

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПОШУКУ МІСТ ДЛЯ
ВІДДАЛЕНОЇ РОБОТИ
REMOTE JOB CITY SEARCH SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-138
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Бірюк В.О.

(прізвище та ініціали)

Керівник Льовкін В.М.

(прізвище та ініціали)

Рецензент Скрупський С.Ю.

(прізвище та ініціали)

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

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Бірюк Владислави Олександрівни
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення пошуку міст для віддаленої роботи
Remote Job City Search Software

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

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Проєктування програмного забезпечення. 3. Розробка програми. 4. Експлуатація та тестування програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Льовкін В.М., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання « 4 » квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис)

Керівник проєкту (роботи)

(підпис)

Бірюк В.О.

(прізвище та ініціали)

Льовкін В.М.

(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 86 с., 26 рис., 3 табл., 3 дод., 12 джерел.

ВІДДАЛЕНА РОБОТА, ПОШУК МІСТ, ОЦІНЮВАННЯ, ВЕБЗАСТОСУНОК, ОБ'ЄКТНО-РЕЛЯЦІЙНЕ ВІДОБРАЖЕННЯ.

Об'єкт роботи – процес пошуку міст для віддаленої роботи. Предмет роботи – програмне забезпечення пошуку міст для віддаленої роботи. Мета роботи – розробити програмне забезпечення пошуку міст для віддаленої роботи для підвищення рівня інформованості працівників перед переїздом в нове місто для віддаленої роботи.

Матеріали, методи та технічні засоби: мова програмування C#, фреймворки ASP.NET Core і Entity Framework Core, система керування базами даних Microsoft SQL Server.

Результати. У результаті виконання роботи проведено повний цикл розробки, що охоплював аналіз предметної області, який призвів до прийняття рішень стосовно проєктування програми з вибором засобів і структуруванням програми і даних, які в свою чергу стали основою реалізації програми, що в свою чергу завершилось успішним проведенням тестуванням виконаної розробки.

Висновки. Розроблена як вебзастосунок програма дозволяє людям, які бажають віддалено працювати з інших міст, подорожуючи або переміщуючись з рідного міста, знайти місто для проживання і віддаленої роботи, що відповідає їхнім вимогам до умов роботи і проживання на основі досвіду інших працівників, які вже працювали віддалено з цього міста раніше.

Галузь використання. Віддалена робота в розрізі пошуку міст, з яких таку віддалену роботу реалізовувати.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 86 pages, 26 figures, 3 tables, 3 appendixes, 12 sources.

REMOTE WORK, CITY SEARCH, EVALUATION, WEB APPLICATION, OBJECT-RELATIONAL MAPPING.

Work object is a process of searching a city for remote work. Subject of the work is a software for searching a city for remote work. The aim of the work is to develop software for searching a city for remote work to raise awareness of employees before moving to a new city for remote work.

Materials, methods and technical means: C# programming language, ASP.NET Core and Entity Framework Core frameworks, Microsoft SQL Server database management system.

Results. As a result of the work, the full development cycle was carried out. The cycle started with analysis of the subject area, which led to the decisions regarding the design of the program with the choice of tools and structuring the software and data, which in turn became the basis of the program development with successful testing of the developed software as a result.

Conclusions. The software, developed as a web application, allows people who want to work remotely from other cities, travelling or moving from their hometown, to find a city to live and work remotely which meets their requirements for working and living conditions based on the experience of other employees who have worked remotely from the city before.

Field of use. Remote work in the context of searching cities from which such remote work could be carried out.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	7
Вступ	8
1 Аналіз предметної області	10
1.1 Особливості організації віддаленої роботи	10
1.2 Огляд існуючих програмних рішень пошуку віддаленої роботи	12
1.3 Висновки за розділом 1	15
2 Проєктування програмного забезпечення	16
2.1 Вибір вебфреймворку	16
2.2 Визначення функціональних вимог до програми	17
2.3 Проєктування бази даних	19
2.4 Визначення структури програми	24
2.5 Висновки за розділом 2	25
3 Розробка програми	26
3.1 Алгоритм функціонування програми	26
3.2 Опис реалізованих класів	27
3.3 Висновки за розділом 3	37
4 Експлуатація та тестування програми	38
4.1 Призначення програми	38
4.2 Умови виконання програми	38
4.3 Експлуатація програми	38
4.4 Тестування програми	43
4.5 Висновки за розділом 4	45
Висновки	46
Перелік джерел посилання	47
Додаток А Технічне завдання	49
Додаток Б Текст програми	53
Додаток В Слайди презентації	78

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

З першого кварталу 2020 року тенденції та очікування щодо віддаленої роботи змінилися, при цьому дуже часто роботодавці не були готові до того, що вся їхня робоча сила почне працювати вдома, проте завдяки технологіям та інноваціям це фактично добре працює для багатьох роботодавців, в результаті у всьому світі все більше роботодавців використовують гнучкий графік для своїх віддалених команд, що призводить до нових тенденцій віддаленої роботи та нових можливостей віддаленої роботи [1].

Дані дослідження Morning Consult, проведеного у травні 2021 року серед 1000 дорослих американців, показали, що майже 40 % респондентів подумали б про те, щоб залишити роботу, якщо б їхні роботодавці не запропонували гнучкі можливості віддаленої роботи, при цьому цей відсоток додатково зріс до 49% для міленіалів і покоління Z [2].

До всіх тих тенденцій у поточному році в Україні зокрема і як результат в Європі загалом додалась війна і вимушене переміщення великої кількості громадян України, що призвело в результаті до додаткового розширення простору віддаленої роботи.

Таким чином, можна виділити велику кількість факторів, до яких можна віднести характер сучасної економіки загалом, пандемію коронавірусу та війну в Україні, що призводять до додаткового розвитку віддаленої роботи. Але всі ці фактори, пошук віддаленої роботи як такої є тільки однією стороною проблеми. Ще однією стороною є подальше проживання в цьому місті зокрема та пошук міста, в якому така віддалена робота може відбуватися, зокрема. Пошук міста пов'язаний з тим, що людина тоді не обмежена і може переміщуватися по світу за власним бажанням і потребам, а також з тим, що людина вимушена переміщуватися і працювати віддалено з інших міст.

Для задоволення окресленої потреби потрібно розробити програмне забезпечення (ПЗ) пошуку міст для віддаленої роботи. Це дозволить підвищити рівень інформованості працівників перед переїздом в нове місто для віддаленої роботи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості організації віддаленої роботи

Віддалена робота – це роль, яку співробітник може виконувати поза типовим офісним середовищем: у віддаленій ролі люди зазвичай можуть виконувати більшість своїх завдань і проєктів за допомогою комп'ютера, вони також можуть спілкуватися з керівниками та іншими членами команди за допомогою телефонних дзвінків, відеодзвінків або систем обміну повідомленнями в чаті, це зменшує потребу співробітників фізично відвідувати офіс для виконання своїх завдань, і замість цього вони виконують їх вдома або у віддаленому офісі [3]-[4].

У США ринок робочої сили досяг максимуму в 164,6 мільйона людей, які працювали в лютому 2020 року, незадовго до того, як пандемія розгорнулася, при цьому статистика віддалених працівників показує, що понад 4,7 мільйона людей працюють віддалено принаймні половину часу, 44% компаній не дозволяють дистанційну роботу і лише 16% компаній наймають тільки віддалених працівників, у той же час згідно з дослідженням Owl labs у всьому світі 16% компаній повністю віддалені, це ж дослідження показало, що близько 62% працівників у віці від 22 до 65 років стверджують, що працюють віддалено принаймні час від часу, а 44% компаній не дозволяють віддалену роботу будь-якого виду [1].

У США кількість публікацій, у яких прямо згадується віддалена робота з січня 2020 року по березень 2022 року, зросла на 319 %, а кількість пошукових запитів зросла на 458 %, у той же час у Великобританії кількість оголошень про віддалену роботу зросла на 329 %, тоді як кількість пошуків різко зросла на 790 %, аналогічні дані продемонстрував LinkedIn про віддалені робочі місця – посади, явно позначені як віддалені, або ті, які містять пов'язані ключові слова, такі як «робота з дому», включаючи гібридні ролі – показали різке збільшення кількості заяв (у березні 2020 року на початку пандемії оплачувані віддалені роботи привернули в 1,8 раза

більше заявок, ніж оплачувані невіддалені роботи, у березні 2022 року цей показник був вищим у 2,6 рази) [2].

10 000 співробітників, опитаних Інститутом економіки Беккера Фрідмана в Чиказькому університеті, вказали, що вони вважають, що вони так само продуктивно працюють вдома, як під час роботи в офісі, при цьому насправді 30 % цих респондентів сказали дослідникам, що вони більш продуктивні та залучені до роботи, знаходячись вдома, до того ж у цьому ж дослідженні було підраховано, що час на дорогу на роботу за віддаленої роботи скоротився на 62,4 мільйона годин на день із загальною економією часу понад 9 мільярдів годин, починаючи з середини березня 2020 року до середини вересня 2020 року [1].

Звіт Owl labs у 2021 році показав, що 55% респондентів кажуть, що працюють більше годин віддалено, ніж у фізичному офісі, при цьому мова йшла не про вимушене збільшення, цей же звіт показав, що лише 36% людей вважають, що офіс найкраще підходить для індивідуальної роботи, окрім того 32% опитаних заявили, що покинуть роботу, якщо не зможуть продовжувати працювати віддалено [1].

Варто відзначити, що незважаючи на особливу популярність сфери інформаційних технологій у віддаленій роботі, це звичайно ж не єдиний напрямок. Зокрема до 10 найпопулярніших робіт початкового рівня належать:

- представник служби підтримки клієнтів;
- штатний бухгалтер;
- адміністративний помічник;
- координатор з підбору персоналу;
- бухгалтер;
- представник колцентру;
- фахівець з розрахунків;
- виконавчий помічник;
- спеціаліст з кредиторської заборгованості;

– координатор з маркетингу [5].

1.2 Огляд існуючих програмних рішень пошуку віддаленої роботи

Проведений огляд дозволив виділити два основні аналоги розробки, що виконується.

Першим аналогом є Digital Nomad World [6] (рис. 1.1).



Рисунок 1.1 – Вебресурс Digital Nomad World

Дане рішення дозволяє виконувати пошук міст, виконуючи сортування за заданим критерієм, але не дозволяючи обмежити значення за певними оцінками. Окрім того надається можливість переглянути повну інформацію про місто, де окрім оцінок транспортуванню, стану охорони

здоров'я, їжі, безпеки, якості життя, робочих місць, інтернету тощо, надається також інформація про ціни в місті.

Варто зазначити, що разом з інформацією про придатність міста для віддаленої роботи надається інформація про вакансії для роботи. Тобто базову інформацію про міста можна отримати, проте це тільки частина сервісу.

Другий аналог – Nomadlist [7] (рис. 1.2). Це сервіс, який надає більше інструментів для отримання інформації про міста для віддаленої роботи. Сервіс надає можливість виконати пошук міст, при цьому пошук має більшу кількість фільтрів.

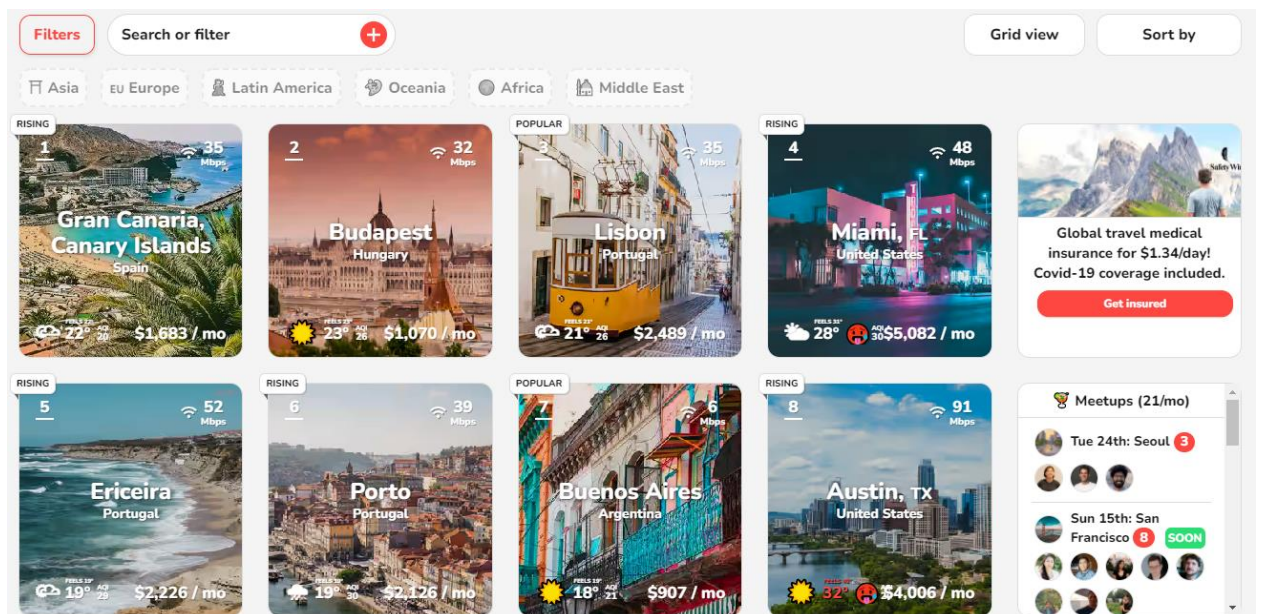


Рисунок 1.2 – Вебресурс Nomadlist

За кожним містом надається достатня широка кількість оцінок, інформація про працівників, які місто відвідали, ціни, фотографії, відгуки. Для авторизованих користувачів надається можливість підписування на інших користувачів, відправлення повідомлень користувачам. Це дозволяє стверджувати, що це ПЗ як інструмент пошуку міст для віддаленої роботи, є більш корисним.

Проте Nomadlist має і ряд недоліків, які варто владнати під час створення відповідного інструменту.

По-перше, слід вказати, що цей інструмент направлений на використання для так званих цифрових кочівників, а оскільки це зазвичай молоде покоління, то питанням облаштування життя сім'ї не надається достатньо уваги. А в цьому розрізі важливо те, на скільки легко влаштувати дитину в садочок, окрім того, на скільки легко орендувати житло та взагалі виконати прокат різних речей, адже сім'ї зазвичай при переїздах потребують прокату і дитячих іграшок, і інших речей. Все це вказує на необхідність значно ширшої кількості критеріїв та оцінок за ними для створення більш придатного інструменту в результаті.

По-друге, такому працівнику зазвичай потрібна інформація про конкретні місця, які вона може використовувати для роботи і життя. В Nomadlist надається інформація тільки про коворкінги. При цьому добре було б, щоб ці працівники на основі власного досвіду оцінювали конкретні місця. Можливість розширення таких місць, внесення конкретних коворкінгів, офісів, дитячих садочків, сервісів прокату є дуже важливим і корисним інструментом.

Також важливо мати можливість розгляду стану міста в динаміці, тобто розуміти на скільки місто активно змінюється і якщо це так, то привертати власну увагу саме до таких міст, адже це створює більше можливостей. Окрім того має бути виділено професії і пов'язано їх з відгуками, це дозволить додати специфіку досвіду до того відгуку, який створив користувач.

Саме на подолання цих недоліків і направлена розробка, яка виконується в роботі. Порівняння існуючих рішень та програми з дипломної роботи проведено в табл. 1.1.

Таблиця 1.1 – Порівняння аналогів

Критерій	Digital Nomad World	Nomadlist	Дипломна робота
Пошук міст для віддаленої роботи	+	+	+
Підписування на користувачів	—	+	+
Багатокритеріальне оцінювання	+	++	+++
Відправлення повідомлень автору відгуку	—	+	+
Рекомендування місць	—	—	+
Історичний розріз даних	—	—	+

1.3 Висновки за розділом 1

Розглянуто особливості організації віддаленої роботи, зокрема актуальність віддаленої роботи як такої в сучасному світі, особливості виконуваних робіт. Проаналізовано існуючі програмні рішення пошуку віддаленої роботи, включаючи Digital Nomad World та Nomadlist, виділено інструменти, які надаються ними, а також сформульовано їх недоліки. Визначено основні особливості програми, яка розробляється.

Завданнями дипломної роботи є такі:

- аналіз предметної області;
- визначення вимог до ПЗ пошуку міст для віддаленої роботи;
- проєктування ПЗ пошуку міст для віддаленої роботи;
- реалізація і налагодження ПЗ пошуку міст для віддаленої роботи.

2 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір вебфреймворку

Під час проєктування програми спершу було виконано вибір засобів для реалізації програми. Оскільки для реалізації потужних вебзастосунків на даний момент достатньо активними рішеннями є вебфреймворки, зокрема Django [8] і ASP.NET Core [9]-[10], то саме їх порівняння і виконувалось.

