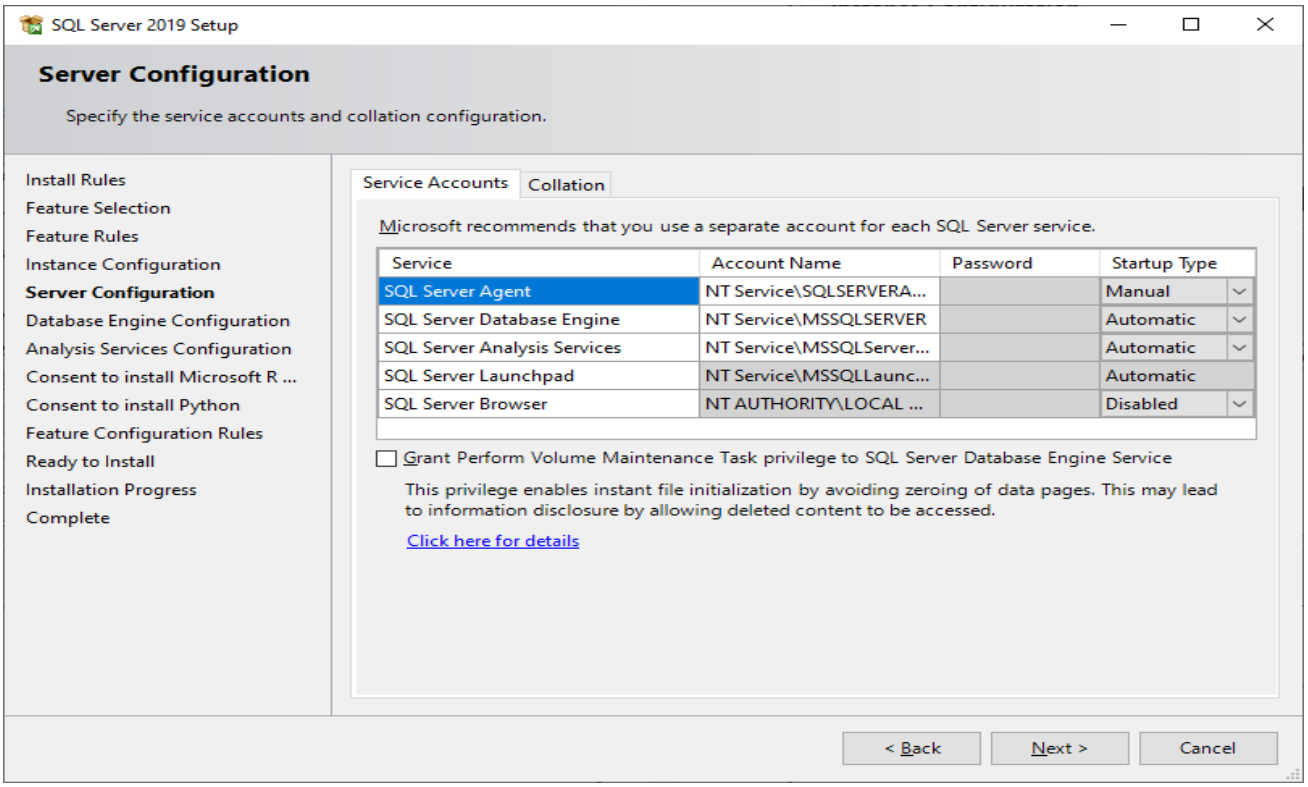


Рисунок 3.16– Запуск служб SQL Server

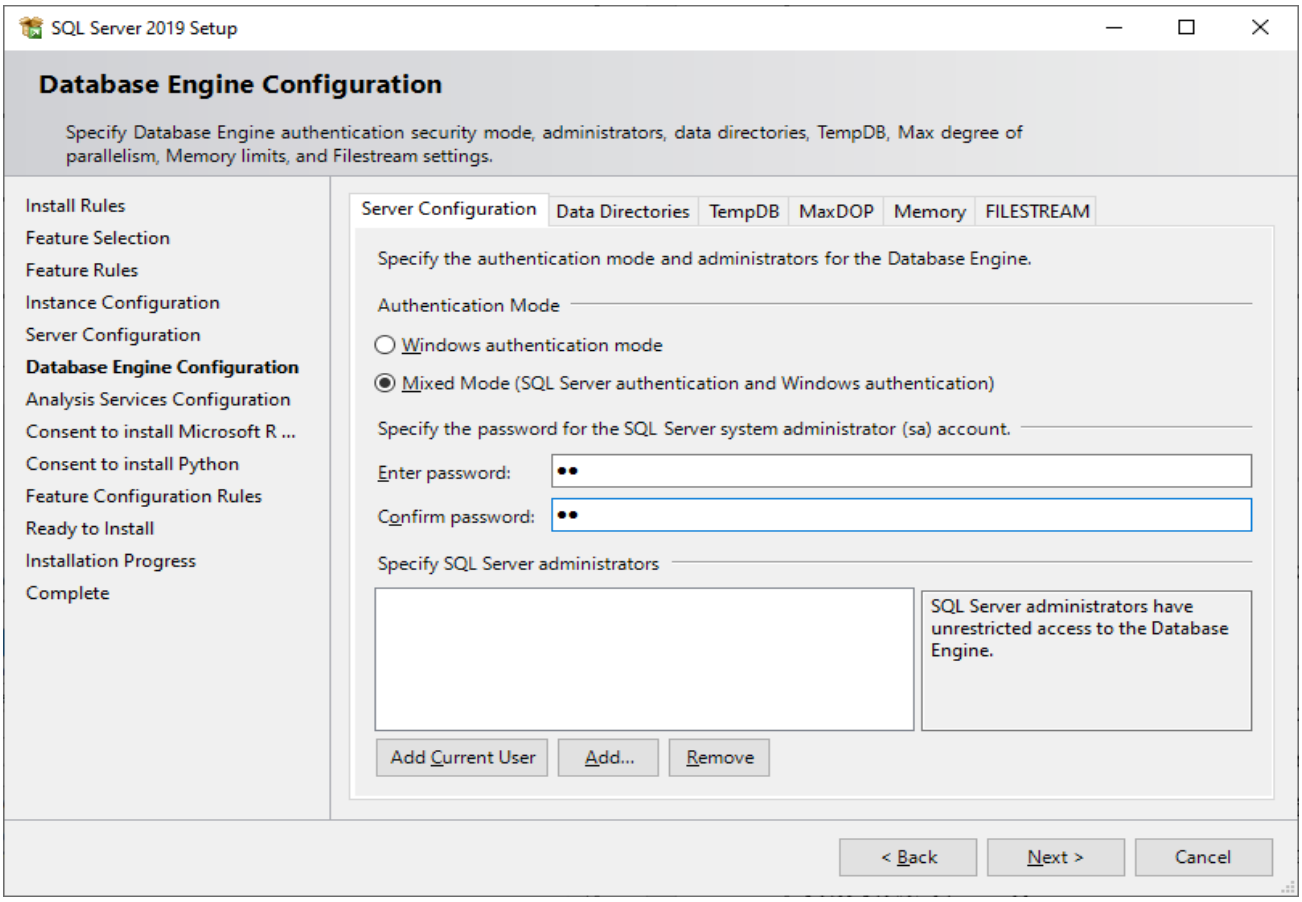


Рисунок 3.17 – Встановлення пароля адміністратора SQL Server

Підключення до сервера здійснюється з допомогою діалогового вікна з'єднання в якому обираємо ім'я сервера системи керування базою даних та введенням паролю адміністратора (Рисунок 3.18).

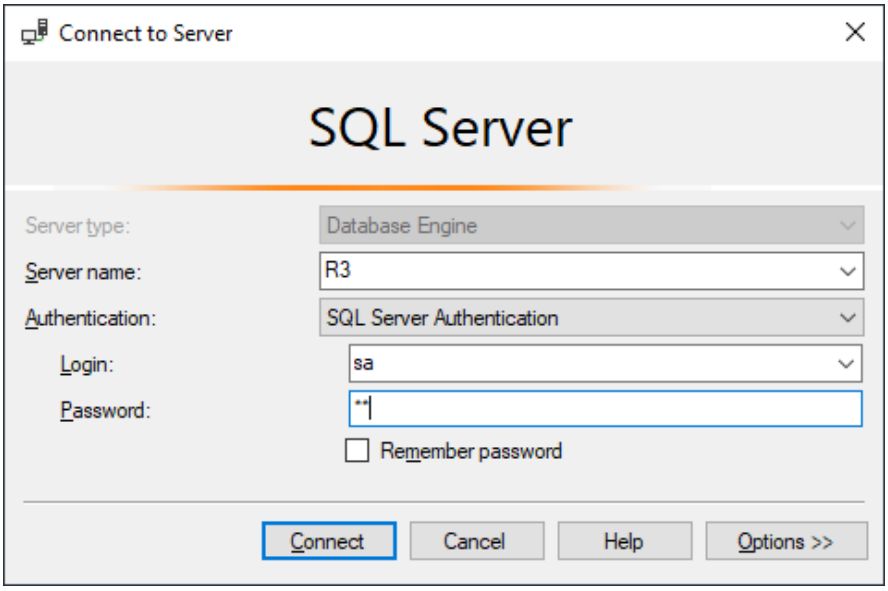


Рисунок 3.18 - Вікно з'єднання з базою даних

Результат успішного під'єднання до сервера бази даних та нашої бази даних наведено на рисунку 3.19

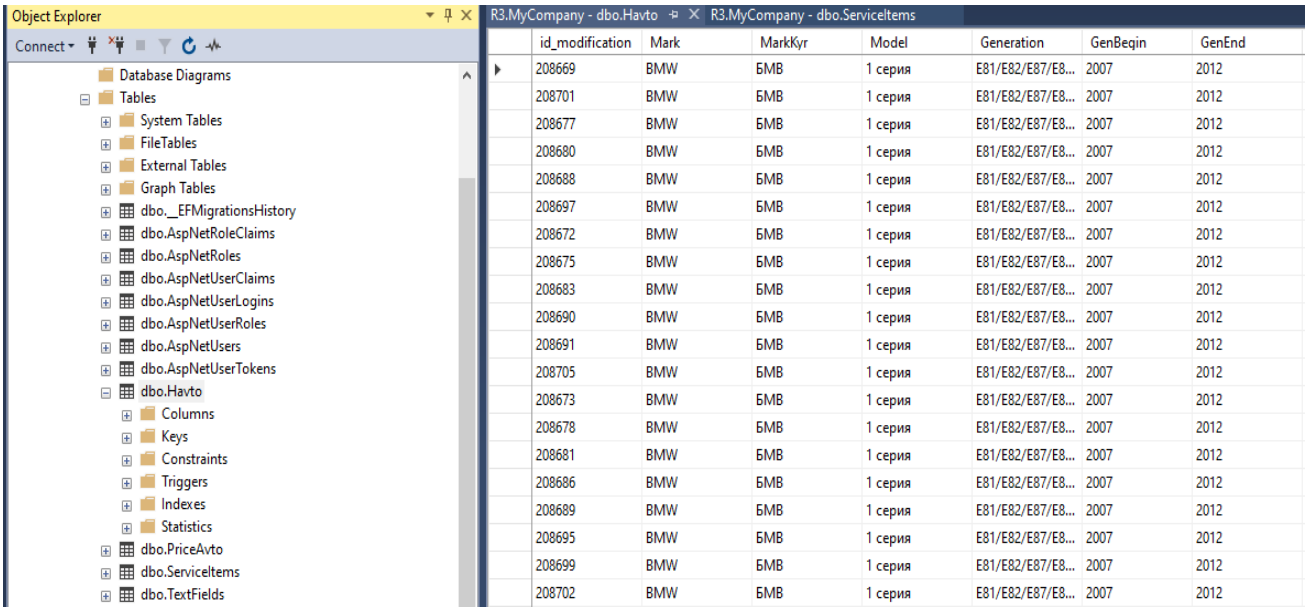


Рисунок 3.19 - База даних в середовищі MS SQL Server Management

В даному розділі здійснено програмну реалізацію та детальне проектування модулів та реалізація бази даних. Детально описано реалізацію модулів системи з застосуванням сучасних технологій проектування та програмування. Описано процедуру розгортання та встановлення системи.

4 НАЛАГОДЖЕННЯ ТА ТЕСТУВАННЯ СИСТЕМИ

4.1 Вибір та обґрунтування методів тестування системи

Тестування програмного забезпечення - це процес виконання його на деякому наборі даних, для яких заздалегідь відомий результат, а також правило поведінки цього програмного забезпечення.

Тестування проводиться вже з програмою, яка працює без помилок. Метою тестування є отримання результатів по конкретних даних, а також контроль якості програми, і переконатися в правильності роботи програмного забезпечення.

Тестування повинне включати перевірку всіх гілок програми і має включати мінімальний набір прикладів. При тестуванні не можна передбачити різні ситуації, проте потрібно домогтися певних результатів.

Для тестування застосовувався ми застосуємо метод ручного контролю. Ручний контроль використовується на ранніх стадіях розробки системи, так як він забезпечує виявлення 40 - 75% помилок. В якості вихідних даних для такого контролю виступають технічне завдання, специфікації, структурні і функціональні схеми, схеми окремих компонентів, а для більш пізніх етапів - алгоритми і тексти програм, а також тестові набори. З методів ручного контролю був обраний метод перевірки за столом. Цей метод не вимагає наявності групи фахівців. Перевірка вихідного коду виконується однією людиною, який читає текст програми, перевіряє його на наявність можливих помилок за списком розповсюджених помилок.