Під час порівняння (табл. 2.1) було виявлено, що ASP.NET Core, на відміну від Django, здатний організовувати багатопотокову роботу, що призводить до оптимізації роботи сервера, і є важливим для застосунків, в яких потенційно може виявлятися велике навантаження на сервер, має можливість потенційного майбутнього масштабування, що може бути важливим в предметній області, яка розглядається в роботі, окрім того направлений більше на середні і великі проєкти, а також має більшу поширеність.

Таблиця 2.1 – Вибір вебфреймворку

Критерій	Django	ASP.NET Core
Мова програмування	Python	C#
Кросплатформність	+	+
Велика підтримка на рівні бібліотек розширення функціональності	+	+
Багатопотокова робота	—	+
Масштабованість	+—	++
Поширеність	+—	++
Основний сегмент ринку	Малий бізнес	Середній

2.2 Визначення функціональних вимог до програми

Огляд і аналіз аналогів програми, що реалізовано в розділі 1, став основою визначення функціональних вимог до програми, які в результаті включають:

- внесення проживань у місті;
- редагування проживань у місті;
- перегляд власних проживань;
- перегляд проживань працівника у місті;
- перегляд проживань у місті;
- перегляд популярних віддалених робіт у місті;
- підписування на працівника;
- відписування під працівника;
- перегляд підписників працівника;
- перегляд підписок працівника;
- перегляд популярних міст за роботою;
- відправка повідомлення працівнику;
- перегляд отриманих повідомлень;
- рекомендування місця;
- перегляд рекомендованих місць у місті;
- перегляд власних рекомендацій;
- обчислення середніх оцінок міст;
- рейтингування міст;
- рейтингування міст за частиною світу;
- рейтингування міст у країні;
- рейтингування міст за критерієм;
- пошук міст;
- пошук місць;
- створення власного відгуку про місто;
- перегляд власних відгуків про міста;

- перегляд відгуків про місто;
- перегляд інформації про віддалену роботу в місті;
- перегляд динаміки зміни оцінок міста;
- перегляд динаміки відвідуваності міста;
- перегляд популярності професій віддаленої роботи;
- реєстрація працівників;
- авторизація працівників.

Для даних вимог було реалізовано їх моделювання для неавторизованого користувача (рис. 2.1) та працівника (рис. 2.2).

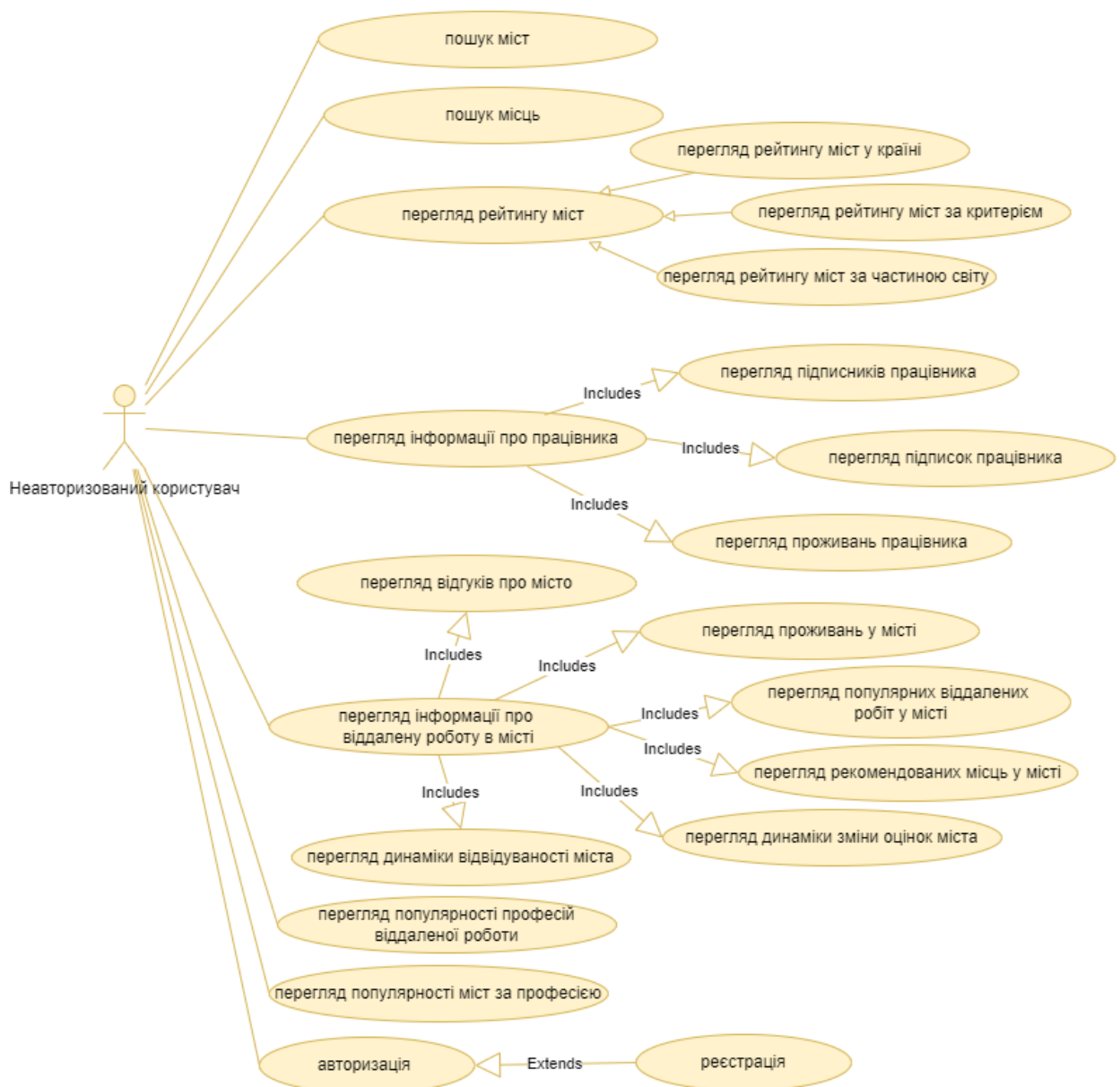


Рисунок 2.1 – Діаграма прецедентів (неавторизований користувач)

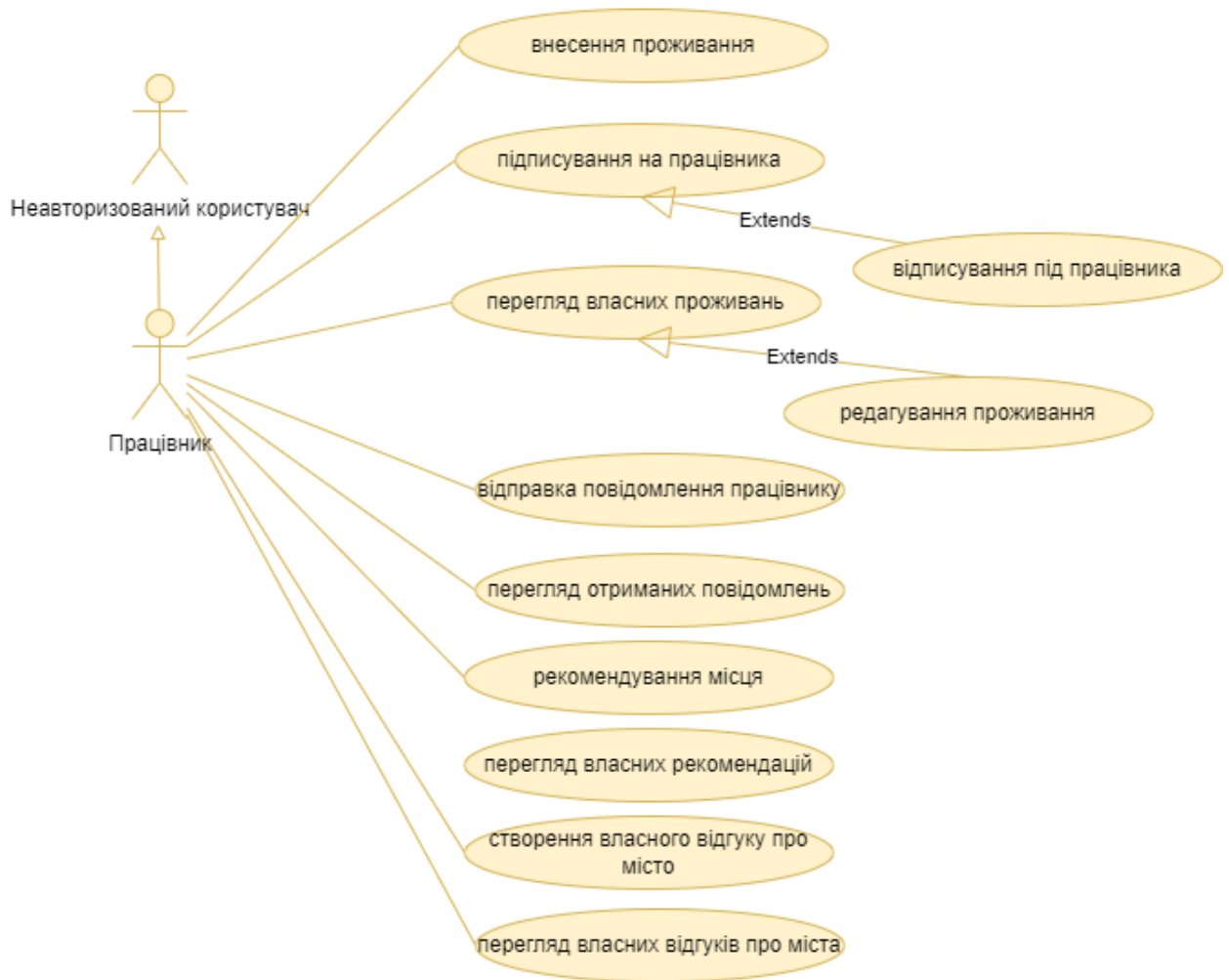


Рисунок 2.2 – Діаграма прецедентів (працівник)

2.3 Проєктування бази даних

Виходячи з функціональних вимог до програми, потенційної необхідності роботи великої кількості користувачів з програмою, а також різних сутностей, якими виражаються ці дані, роботу з даними логічним було реалізовувати через створення бази даних (БД). Потужною і зручною для платформи .NET є система керування БД Microsoft SQL Server [11]-[12], яку і було використано у створюваному вебзастосунку.

Однією з базових сутностей предметної області є працівник (сутність Worker), який за підсумками свого досвіду формує базу інформації, що використовується для накопичення статистики за містами з метою подальшого пошуку тих міст іншими працівниками, які найбільше підходять

для їх віддаленої роботи та умов, які вони для цього та для життя загалом потребують.

Основою визначення досвіду працівника є його проживання (сутність Living) в різних містах з метою віддаленої роботи. Проживання має зафіксувати, коли користувачем було отримано відповідний досвід, де саме це відбувалося та ким саме працюючи. Працівник може оцінювати конкретно це проживання або ні, проживання використовуються зокрема для збереження історії випадків, коли працівник працював в інших містах.

Відповідно сутністю, яка є самостійною, але також витікає і з проживання, є робота (сутність Job). Робота необхідна зокрема для того, щоб можна було переглянути, які саме працівники зазвичай працюють в певному місті, а також для того, щоб розуміти загалом популярність професій для віддаленої роботи.

Окремо було виділено сутність сфери роботи (сутність Sphere), щоб об'єднати в певні групи роботи з метою спрощення розуміння суті та покращення швидкості доступу до потрібної професії і підвищення зручності пошуку.

Працівник фіксує результати свого досвіду проживання у відгуках (сутність Review). Потенційно кожне проживання може фіксуватися окремим відгуком. Проте працівник може їздити в одне й те саме місто багато разів, а стільки відгуків не створювати, тому було остаточно вирішено відділити проживання від відгуку.

Звичайно, що основною сутністю всієї предметної області є місто (сутність City). Саме в містах реалізується проживання працівників і робота, і саме міста шукають інші працівники для визначення з містами для майбутнього власного проживання. Окрім того в даній сутності мають визначатися усереднені оцінки працівників на основі оцінок, виставлених у їхніх відгуках.

Ще однією сутністю є країна (сутність Country), адже кожне місто розташовується в певній країні і вони в свою чергу дозволяють знову ж таки

групувати міста для зручності пошуку. Окрім того країна характеризує стандартне проживання працівника.

Оскільки працівнику, який приїжджає для віддаленої роботи в місто, важливо не тільки початково визначити, куди йому їхати, але і розуміти зручні для нього місця в цьому місті, то в предметній області було виділено сутність місце (сутність Place).

Для групування місць було виділено сутність PlaceCategory для забезпечення універсального доступу до місць. Це дозволить швидко знаходити потрібні місця в місті для пришвидшення доступу. Наприклад, зручні коворкінги, офіси, зали тощо.

Також було виділено таблицю UserPlace, щоб зберегти в ній інформацію про враження (рекомендацію) працівника відносно конкретного місця.

Сутність підписка (сутність Follow) визначає можливість перегляду працівником проживань та відгуків інших працівників. Таким чином, довіряючи думці певного працівника, можна підписатися на його проживання і відгуки і аналізувати їх для визначення власного майбутнього проживання.

Іноді замість отримання всієї інформації про місто набагато корисніше отримати відповідь на запитання від людини, яка мала досвід проживання і роботи у відповідному місті. Для цього було виділено сутність повідомлення (сутність Message), що визначає запитання і відповіді від одного працівника до іншого.

За даними сутностями було визначено відповідні таблиці БД, а між таблицями визначено зв'язки.

Таблиця Country має зв'язок з таблицею City один до багатьох через те, що одній країні може відповідати багато міст, проте кожне місто розташоване у чітко визначеній країні.

Таблиця Country має зв'язок з таблицею Worker один до багатьох через те, що з однієї країни може походити багато працівників, проте кожен

працівник повинен бути ідентифікований за країною свого постійного проживання.

Таблиця City має зв'язок з таблицею Living один до багатьох через те, що у кожному місті відбувається багато проживань, проте кожне проживання відбувається в конкретному місті.

Таблиця Job має зв'язок з таблицею Living один до багатьох через те, що працівники з однією і тією самою роботою мають багато проживань, проте кожне проживання має бути визначено однією роботою. Якщо під час проживання працівник змінив декілька робіт, то має бути створено окремі записи для різних робіт, щоб характеризувати її. Якщо ж працівник одночасно виконує декілька робіт, то передбачається, що він має вибрати якусь роботу як основну і її визначити для себе.

Таблиця Sphere має зв'язок з таблицею Job один до багатьох через те, що на одну сферу припадає багато робіт, проте одна робота може бути зв'язана тільки з однією конкретною сферою.

Таблиця City має зв'язок з таблицею Place один до багатьох через те, що в одному місті розташовано багато різних місць, проте одне місце може знаходитись тільки в одному конкретному місті.

Таблиця PlaceCategory має зв'язок з таблицею Place один до багатьох через те, що на одну категорію припадає багато конкретних місць, проте кожне місце відповідає одній конкретній категорії.

Таблиця Worker має зв'язок з таблицею Message один до багатьох через те, що кожен працівник може мати багато відправлених і багато отриманих повідомлень, проте кожне повідомлення може мати тільки одного відправника і одного отримувача.

Таблиця Worker має зв'язок з таблицею Follow один до багатьох через те, що кожен працівник може мати багато підписок на інших працівників, проте кожна підписка може мати тільки одне джерело і одного підписника.

Таблиця Worker має зв'язок з таблицею WorkerPlace один до багатьох через те, що кожен працівник може мати багато рекомендацій різних місць, проте кожна рекомендація належить тільки одному працівнику.

Таблиця Place має зв'язок з таблицею WorkerPlace один до багатьох через те, що кожне місце може мати багато рекомендацій, проте кожна рекомендація належить тільки одному місцю.

Таблиця Worker має зв'язок з таблицею Living один до багатьох через те, що кожен працівник може мати багато проживань, проте кожне проживання характеризує тільки одного працівника.

Схема створеної БД подана на рис. 2.3.