З функціональних методів був обраний метод граничних значень. Граничні значення - це значення на границях класів еквівалентності вхідних значень або біля них. Аналіз показує, що в цих місцях різко збільшується можливість виявлення помилок. Аналіз граничних значень, якщо він застосований правильно, є одним з найбільш корисних методів проектування тестів. Однак слід пам'ятати, що граничні значення можуть бути ледь вловимі і визначення їх пов'язано з великими труднощами, що є недоліком цього методу.

Після написання програмного забезпечення проводилося його тестування. В ході проведення тестових випробувань нашої довідково-інформаційної системи, було виявлено кілька невідповідностей і ситуацій, при яких програма не функціонує належним чином. Результати тестування представлені в таблиці 4.1.

Таблиці 4.1 - Результати тестування

Дата	Тестування проводив	Метод тестування	Назва тесту	Опис тесту	Результат
14.05.21	Проектувальник	ручний	Робота системи	Спроба запуску головної форми з модулями	успіх
14.05.21	Проектувальник	функціональний	Перевірка з'єднання	Спроба отримання інформації в ПЗ від сервера БД	Помилка. Невірно вказано кодування
15.05.21	Проектувальник	функціональний	Перевірка з'єднання	Спроба отримання інформації в ПЗ від сервера БД	успіх
17.05.21	Проектувальник	ручний	авторизація	Спроба авторизації з логіном / паролем користувача	успіх
19.05.21	Проектувальник	функціональний	некоректний пароль	Спроба авторизації з логіном користувача і неправильним паролем	успіх
20.05.21	Проектувальник	ручний	неправильна адреса	Спроба при налаштуванні розсилки вказати некоректний формат електронної адреси	Помилка. Форма прийняла електронну адресу
22.05.21	Проектувальник	ручний	неправильна адреса	Спроба при налаштуванні розсилки вказати некоректний формат електронної адреси	успіх

На тесті «Перевірка з'єднання» була виявлена проектувальником помилка. Зокрема, файлі конфігурації налаштування підключення до бази даних було невірно вказано кодування WIN-1251. Для виправлення цієї помилки в конфігураційному файлі було вказана вірне UTF-8 кодування.

На тесті «Неправильна адреса» виникла помилка при спробі під час налаштування розсилки вказати некоректний формат електронної адреси. Форма прийняла адреса для обробки. Для виправлення помилки в поле адреси була додана маска виду «ім'я@домен.домен».

На тесті «Створення PDF-файлу» виникла помилка при спробі створення PDF-файлу. Замість зрозумілого тексту, що читається був виведений набір ієрогліфів. Причиною послужили не вказаних параметри кодування в початковому тексті створення звіту, в результаті програма застосувала параметри, що задаються за умовчанням. Для виправлення помилки були вказані параметри необхідної кодування.

4.2 Валідація та верифікація системи

Запуск проекту системою Microsoft Visual Studio після збірки та перевірки на наявність виключених ситуацій (Рисунок 4.1)

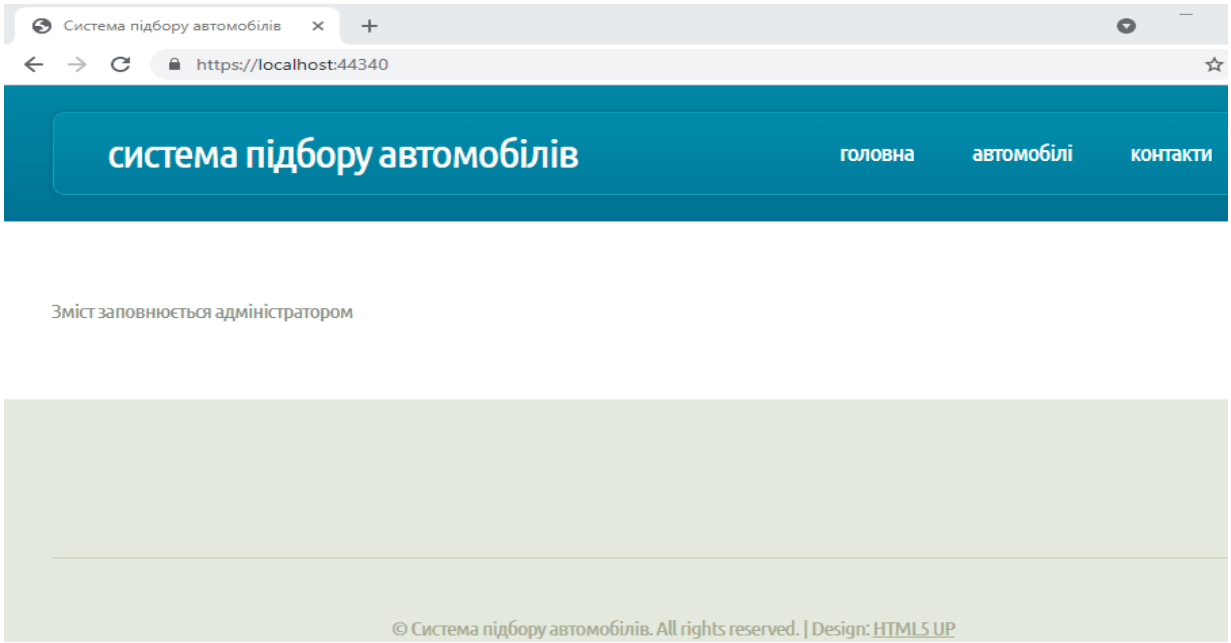


Рисунок 4.1 – Головна сторінка системи

Вхід в особистий кабінет сайту – введення невірно введених даних паролю користувача (рисунок 4.2)

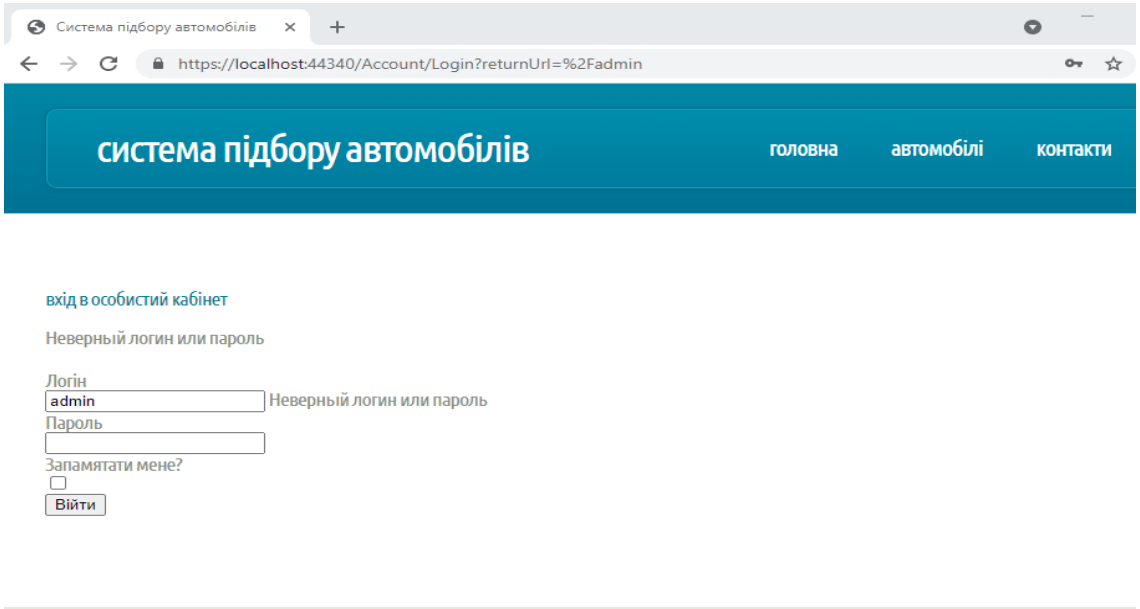


Рисунок 4.2 – Введення некоректних даних користувача

Успішний вхід в панель адміністрування сайту, виконаний в ролі адміністратора сайту (Рисунок 4.3)

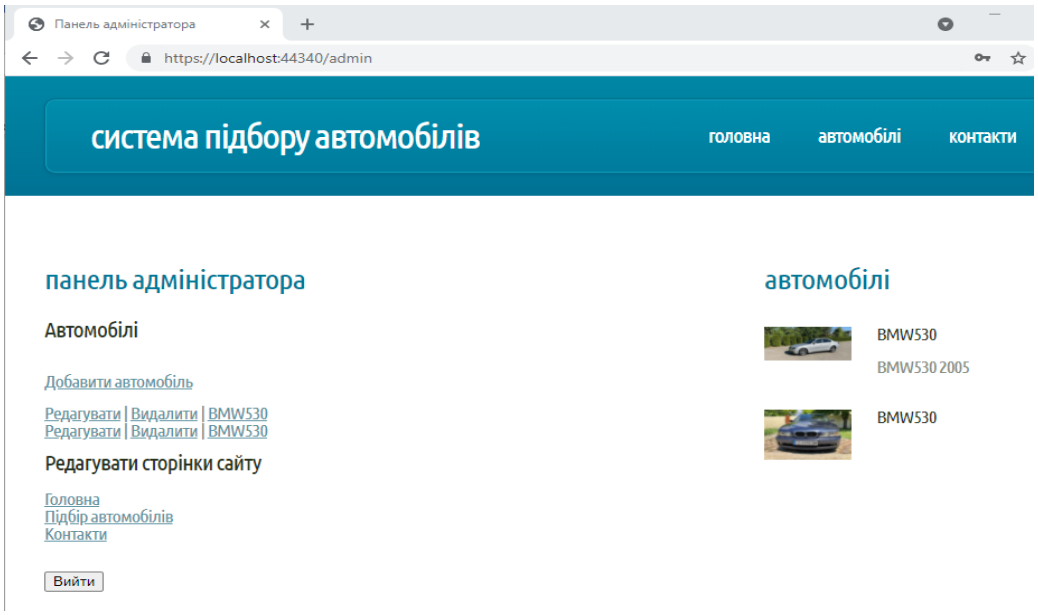


Рисунок 4.3 – Панель адміністрування сайту

Додавання характеристик про автомобіль користувачем наведено на рисунку 4.4.

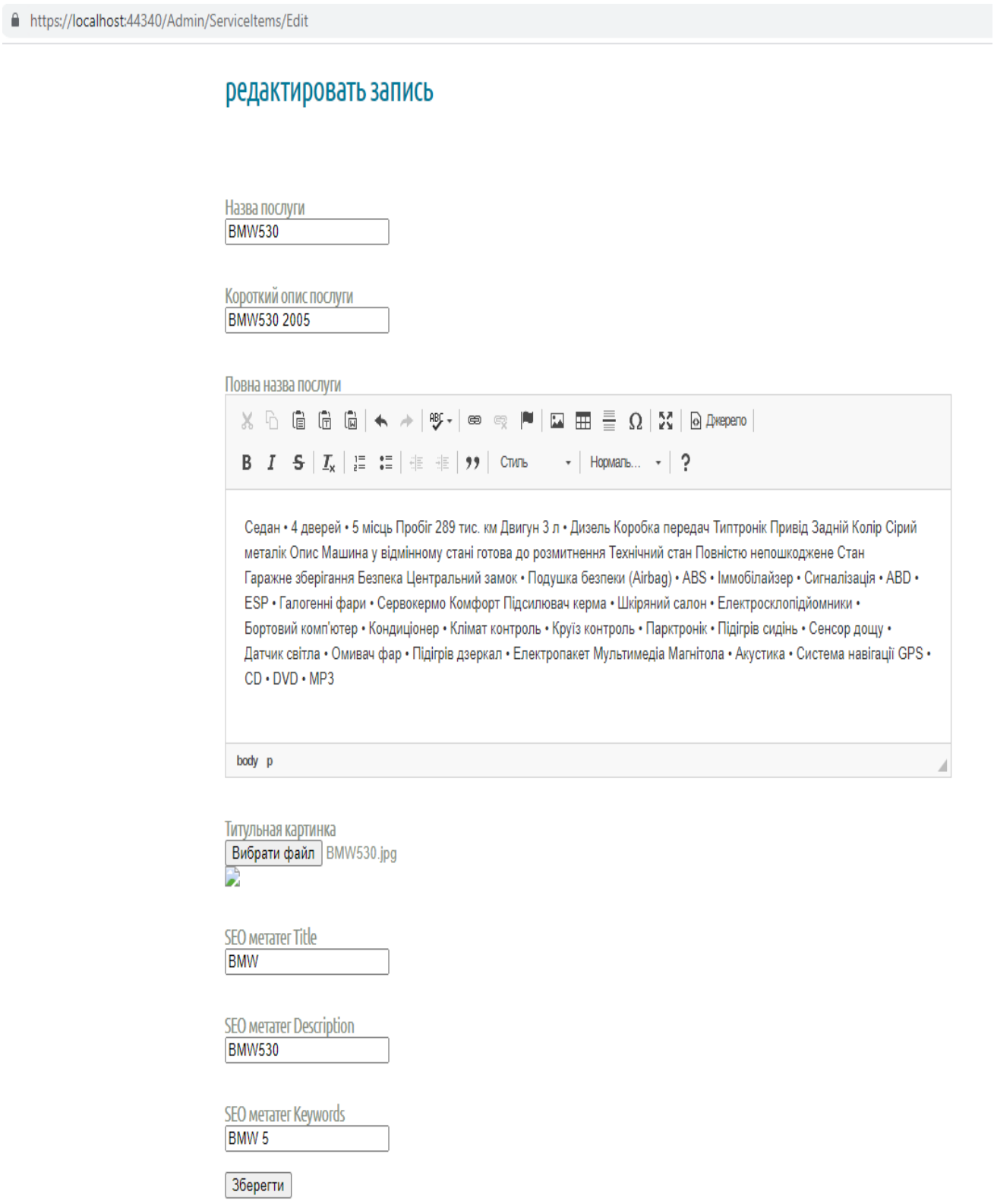


Рисунок 4.4 – Додавання запису про дані автомобіля

Редагування даних та характеристик автомобіля з та вставкою фото наведено на рисунку 4.5.

металік Опис Машина у відмінному стані готова до розмитнення Технічний стан Повністю непошкоджене Стан
Гаражне зберігання Безпека Центральний замок • Подушка безпеки (Airbag) • ABS • Іммобілайзер • Сигналізація • ABD • ESP • Галогенні фари • Сервокермо Комфорт Підсилювач керма • Шкіряний салон • Електросклопідйомники • Бортовий комп'ютер • Кондиціонер • Клімат контроль • Круїз контроль • Парктронік • Підігрів сидінь • Сенсор дощу • Датчик світла • Омивач фар • Підігрів дзеркал • Електропакет Мультимедіа Магнітола • Акустика • Система навігації GPS • CD • DVD • MP3

Титульная картинка

Вибрати файл Файл не вибрано

SEO метатег Title

BMW

SEO метатег Description

BMW530

SEO метатег Keywords

BMW 5

Зберегти

Рисунок 4.5 – Редагування даних автомобіля

Редагування даних «Контакти» наведено на рисунку 4.6, а сторінка контакти відповідно на рисунку 4.7

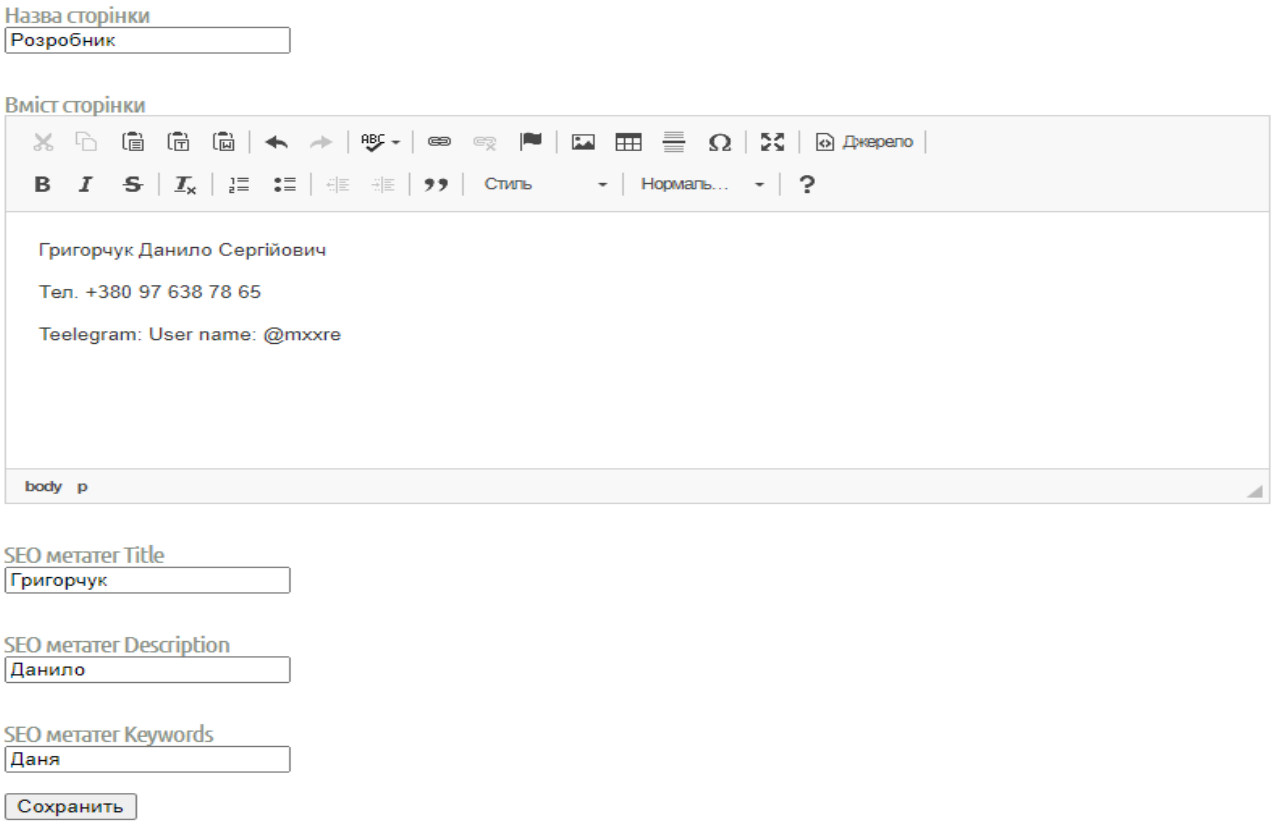


Рисунок 4.6 - Редагування даних «Контакти»

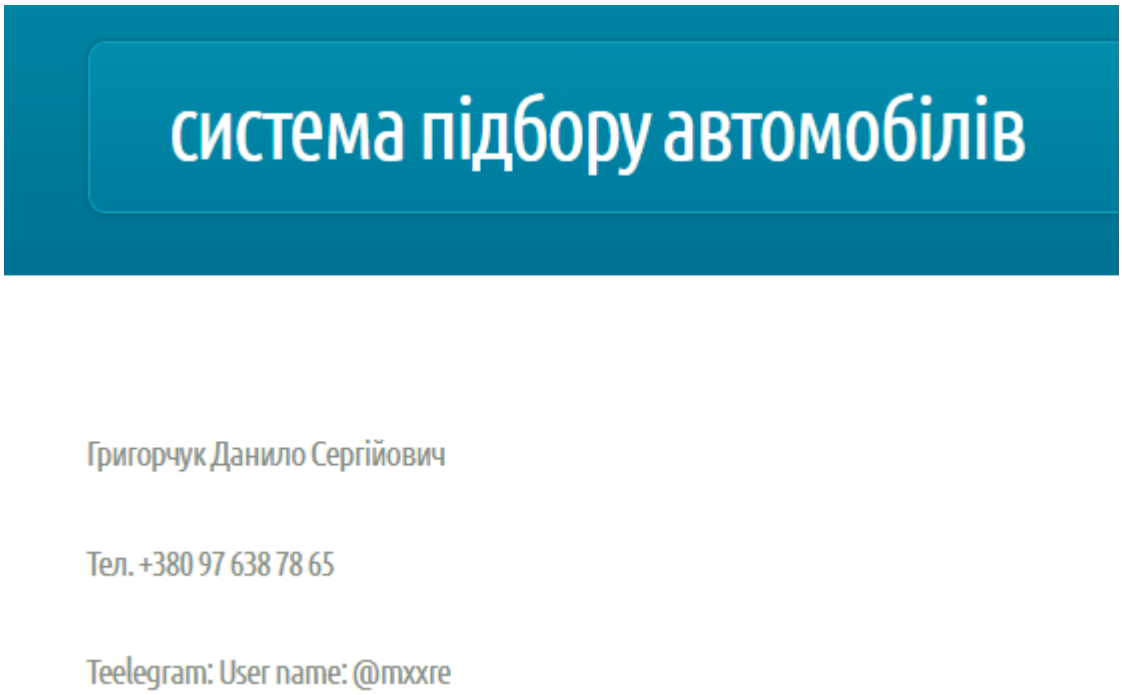


Рисунок 4.7 – Сторінка «Контакти»

4.3 Аналіз результатів тестування системи

Аналіз результатів тестування системи дипломного проекту полягає у підборі певних критеріїв запроектованої стратегії рекомендацій, які б надали найкращі результати пошуку вподобань автомобілів, щодо прогнозу побажань користувача. При чому, результати рекомендацій порівнюються шляхом зіставлення різних варіантів кінцевих рекомендацій та визначення рекомендації, що в повній мірі враховує побажання користувача.

Вибір критеріїв відбувається шляхом законів функціонування нашої системи вибору автомобілів, уподобань усередненого користувача. Переваги та вподобання такого користувача встановлюється при цьому експертом, які він надає певним критеріям під час аналізу та пошуку.

Значення n, що приймається рівним кількості елементів, з яких складається «ковзаюче вікно» відрізнятися в залежності від певної кількості рекомендованих користувачам автомобілів.

У рекомендаційній системі для підбору автомобілів згідно вподобань користувача використовуються наступні критерії важливості:

- марка авто – 0.3;
- рік випуску – 0.15;
- ціна – 0.20;
- тип авто– 0.025;
- тип двигуна– 0.03;
- об’єм двигуна -0.02;
- комплектація – 0.05;
- наявність пошкоджень 0.025;
- витрата пального – 0.025;
- колір – 0.01;
- наявність запчастин – 0.02.

Відповідні дані параметрів, які застосовні для розрахунку близькості за формулою 2.6, для кожного критерію відповідно, наведенні кількісною шкалою (таблиця 4.2).

Таблиця 4.2 – Значення параметрів для розрахунку оцінки вподобань

Назва критерію (одиниця вимірювання)	Назви параметрів		
	v	*w*	*s*
Марка авто	1	5	0,5
Рік випуску	0	1	0,5
Ціна, грн	4% (відмінність від ціни)	15% (відмінність від ціни)	0,5
Тип авто	0	2	0,5
Тип двигуна	6%	18%	0,5
Об’єм двигуна	5%	12%	0,5
Комплектація	4%	12%	0,5
Наявність пошкоджень	0	1	0,5
Витрата пального	5%	15%	0,5

В розділі здійснено налагодження та тестування системи. Зокрема здійснено вибір та обґрунтовано методи тестування системи. Проведено аналіз результатів тестування системи.

ВИСНОВКИ

В ході проведеного аналізу предметної області та сучасного стану досліджуваної проблеми, нами були розглянуто основні підходи до проектування та створення рекомендаційних систем на веб-платформах

У якості предметної сфери була обрана предметна область купівлі, продажу та пошуку автомобілів за вподобаннями користувача. Тому у якості об’єкту дослідження були обрані системи, що спеціалізуються на пошуку автомобілів. Проведено їх аналіз та виділено основні можливості та принципи функціонування. В ході роботи було вирішено проблему проектування та конструювання інформаційного забезпечення такої систем та здійснено проектування довідково-інформаційної системи, яка спеціалізується на пошуку автомобілів за вподобаннями користувача з використанням та реалізацією та розробленого алгоритму по створенню рекомендацій.

Зокрема, при проектуванні системи, в першому розділі було проведено досліджено предметну область та здійснено її аналіз. Розкрито поняття рекомендаційних систем. Здійснено постановку завдання та описано математичну модель. Також у ході аналізу предметної області було розроблено технічне завдання

В другому розділі здійснено проектування архітектури та структури системи. Запропоновано вибір технології проектування, зокрема у якості архітектурного шаблону сервісу була обрана MVC модель. Здійснено аналіз та вибір СКБД та здійснено проектування логічної структури бази даних.

В третьому розділі здійснено програмну реалізацію та детальне проектування модулів та реалізація бази даних. Детально описано реалізацію модулів системи з застосуванням сучасних технологій проектування та програмування. Описано процедуру розгортання та встановлення системи.

У четвертому розділі В розділі здійснено налагодження та тестування системи. Зокрема здійснено вибір та обґрунтовано методи тестування системи. Проведено аналіз результатів тестування системи.

Результатом проектування стала програмна реалізація рекомендаційної довідково-інформаційної системи вибору автомобілів у вигляді веб-сайту

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Бедратюк Л. П. Дипломний проект : методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення» / Л. П. Бедратюк, Г. І. Радельчук, Ю. В. Форкун, О. М. Яшина. – Хмельницький: ХНУ, 2020. – 77 с.
2. Спирли Э. Корпоративные хранилища данных. Планирование, разработка и реализация. У т. 1 / Є. Спирли - Издательство: Вильямс Т. 3.: Корпоративные хранилища данных. Планирование, разработка и реализация - 2001. – 400 с.
3. Брауде Э. Технология разработки программного обеспечения / Э. Брауде – СПб.: Питер, 2004. - 655 с.
4. Мартин Р. «Принципы, паттерны и методики гибкой разработки на языке C#» / Р.Мартин, М. Мартин, - СПб.: Символ-Плюс, 2011 - 768с.
5. Форкун Ю.В. Архітектура та проектування програмного забезпечення: методичні вказівки до курсового проектування для студентів напряму підготовки "Програмна інженерія" [текст]/ Ю.В. Форкун - Хмельницький: ХНУ, 2015. - 55 с.
6. Ших К. Эра Facebook. Как использовать возможности социальных сетей для развития вашего бизнеса / Клара Ших. – М.: Манн, Иванов и Фербер, 2011. – 304 с.
7. Нейгел К. C# 5.0 и платформа .NET 4.5 для профессионалов/ К. Нейгел - М.: «Диалектика», 2013. - 1440 с.
8. Гохберг Г. С. Информационньїе технологии : учебник / Г. С. Гохберг. А. В. Зафиевский, А. А. Короткий. - М, 2007. – 206 с.
9. Саак А. 3. Информационньїе технологии управлення : [учебник для вузов по специальности "Государственное и муниципальное управление"] / А. 3. Саак, Е. В. Пахомов, В. Н. Тюшняков СПб.: Питер, 2008. - 318 с.