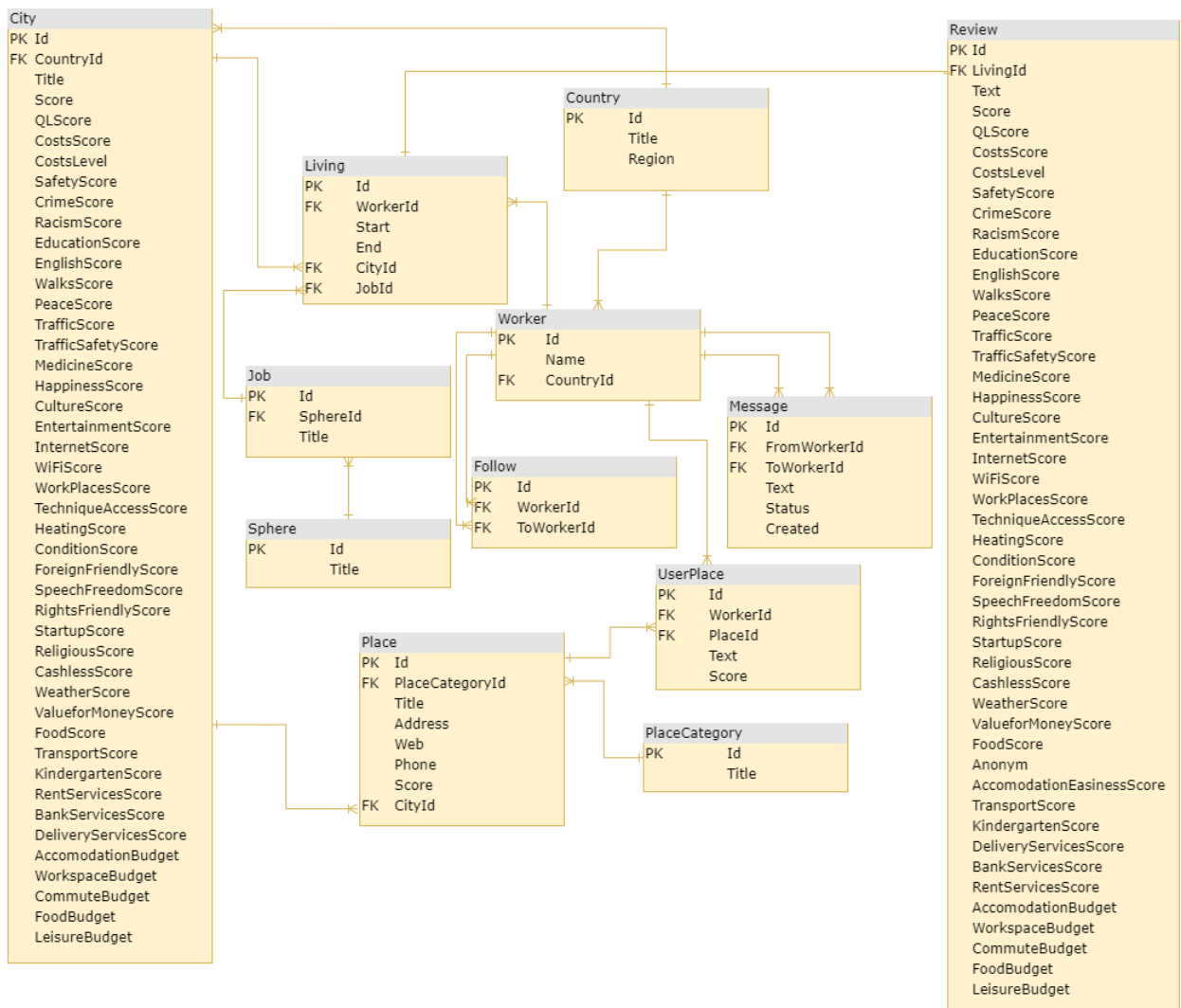


Рисунок 2.3 – Схема БД

Таблиця Review має зв'язок з таблицею Living один до одного через те, що кожний відгук може характеризувати тільки одне проживання, проте одне проживання може бути охарактеризовано тільки одним відгуком або жодним.

2.4 Визначення структури програми

Структуру вебзастосунку, який розробляється, було запроєктовано на основі використання логіки, яка відповідає вебфреймворку ASP.NET Core (рис. 2.4).

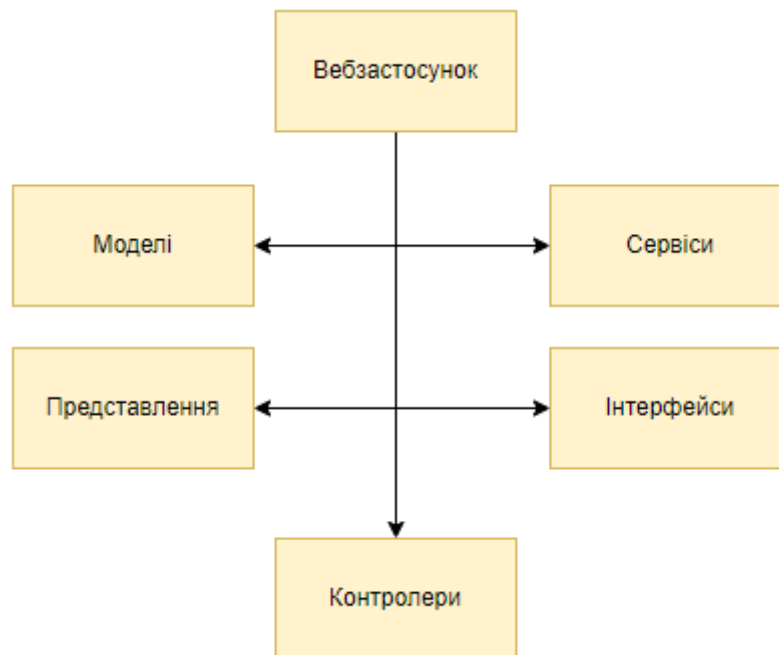


Рисунок 2.4 – Структурна схема вебзастосунку

До основної структури вебзастосунку входять моделі, які передбачають визначення класів у відповідності з таблицями БД вебзастосунку, які будуть побудовані на основі цих моделей. Для доступу до цих даних мають бути визначені інтерфейси. Інтерфейси реалізуються безпосередньо у сервісах. Сервіси необхідні безпосередньо для забезпечення доступу до даних БД напряму. Сервіси мають бути згруповані аналогічно

моделям, проте додатково до цього має бути передбачено сервіс DynamicService, який необхідний для групування даних, пов'язаних з реалізацією функцій застосунку, де дані розглядаються в динаміці, тобто в часовому розрізі.

Безпосередньо робота користувача з вебзастосунком реалізується через предсталення вебзастосунку. Представлення відповідають вебсторінкам, які забезпечують кожну з функціональних вимог. Взаємодія між представленнями і даними реалізується через контролери. Контролери мають бути реалізовані для кожної вебсторінки, визначаючи формування даних з сервісів вебзастосунку для відображення на відповідній вебсторінці.

2.5 Висновки за розділом 2

Рішення щодо вибору вебфреймворку виконано на користь фреймворку ASP.NET Core і відповідно мови програмування C#. Забезпечено моделювання роботи неавторизованого користувача і працівника на основі визначених функціональних вимог до програми. Побудовано структуру БД, визначено структуру вебзастосунку.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм функціонування програми

Функціонування програми виконується головним чином на основі взаємодії користувача з програмою, тому саме на цій процедурі була побудована основна схема функціонування програми (рис. 3.1).

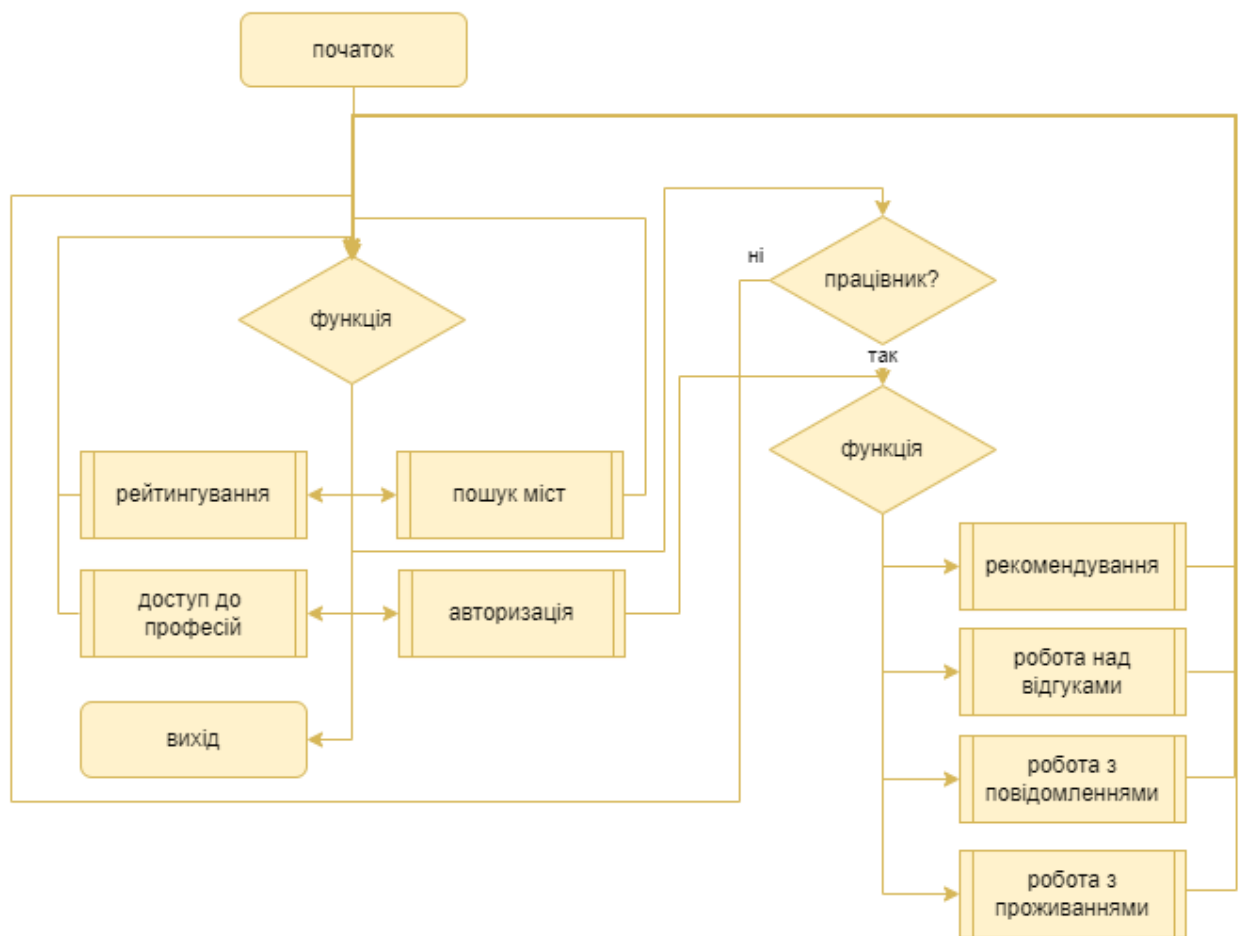


Рисунок 3.1 – Схема функціонування програми

Обрати потрібну функцію може сам користувач. Користувачу доступні всі функції, окрім тих для яких потрібно авторизуватися, окрім випадку, коли користувач уже авторизувався. Кожна з цих функцій передбачає виконання цілої послідовності дій або роботу одразу з декількома функціями.

3.2 Опис реалізованих класів

Структура БД була визначена на основі створення коду моделей, які за допомогою фреймворку Entity Framework Core були перетворені початково на відповідні таблиці БД.

Для реалізації моделі відгуку про проживання і віддалену роботу в певному місті створено клас Review з полями:

- int Id – номер відгуку;
- int LivingId – номер проживання, за яким було створено відгук щодо віддаленої роботи, знаходячись у місті;
- string Text – повний текст відгуку;
- byte Score – загальна оцінка місту;
- byte QLScore – оцінка якості життя в місті;
- byte CostsScore – оцінка рівня витрат у місті;
- float CostsLevel – сума витрат у місті;
- byte SafetyScore – оцінка безпеки у місті;
- byte CrimeScore – оцінка кримінального рівня у місті;
- byte RacismScore – оцінка проблем з расизмом у місті;
- byte EducationScore – оцінка рівня освіти у місті;
- byte EnglishScore – оцінка рівня володіння жителями міста англійською мовою;
- byte WalksScore – оцінка комфортності прогулянок в місті;
- byte PeaceScore – оцінка мирної ситуації в місті;
- byte TrafficScore – оцінка інтенсивності дорожнього руху в місті;
- byte TrafficSafetyScore – оцінка безпечності дорожнього руху в місті;
- byte MedicineScore – оцінка рівня медицини;
- byte HappinessScore – оцінка сприйняття людей у місті себе щасливими;
- byte CultureScore – оцінка культури і культурних подій у місті;
- byte EntertainmentScore – оцінка рівня розваг у місті;

- byte InternetScore – оцінка якості інтернету в місті;
- byte WiFiScore – оцінка бездротового доступу до інтернету в місті;
- byte WorkPlacesScore – оцінка робочих місць у місті;
- byte TechniqueAccessScore – оцінка доступності потрібної для роботи техніки в місті;
- byte HeatingScore – оцінка якості обігріву в приміщеннях міста;
- byte ConditionScore – оцінка доступності кондиціонування в приміщеннях міста;
- byte ForeignFriendlyScore – оцінка дружності до іноземців у місті;
- byte SpeechFreedomScore – оцінка свободи слова в місті;
- byte RightsFriendlyScore – оцінка прав людини в місті;
- byte StartupScore – оцінка рівня розвитку стартапів у місті;
- byte ReligiousScore – оцінка релігійності міста;
- byte CashlessScore – оцінка доступності безготівкових розрахунків;
- byte WeatherScore – оцінка погоди в місті;
- byte ValueforMoneyScore – оцінка відношення ціни і якості життя в місті;
- byte FoodScore – оцінка їжі в місті;
- byte TransportScore – оцінка громадського транспорту в місті;
- byte KindergartenScore – оцінка доступності дитячих садків у місті;
- byte RentServicesScore – оцінка послуг з прокату і оренди в місті;
- byte BankServicesScore – оцінка банківських послуг у місті;
- byte DeliveryServicesScore – оцінка сервісів доставки у місті;
- byte AccomodationEasinessScore – оцінка легкості оренди житла в місті;
- bool Anonym – анонімність витрат;
- float AccomodationBudget – частина витрат на житло в бюджеті витрат у місті;
- float WorkspaceBudget – частина витрат на роботу та робоче місце в бюджеті витрат у місті;

- float CommuteBudget – частина витрат на транспорт в бюджеті витрат у місті;

- float FoodBudget – частина витрат на їжу в бюджеті витрат у місті;

- float LeisureBudget – частина витрат на відпочинок на їжу в бюджеті витрат у місті;

- virtual Living Living – об’єкт проживання, за яким було створено відгук щодо віддаленої роботи, знаходячись у місті.

Для реалізації моделі міста створено клас City з полями:

- int Id – номер відгуку;

- int CountryId – номер країни, в якій розташовано місто;

- string Title – назва міста;

- float Score – середня загальна оцінка місту;

- float QLScore – середня оцінка якості життя в місті;

- float CostsScore – середня оцінка рівня витрат у місті;

- float CostsLevel – середня сума витрат у місті;

- float SafetyScore – середня оцінка безпеки у місті;

- float CrimeScore – середня оцінка кримінального рівня у місті;

- float RacismScore – середня оцінка проблем з расизмом у місті;

- float EducationScore – середня оцінка рівня освіти у місті;

- float EnglishScore – середня оцінка рівня володіння жителями міста англійською мовою;

- float WalksScore – середня оцінка комфортності прогулянок в місті;

- float PeaceScore – середня оцінка мирної ситуації в місті;

- float TrafficScore – середня оцінка інтенсивності дорожнього руху в місті;

- float TrafficSafetyScore – середня оцінка безпечності дорожнього руху в місті;

- float MedicineScore – середня оцінка рівня медицини;

- float HappinessScore – середня оцінка сприйняття людей у місті себе щасливими;

- float CultureScore – середня оцінка культури і культурних подій у місті;
- float EntertainmentScore – середня оцінка рівня розваг у місті;
- float InternetScore – середня оцінка якості інтернету в місті;
- float WiFiScore – середня оцінка бездротового доступу до інтернету в місті;
- float WorkPlacesScore – середня оцінка робочих місць у місті;
- float TechniqueAccessScore – середня оцінка доступності потрібної для роботи техніки в місті;
- float AccomodationEasinessScore – середня оцінка легкості оренди житла в місті;
- float HeatingScore – середня оцінка якості обігріву в приміщеннях міста;
- float ConditionScore – середня оцінка доступності кондиціонування в приміщеннях міста;
- float ForeignFriendlyScore – середня оцінка дружності до іноземців у місті;
- float SpeechFreedomScore – середня оцінка свободи слова в місті;
- float RightsFriendlyScore – середня оцінка прав людини в місті;
- float StartupScore – середня оцінка рівня розвитку стартапів у місті;
- float ReligiousScore – середня оцінка релігійності міста;
- float CashlessScore – середня оцінка доступності безготівкових розрахунків;
- float WeatherScore – середня оцінка погоди в місті;
- float ValueforMoneyScore – середня оцінка відношення ціни і якості життя в місті;
- float FoodScore – середня оцінка їжі в місті;
- float TransportScore – середня оцінка громадського транспорту в місті;

- float KindergartenScore – середня оцінка доступності дитячих садків у місті;
 - float RentServicesScore – середня оцінка послуг з прокату і оренди в місті;
 - float BankServicesScore – середня оцінка банківських послуг у місті;
 - float DeliveryServicesScore – середня оцінка сервісів доставки у місті;
 - float AccomodationBudget – усереднена частина витрат на житло в бюджеті витрат у місті;
 - float WorkspaceBudget – усереднена частина витрат на роботу та робоче місце в бюджеті витрат у місті;
 - float CommuteBudget – усереднена частина витрат на транспорт в бюджеті витрат у місті;
 - float FoodBudget – усереднена частина витрат на їжу в бюджеті витрат у місті;
 - float LeisureBudget – усереднена частина витрат на відпочинок на їжу в бюджеті витрат у місті;
 - virtual ICollection<Living> Livings – колекція проживань працівників у цьому місті;
 - virtual ICollection<Place> Places – колекція місць, які було рекомендовано працівниками у цьому місті;
 - virtual Country Country – об’єкт країни, в якій знаходиться місто.
- Для реалізації моделі місця створено клас Place з полями:
- int Id – номер місця;
 - int PlaceCategoryId – номер категорії місць;
 - string Title – назва місця;
 - string Address – адреса, за якою розташовано дане місце;
 - string Web – вебадреса даного місця;
 - string Phone – номер телефону;
 - int CityId – номер міста, в якому знаходиться місце;
 - byte Score – середня оцінка, виставлена місцю;

- virtual PlaceCategory PlaceCategory – об’єкт категорії, до якої належить місце;

- virtual City City – об’єкт міста, в якій знаходиться місце;

- virtual ICollection<UserPlace> UserPlaces – рекомендації працівників відносно даного місця.