10. Исаев Г. Н. Информационньїе системи в зкономике : [учебник для вузов по специальностям "Финансьі и кредит", "Бухгалтерский учет, анализ и аудит"] / Г. Н. Исаев. -М., 2008. - 462 с

11. Автоматизация управлення предприятием / [В. В. Баронов, Г. Н. Калянов, Ю. И. Попов и др.]. - М., 2000. - 239 с.

12. Калянов Г. Н. Моделирование, анализ, реорганизация и автоматизация бизнес-процессов: учебное пособие для вузов по специальности 080801 "Прикладная информатика (по областям)" и другим зкономическим специальностям / Г. Н. Калянов. - М., 2006. – 238 с.

13. Тельнов Ю. Ф. Реинжиниринг бизнес-процессов. Компонентная методология / Ю. Ф. Тельнов. - М., 2005. – 318 с.

14. Гаджинский А. М. Логистика : учебник [для вузов по направленню "Зкономика"] / А. М. Гаджинский. М., 2011. - 481 с.

15. Благодатских В. А. Стандартизацій разработки программньїх средств : учебное пособие по специальности "Прикладная информатика (в зкономике)" / В. А. Благодатских, В. А. Волнин, К. Ф. Поскакалов ; под ред. О. С. Разумова. - М., 2006. – 283 с.

16. Идеальная архитектура. Ведущие специалистьі о красоте программньїх архитектур. - Пер. с англ. - СПб.: Символ-Плюс, 2010. - 528 с., ил.

17. Фаулер М. UML. Основи, 3-є издание. - Пер. с англ. - СПб: Символ-Плюс, 2004. - 192 с..

18. NET Fundamentals [Electronic resource] / .NET Documentation. – Available from: https://docs.microsoft.com/en-us/dotnet/fundamentals

19. Introduction to ASP.NET Core [Electronic resource] / Microsoft Docs. – Available from: Зм. Арк № докум. Підпис Дата ДПІПЗ.170101.01.01.ПЗ Арк. 92 https://docs.microsoft.com/en-us/aspnet/core/introduction-to aspnetcore?view=aspnetcore-5.0

ДОДАТОК А

(обов'язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Вступ

Робота виконується в рамках проекту розробки довідкової інформаційної системи «Інтелектуальна рекомендаційна система вибору автомобілів»

1. Підстава для розробки

Підставою для розробки є «Завдання на дипломний проект», затверджене завідувачем кафедри інженерії програмного забезпечення.

Найменування розробки: Інтелектуальна рекомендаційна система вибору автомобілів.

2. Призначення розробки

Довідково-інформаційна система «Інтелектуальна рекомендаційна система вибору автомобілів» призначена для здійсненя вибору аавтомобілв на сайті та допомога у підборі автомобілів користувачам

Користувачами системи є користувачі які хочуть підібрати автомобіль для покупки чи оренди та адміністрування системи

Функціональне призначення передбачає: реєстрація користувачів у системі, перегляд та пошук автомобілів за відповідними критеріями, фільтрація автомобілів за різними критеріями; додавання/видалення оголошень за відповідними рубриками; можливість напису відгуків, надання користувачу рекомендацій згідно його вподобань.

Програмне забезпечення має працюватиме сервері та клієнті під управлінням будь-якої операційної системи.

3. Вимоги до програми

3.1 Вимоги до функціональних характеристик

Програмне забезпечення інтелектуальної рекомендаційна система вибору автомобілів повинне виконувати функції, подані нижче.

– реєстрація користувачів у системі;
– перегляд та пошук автомобілів за відповідними критеріями;
– фільтрація автомобілів за різними критеріями;
– додавання/видалення автомобілів за відповідними характеристиками;
– можливість напису відгуків;
– надання користувачу рекомендацій згідно його вподобань.

3.1. Вимоги до надійності

Програмне забезпечення повинно виконувати наступні вимоги:

– обробка помилок;
– виконання функціональних вимог;
– виведення результату на екран.

3.2. Умови експлуатації

В ході розробки програмного забезпечення повинні бути підготовлені: текст та опис програми, програма та керівництво користувача.

3.3. Вимоги до складу та параметрів технічних засобів

– платформа: Windows 10, 8, 8.1, 7;
– розрядність: x64 (64-bit);

– процесор: Pentium 4 з SSE2;
– відеоадаптер: nVidia, Intel, AMD / ATI;
– жорсткий диск: 10 Gb;
– оперативна пам'ять: 1 GB;
– контролер: миша, клавіатура, мікрофон;
– аудіокарта: будь-яка;
– інтернет: стабільне з'єднання;
– роздільна здатність екрану: SVGA 800x600.

3.4. Вимоги до інформаційної та програмної сумісності

Для розробки програмного забезпечення будуть використовуватися такі технології:

– MVC;
– Entity Framework;
– Microsoft SQL Server;
– Microsoft Visual Studio.

3.5. Спеціальні вимоги

Програма повинна мати зручний дизайн інтерфейсу для поліпшення сприйняття користувачем.

4. Вимоги до програмної документації

Програмна документація повинна включати такі документи наступний набір документів:

– текст програми з коментарями та поясненнями;
– відомості про функціонування програми;
– технічне завдання;
– інструкція користувача;
– керівництво програмісту.

5. Стадії та етапи розробки

Стадії та етапи розробки програмного забезпечення для забезпечення інтелектуальної рекомендаційна система вибору автомобілів показані у таблиці 1.4.

Таблиця 1.4 – Стадії та етапи розробки проекту

Стадія розробки	Етапи робіт	Зміст робіт
Технічне завдання 02.01.21 – 31.01.21	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання
Ескізний проект 01.02.21 – 14.02.21	Розробка ескізного проекту	Попередня розробка структури вхідних і вихідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної

		структури системи, що буде розроблюватися
Технічний проект 15.02.21 – 28.02.21	Розробка технічного проекту	Уточнення структури вхідних і вихідних даних; розробка докладного алгоритму; розробка структури програми; остаточне визначення конфігурації технічних засобів
Робочий проект 01.03.21 – 10.04.21	Розробка програмного забезпечення	Реалізація програмного забезпечення; відлагодження; проведення попереднього тестування
Розробка програмної документації 11.04.21 – 20.04.21	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи 21.04.21 – 30.04.21	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і розгортання програмного забезпечення

6. Порядок контролю та приймання

Контроль здійснюється кінцевими користувачами системи, підключеними на етапі тестування програмного забезпечення. Прийом комплексу здійснюється після налаштування програмного забезпечення та окремих його компонентів для нормального функціонування.

ДОДАТОК Б

(обов’язковий)

ДАТАЛОГІЧНА МОДЕЛЬ

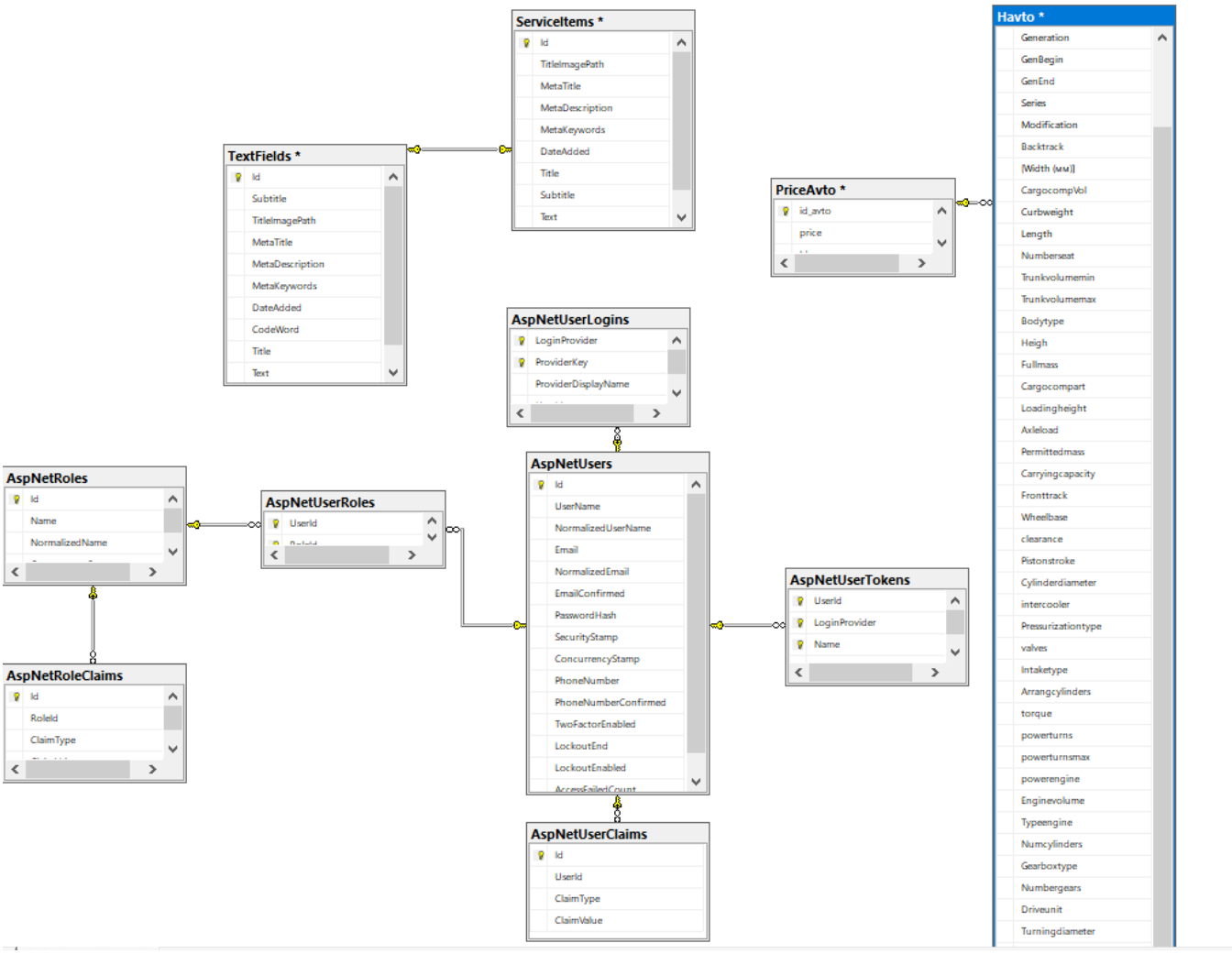


Рисунок Б.1 – Даталогічна модель бази даних

ДОДАТОК Б

(обов'язковий)

ФРАГМЕНТИ КОДУ ПРОГРАМНОЇ СИСТЕМИ

Код файлу startup.sh

```
namespace MyCompany
{
   public class Startup
   {
      public IConfiguration Configuration { get; }
      public Startup(IConfiguration configuration) => Configuration = configuration;

      public void ConfigureServices(IServiceCollection services)
      {
         //пздключаем конфзг sз appsetting.json
         Configuration.Bind("Project", new Config());

         //подключаем нужный функционал приложения в качестве сервисов
         services.AddTransient<ITextFieldsRepository, EFTextFieldsRepository>();
         services.AddTransient<IServiceItemsRepository,
EFServiceItemsRepository>();
         services.AddTransient<DataManager>();

         //підключаем контекст БД
         services.AddDbContext<AppDbContext>(x                                    =>
x.UseSqlServer(Config.ConnectionString));
```

```
        //identity система
        services.AddIdentity<IdentityUser, IdentityRole>(opts =>
        {
            opts.User.RequireUniqueEmail = true;
            opts.Password.RequiredLength = 6;
            opts.Password.RequireNonAlphanumeric = false;
            opts.Password.RequireLowercase = false;
            opts.Password.RequireUppercase = false;
            opts.Password.RequireDigit = false;

}).AddEntityFrameworkStores<AppDbContext>().AddDefaultTokenProviders();

        //налаштовуємо authentication cookie
        services.ConfigureApplicationCookie(options =>
        {
            options.Cookie.Name = "myCompanyAuth";
            options.Cookie.HttpOnly = true;
            options.LoginPath = "/account/login";
            options.AccessDeniedPath = "/account/accessdenied";
            options.SlidingExpiration = true;
        });

        //налаштовуємо політику авторизації для Admin area
        services.AddAuthorization(x =>
        {
            x.AddPolicy("AdminArea", policy => { policy.RequireRole("admin"); });
        });

        //добавляем сервисы для контроллеров и представлений (MVC)
```

```
        services.AddControllersWithViews(x =>
            {
                x.Conventions.Add(new AdminAreaAuthorization("Admin",
"AdminArea"));
            })
            //выставляем совместимость с asp.net core 3.0

.SetCompatibilityVersion(CompatibilityVersion.Version_3_0).AddSessionStateTemp
DataProvider();
    }

    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        //!!! порядок реєстрації middleware

        if (env.IsDevelopment())
            app.UseDeveloperExceptionPage();

        //підключаємо підримку  в додатку (css, js і т.д.)
        app.UseStaticFiles();

        // підключаємо систему маршрутизації
        app.UseRouting();

        //подключаємо аутентифікацію и авторизацію
        app.UseCookiePolicy();
        app.UseAuthentication();
        app.UseAuthorization();
```

```
            //реєструємо потрібні нам маршрути (ендпоінти)
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllerRoute("admin",
"{area:exists}/{controller=Home}/{action=Index}/{id?}");
                endpoints.MapControllerRoute("default",
"{controller=Home}/{action=Index}/{id?}");
            });
        }
    }
}
```

Файл Program.cs

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace MyCompany
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
```

```
                webBuilder.UseStartup<Startup>();
            });
    }
}
```

Файл appsettings.json

```
{
  "Project": {
    "ConnectionString": "Data Source=R3; Database=MyCompany; Persist Security Info=false; User ID='sa'; Password='sa'; MultipleActiveResultSets=True; Trusted_Connection=False;",
    "CompanyName": "Система підбору автомобілів",
    "CompanyPhone": "+38 (097) 638-78-65",
    "CompanyPhoneShort": "+380976387865",
    "CompanyEmail": "Mr.LonelyW1zard@gmail.com"
  }
}
```

Файл bundleconfig.json

```
[
  {
    "outputFileName": "wwwroot/js/scripts.js",
    "inputFiles": [
      "wwwroot/js/jquery.min.js",
      "wwwroot/js/browser.min.js",
      "wwwroot/js/breakpoints.min.js",
      "wwwroot/js/util.js",
      "wwwroot/js/main.js"
    ]
```

```
  }
]
```

Файл compilerconfig.json

```
[
  {
    "outputFile": "wwwroot/css/main.css",
    "inputFile": "wwwroot/sass/main.scss"
  }
]
```

Файл HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using MyCompany.Domain;

namespace MyCompany.Areas.Admin.Controllers
{
    [Area("Admin")]
    public class HomeController : Controller
    {
        private readonly DataManager dataManager;

        public HomeController(DataManager dataManager)
        {
            this.dataManager = dataManager;
        }

        public IActionResult Index()
        {
            return View(dataManager.ServiceItems.GetServiceItems());
```

```
        }
    }
}
```

Файл TextFieldsController.cs

```
using Microsoft.AspNetCore.Mvc;
using MyCompany.Domain;
using MyCompany.Domain.Entities;
using MyCompany.Service;

namespace MyCompany.Areas.Admin.Controllers
{
    [Area("Admin")]
    public class TextFieldsController : Controller
    {
        private readonly DataManager dataManager;
        public TextFieldsController(DataManager dataManager)
        {
            this.dataManager = dataManager;
        }

        public IActionResult Edit(string codeWord)
        {
            var entity = dataManager.TextFields.GetTextFieldByCodeWord(codeWord);
            return View(entity);
        }

        [HttpPost]
        public IActionResult Edit(TextField model)
```

```
        {
            if (ModelState.IsValid)
            {
                dataManager.TextFields.SaveTextField(model);
                return RedirectToAction(nameof(HomeController.Index),
nameof(HomeController).CutController());
            }
            return View(model);
        }
    }
}
```

Файл ServiceItemsController.cs

```
using System;
using System.IO;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using MyCompany.Domain;
using MyCompany.Domain.Entities;
using MyCompany.Service;

namespace MyCompany.Areas.Admin.Controllers
{
    [Area("Admin")]
    public class ServiceItemsController : Controller
    {
        private readonly DataManager dataManager;
        private readonly IWebHostEnvironment hostingEnvironment;
```

```
public ServiceItemsController(DataManager dataManager, IWebHostEnvironment hostingEnvironment)
{
    this.dataManager = dataManager;
    this.hostingEnvironment = hostingEnvironment;
}

public IActionResult Edit(Guid id)
{
    var entity = id == default ? new ServiceItem() : dataManager.ServiceItems.GetServiceItemById(id);
    return View(entity);
}
[HttpPost]
public IActionResult Edit(ServiceItem model, IFormFile titleImageFile)
{
    if (ModelState.IsValid)
    {
        if (titleImageFile != null)
        {
            model.TitleImagePath = titleImageFile.FileName;
            using (var stream = new FileStream(Path.Combine(hostingEnvironment.WebRootPath, "images/", titleImageFile.FileName), FileMode.Create))
            {
                titleImageFile.CopyTo(stream);
            }
        }
        dataManager.ServiceItems.SaveServiceItem(model);
```

```
            return RedirectToAction(nameof(HomeController.Index),
nameof(HomeController).CutController());
        }
        return View(model);
    }

    [HttpPost]
    public IActionResult Delete(Guid id)
    {
        dataManager.ServiceItems.DeleteServiceItem(id);
        return RedirectToAction(nameof(HomeController.Index),
nameof(HomeController).CutController());
    }
  }
}
```

Файл index.cshtml

```
@model IQueryable<ServiceItem>
@{
  string strTitle = "Панель адміністратора";
  ViewBag.Title = strTitle;
}

<div>
  <h2>@strTitle</h2>
  <div>
    <h3>Автомобілі</h3>
    <div class="div-box">
```

```
        <a asp-area="Admin" asp-controller="ServiceItems" asp-action="Edit" asp-route-id="">Добавити автомобіль</a>
    </div>
    @if (Model.Any())
    {
        <div>
            @foreach (ServiceItem entity in Model)
            {
                <div>
                    <a asp-area="Admin" asp-controller="ServiceItems" asp-action="Edit" asp-route-id="@entity.Id">Редагувати</a>
                    |
                    <form style="display: inline-block;" id="form-@entity.Id" asp-area="Admin" asp-controller="ServiceItems" asp-action="Delete" method="post">
                        <input type="hidden" name="id" value="@entity.Id">
                        <a href="#" onclick="document.getElementById('form-@entity.Id').submit();">Видалити</a>
                    </form>
                    |
                    <a asp-area="" asp-controller="Services" asp-action="Index" asp-route-id="@entity.Id">
                        @($"{entity.Title}")
                    </a>
                </div>
            }
        </div>
    }
</div>
<div class="div-box">
```

```
        <h3>Редагувати сторінки сайту</h3>
        <a asp-area="Admin" asp-controller="TextFields" asp-action="Edit" asp-route-codeWord="PageIndex">Головна</a>
        <a asp-area="Admin" asp-controller="TextFields" asp-action="Edit" asp-route-codeWord="PageServices">Підбір автомобілів</a>
        <a asp-area="Admin" asp-controller="TextFields" asp-action="Edit" asp-route-codeWord="PageContacts">Контакти</a>
    </div>
    <div class="div-box">
        <form asp-area="" asp-controller="Account" asp-action="Logout" method="post">
            <input type="submit" value="Вийти" />
        </form>
    </div>
</div>
```

Файл edit.cshtml

```
@model ServiceItem
@{
    string strTitle = "Редагувати запис";
    ViewBag.Title = strTitle;
}

<script src="~/js/ckeditor/ckeditor.js"></script>

<div>
    <h2>@strTitle</h2>
    <div>
```

```
<form asp-area="Admin" asp-controller="ServiceItems" asp-action="Edit" method="post" enctype="multipart/form-data">
    <input type="hidden" asp-for="Id" />
    <input type="hidden" asp-for="DateAdded" />
    <input type="hidden" asp-for="TitleImagePath" />

    <div asp-validation-summary="All"></div>
    <div class="div-box">
        <label asp-for="Title"></label>
        <input asp-for="Title" />
        <span asp-validation-for="Title"></span>
    </div>
    <div class="div-box">
        <label asp-for="Subtitle"></label>
        <input asp-for="Subtitle" />
        <span asp-validation-for="Subtitle"></span>
    </div>
    <div class="div-box">
        <label asp-for="Text"></label>
        <textarea asp-for="Text"></textarea>
        <span asp-validation-for="Text"></span>
    </div>
    <div class="div-box">
        <label asp-for="TitleImagePath"></label>
        <input type="file" name="titleImageFile" id="titleImageFile" />
        <div>
            <img class="img-block" src="~/images/@Model.TitleImagePath" />
        </div>
    </div>
```

```
            <div class="div-box">
                <label asp-for="MetaTitle"></label>
                <input asp-for="MetaTitle" />
                <span asp-validation-for="MetaTitle"></span>
            </div>
            <div class="div-box">
                <label asp-for="MetaDescription"></label>
                <input asp-for="MetaDescription" />
                <span asp-validation-for="MetaDescription"></span>
            </div>
            <div class="div-box">
                <label asp-for="MetaKeywords"></label>
                <input asp-for="MetaKeywords" />
                <span asp-validation-for="MetaKeywords"></span>
            </div>
            <input type="submit" value="Зберегти" />
        </form>
    </div>
</div>



<script>
    window.onload = function() {
        var newCKEdit = CKEDITOR.replace('@Html.IdFor(x=>x.Text)');
        newCKEdit.updateElement();
    }
</script>
```

Файл HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using MyCompany.Domain;

namespace MyCompany.Controllers
{
   public class HomeController : Controller
   {
      private readonly DataManager dataManager;

      public HomeController(DataManager dataManager)
      {
         this.dataManager = dataManager;
      }

      public IActionResult Index()
      {
         return
View(dataManager.TextFields.GetTextFieldByCodeWord("PageIndex"));
      }

      public IActionResult Contacts()
      {
         return
View(dataManager.TextFields.GetTextFieldByCodeWord("PageContacts"));
      }
   }
}
```

Файл AccountController.cs

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using MyCompany.Models;

namespace MyCompany.Controllers
{
   [Authorize]
   public class AccountController : Controller
   {
      private readonly UserManager<IdentityUser> userManager;
      private readonly SignInManager<IdentityUser> signInManager;
      public AccountController(UserManager<IdentityUser> userMgr,
SignInManager<IdentityUser> signinMgr)
      {
         userManager = userMgr;
         signInManager = signinMgr;
      }

      [AllowAnonymous]
      public IActionResult Login(string returnUrl)
      {
         ViewBag.returnUrl = returnUrl;
         return View(new LoginViewModel());
      }
      [HttpPost]
      [AllowAnonymous]
```

```
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl)
{
    if (ModelState.IsValid)
    {
        IdentityUser user = await userManager.FindByNameAsync(model.UserName);
        if (user != null)
        {
            await signInManager.SignOutAsync();
            Microsoft.AspNetCore.Identity.SignInResult result = await signInManager.PasswordSignInAsync(user, model.Password, model.RememberMe, false);
            if (result.Succeeded)
            {
                return Redirect(returnUrl ?? "/");
            }
        }
        ModelState.AddModelError(nameof(LoginViewModel.UserName), "Неверный логин или пароль");
    }
    return View(model);
}

[Authorize]
public async Task<IActionResult> Logout()
{
    await signInManager.SignOutAsync();
    return RedirectToAction("Index", "Home");
```

```
        }
    }
}
```

ServicesController.cs

```
using System;
using Microsoft.AspNetCore.Mvc;
using MyCompany.Domain;

namespace MyCompany.Controllers
{
    public class ServicesController : Controller
    {
        private readonly DataManager dataManager;

        public ServicesController(DataManager dataManager)
        {
            this.dataManager = dataManager;
        }

        public IActionResult Index(Guid id)
        {
            if (id != default)
            {
                return View("Show", dataManager.ServiceItems.GetServiceItemById(id));
            }

            ViewBag.TextField                                                    =
dataManager.TextFields.GetTextFieldByCodeWord("PageServices");
```

```
            return View(dataManager.ServiceItems.GetServiceItems());
        }
    }
}
```

Файл AppDbContext.cs

```
using System;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using MyCompany.Domain.Entities;

namespace MyCompany.Domain
{
    public class AppDbContext : IdentityDbContext<IdentityUser>
    {
        public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options) { }

        public DbSet<TextField> TextFields { get; set; }
        public DbSet<ServiceItem> ServiceItems { get; set; }

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

            modelBuilder.Entity<IdentityRole>().HasData(new IdentityRole
            {
                Id = "44546e06-8719-4ad8-b88a-f271ae9d6eab",
```

```
            Name = "admin",
            NormalizedName = "ADMIN"
        });

        modelBuilder.Entity<IdentityUser>().HasData(new IdentityUser
        {
            Id = "3b62472e-4f66-49fa-a20f-e7685b9565d8",
            UserName = "admin",
            NormalizedUserName = "ADMIN",
            Email = "my@email.com",
            NormalizedEmail = "MY@EMAIL.COM",
            EmailConfirmed = true,
            PasswordHash = new PasswordHasher<IdentityUser>().HashPassword(null,
"superpassword"),
            SecurityStamp = string.Empty
        });

        modelBuilder.Entity<IdentityUserRole<string>>().HasData(new
IdentityUserRole<string>
        {
            RoleId = "44546e06-8719-4ad8-b88a-f271ae9d6eab",
            UserId = "3b62472e-4f66-49fa-a20f-e7685b9565d8"
        });

        modelBuilder.Entity<TextField>().HasData(new TextField {
            Id = new Guid("63dc8fa6-07ae-4391-8916-e057f71239ce"),
            CodeWord = "PageIndex",
            Title = "Головна"
        });
```

```
            modelBuilder.Entity<TextField>().HasData(new TextField
            {
                Id = new Guid("70bf165a-700a-4156-91c0-e83fce0a277f"),
                CodeWord = "PageServices",
                Title = "Автомобілі"
            });
            modelBuilder.Entity<TextField>().HasData(new TextField
            {
                Id = new Guid("4aa76a4c-c59d-409a-84c1-06e6487a137a"),
                CodeWord = "PageContacts",
                Title = "Контакти"
            });
        }
    }
}
```

Файл DataManager.cs

```
using MyCompany.Domain.Repositories.Abstract;

namespace MyCompany.Domain
{
    public class DataManager
    {
        public ITextFieldsRepository TextFields { get; set; }
        public IServiceItemsRepository ServiceItems { get; set; }

        public DataManager(ITextFieldsRepository textFieldsRepository,
IServiceItemsRepository serviceItemsRepository)
        {
```

```
            TextFields = textFieldsRepository;
            ServiceItems = serviceItemsRepository;
        }
    }
}
```