Для реалізації моделі проживання створено клас Living з полями:

- int Id – номер проживання;

- int WorkerId – номер працівника, до якого відноситься дане проживання;

- DateTime Start – дата початку проживання;

- DateTime End – дата завершення проживання;

- int CityId – номер міста, в якому відбулось проживання;

- int JobId – номер роботи, яку виконував працівник під час даного проживання;

- virtual Worker Worker – об’єкт працівника, який проживав у даному випадку;

- virtual City City – об’єкт міста, в якому відбулось проживання;

- virtual Job Job – об’єкт роботи, яку виконував працівник під час даного проживання.

Для реалізації моделі працівника створено клас Worker шляхом наслідування класу IdentityUser, який забезпечує базову ідентифікацію користувача в ASP.NET Microsoft.AspNetCore.Identity, з полями:

- string Name – ім’я працівника;

- int CountryId – номер країни;

- virtual Country Country – об’єкт країни, національність якої має працівник;

- virtual ICollection<Follow> Follows – підписки даного працівника на інших працівників;

- virtual ICollection<Follow> Followers – підписники даного працівника;

- virtual ICollection<UserPlace> UserPlaces – місця, рекомендовані даним працівником;
- virtual ICollection<Message> ToMessages – повідомлення, отримані даним працівником;
- virtual ICollection<Message> FromMessages – повідомлення, відправлені даним працівником.

Для реалізації моделі повідомлення створено клас Message з полями:

- int Id – номер повідомлення;
- int FromWorkerId – номер працівника, який є автором повідомлення;
- int ToWorkerId – номер працівника, якому адресовано повідомлення;
- string Text – текст повідомлення;
- bool State – стан повідомлення (прочитане або непрочитане);
- DateTime Created – дата, час створення повідомлення.
- virtual Worker FromWorker – об'єкт працівника, який є автором повідомлення;
- virtual Worker ToWorker – об'єкт працівника, який є адресатом повідомлення.

Аналогічно влаштовані інші класи моделей.

Клас сервісів міста CityService включає наступні методи:

- CityService (DbContext con) – визначає в якості контексту БД, який використовується для прямого доступу до даних, con;
- City GetById(int city_id) – повертає об'єкт міста, яке має номер city_id;
- IEnumerable<City> GetAll() – повертає список всіх міст, які є в БД;
- IEnumerable<City> GetCitiesByRegion(byte region) – повертає список всіх міст, які належать до частини світу region;
- IEnumerable<City> GetCitiesByCountry(int country_id) – повертає список міст, які знаходяться в країні з номером country_id;

- `IEnumerable<City> GetRating()` – повертає рейтинг міст, відсортованих за середньою загальною оцінкою, у вигляді відповідного переліку;
- `IEnumerable<City> GetRatingByCountry(int country_id)` – повертає рейтинг міст у країні з номером `country_id`, відсортованих за середньою загальною оцінкою, у вигляді відповідного переліку;
- `IEnumerable<City> GetRatingByRegion(int region)` – повертає рейтинг міст у частині звіту з номером `region`, відсортованих за середньою загальною оцінкою, у вигляді відповідного переліку;
- `IEnumerable<City> GetRatingByCriterion(int criterion, List <byte> measures)` – повертає перелік міст, що представляють рейтинг, сформований за критерієм з номером `criterion` (визначає одну з середніх оцінок, якими оцінюються міста), вносячи до рейтингу тільки ті міста, що задовольняють умовам переданим в якості параметра `measures`, який є переліком значень оцінок міста за кожним критерієм (нульове значення означає, що перевірка за даним критерієм пропущена);
- `IEnumerable<City> Search(string city_title, string country_title, byte region, List <byte> measures, float mincosts, float maxcosts, float minaccom, float maxaccom, float minworkspace, float maxworkspace, float mincom, float maxcom, float minfood, float maxfood, float minleisure, float maxleisure)` – виконує пошук міст, що мають в своїй назві рядок `city_title`, в назві країни рядок `country_title`, належать до частини світу `region`, мають значення оцінок `measures`, мають сумарні витрати в інтервалі від `mincosts` до `maxcosts`, витрати на оренду житла в інтервалі від `minaccom` до `maxaccom`, витрати на робоче місце в інтервалі від `minworkspace` до `maxworkspace`, витрати на транспорт в інтервалі від `mincom` до `maxcom`, витрати на їжу в інтервалі від `minfood` до `maxfood`, витрати на відпочинок в інтервалі від `minleisure` до `maxleisure`;
- `async Task CityScores(int city_id)` – виконує обчислення середніх оцінок і витрат в місті на основі всіх відгуків працівників про проживання в даному місті.

Клас сервісів відгуку ReviewService включає наступні методи:

- ReviewService (BaseDbContext con) – визначає в якості контексту БД, який використовується для прямого доступу до даних, con;
- Review GetById(int rev_id) – повертає об’єкт відгуку, який має номер rev_id;
- IEnumerable<Review> GetAll() – повертає список всіх відгуків, які є в БД;
- IEnumerable<Review> GetWorkerReviews(int worker_id) – повертає список всіх відгуків, створених працівником з номером worker_id;
- IEnumerable<Review> GetCityReviews(int city_id) – повертає список всіх відгуків стосовно міста з номером city_id;
- async Task Add(Review rev) – вносить відгук rev до БД;
- async Task Update(Review rev) – оновлює відгук rev у БД.

Клас сервісів місця PlaceService включає наступні методи:

- PlaceService (BaseDbContext con) – визначає в якості контексту БД, який використовується для прямого доступу до даних, con;
- Place GetById(int pid) – повертає об’єкт місця, який має номер pid;
- IEnumerable<Place> GetAll() – повертає список всіх місць, які є в БД;
- IEnumerable<Place> GetCityPlaces(int city_id) – повертає список всіх місць, що знаходяться в місті city_id;
- IEnumerable<Place> GetWorkerPlaces(int worker_id) – повертає список всіх місць, рекомендованих працівником з номером worker_id;
- IEnumerable<Place> Search(string place_title, int city_id, string address, byte minscore, byte maxscore) – виконує пошук місць, що мають в своїй назві рядок place_title, знаходяться в місті з номером city_id за адресою, яка містить address, а середній виставлений бал в рекомендаціях працівників знаходиться в інтервалі від minscore до maxscore;
- async Task Add(Place p) – вносить місце p до БД;
- async Task Update(Review rev) – оновлює місце p у БД;

– `async Task Scores(int pid)` – виконує обчислення середніх оцінок міста з номером `pid` на основі всіх рекомендацій працівників.

Аналогічно реалізовано інші класи сервісів.

Окрім того для роботи з функціями програми було реалізовано всі необхідні представлення, а для заповнення представлень даними було створено відповідні контролери.

Клас контролера міста `CityController` включає наступні основні методи:

– `ActionResult Index(int city_id)` – заповнює даними міста з номером `city_id` представлення перегляду повної інформації про місто;

– `ActionResult CitiesByRegion(int region)` – заповнює даними представлення перегляду всіх міст з частини світу `region`;

– `ActionResult CitiesByCountry(int country_id)` – заповнює даними представлення перегляду всіх міст з країни з номером `country_id`;

– `ActionResult CitiesRating()` – заповнює даними представлення перегляду загального рейтингу міст;

– `ActionResult CitiesRatingByCountry(int country_id)` – заповнює даними представлення перегляду рейтингу міст в країні з номером `country_id`;

– `ActionResult CitiesRatingByRegion(int region)` – заповнює даними представлення перегляду рейтингу міст у частині світу `region`;

– `ActionResult Search()` – заповнює підготовчими даними сторінку, на якій заповнюються параметри пошуку міста;

– `ActionResult CitiesRatingByCriterion(int criterion, string measures)` – заповнює даними представлення перегляду рейтингу міст, сортуючи їх за оцінкою з номером `criterion`, обмежуючи інші параметри пошуку значеннями, переданими як рядок `measures`;

– `ActionResult SearchResults(string city_title, string country_title, byte region, string measures, float mincosts, float maxcosts, float minaccom, float maxaccom, float minworkspace, float maxworkspace, float mincom, float`

maxcom, float minfood, float maxfood, float minleisure, float maxleisure) – заповнює даними представлення перегляду результатів пошуку міст, які мають в назві city_title, знаходяться в країні з назвою country_title, в частині світу region, мають сумарні витрати в інтервалі від mincosts до maxcosts, витрати на оренду житла в інтервалі від minasscom до maxasscom, витрати на робоче місце в інтервалі від minworkspace до maxworkspace, витрати на транспорт в інтервалі від mincom до maxcom, витрати на їжу в інтервалі від minfood до maxfood, витрати на відпочинок в інтервалі від minleisure до maxleisure, обмежуючи інші параметри пошуку значеннями, переданими як рядок measures.

3.3 Висновки за розділом 3

Представлено порядок функціонування програми. У відповідності зі структурною схемою програми реалізовано класи моделей, сервісів, контролерів, основні з яких описано, визначено візуальну частину вебзастосунку.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для пошуку міст, в яких людина, яка працює віддалено, може проживати і працювати, виходячи з власних вимог, які стосуються роботи і проживання. Відповідність вимогам визначається на основі досвіду інших працівників, які також проживали і працювали віддалено в цих містах раніше.

4.2 Умови виконання програми

Для забезпечення підтримки роботи вебсервера та сервера БД необхідно надати апаратне забезпечення, що включає мінімально процесор, що має не менше ніж 4 ядра з тактовою частотою не менше 2,1 ГГц, підтриманий оперативною пам'яттю обсягом не менше 8 Гб та жорстким диском з доступними 10 Гб простору на ньому.

4.3 Експлуатація програми

Експлуатація програми побудована на основі стандартного для вебзастосунків механізму. Після розгортання вебзастосунку на вебсервері потрібно звернутися за адресою вебсайту. Доступ неавторизованого користувача забезпечується через роботу з вебсторінками на основі меню:

- пошук: дозволяє реалізовувати пошук міст на основі набору критеріїв для упорядкування, а також набору умов, що включають оцінки, інтервали витрат, назву міста, країну або частину світу (окрема кнопка «Шукати місця» дозволяє перейти до пошуку місць);
- рейтинги (рис. 4.1): дозволяє переглядати рейтинг міст за параметрами, між якими можна переключатися за допомогою відповідних кнопок («Загальний» для перегляду рейтингу за загальною оцінкою міста,

«Частина світу» для перегляду такого ж рейтингу, але обмеженого містами заданої з переліку частини світу, «Країна» для перегляду такого ж рейтингу, але обмеженого містами обраної зі списку країни, «Критерій» для перегляду рейтингу з можливістю обмеження окремих оцінок, а також вибору оцінки, за якою складається рейтинг);

– професії: дозволяє переглянути список професій, відсортованих за кількістю проживань, які відповідали даній професії, а після натискання на відповідну професію буде відкриватися рейтинг міст за кількістю проживань у них за відповідною роботою.

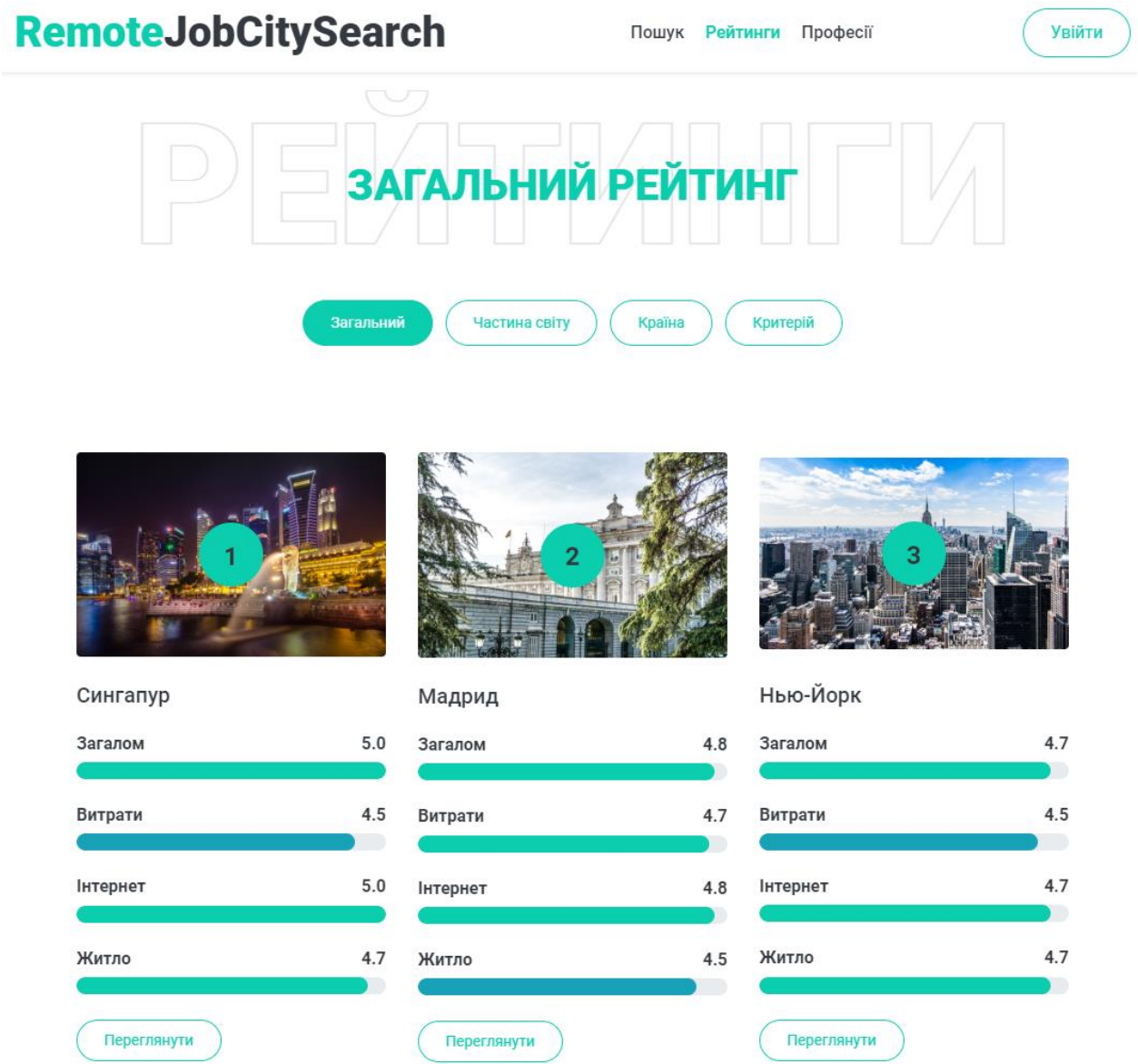


Рисунок 4.1 – Рейтинг міст

Якщо на сторінці рейтингу, результатів пошуку або будь-якої іншої сторінки натиснути на назву міста, то відкриється вебсторінка окремого міста (рис. 4.2).

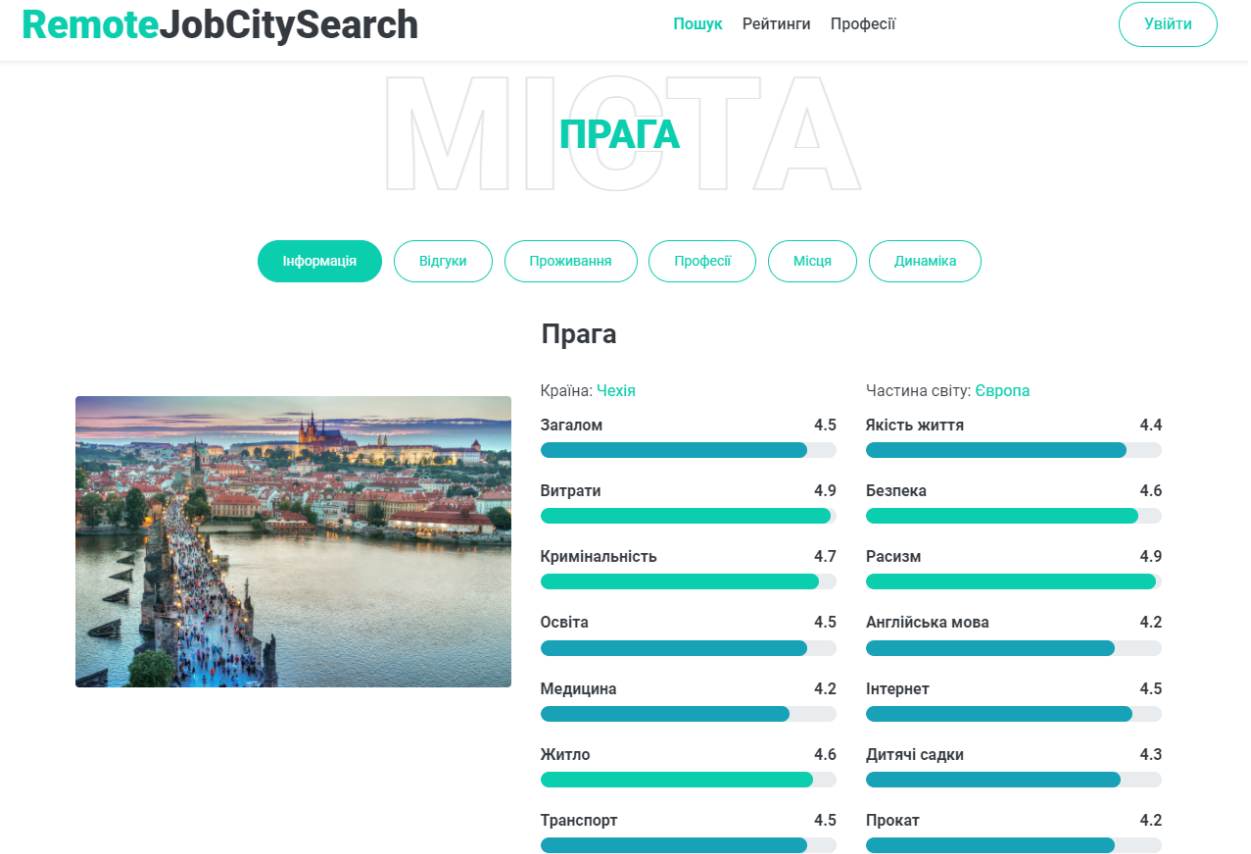


Рисунок 4.2 – Інформація про місто

На вебсторінці окремого міста доступна інформація, розділена за наступними розділами на основі наступних кнопок:

- інформація: середні оцінки за всіма критеріями та середні витрати загалом і в розрізі критеріїв;
- відгуки: містить посилання на різні відгуки про проживання працівників у даному місті, відсортовані починаючи з останнього;
- проживання: містить усі проживання різних працівників зі вказанням періоду проживання у даному місті, сортуючи їх за часом початку проживання;

– професії: містить список всіх професій, відсортованих за кількістю проживань у даному місті за даною професією, відповідно можна відкрити сторінку професії;

– місця: містить основні категорії місць (на даний момент виділено коворкінги, дитячі садочки, спортивні зали, офіси та сервіси прокату), для відображення місць відповідної категорії в місті відсортованих за оцінками, а також надає можливість переходу до пошуку місць за кнопкою «Шукати місця»;

– динаміка (рис. 4.3): дозволяє переглянути зміни протягом останнього року помісячно загальної оцінки, виставленої місту (використовується кнопка «Оцінки»), або кількості проживань у місті (використовується кнопка «Відвідуваність»).

Оцінки відображаються на шкалі бірюзовим, якщо оцінка вище ніж 4.5, блакитним, якщо оцінка вище ніж 4.0, помаранчевим, якщо оцінка вище 3.5, червоним, якщо оцінка вище 3.0, і чорним у всіх інших випадках.

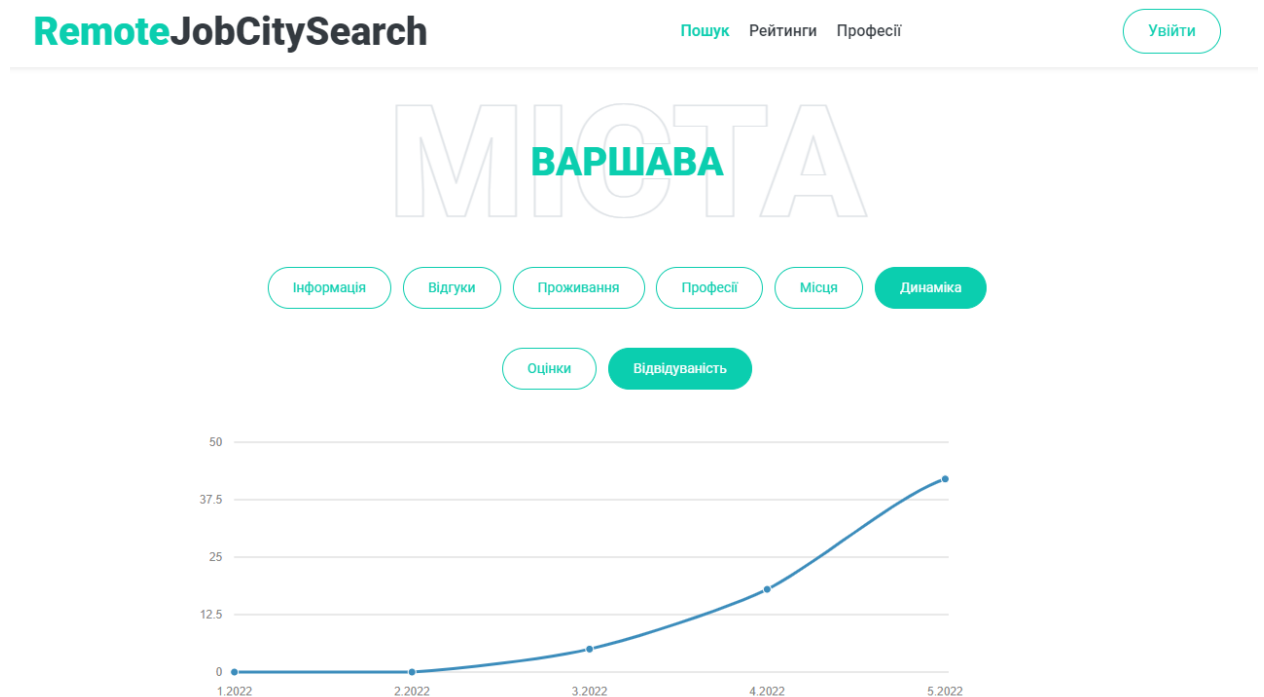


Рисунок 4.3 – Динаміка відвідуваності міста

За допомогою кнопки «Увійти» можна перейти до вебсторінки авторизації користувача з можливістю за потреби реєстрації. Після авторизації працівник отримує розширене меню, що додатково містить наступні пункти меню:

- проживання (рис. 4.4): працівнику надається список всіх його проживань, починаючи з останнього і далі за віддаленістю, а за кожним проживанням працівник може або його відредагувати за допомогою натискання на відповідну кнопку і внесення змін на окремій сторінці, перейти до детального перегляду інформації за проживанням (звідти також можна перейти до редагування або створення відгуку), а також переглянути відгук (за допомогою кнопки «Відгук»), якщо він вже за проживанням був створений, або створити цей відгук (за допомогою кнопки «Створити відгук»), якщо цей відгук не був створений;

- відгуки, де міститься аналогічний список зі створеними відгуками за різними містами, де проживав працівник;

- місця, де містяться рекомендації створені даним працівником, а також доступна кнопка «Рекомендувати» для створення нової рекомендації;

- чат, де міститься список всіх користувачів, які відправили повідомлення даному працівнику, натискаючи на ім'я працівника, можна переглянути список отриманих і відправлених повідомлень, а також створити нове повідомлення;

- підписки, де містяться переліки проживань і відгуків працівників (перемикання за допомогою однойменних кнопок), на які авторизований працівник підписався;

- профіль, де міститься інформація про авторизованого працівника, список працівників, на які підписався працівник, а також працівників, на яких він підписався (аналогічні засоби разом з відправленням повідомлення доступні при відкритті сторінки будь-якого працівника при перегляді його проживання або відгуку).

ПРОЖИВАННЯ

ВЛАСНІ ПРОЖИВАННЯ

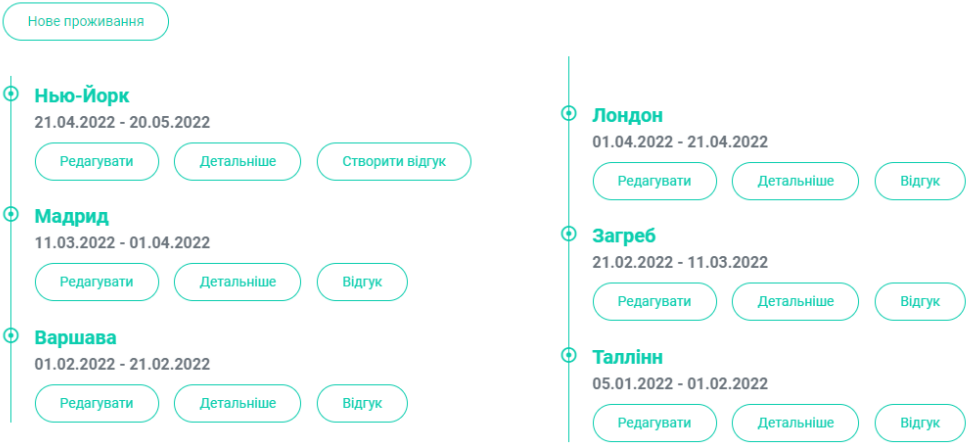


Рисунок 4.4 – Робота з власними проживаннями

4.4 Тестування програми

Під час тестування перевірено виконання всіх визначених функцій (табл. 4.1). Тестування завершено успішно.

Таблиця 4.1 – Тестування програми

Функція	Неавторизований користувач	Працівник
Внесення проживання в місті		+
Редагування внесеного проживання в місті		+
Перегляд власних проживань користувачем		+
Перегляд проживань заданого працівника в місті	+	+
Перегляд проживань в місті	+	+
Перегляд популярних віддалених робіт у місті	+	+

Продовження таблиці 4.1

Функція	Неавторизований користувач	Працівник
Створення відгуку про місто		+
Підписування на працівника		+
Відписування під працівника		+
Перегляд підписників працівника	+	+
Перегляд підписок працівника	+	+
Перегляд популярних міст за заданою роботою	+	+
Відправка повідомлення працівнику		+
Перегляд отриманих повідомлень		+
Рекомендування місця		+
Перегляд рекомендованих місць у місті	+	+
Перегляд власних рекомендацій		+
Перегляд загального рейтингу міст	+	+
Перегляд рейтингу міст за частиною світу	+	+
Перегляд рейтингу міст у країні	+	+
Перегляд рейтингу міст за критерієм	+	+
Пошук міст	+	+
Пошук місць за містом	+	+
Пошук місць загалом	+	+
Перегляд власних відгуків про міста		+
Перегляд відгуків про місто	+	+
Перегляд віддаленої роботи в місті	+	+
Перегляд динаміки зміни оцінок міста	+	+
Перегляд динаміки відвідуваності міста	+	+
Перегляд популярності професій	+	+
Авторизація працівника		+
Реєстрація працівника		+

4.5 Висновки за розділом 4

Усі передбачені функції програми було успішно реалізовано: їх виконання перевірено під час тестування, а процес роботи з програмою над виконанням цих функцій описано.

ВИСНОВКИ

За результатами виконання дипломної роботи було розроблено ПЗ пошуку міст для віддаленої роботи, що дозволило підвищити рівень інформованості працівників перед переїздом в нове місто для віддаленої роботи.

Для досягнення цього було спочатку виконано огляд проблеми віддаленої роботи, підтверджено актуальність її в розрізі пошуку міст для віддаленої роботи, виконано огляд існуючих для цього рішень. Проведений аналіз став основою для виконання проєктування програми, за результатами якого зокрема було виконано прийняття набору рішень, пов'язаних з використанням мови програмування C#, фреймворків ASP.NET Core і Entity Framework Core, системи керування базами даних Microsoft SQL Server для розробки, а також інших додаткових засобів. За результатами проєктування програми також виконано структурування програми і даних, що відобразилось у створеній схемі БД.

Грунтуючись на створеній структурній схемі програми було реалізовано окремі одиниці, основні класи яких було детально описано, що в підсумку дозволило реалізувати всю програму загалом. Проведену реалізацію програми було підтримано і продовжено успішним проведенням тестуванням виконаної розробки.

Розроблена як вебзастосунок програма дозволяє людям, які бажають віддалено працювати з інших міст, подорожуючи або переміщуючись з рідного міста, знайти місто для проживання і віддаленої роботи, що відповідає їхнім вимогам до умов роботи і проживання на основі досвіду інших працівників, які вже працювали віддалено з цього міста раніше. Таким чином, виконана розробка дозволяє в підсумку підготуватися людині до переїзду в інше місто, в якому вона бажає працювати віддалено, на короткий або тривалий строк.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Statistics on Remote Workers That Will Surprise You (2022) [Electronic resource]. – Access mode : <https://www.apollotechnical.com/statistics-on-remote-workers/>.
2. Turits, M. Are there enough remote jobs for everyone who wants one? [Electronic resource] / M. Turits. – Access mode : <https://www.bbc.com/worklife/article/20220421-are-there-enough-remote-jobs-for-everyone-who-wants-one>.
3. 15 Types of Remote Jobs [Electronic resource]. – Access mode : <https://www.indeed.com/career-advice/finding-a-job/types-of-remote-jobs>.
4. Peek, S. Communication Technology and Inclusion Will Shape the Future of Remote Work [Text] / S. Peek. – Access mode : <https://www.businessnewsdaily.com/8156-future-of-remote-work.html>.
5. Smith, M. The 10 most in-demand entry-level remote jobs to land right now – and where to find them [Electronic resource] / M. Smith. – Access mode : <https://www.cnbc.com/2022/05/10/the-10-most-in-demand-entry-level-remote-jobs-to-land-right-now.html>.
6. Digital Nomad World – The Fastest Growing Digital Nomads Community [Electronic resource]. – Access mode : <https://digitalnomads.world/>.
7. Nomad List – Best Places to Live for Digital Nomads [Electronic resource]. – Access mode : <https://nomadlist.com/>.
8. George, N. Mastering Django [Electronic resource] / N. George. – Hamilton, Australia : GNW Independent Publishing, 2020. – 638 p.
9. Duldulao, D. B. ASP.NET Core and Vue.js [Text] : Build real-world, scalable, full-stack applications using Vue.js 3, TypeScript, .NET 5, and Azure / D. B. Duldulao. – Birmingham : Packt Publishing, 2021. – 478 p.
10. Murach, J. Murach's ASP.NET Core MVC [Text] / J. Murach. – California : Mike Murach & Associates, 2020. – 780 p.
11. Petkovic, D. Microsoft SQL Server 2019 [Text] : A Beginner's Guide,

Seventh Edition / D. Petkovic. – New York : McGraw Hill, 2020. – 864 p.

12. Petkovic, D. Microsoft SQL Server 2016 [Text] : A Beginner's Guide / D. Petkovic. – New York : McGraw Hill, 2016. – 896 p.

ДОДАТОК А
Технічне завдання

Вступ

Віддалена робота є вже навіть не тенденцію, а постійною складовою ринку праці. Велика кількість людей у всьому світі працює віддалено. Віддалена робота дає певну свободу, яку частина з таких працівників застосовує у постійних переміщеннях між містами світу, а інша частина змінює міста рідше, проте не завжди залишається у рідному місті. Саме для реалізації цієї потреби стосовно пошуку потенційних міст проживання і роботи і призначена дана програма.