Файл AppDbContextModelSnapshot.cs

```
using System;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;
using Microsoft.EntityFrameworkCore.Metadata;
using MyCompany.Domain;

namespace MyCompany.Migrations
{
    [DbContext(typeof(AppDbContext))]
    partial class AppDbContextModelSnapshot : ModelSnapshot
    {
        protected override void BuildModel(ModelBuilder modelBuilder)
        {
#pragma warning disable 612, 618
            modelBuilder
                .HasAnnotation("ProductVersion", "3.1.1")
                .HasAnnotation("Relational:MaxIdentifierLength", 128)
                .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

            modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRole", b =>
                {
```

```
b.Property<string>("Id")
    .HasColumnType("nvarchar(450)");

b.Property<string>("ConcurrencyStamp")
    .IsConcurrencyToken()
    .HasColumnType("nvarchar(max)");

b.Property<string>("Name")
    .HasColumnType("nvarchar(256)")
    .HasMaxLength(256);

b.Property<string>("NormalizedName")
    .HasColumnType("nvarchar(256)")
    .HasMaxLength(256);

b.HasKey("Id");

b.HasIndex("NormalizedName")
    .IsUnique()
    .HasName("RoleNameIndex")
    .HasFilter("[NormalizedName] IS NOT NULL");

b.ToTable("AspNetRoles");

b.HasData(
    new
    {
        Id = "44546e06-8719-4ad8-b88a-f271ae9d6eab",
        ConcurrencyStamp = "1c07fb23-ced7-48f9-bf0a-e0df233cd7a3",
```

```
                Name = "admin",
                NormalizedName = "ADMIN"
            });
        });


modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>",  b
=>
        {
            b.Property<int>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("int")
                .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);

            b.Property<string>("ClaimType")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("ClaimValue")
                .HasColumnType("nvarchar(max)");

            b.Property<string>("RoleId")
                .IsRequired()
                .HasColumnType("nvarchar(450)");

            b.HasKey("Id");

            b.HasIndex("RoleId");
```

```
        b.ToTable("AspNetRoleClaims");
    });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUser", b =>
    {
        b.Property<string>("Id")
            .HasColumnType("nvarchar(450)");

        b.Property<int>("AccessFailedCount")
            .HasColumnType("int");

        b.Property<string>("ConcurrencyStamp")
            .IsConcurrencyToken()
            .HasColumnType("nvarchar(max)");

        b.Property<string>("Email")
            .HasColumnType("nvarchar(256)")
            .HasMaxLength(256);

        b.Property<bool>("EmailConfirmed")
            .HasColumnType("bit");

        b.Property<bool>("LockoutEnabled")
            .HasColumnType("bit");

        b.Property<DateTimeOffset?>("LockoutEnd")
            .HasColumnType("datetimeoffset");
        b.Property<string>("NormalizedEmail")
            .HasColumnType("nvarchar(256)")
```

```
    .HasMaxLength(256);

b.Property<string>("NormalizedUserName")
    .HasColumnType("nvarchar(256)")
    .HasMaxLength(256);

b.Property<string>("PasswordHash")
    .HasColumnType("nvarchar(max)");

b.Property<string>("PhoneNumber")
    .HasColumnType("nvarchar(max)");

b.Property<bool>("PhoneNumberConfirmed")
    .HasColumnType("bit");

b.Property<string>("SecurityStamp")
    .HasColumnType("nvarchar(max)");

b.Property<bool>("TwoFactorEnabled")
    .HasColumnType("bit");

b.Property<string>("UserName")
    .HasColumnType("nvarchar(256)")
    .HasMaxLength(256);

b.HasKey("Id");

b.HasIndex("NormalizedEmail")
    .HasName("EmailIndex");
```

```
            b.HasIndex("NormalizedUserName")
                .IsUnique()
                .HasName("UserNameIndex")
                .HasFilter("[NormalizedUserName] IS NOT NULL");
            b.ToTable("AspNetUsers");
            b.HasData(
                new
                {
                    Id = "3b62472e-4f66-49fa-a20f-e7685b9565d8",
                    AccessFailedCount = 0,
                    ConcurrencyStamp = "091be5bf-9d58-40e6-98a8-e82859894571",
                    Email = "my@email.com",
                    EmailConfirmed = true,
                    LockoutEnabled = false,
                    NormalizedEmail = "MY@EMAIL.COM",
                    NormalizedUserName = "ADMIN",
                    PasswordHash =
"AQAAAAEAACcQAAAAECBA6mt3xGNgyLjwyRhhtI3PI2IBKsm00Y3bAJfjdVr
gT0++e45OV5Vh4SLTLPDHEQ==",
                    PhoneNumberConfirmed = false,
                    SecurityStamp = "",
                    TwoFactorEnabled = false,
                    UserName = "admin"
                });
        });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>", b
=>
```

```
            {
                b.Property<int>("Id")
                    .ValueGeneratedOnAdd()
                    .HasColumnType("int")
                    .HasAnnotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn);
                b.Property<string>("ClaimType")
                    .HasColumnType("nvarchar(max)");
                b.Property<string>("ClaimValue")
                    .HasColumnType("nvarchar(max)");
                b.Property<string>("UserId")
                    .IsRequired()
                    .HasColumnType("nvarchar(450)");
                b.HasKey("Id");
                b.HasIndex("UserId");
                b.ToTable("AspNetUserClaims");
            });

modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>",    b
=>
            {
                b.Property<string>("LoginProvider")
                    .HasColumnType("nvarchar(450)");
                b.Property<string>("ProviderKey")
                    .HasColumnType("nvarchar(450)");
                b.Property<string>("ProviderDisplayName")
                    .HasColumnType("nvarchar(max)");
                b.Property<string>("UserId")
                    .IsRequired()
```

```
                .HasColumnType("nvarchar(450)");
            b.HasKey("LoginProvider", "ProviderKey");
            b.HasIndex("UserId");
            b.ToTable("AspNetUserLogins");
        });
modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>",  b
=>
        {
            b.Property<string>("UserId")
                .HasColumnType("nvarchar(450)");
            b.Property<string>("RoleId")
                .HasColumnType("nvarchar(450)");
            b.HasKey("UserId", "RoleId");
           b.HasIndex("RoleId");
            b.ToTable("AspNetUserRoles");
            b.HasData(
                new
                {
                    UserId = "3b62472e-4f66-49fa-a20f-e7685b9565d8",
                    RoleId = "44546e06-8719-4ad8-b88a-f271ae9d6eab"
                });
        });


modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>",  b
=>
        {
            b.Property<string>("UserId")
                .HasColumnType("nvarchar(450)");
```

```
            b.Property<string>("LoginProvider")
                .HasColumnType("nvarchar(450)");
            b.Property<string>("Name")
                .HasColumnType("nvarchar(450)");
            b.Property<string>("Value")
                .HasColumnType("nvarchar(max)");
            b.HasKey("UserId", "LoginProvider", "Name");
            b.ToTable("AspNetUserTokens");
        });
    modelBuilder.Entity("MyCompany.Domain.Entities.ServiceItem", b =>
        {
            b.Property<Guid>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("uniqueidentifier");
            b.Property<DateTime>("DateAdded")
                .HasColumnType("datetime2");
            b.Property<string>("MetaDescription")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("MetaKeywords")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("MetaTitle")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("Subtitle")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("Text")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("Title")
                .IsRequired()
                .HasColumnType("nvarchar(max)");
```

```
            b.Property<string>("TitleImagePath")
                .HasColumnType("nvarchar(max)");
            b.HasKey("Id");
            b.ToTable("ServiceItems");
        });
    modelBuilder.Entity("MyCompany.Domain.Entities.TextField", b =>
        {
            b.Property<Guid>("Id")
                .ValueGeneratedOnAdd()
                .HasColumnType("uniqueidentifier");
            b.Property<string>("CodeWord")
                .IsRequired()
                .HasColumnType("nvarchar(max)");
            b.Property<DateTime>("DateAdded")
                .HasColumnType("datetime2");
            b.Property<string>("MetaDescription")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("MetaKeywords")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("MetaTitle")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("Subtitle")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("Text")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("Title")
                .HasColumnType("nvarchar(max)");
            b.Property<string>("TitleImagePath")
                .HasColumnType("nvarchar(max)");
```

```
            b.HasKey("Id");
            b.ToTable("TextFields");
            b.HasData(
               new
               {
                  Id = new Guid("63dc8fa6-07ae-4391-8916-e057f71239ce"),
                  CodeWord = "PageIndex",
                  DateAdded = new DateTime(2020, 2, 8, 6, 13, 50, 96,
DateTimeKind.Utc).AddTicks(9537),
                  Text = "Содержание заполняется администратором",
                  Title = "Главная"
               },
               new
               {
                  Id = new Guid("70bf165a-700a-4156-91c0-e83fce0a277f"),
                  CodeWord = "PageServices",
                  DateAdded = new DateTime(2020, 2, 8, 6, 13, 50, 97,
DateTimeKind.Utc).AddTicks(2218),
                  Text = "Содержание заполняется администратором",
                  Title = "Наши услуги"
               },
               new
               {
                  Id = new Guid("4aa76a4c-c59d-409a-84c1-06e6487a137a"),
                  CodeWord = "PageContacts",
                  DateAdded = new DateTime(2020, 2, 8, 6, 13, 50, 97,
DateTimeKind.Utc).AddTicks(2284),
                  Text = "Содержание заполняется администратором",
                  Title = "Контакты"
```

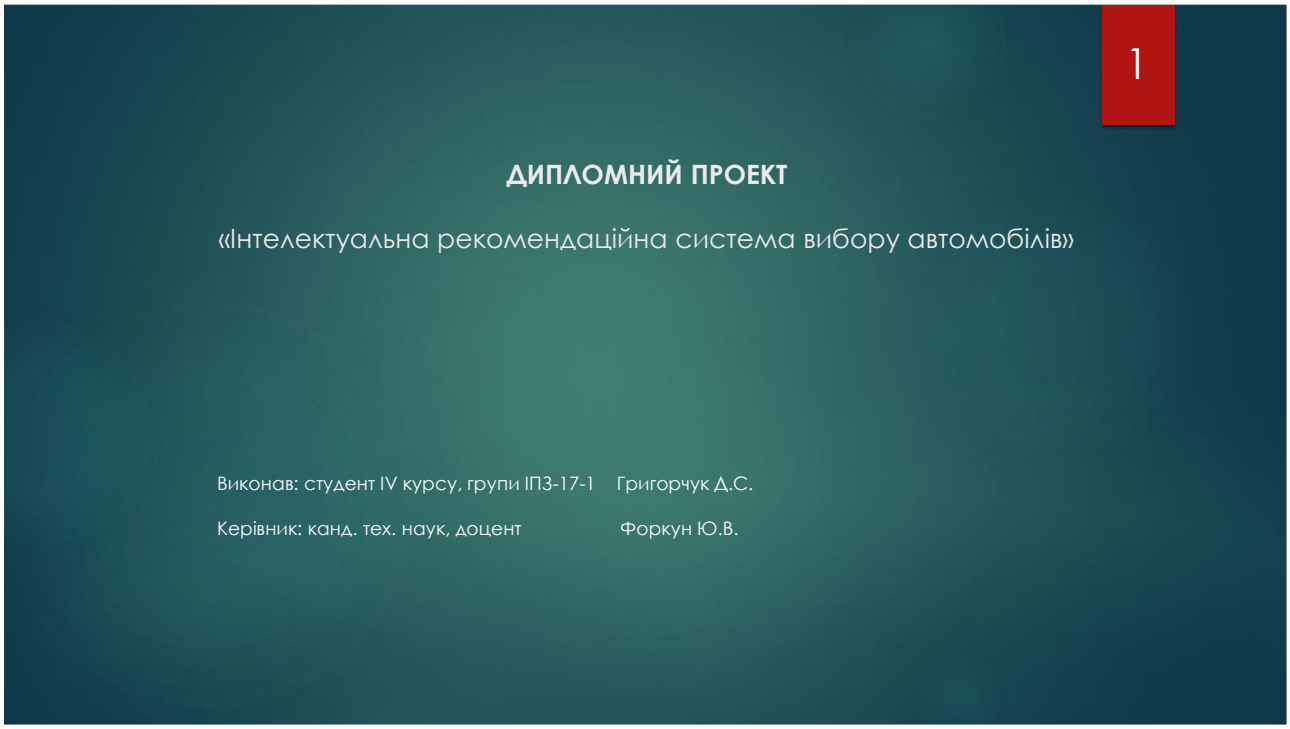
```
            });
        });
modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityRoleClaim<string>", b
=>
        {
            b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole", null)
                .WithMany()
                .HasForeignKey("RoleId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();
        });
modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserClaim<string>", b
=>
        {
            b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser", null)
                .WithMany()
                .HasForeignKey("UserId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();
        });
modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserLogin<string>", b
=>
        {
            b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser", null)
                .WithMany()
                .HasForeignKey("UserId")
                .OnDelete(DeleteBehavior.Cascade)
                .IsRequired();
        });
```