A.1 Підстави для розробки

Дипломна кваліфікаційна робота виконується за темою «Програмне забезпечення пошуку міст для віддаленої роботи» у відповідності з наказом № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програма призначена для пошуку міст, в яких людина, яка працює віддалено, може проживати і працювати, виходячи з власних вимог, які стосуються роботи і проживання. Відповідність вимогам визначається на основі досвіду інших працівників, які також проживали і працювали віддалено в цих містах раніше.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Від набору функціональних характеристик вимагається виконання наступного:

- внесення проживань у місті;
- редагування проживань у місті;
- перегляд власних проживань;
- перегляд проживань працівника у місті;
- перегляд проживань у місті;
- перегляд популярних віддалених робіт у місті;
- підписування на працівника;
- відписування під працівника;
- перегляд підписників працівника;
- перегляд підписок працівника;
- перегляд популярних міст за роботою;
- відправка повідомлення працівнику;
- перегляд отриманих повідомлень;
- рекомендування місця;
- перегляд рекомендованих місць у місті;
- перегляд власних рекомендацій;
- обчислення середніх оцінок міст;
- рейтингування міст;
- рейтингування міст за частиною світу;
- рейтингування міст у країні;
- рейтингування міст за критерієм;
- пошук міст;
- пошук місць;
- створення власного відгуку про місто;
- перегляд власних відгуків про міста;
- перегляд відгуків про місто;
- перегляд інформації про віддалену роботу в місті;
- перегляд динаміки зміни оцінок міста;
- перегляд динаміки відвідуваності міста;
- перегляд популярності професій віддаленої роботи;

- реєстрація працівників;
- авторизація працівників.

A.3.2 Умови експлуатації

Для експлуатації ПЗ потрібно забезпечити окремий сервер, на якому буде запущено вебсервер і сервер БД для постійного доступу користувачів до сервісу пошуку міст для віддаленої роботи.

A.3.3 Вимоги до складу та параметрів технічних засобів

Для забезпечення підтримки роботи вебсервера та сервера БД необхідно надати апаратне забезпечення, що включає мінімально процесор, що має не менше ніж 4 ядра з тактовою частотою не менше 2,1 ГГц, підтриманий оперативною пам'яттю обсягом не менше 8 Гб та жорстким диском з доступними 10 Гб простору на ньому.

A.4 Порядок контролю та приймання

Для контролю виконання дипломної кваліфікаційної роботи повинен використовуватися відповідний календарний план, який затверджено викладачем у завданні на роботу. Приймання роботи відбувається на основі проведення процедури захисту роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла City.cs

```
using System;
using System.Collections.Generic;

namespace RemoteJobCitySearch.Models
{
    public class City
    {
        public int Id { get; set; }
        public int CountryId { get; set; }
        public string Title { get; set; }
        public float Score { get; set; }
        public float QLScore { get; set; }
        public float CostsScore { get; set; }
        public float CostsLevel { get; set; }
        public float SafetyScore { get; set; }
        public float CrimeScore { get; set; }
        public float RacismScore { get; set; }
        public float EducationScore { get; set; }
        public float EnglishScore { get; set; }
        public float WalksScore { get; set; }
        public float PeaceScore { get; set; }
        public float TrafficScore { get; set; }
        public float TrafficSafetyScore { get; set; }
        public float MedicineScore { get; set; }
        public float HappinessScore { get; set; }
        public float CultureScore { get; set; }
        public float EntertainmentScore { get; set; }
        public float InternetScore { get; set; }
        public float WiFiScore { get; set; }
        public float WorkPlacesScore { get; set; }
        public float TechniqueAccessScore { get; set; }
        public float AccomodationEasinessScore { get; set; }
        public float HeatingScore { get; set; }
        public float ConditionScore { get; set; }
        public float ForeignFriendlyScore { get; set; }
        public float SpeechFreedomScore { get; set; }
        public float RightsFriendlyScore { get; set; }
        public float StartupScore { get; set; }
        public float ReligiousScore { get; set; }
        public float CashlessScore { get; set; }
        public float WeatherScore { get; set; }
        public float ValueforMoneyScore { get; set; }
```

```

        public float FoodScore { get; set; }
        public float TransportScore { get; set; }
        public float KindergartenScore { get; set; }
        public float RentServicesScore { get; set; }
        public float BankServicesScore { get; set; }
        public float DeliveryServicesScore { get; set; }
        public float AccomodationBudget { get; set; }
        public float WorkspaceBudget { get; set; }
        public float CommuteBudget { get; set; }
        public float FoodBudget { get; set; }
        public float LeisureBudget { get; set; }
        public virtual ICollection<Living> Livings
        { get; set; }
        public virtual ICollection<Place> Places
        { get; set; }
        public virtual Country Country { get; set; }
    }
}

```

Б.2 Текст файла Review.cs

```

using System;
using System.Collections.Generic;

namespace RemoteJobCitySearch.Models
{
    public class Review
    {
        public int Id { get; set; }
        public int LivingId { get; set; }
        public string Text { get; set; }
        public byte Score { get; set; }
        public byte QLScore { get; set; }
        public byte CostsScore { get; set; }
        public float CostsLevel { get; set; }
        public byte SafetyScore { get; set; }
        public byte CrimeScore { get; set; }
        public byte RacismScore { get; set; }
        public byte EducationScore { get; set; }
        public byte EnglishScore { get; set; }
        public byte WalksScore { get; set; }
        public byte PeaceScore { get; set; }
        public byte TrafficScore { get; set; }
    }
}

```

```

        public byte TrafficSafetyScore { get; set; }
        public byte MedicineScore { get; set; }
        public byte HappinessScore { get; set; }
        public byte CultureScore { get; set; }
        public byte EntertainmentScore { get; set; }
        public byte InternetScore { get; set; }
        public byte WiFiScore { get; set; }
        public byte WorkPlacesScore { get; set; }
        public byte TechniqueAccessScore { get; set; }
        public byte AccomodationEasinessScore { get; set; }
        public byte HeatingScore { get; set; }
        public byte ConditionScore { get; set; }
        public byte ForeignFriendlyScore { get; set; }
        public byte SpeechFreedomScore { get; set; }
        public byte RightsFriendlyScore { get; set; }
        public byte StartupScore { get; set; }
        public byte ReligiousScore { get; set; }
        public byte CashlessScore { get; set; }
        public byte WeatherScore { get; set; }
        public byte ValueforMoneyScore { get; set; }
        public byte FoodScore { get; set; }
        public byte TransportScore { get; set; }
        public byte KindergartenScore { get; set; }
        public byte RentServicesScore { get; set; }
        public byte BankServicesScore { get; set; }
        public byte DeliveryServicesScore { get; set; }
        public bool Anonym { get; set; }
        public float AccomodationBudget { get; set; }
        public float WorkspaceBudget { get; set; }
        public float CommuteBudget { get; set; }
        public float FoodBudget { get; set; }
        public float LeisureBudget { get; set; }
        public virtual Living Living { get; set; }
    }
}

```

Б.3 Текст файла Place.cs

```

using System;
using System.Collections.Generic;

namespace RemoteJobCitySearch.Data.Models
{

```

```

public class Place
{
    public int Id { get; set; }
    public int PlaceCategoryId { get; set; }
    public string Title { get; set; }
    public string Address { get; set; }
    public string Web { get; set; }
    public string Phone { get; set; }
    public int CityId { get; set; }
    public byte Score { get; set; }
    public virtual PlaceCategory PlaceCategory { get; set; }
    public virtual City City { get; set; }
    public virtual ICollection<UserPlace> UserPlaces
    { get; set; }
}
}

```

Б.4 Текст файла Living.cs

```

using System;
using System.Collections.Generic;

namespace RemoteJobCitySearch.Models
{
    public class Living
    {
        public int Id { get; set; }
        public int WorkerId { get; set; }
        public DateTime Start { get; set; }
        public DateTime End { get; set; }
        public int CityId { get; set; }
        public int JobId { get; set; }
        public virtual Worker Worker { get; set; }
        public virtual City City { get; set; }
        public virtual Job Job { get; set; }
    }
}

```

Б.5 Текст файла Worker.cs

```

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;

namespace RemoteJobCitySearch.Models
{
    public class Worker : IdentityUser
    {
        public string Name { get; set; }
        public int CountryId { get; set; }
        public virtual Country Country { get; set; }
        public virtual ICollection<Follow> Follows
        { get; set; }
        public virtual ICollection<Follow> Followers
        { get; set; }
        public virtual ICollection<UserPlace> UserPlaces
        { get; set; }
        public virtual ICollection<Message> ToMessages
        { get; set; }
        public virtual ICollection<Message> FromMessages
        { get; set; }
    }
}

```

Б.6 Текст файла CityService.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using RemoteJobCitySearch.Data;
using RemoteJobCitySearch.Data.Interfaces;
using RemoteJobCitySearch.Data.Models;
using Microsoft.EntityFrameworkCore;

namespace RemoteJobCitySearch.Service
{
    public class CityService : ICity
    {
        private readonly BaseDbContext basecontext;

        public CityService (BaseDbContext con)

```

```

{
    basecontext = con;
}

public City GetById(int city_id)
{
    return basecontext.Citys.Where(c => c.Id == city_id)
        .Include(c => c.Title)
        .Include(c => c.Score)
        .Include(c => c.QLScore)
        .Include(c => c.CostsScore)
        .Include(c => c.CostsLevel)
        .Include(c => c.SafetyScore)
        .Include(c => c.CrimeScore)
        .Include(c => c.RacismScore)
        .Include(c => c.EducationScore)
        .Include(c => c.EnglishScore)
        .Include(c => c.WalksScore)
        .Include(c => c.PeaceScore)
        .Include(c => c.TrafficScore)
        .Include(c => c.TrafficSafetyScore)
        .Include(c => c.MedicineScore)
        .Include(c => c.HappinessScore)
        .Include(c => c.CultureScore)
        .Include(c => c.EntertainmentScore)
        .Include(c => c.InternetScore)
        .Include(c => c.WiFiScore)
        .Include(c => c.WorkPlacesScore)
        .Include(c => c.TechniqueAccessScore)
        .Include(c => c.HeatingScore)
        .Include(c => c.ConditionScore)
        .Include(c => c.ForeignFriendlyScore)
        .Include(c => c.SpeechFreedomScore)
        .Include(c => c.RightsFriendlyScore)
        .Include(c => c.StartupScore)
        .Include(c => c.ReligiousScore)
        .Include(c => c.CashlessScore)
        .Include(c => c.WeatherScore)
        .Include(c => c.ValueforMoneyScore)
        .Include(c => c.FoodScore)
        .Include(c => c.TransportScore)
        .Include(c => c.KindergartenScore)
        .Include(c => c.AccomodationEasinessScore)
        .Include(c => c.RentServicesScore)
        .Include(c => c.BankServicesScore)

```

```

        .Include(c => c.DeliveryServicesScore)
        .Include(c => c.Anonym)
        .Include(c => c.AccommodationBudget)
        .Include(c => c.WorkspaceBudget)
        .Include(c => c.CommuteBudget)
        .Include(c => c.FoodBudget)
        .Include(c => c.L leisureBudget)
        .Include(c => c.Livings).ThenInclude(p =>
p.Worker)
        .Include(c => c.Livings).ThenInclude(p =>
p.Job)
        .Include(c => c.Places).ThenInclude(p =>
p.PlaceCategory)
        .First();
    }

    public IEnumerable<City> GetAll()
    {
        return basecontext.Citys
            .Include(c => c.Title)
            .Include(c => c.Score)
            .Include(c => c.CostsScore)
            .Include(c => c.InternetScore)
            .Include(c => c.AccommodationEasinessScore)
    }

    public IEnumerable<City> GetCitiesByRegion(byte region)
    {
        return basecontext.Citys
            .Where(c => c.Country.Region == region)
            .Include(c => c.Title)
            .Include(c => c.Score)
            .Include(c => c.CostsScore)
            .Include(c => c.InternetScore)
            .Include(c => c.AccommodationEasinessScore)
    }

    public IEnumerable<City> GetCitiesByCountry(int
country_id)
    {
        return basecontext.Citys
            .Where(c => c.Country.Id == country_id)
            .Include(c => c.Title)
            .Include(c => c.Score)
            .Include(c => c.CostsScore)
    }

```

```

        .Include(c => c.InternetScore)
        .Include(c => c.AccommodationEasinessScore)
    }

    public IEnumerable<City> GetRating()
    {
        cities = basecontext.Citys
            .Include(c => c.Title)
            .Include(c => c.Score)
            .Include(c => c.CostsScore)
            .Include(c => c.InternetScore)
            .Include(c => c.AccommodationEasinessScore)
        return cities.order_by('-Score');
    }

    public IEnumerable<City> GetRatingByCountry(int
country_id)
    {
        cities = basecontext.Citys
            .Where(c => c.Country.Id == country_id)
            .Include(c => c.Title)
            .Include(c => c.Score)
            .Include(c => c.CostsScore)
            .Include(c => c.InternetScore)
            .Include(c => c.AccommodationEasinessScore)
        return cities.order_by('-Score');
    }

    public IEnumerable<City> GetRatingByRegion(int region)
    {
        cities = basecontext.Citys
            .Where(c => c.Country.Region == region)
            .Include(c => c.Title)
            .Include(c => c.Score)
            .Include(c => c.CostsScore)
            .Include(c => c.InternetScore)
            .Include(c => c.AccommodationEasinessScore)
        return cities.order_by('-Score');
    }

    public IEnumerable<City> GetRatingByCriterion(int
criterion, List <byte> measures)
    {
        var expression = basecontext.Citys;
        if (measures[0] != 0)

```

```

        {
            expression = expression.Where(c =>
Convert.ToInt32(c.QLScore) == measures[0]);
        }
        else if (measures[1] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.CostsScore) == measures[1]);
        }
        else if (measures[2] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.SafetyScore) == measures[2]);
        }
        else if (measures[3] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.CrimeScore) == measures[3]);
        }
        else if (measures[4] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.RacismScore) == measures[4]);
        }
        else if (measures[5] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.EducationScore) == measures[5]);
        }
        else if (measures[6] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.EnglishScore) == measures[6]);
        }
        else if (measures[7] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.WalksScore) == measures[7]);
        }
        else if (measures[8] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.PeaceScore) == measures[8]);
        }
        else if (measures[9] != 0)

```

```

        {
            expression = expression.Where(c =>
Convert.ToInt32(c.TrafficScore) == measures[9]);
        }
        else if (measures[10] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.TrafficSafetyScore) == measures[10]);
        }
        else if (measures[11] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.MedicineScore) == measures[11]);
        }
        else if (measures[12] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.HappinessScore) == measures[12]);
        }
        else if (measures[13] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.CultureScore) == measures[13]);
        }
        else if (measures[14] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.EntertainmentScore) == measures[14]);
        }
        else if (measures[15] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.InternetScore) == measures[15]);
        }
        else if (measures[16] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.WiFiScore) == measures[16]);
        }
        else if (measures[17] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.WorkPlacesScore) == measures[17]);
        }
        else if (measures[18] != 0)

```

```

        {
            expression = expression.Where(c =>
Convert.ToInt32(c.TechniqueAccessScore) == measures[18]);
        }
        else if (measures[19] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.AccomodationEasinessScore) == measures[19]);
        }
        else if (measures[20] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.HeatingScore) == measures[20]);
        }
        else if (measures[21] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.ConditionScore) == measures[21]);
        }
        else if (measures[22] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.ForeignFriendlyScore) == measures[22]);
        }
        else if (measures[23] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.SpeechFreedomScore) == measures[23]);
        }
        else if (measures[24] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.RightsFriendlyScore) == measures[24]);
        }
        else if (measures[25] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.StartupScore) == measures[25]);
        }
        else if (measures[26] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.ReligiousScore) == measures[26]);
        }
        else if (measures[27] != 0)

```

```

        {
            expression = expression.Where(c =>
Convert.ToInt32(c.CashlessScore) == measures[27]);
        }
        else if (measures[28] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.WeatherScore) == measures[28]);
        }
        else if (measures[29] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.ValueforMoneyScore) == measures[29]);
        }
        else if (measures[30] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.FoodScore) == measures[30]);
        }
        else if (measures[31] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.KindergartenScore) == measures[31]);
        }
        else if (measures[32] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.RentServicesScore) == measures[32]);
        }
        else if (measures[33] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.BankServicesScore) == measures[33]);
        }
        else if (measures[34] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.DeliveryServicesScore) == measures[34]);
        }
        else if (measures[35] != 0)
        {
            expression = expression.Where(c =>
Convert.ToInt32(c.Score) == measures[35]);
        }
        if (criterion == -1)

```

```

        {
            expression = expression.OrderBy(d => d.Score);
        }
        else if (criterion == 0)
        {
            expression = expression.OrderBy(d => d.QLScore);
        }
        else if (criterion == 1)
        {
            expression = expression.OrderBy(d =>
d.CostsScore);
        }
        else if (criterion == 2)
        {
            expression = expression.OrderBy(d =>
d.SafetyScore);
        }
        else if (criterion == 3)
        {
            expression = expression.OrderBy(d =>
d.CrimeScore);
        }
        else if (criterion == 4)
        {
            expression = expression.OrderBy(d =>
d.RacismScore);
        }
        else if (criterion == 5)
        {
            expression = expression.OrderBy(d =>
d.EducationScore);
        }
        else if (criterion == 6)
        {
            expression = expression.OrderBy(d =>
d.EnglishScore);
        }
        else if (criterion == 7)
        {
            expression = expression.OrderBy(d =>
d.WalksScore);
        }
        else if (criterion == 8)
        {

```