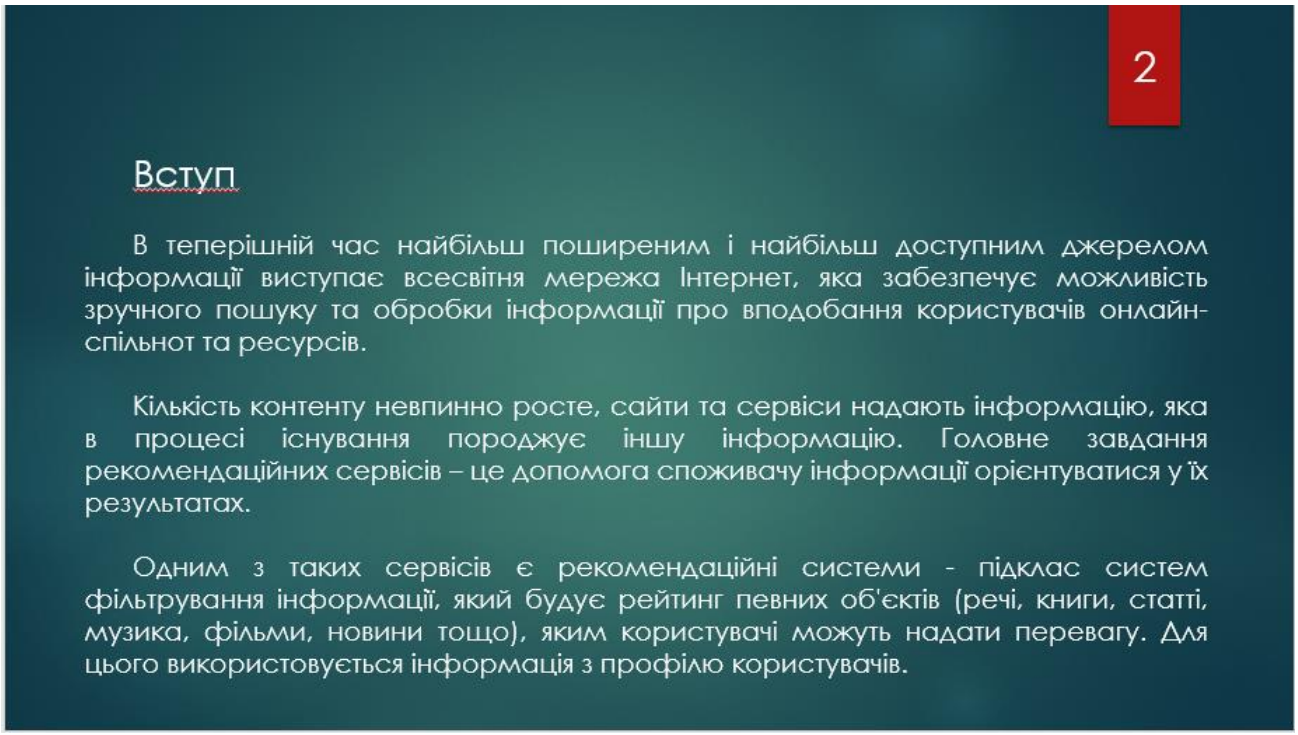
```
modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserRole<string>",      b
=>
            {
                b.HasOne("Microsoft.AspNetCore.Identity.IdentityRole", null)
                    .WithMany()
                    .HasForeignKey("RoleId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
                b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser", null)
                    .WithMany()
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });
modelBuilder.Entity("Microsoft.AspNetCore.Identity.IdentityUserToken<string>",   b
=>
            {
                b.HasOne("Microsoft.AspNetCore.Identity.IdentityUser", null)
                    .WithMany()
                    .HasForeignKey("UserId")
                    .OnDelete(DeleteBehavior.Cascade)
                    .IsRequired();
            });
#pragma warning restore 612, 618
        }
    }
}
```

ДОДАТОК Г

(обов'язковий)

ПРЕЗЕНТАЦІЙНІ СЛАЙДИ





3

Мета, об’єкт та предмет дослідження

Головною метою проекту є надання користувачу рекомендацій на основі його вподобань.

Об’єктом дослідження роботи виступають веб-сервіси, що спеціалізуються на продажі деякого продукту, який входить до предметної області.

Предметом дослідження виступають алгоритми та методи надання рекомендацій, які використовуються для прогнозування уподобань користувачів для надання їм пропозицій згідно їх уподобань.

4

Рекомендаційні системи – це підклас систем фільтрування інформації, який будує рейтинг певних об'єктів (речі, продукти, книги, статі, музика, фільми, новини тощо), яким користувачі можуть надати певну перевагу на основі власних вподобань, та рекомендацій і порад інших користувачів. За основу, в таких системах використовується інформація з профілю користувача, історій його покупок, переглядів тощо .

Рекомендаційні системи широко застосовуються в таких сферах, зокрема, як електронна комерція, соціальні мережі, веб-додатки тощо, де акцент робиться на різного роду інформації про дані користувача.

Рекомендаційні системи виступають альтернативою пошуковим алгоритмам, оскільки вони допомагають користувачам виявляти елементи, які в іншому випадку вони могли і не знайти. Для інтернет-магазинів це надзвичайно важлива функція та один з небагатьох способів якісно працювати. Рекомендації в даному випадку не виступають звичайною додатковою опцією платформи, а забезпечують зручність навігації користувачам по відповідному веб-ресурсу.

7

Задачу створення рекомендаційної системи будемо виконувати на прикладі вибору автомобілів. В даний час існує безліч систем, які здійснюють посередницькі послуги між продавцем і покупцем на ринку продажу та оренди автомобілів.

Сервіси з пошуку автомобілів надають своїм користувачам наступні загальні можливості:

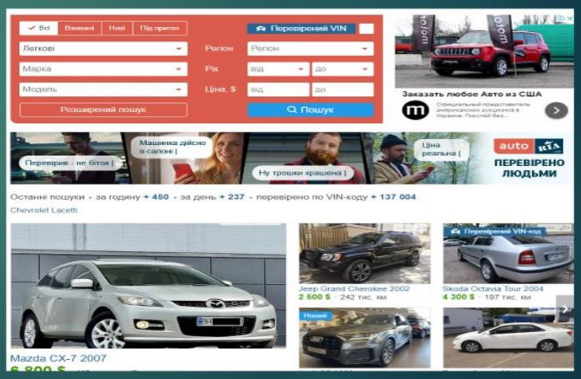
– перегляд та пошук оголошень за відповідними рубриками;

– фільтрація оголошень за різними критеріями;

– реєстрація користувачів у системі;

– додавання/видалення оголошень за відповідними рубриками;

– надання користувачу рекомендацій.



8

Архітектура системи

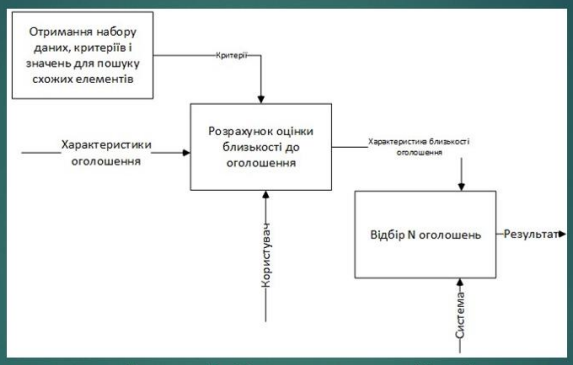
Виходячи з цілей системи, до неї висуваються наступні системні вимоги. Система має складатися з клієнтської частини, яка відповідає за отримання даних з сервера для подальшої візуалізації в зручному вигляді для користувача і відправки даних на сервер, а також сервера, який має приймати запити клієнта, обробляти їх, зберігати інформацію про користувацьку активність та прогнозувати клієнтські уподобання, шляхом виконання алгоритмів стратегії рекомендацій.

Для визначення функціональних вимог до системи була розроблена її функціональна модель з використанням стандарту IDEF0. У рамках цієї методології був створений набір діаграм, що описують систему, визначають її межі та основні процеси.

9

Концептуальна діаграма інформаційної системи

10



Діаграма функції «Рекомендаційної системи»

11

Представлена діаграма декомпозиції функції «Застосування рекомендаційної системи». Для її виконання спочатку необхідно отримати набір критеріїв, що описують заданий тип об'яви та значення їх відносної важливості, характеристику еталонного оголошення, та характеристики всіх оголошень для яких буде розраховуватися значення близькості.

Таким чином, проведене функціональне моделювання та аналіз отриманих діаграм дозволили уточнити функціональні вимоги до розроблюваної системи:

– реєстрація нових користувачів;
– перевірка введених даних;
– створення нових оголошень;
– редагування оголошень;
– написання відгуків у оголошенні;
– редагування профілю користувача;
– рейтинг у профілю;
– фільтрація оголошень за різними критеріями;
– надання рекомендацій;
– відображення помилок.

12

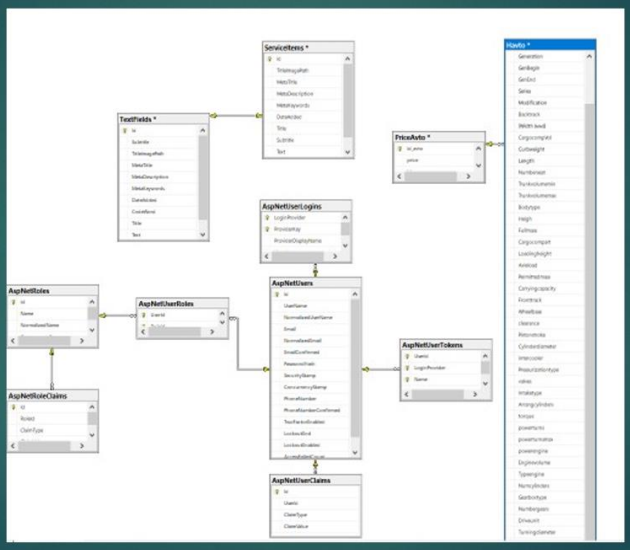
Вибір технологій і методів реалізації системи

В основі буде база даних зі своєю логікою, що може бути застосована до будь-якого додатку такого типу.

Для реалізації нашого проекту була обрана СКБД MS SQL Server 2019. MS SQL Server є однією з найбільш популярних систем керування базами даних в світі. Дана СКБД, як і MySQL підходить для різних проектів: від невеликих застосунків до великих високонавантажених проектів.

Для роботи з базою даних використовується фреймворк Entity Framework.

13



Даталогічна модель бази даних

14

Оскільки наш проект має клієнт-серверну архітектуру, то оперувати базою даних буде сервер в залежності від запитів клієнта. В результаті потрібно два застосунки, зокрема сам сервер та клієнтський застосунок, який буде з ним взаємодіяти.

Для розробки даного проекту я обрав мову програмування C#. C# має багато можливостей для зручної роботи з різними СКБД.

C# використовується, як для розробки програм для настільних ПК, так і для розробки веб-сервісів, сайтів та інших застосунків. Дана мова є універсальною. На ній можна розробляти, як і прості проекти, так і досить потужні додатки.

Серверну частину пропонується розробляти на основі технології ASP.NET Core на основі шаблону WEB API.

Клієнтську частину пропонується розробляти для платформи .NET Framework, як веб-додаток.

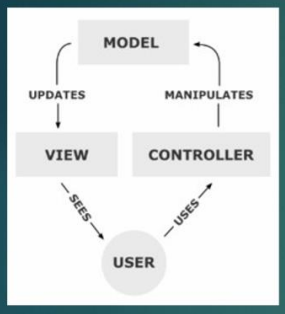
15

Для роботи з базою даних C# та інструментарій Microsoft Visual Studio мають багато можливостей для зручної роботи з різними типами СКБД.

В даному проекті, для підключення до бази даних та маніпуляцій з нею, доцільно використовувати засоби Microsoft Visual Studio для роботи з СКБД Microsoft SQL Server, а також інструментарій Microsoft SQL Server Management Studio.

Для створення веб-додатку використовується WEBAPI шаблон проекту ASP.NET Core. Вся основна логіка для роботи з базою даних знаходиться у веб-додатку. Для проекту також була розроблена адміністративна веб-сторінка для управління контентом проектованого програмного забезпечення.

16



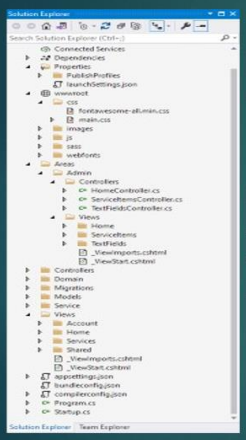
Для реалізації нашої системи ми будемо створювати веб-додаток використовуючи сучасні технології проектування, зокрема спроектуємо наш додаток використовуючи концепцію MVC (модель – представлення - контролер).

Цей шаблон передбачає поділ системи на три взаємопов'язані частини: модель даних, вигляд (інтерфейс користувача) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (вигляду) так, щоб зміни інтерфейсу користувача мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін інтерфейсу користувача.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми.

18

Реалізація модулів системи



Структура модулів сайту

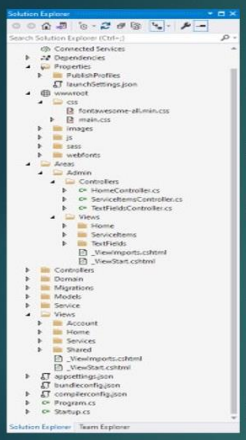
Першою папкою виступає папка wwwroot – стандартна папка ASP.NET Core, де будуть зберігатись статичні файли: стилі, скрипти, зображення тощо.

Areas – область для всього додатку ASP.NET Core , в якій визначені окремий набір контролерів та представлень. В цій директорії знаходиться папка Admin. В цій області будуть всі операції по адмініструванню сайту , яка в свою чергу містить папки Controllers (контролери) та Views (представлення).

Кореневі директорії:

- Controllers – контролери для основної частини додатку;
- Views/Shared – представлення для основної частини додатку;

19



Структура модулів сайту

- Layout.cshtml – майстер сторінка;
- Domain - для збереження доменної моделі, так звані бізнес-сутності (контекст підключення до бази даних, визначенні поля - тощо ;
- Models – моделі на рівні додатку, яка в свою чергу буде містити підпапку ViewComponents – компоненти представлення;
- Service - бізнес-логіка для обслуговування додатку, для роботи з функціоналом самого додатку.