```

        expression = expression.OrderBy(d =>
d.PeaceScore);
    }
    else if (criterion == 9)
    {
        expression = expression.OrderBy(d =>
d.TrafficScore);
    }
    else if (criterion == 10)
    {
        expression = expression.OrderBy(d =>
d.TrafficSafetyScore);
    }
    else if (criterion == 11)
    {
        expression = expression.OrderBy(d =>
d.MedicineScore);
    }
    else if (criterion == 12)
    {
        expression = expression.OrderBy(d =>
d.HappinessScore);
    }
    else if (criterion == 13)
    {
        expression = expression.OrderBy(d =>
d.CultureScore);
    }
    else if (criterion == 14)
    {
        expression = expression.OrderBy(d =>
d.EntertainmentScore);
    }
    else if (criterion == 15)
    {
        expression = expression.OrderBy(d =>
d.InternetScore);
    }
    else if (criterion == 16)
    {
        expression = expression.OrderBy(d =>
d.WiFiScore);
    }
    else if (criterion == 17)
    {

```

```

        expression = expression.OrderBy(d =>
d.WorkPlacesScore);
    }
    else if (criterion == 18)
    {
        expression = expression.OrderBy(d =>
d.TechniqueAccessScore);
    }
    else if (criterion == 19)
    {
        expression = expression.OrderBy(d =>
d.AccomodationEasinessScore);
    }
    else if (criterion == 20)
    {
        expression = expression.OrderBy(d =>
d.HeatingScore);
    }
    else if (criterion == 21)
    {
        expression = expression.OrderBy(d =>
d.ConditionScore);
    }
    else if (criterion == 22)
    {
        expression = expression.OrderBy(d =>
d.ForeignFriendlyScore);
    }
    else if (criterion == 23)
    {
        expression = expression.OrderBy(d =>
d.SpeechFreedomScore);
    }
    else if (criterion == 24)
    {
        expression = expression.OrderBy(d =>
d.RightsFriendlyScore);
    }
    else if (criterion == 25)
    {
        expression = expression.OrderBy(d =>
d.StartupScore);
    }
    else if (criterion == 26)
    {

```

```

        expression = expression.OrderBy(d =>
d.ReligiousScore);
    }
    else if (criterion == 27)
    {
        expression = expression.OrderBy(d =>
d.CashlessScore);
    }
    else if (criterion == 28)
    {
        expression = expression.OrderBy(d =>
d.WeatherScore);
    }
    else if (criterion == 29)
    {
        expression = expression.OrderBy(d =>
d.ValueforMoneyScore);
    }
    else if (criterion == 30)
    {
        expression = expression.OrderBy(d =>
d.FoodScore);
    }
    else if (criterion == 31)
    {
        expression = expression.OrderBy(d =>
d.KindergartenScore);
    }
    else if (criterion == 32)
    {
        expression = expression.OrderBy(d =>
d.RentServicesScore);
    }
    else if (criterion == 33)
    {
        expression = expression.OrderBy(d =>
d.BankServicesScore);
    }
    else if (criterion == 34)
    {
        expression = expression.OrderBy(d =>
d.DeliveryServicesScore);
    }
    expression = expression.Include(c => c.Title)
        .Include(c => c.Score)

```

```

        .Include(c => c.QLScore)
        .Include(c => c.CostsScore)
        .Include(c => c.InternetScore)
        .Include(c => c.TransportScore)
        .Include(c => c.AccommodationEasinessScore)
    return expression;
}

public IEnumerable<City> Search(string city_title,
string country_title, byte region, List<byte> measures, float
mincosts, float maxcosts, float minaccom, float maxaccom, float
minworkspace, float maxworkspace, float mincom, float maxcom, float
minfood, float maxfood, float minleisure, float maxleisure)
{
    var expression = GetRatingByCriterion(-1, measures);
    expression = expression.Where(c =>
c.Title.Contains(city_title))
        .Where(c =>
c.Country.Title.Contains(country_title))
        .Where(c => c.Country.Region == region)
        .Where(c => c.AccommodationBudget * c.CostsLevel
>= minaccom && c.AccommodationBudget * c.CostsLevel <= maxaccom)
        .Where(c => c.WorkspaceBudget * c.CostsLevel >=
minworkspace && c.WorkspaceBudget * c.CostsLevel <= maxworkspace)
        .Where(c => c.CommuteBudget * c.CostsLevel >=
mincom && c.CommuteBudget * c.CostsLevel <= maxcom)
        .Where(c => c.FoodBudget * c.CostsLevel >=
minfood && c.FoodBudget * c.CostsLevel <= maxfood)
        .Where(c => c.LeisureBudget * c.CostsLevel >=
minleisure && c.LeisureBudget * c.CostsLevel <= maxleisure)
        .Where(c => c.CostsLevel >= mincosts &&
c.CostsLevel <= maxcosts);
    return expression;
}

public async Task CityScores(int city_id)
{
    var justcity = GetById (city_id);
    var citylivings = justcity.Livings.ToList();
    float score = 0;
    int q = citylivings.Count;
    foreach (var liv in citylivings)
    {
        score = score + liv.Score / q;
    }
}

```

```
justcity.Score = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.QLScore / q;
}
justcity.QLScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.CostsScore / q;
}
justcity.CostsScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.CostsLevel / q;
}
justcity.CostsLevel = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.SafetyScore / q;
}
justcity.SafetyScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.CrimeScore / q;
}
justcity.CrimeScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.RacismScore / q;
}
justcity.RacismScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.EducationScore / q;
}
justcity.EducationScore = score;
score = 0;
foreach (var liv in citylivings)
```

```

{
    score = score + liv.EnglishScore / q;
}
justcity.EnglishScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.WalksScore / q;
}
justcity.WalksScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.PeaceScore / q;
}
justcity.PeaceScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.TrafficScore / q;
}
justcity.TrafficScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.TrafficSafetyScore / q;
}
justcity.TrafficSafetyScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.MedicineScore / q;
}
justcity.MedicineScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.HappinessScore / q;
}
justcity.HappinessScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.CultureScore / q;
}

```

```

justcity.CultureScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.EntertainmentScore / q;
}
justcity.EntertainmentScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.InternetScore / q;
}
justcity.InternetScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.WiFiScore / q;
}
justcity.WiFiScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.WorkPlacesScore / q;
}
justcity.WorkPlacesScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.TechniqueAccessScore / q;
}
justcity.TechniqueAccessScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.AccomodationEasinessScore /
q;

}
justcity.AccomodationEasinessScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.HeatingScore / q;
}
justcity.HeatingScore = score;
score = 0;

```

```

foreach (var liv in citylivings)
{
    score = score + liv.ConditionScore / q;
}
justcity.ConditionScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.ForeignFriendlyScore / q;
}
justcity.ForeignFriendlyScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.SpeechFreedomScore / q;
}
justcity.SpeechFreedomScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.RightsFriendlyScore / q;
}
justcity.RightsFriendlyScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.StartupScore / q;
}
justcity.StartupScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.ReligiousScore / q;
}
justcity.ReligiousScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.CashlessScore / q;
}
justcity.CashlessScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.WeatherScore / q;
}

```

```

}
justcity.WeatherScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.ValueforMoneyScore / q;
}
justcity.ValueforMoneyScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.FoodScore / q;
}
justcity.FoodScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.TransportScore / q;
}
justcity.TransportScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.KindergartenScore / q;
}
justcity.KindergartenScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.RentServicesScore / q;
}
justcity.RentServicesScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.BankServicesScore / q;
}
justcity.BankServicesScore = score;
score = 0;
foreach (var liv in citylivings)
{
    score = score + liv.DeliveryServicesScore / q;
}
justcity.DeliveryServicesScore = score;
score = 0;

```

```

        foreach (var liv in citylivings)
        {
            score = score + liv.CostsLevel *
liv.AccomodationBudget / q;
        }
        justcity.AccomodationBudget = score;
        score = 0;
        foreach (var liv in citylivings)
        {
            score = score + liv.CostsLevel *
liv.WorkspaceBudget / q;
        }
        justcity.WorkspaceBudget = score;
        score = 0;
        foreach (var liv in citylivings)
        {
            score = score + liv.CostsLevel *
liv.CommuteBudget / q;
        }
        justcity.CommuteBudget = score;
        score = 0;
        foreach (var liv in citylivings)
        {
            score = score + liv.CostsLevel * liv.FoodBudget
/ q;
        }
        justcity.FoodBudget = score;
        score = 0;
        foreach (var liv in citylivings)
        {
            score = score + liv.CostsLevel *
liv.LeisureBudget / q;
        }
        justcity.LeisureBudget = score;
        score = justcity.AccomodationBudget +
justcity.WorkspaceBudget + justcity.CommuteBudget +
justcity.FoodBudget + justcity.LeisureBudget;
        justcity.AccomodationBudget =
justcity.AccomodationBudget / score;
        justcity.WorkspaceBudget = justcity.WorkspaceBudget
/ score;
        justcity.CommuteBudget = justcity.CommuteBudget /
score;
        justcity.FoodBudget = justcity.FoodBudget / score;

```

```
        justcity.LeisureBudget = justcity.LeisureBudget /  
score;  
        basecontext.Update(justcity);  
        await basecontext.SaveChangesAsync();  
    }  
}
```

ДОДАТОК В
Слайди презентації

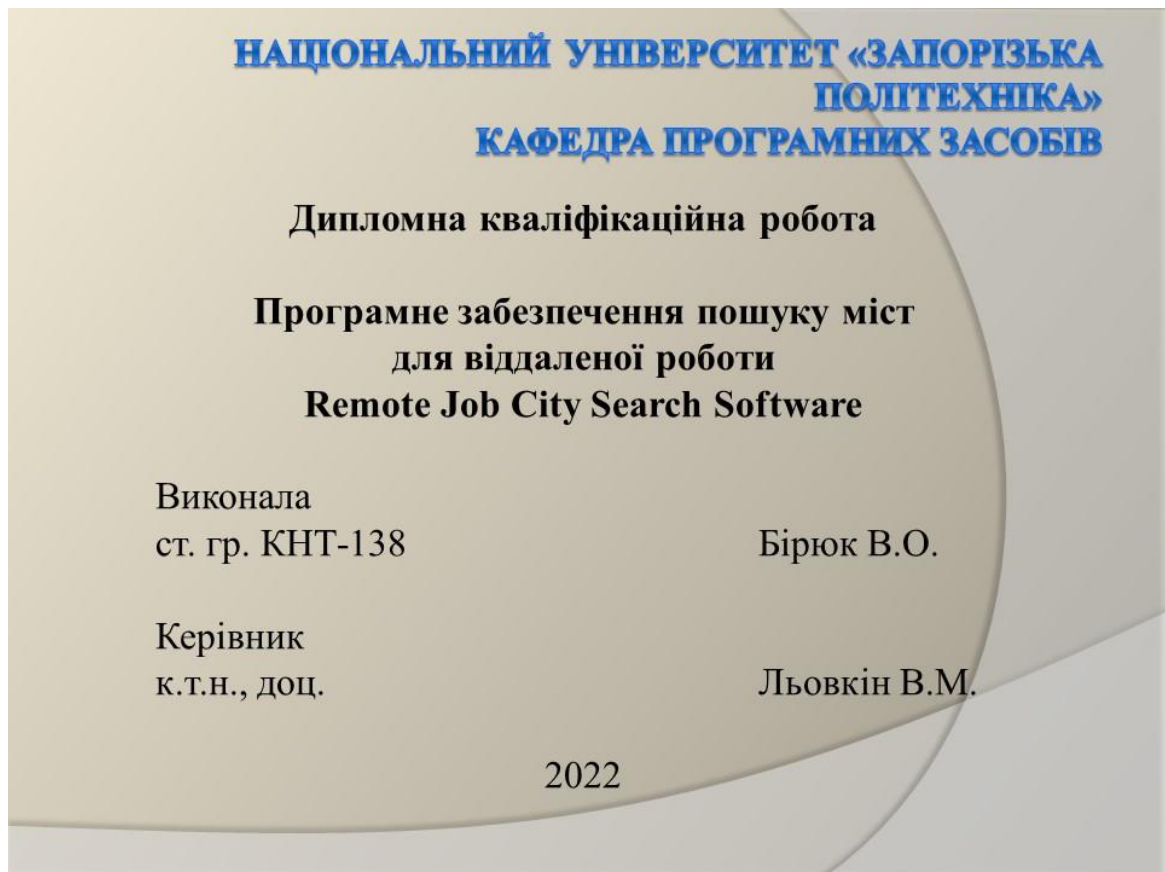


Рисунок В.1 – Слайд № 1

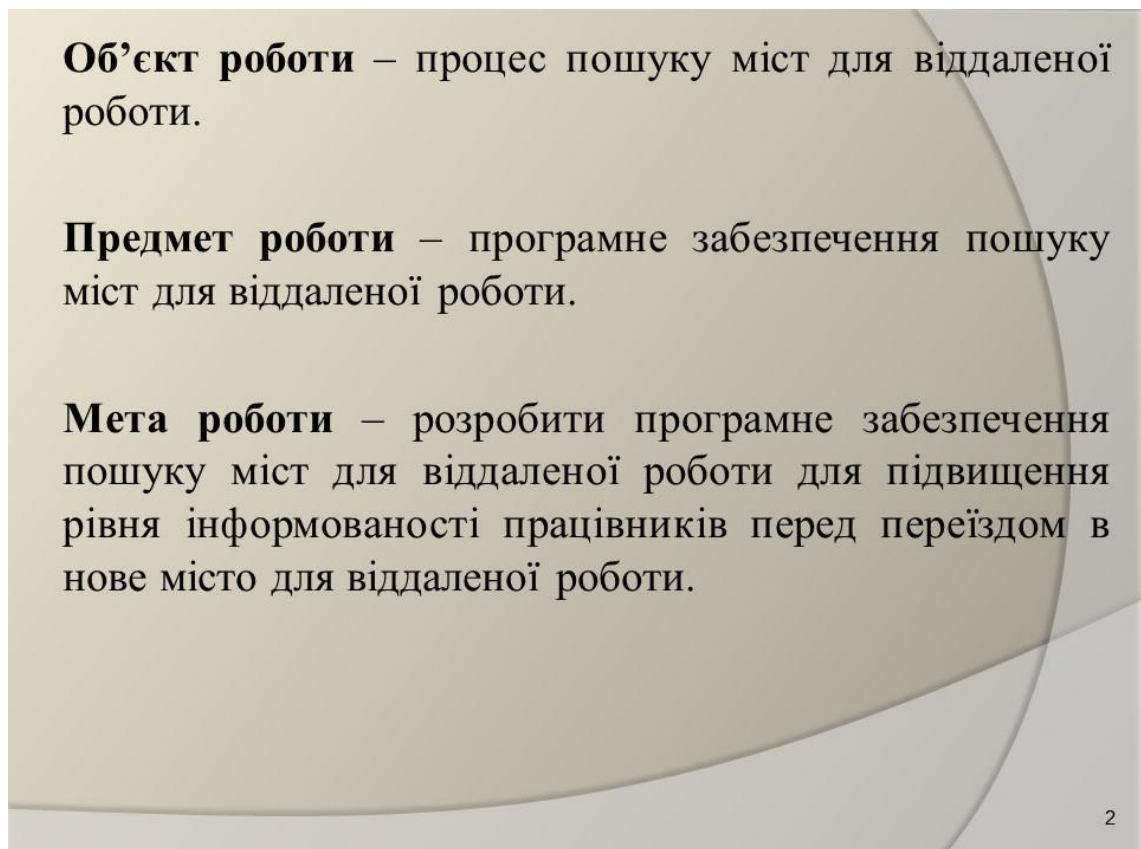


Рисунок В.2 – Слайд № 2

Завдання дипломної кваліфікаційної роботи

- аналіз предметної області;
- визначення вимог до програмного забезпечення пошуку міст для віддаленої роботи;
- проєктування програмного забезпечення пошуку міст для віддаленої роботи;
- реалізація і налагодження програмного забезпечення пошуку міст для віддаленої роботи.

3

Рисунок В.3 – Слайд № 3

Порівняння аналогів

Критерій	Digital Nomad World	Nomadlist	Дипломна робота
Пошук міст для віддаленої роботи	+	+	+
Підписування на користувачів	–	+	+
Багатокритеріальне оцінювання	+	++	+++
Відправлення повідомлень автору відгуку	–	+	+
Рекомендування місць	–	–	+
Історичний розріз даних	–	–	+

4

Рисунок В.4 – Слайд № 4

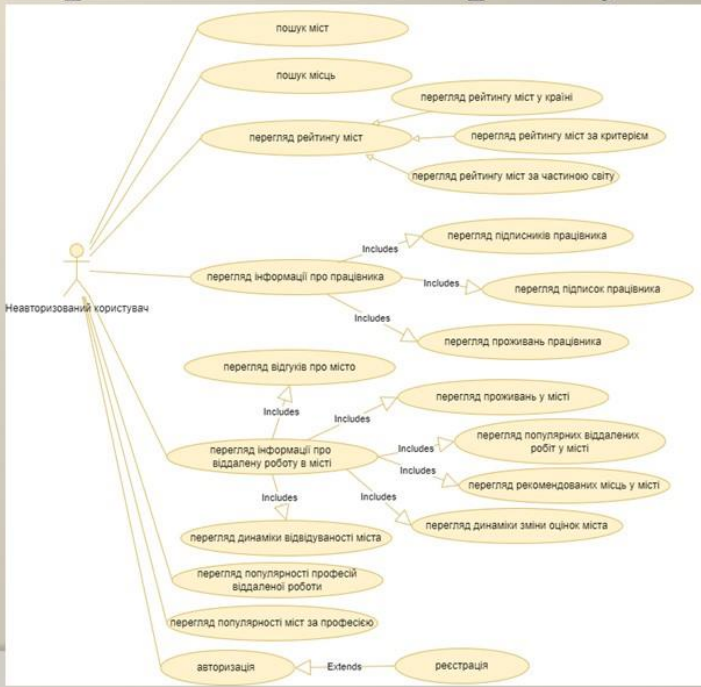
Вибір вебфреймворку

Критерій	Django	ASP.NET Core
Мова програмування	Python	C#
Кросплатформність	+	+
Велика підтримка на рівні бібліотек розширення функціональності	+	+
Багатопотокова робота	–	+
Масштабованість	+-	++
Поширеність	+-	++
Основний сегмент ринку	Малий бізнес	Середній

5

Рисунок В.5 – Слайд № 5

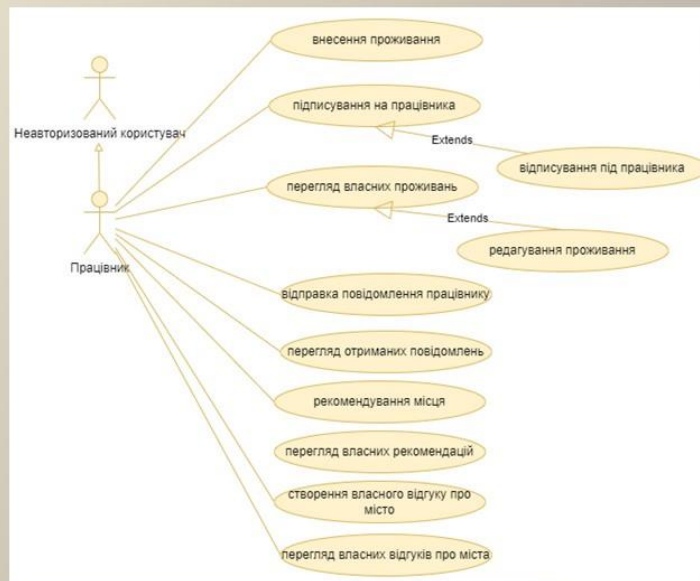
Діаграма прецедентів неавторизованого користувача



6

Рисунок В.6 – Слайд № 6

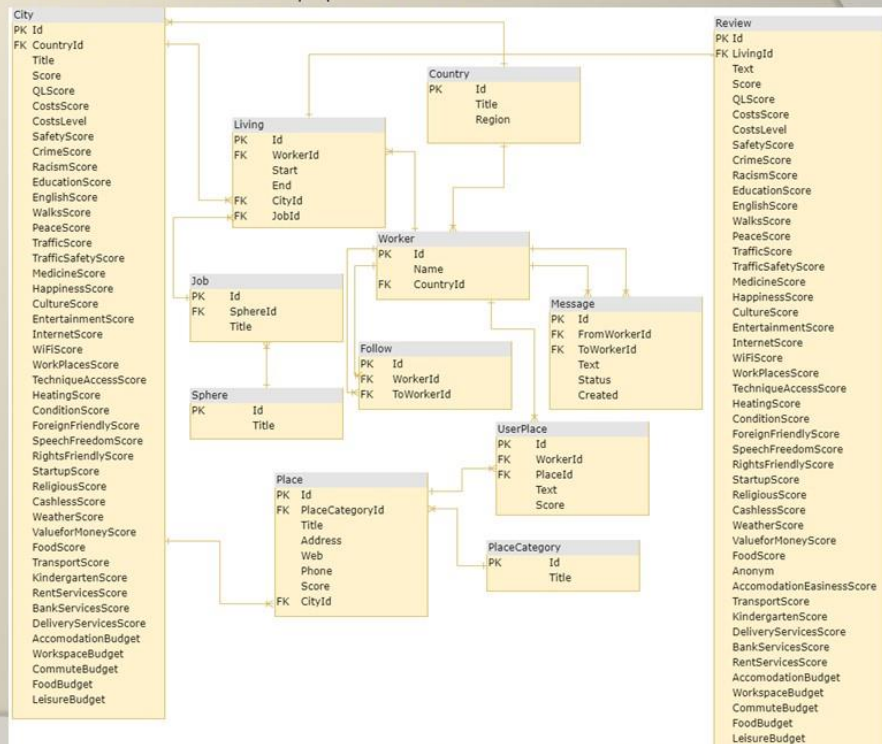
Діаграма прецедентів працівника



7

Рисунок В.7 – Слайд № 7

Схема бази даних



8

Рисунок В.8 – Слайд № 8

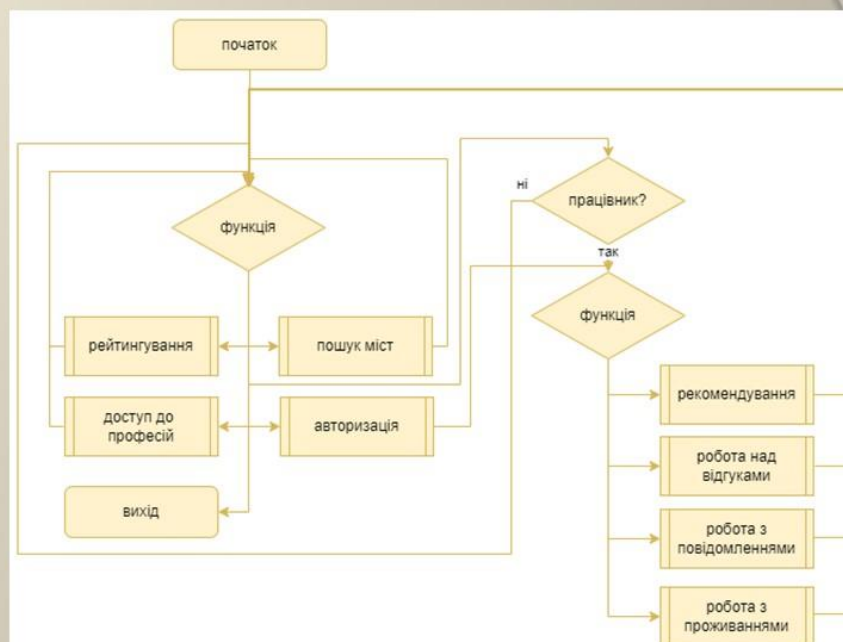
Структурна схема вебзастосунку



9

Рисунок В.9 – Слайд № 9

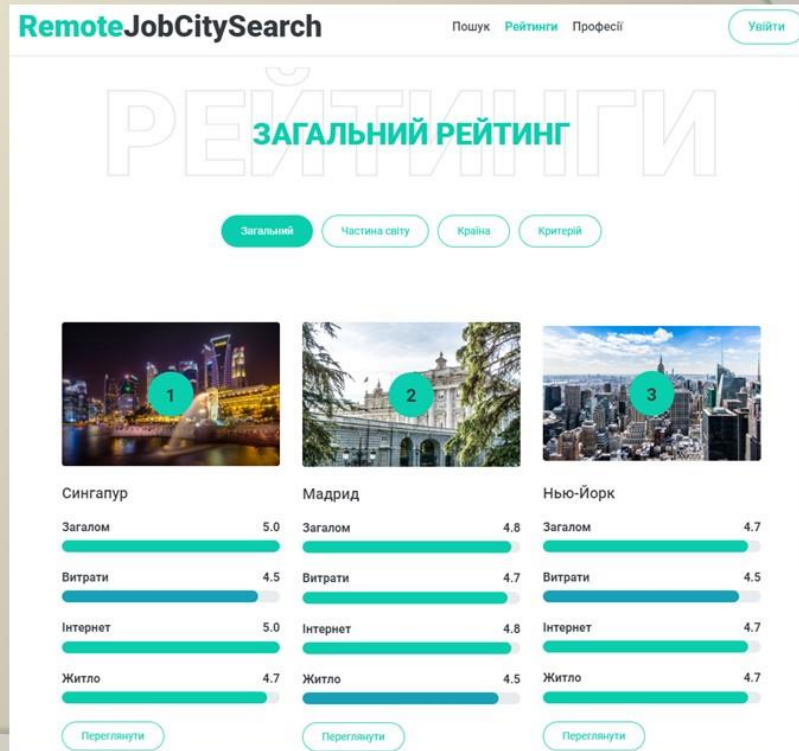
Схема функціонування програми



10

Рисунок В.10 – Слайд № 10

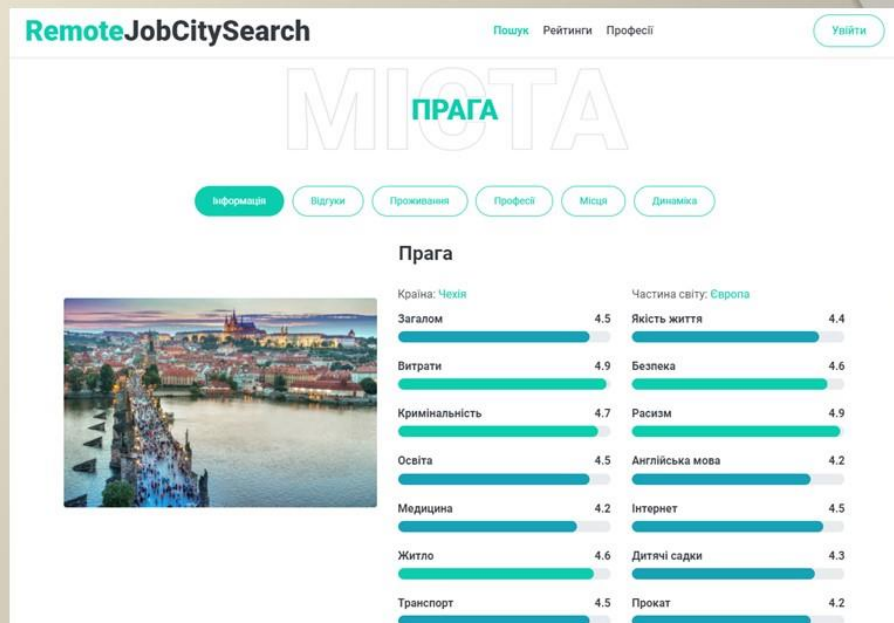
Робота з програмою



11

Рисунок В.11 – Слайд № 11

Робота з програмою



12

Рисунок В.12 – Слайд № 12

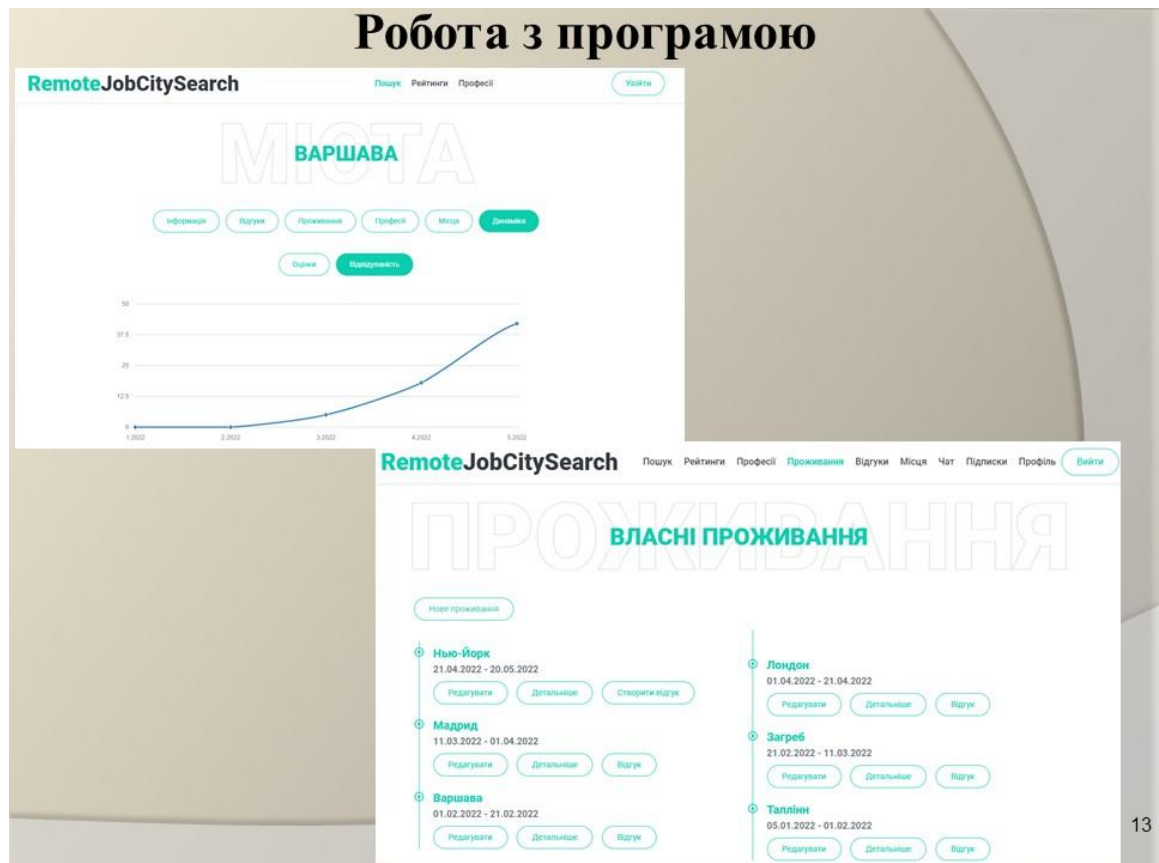


Рисунок В.13 – Слайд № 13

Результати тестування програми

Функція	Неавторизований користувач	Працівник
Внесення проживання в місті		+
Редагування внесеного проживання в місті		+
Перегляд власних проживань користувачем		+
Перегляд проживань заданого працівника в місті	+	+
Перегляд проживань в місті	+	+
Перегляд популярних віддалених робіт у місті	+	+
Створення відгуку про місто		+
Підписування на працівника		+
Відписування під працівника		+
Перегляд підписників працівника	+	+
Перегляд підписок працівника	+	+
Перегляд популярних міст за заданою роботою	+	+
Відправка повідомлення працівнику		+
Перегляд отриманих повідомлень		+
Рекомендування місця		+
Перегляд рекомендованих місць у місті	+	+
Перегляд власних рекомендацій		+
Перегляд загального рейтингу міст	+	+
Перегляд рейтингу міст за частиною світу	+	+
Перегляд рейтингу міст у країні	+	+
Перегляд рейтингу міст за критерієм	+	+
Пошук міст	+	+
Пошук місць за містом	+	+
Пошук місць загалом	+	+
Перегляд власних відгуків про міста		+
Перегляд відгуків про місто	+	+
Перегляд інформації про віддалену роботу в місті	+	+
Перегляд динаміки зміни оцінок міста	+	+
Перегляд динаміки відвідуваності міста	+	+
Перегляд популярності професій	+	+
Авторизація працівника		+
Реєстрація працівника		+

Рисунок В.14 – Слайд № 14

Висновки

За результатами виконання дипломної роботи було розроблено програмне забезпечення пошуку міст для віддаленої роботи, що дозволило підвищити рівень інформованості працівників перед переїздом в нове місто для віддаленої роботи.

Для досягнення цього було спочатку виконано огляд проблеми віддаленої роботи, підтверджено актуальність її в розрізі пошуку міст для віддаленої роботи, виконано огляд існуючих для цього рішень. Проведений аналіз став основою для виконання проектування програми, за результатами якого зокрема було виконано прийняття набору рішень, пов'язаних з використанням мови програмування C#, фреймворків ASP.NET Core і Entity Framework Core, системи керування базами даних Microsoft SQL Server для розробки, а також інших додаткових засобів. За результатами проектування програми також виконано структурування програми і даних, що відобразилось у створеній схемі БД. Ґрунтуючись на створеній структурній схемі програми було реалізовано окремі одиниці, основні класи яких було детально описано, що в підсумку дозволило реалізувати всю програму загалом. Проведену реалізацію програми було підтримано і продовжено успішним проведенням тестуванням виконаної розробки.

15

Рисунок В.15 – Слайд № 15