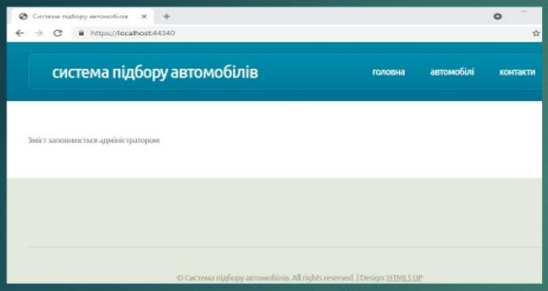
У файлі кореневої директорії розміщуємо файл appsettings.json – всі налаштування додатку та рядок підключення до бази даних.

В кореневій директорії також розміщуються файли:

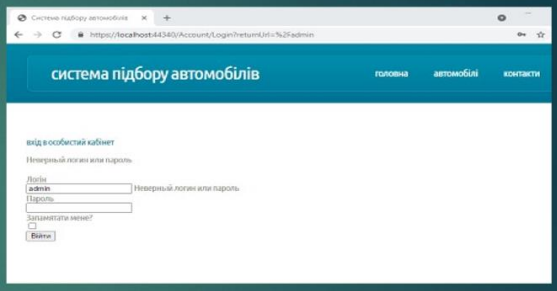
- Program.cs – головний файл, в якому визначено однойменний клас Program і з якого, по суті, починається виконання додатку;
- startup.cs – це також точка входу і він викликається після виконання файлів Program.cs на рівні додатків.

20

Інтерфейс програми



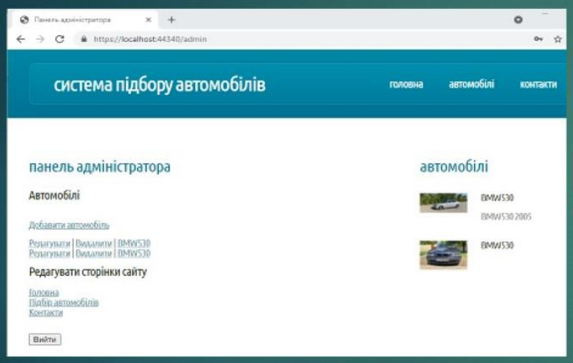
Головна сторінка



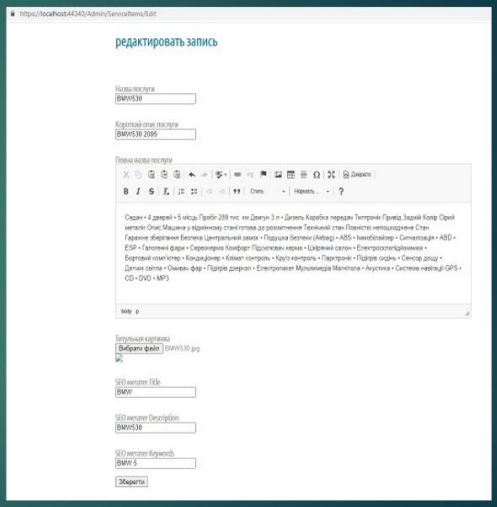
Введення некоректних даних користувача

21

Інтерфейс програми

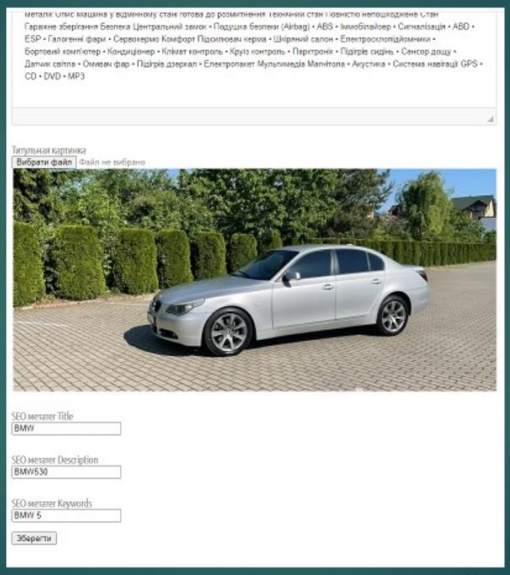


Панель адміністрування



Додавання даних про автомобіль

22



Редагування даних про автомобіль

23

Висновки

В ході проведеного аналізу предметної області та сучасного стану досліджуваної проблеми, нами були розглянуті основні підходи до проектування та створення рекомендаційних систем на веб-платформах.

У якості предметної сфери була обрана предметна область купівлі, продажу та пошуку автомобілів за вподобаннями користувача. Тому у якості об'єкту дослідження були обрані системи, що спеціалізуються на пошуку автомобілів. Проведено їх аналіз та виділено основні можливості та принципи функціонування.

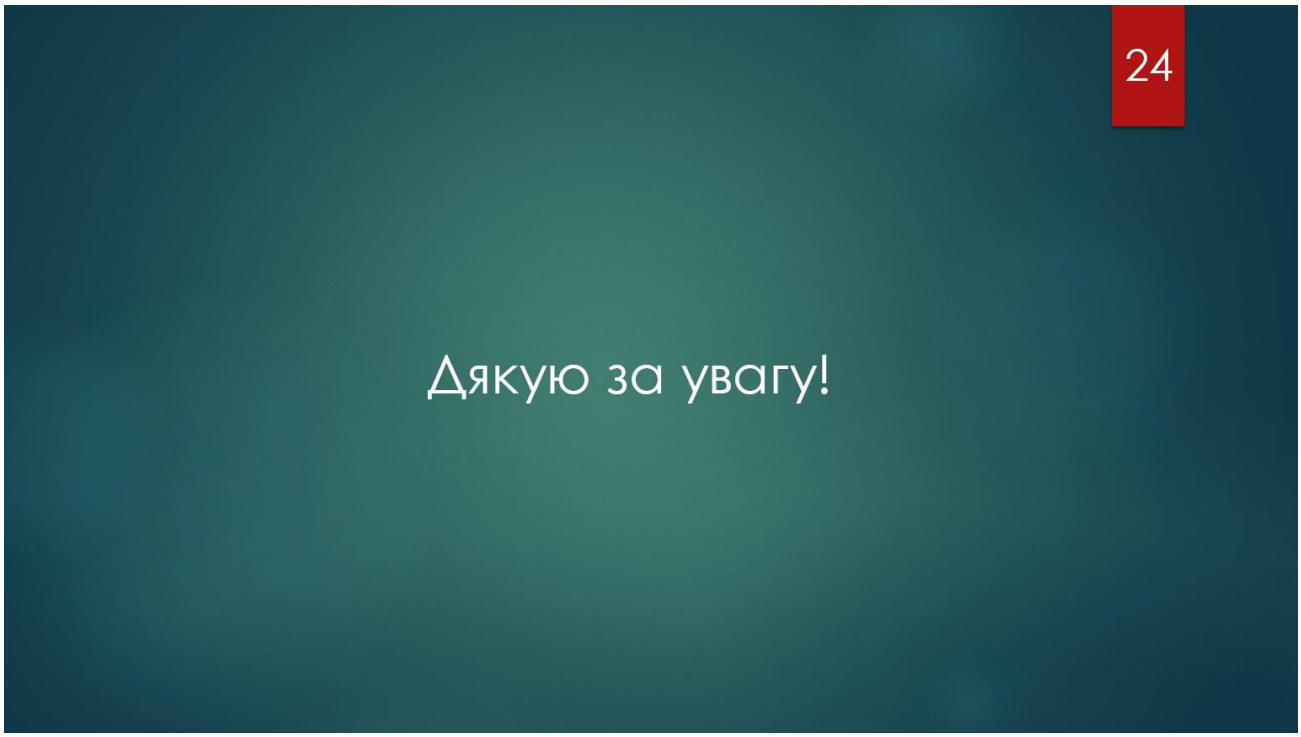
В ході роботи було вирішено проблему проектування та конструювання інформаційного забезпечення такої системи та здійснено проектування довідково-інформаційної системи, яка спеціалізується на пошуку автомобілів за вподобаннями користувача з використанням та реалізацією розробленого алгоритму по створенню рекомендацій.

Запропоновано вибір технології проектування. У якості архітектурного шаблону сервісу була обрана MVC модель.

Здійснено аналіз та вибір СКБД та здійснено проектування логічної структури бази даних.

Було проведено тестування системи та аналіз його результатів.

Результатом проектування стала програмна реалізація рекомендаційної довідково-інформаційної системи вибору автомобілів у вигляді веб-сайту.


24
Дякую за увагу!

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗ-17-1
Григорчука Д.С.
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему дипломного проекту освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»:
Інтелектуальна рекомендаційна система вибору автомобілів

(керівник дипломного проекту – Форкун Юрій Вікторович)
Прізвище, ім'я, по батькові

01.02.2021 р.
Дата

Підпис студента

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти

Григорчука Д. С.

Прізвище, ініціали

факультет ПКТС, 4 курс, група ІПЗ-17-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

16.06.2021

дата

підпис

16.06.2021 Григорчук.html

Wed Jun 16 08:26:03 EEST 2021, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 4.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 10%**

ID: 94117 Назва: Інтелектуальна рекомендаційна система вибору автомобілів Додано в БД: 2021-06-16 Автора: Д.С. Григорчук Керівники: Ю.В.Форкун Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	72434	671	11236 (16%)	134 (20%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1008308700

Дата перевірки:
16.06.2021 09:15:10 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
16.06.2021 09:16:34 EEST

ID користувача:
100005589

Назва документа: **ДП Григорчук без додатків**

Кількість сторінок: **79** Кількість слів: **11602** Кількість символів: **92877** Розмір файлу: **3.05 MB** ID файлу: **1008376591**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

16%
Схожість

Найбільша схожість: **4.98%** з джерелом з Бібліотеки (ID файлу: **1008265198**)

10.9% Джерела з Інтернету | 354 ..... Сторінка 81

6.69% Джерела з Бібліотеки | 91 ..... Сторінка 84

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи | 15

Підозріле форматування | 17 сторінок

КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Інтелектуальна рекомендаційна система вибору автомобілів»

Автор: Григорчук Данило Сергійович

Спеціальність: 121 – Інженерія програмного забезпечення

Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»

Науковий керівник: Гурман Іван Васильович, канд. тех. наук, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті дипломного проекту системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень в бланках (титульного аркушу, бланку завдання, в структурі підрозділів, ВСТУПУ);
2) В якості запозичень системою було зафіксовано стандартні конструкції та послідовності коду, а також посилання на використання бібліотек які є спільними для великої кількості мобільних додатків та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;
3) усі запозичення фрагментарні, або мають належним чином оформленні посилання.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності / схожості, складає 16% і адресується до 354 першоджерела, що з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь дипломного проекту.

Керівник _____ Ю.В. Форкун

Гарант ОП _____ Л.П. Бедратюк

Завідувач кафедри _____ Л.П. Бедратюк

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНИЙ ПРОЕКТ
рівня вищої освіти «Бакалавр»

Дипломник Григорчук Данило Сергійович

Тема «Інтелектуальна рекомендаційна система вибору автомобілів»

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломного проекту:

Кількість листів креслень _____ кількість сторінок записки 65

1.Короткий зміст пояснювальної записки та прийнятих рішень У бакалаврській роботі виконано аналіз рекомендаційна система вибору автомобілів. Розроблено алгоритми роботи системи. Спроектовано та реалізовано архітектуру додатку. Реалізовано програмний засіб для рекомендацій вибору автомобілів користувачам веб-сайтів.

2. Висновок про відповідність проекту поставленому завданню Дипломний проект рівня вищої освіти «бакалавр» в повній мірі відповідає поставленому завданню у теоретичній практичній реалізації.

3. Характеристика виконання кожного розділу проекту, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі обґрунтовується актуальність теми проекту, проводиться попередній аналіз досліджуваної проблеми, встановлюється мета та завдання на виконання проекту. У першому розділі в повній мірі досліджено рекомендаційні системи, проаналізоване наявне програмне забезпечення, визначені вимоги до проектування та реалізації системи. В другому розділі здійснено проектування архітектури системи та реалізацію бази даних.. В третьому розділі наведено процес реалізації системи та її компонентів .. В четвертому розділі проведений аналіз методів тестування з застосуванням певних підходів до тестування програмного забезпечення.

4. Позитивні сторони проекту Дипломний проект спроектований засобами за сучасними вимогами до створення такого програмного забезпечення, що дозволяє його підтримувати в актуальному стані та швидко здійснювати модернізацію відповідно до вимог сьогодення, а також використовувати на різних платформах

5. Негативні сторони проекту В реалізації програмного забезпечення доцільно було застосувати мультипарадигмове програмування.

6. Оцінка графічного оформлення та пояснювальної записки проекту Графічне оформлення виконано відповідно до теми дипломного проекту з дотриманням вимог стандартів. Рівень графічного оформлення достатній для ілюстрування всіх методів, алгоритмів та процесів, які подані в дипломному проекті.

7. Відгук про дипломний проект в цілому Загалом дипломний проект повноцінно описує вирішення поставленої задачі. Викладений матеріал у проекті послідовний та логічний. Всі розділи логічні та повноцінно описані, що дозволяє зрозуміти весь проект. Графічні матеріали наочно показують всі особливі моменти, що спрощує розуміння вирішення задачі. В цілому проект заслуговує позитивної оцінки.

8. Інші зауваження

9. Оцінка дипломного проекту Розглянувши всі позитивні та негативні сторони представленого дипломного проекту можна зробити висновок, що він заслуговує оцінки «добре» (3.75/C).

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи)

Говорущенко Тетяна Олександрівна, доктор технічних наук, професор, завідувач кафедри комп'ютерної інженерії та системного програмування Хмельницького національного університету

«2» червня 2021 р. _____
(підпис)