Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра програмного забезпечення
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка методів і програмних засобів підвищення якості експертної багатокритеріальної оцінки житлової нерухомості»

Виконав: студент 2-го курсу, групи 2ПІ-22м
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Сєріков А.І.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ
Катєльніков Д.І.
(прізвище та ініціали)
«08» грудня 2023 р.

Опонент: к.т.н., доц. каф. ОТ
Богомолов С.В.
(прізвище та ініціали)
«08» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
(прізвище та ініціали)
«08» грудня 2023 р.

Вінниця ВНТУ - 2023 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти II-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«19» вересня 2023 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Сєрікову Антону Ігоровичу

1. Тема роботи – розробка методів і програмних засобів підвищення якості експертної багатокритеріальної оцінки житлової нерухомості.

Керівник роботи: Катєльніков Денис Іванович, к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від «18» вересня 2023 р. № 247.

2. Строк подання студентом роботи
5 грудня 2023 р.

3. Вихідні дані до роботи: методи регресійного прогнозування даних – методи градієнтного бустінгу на основі дерева рішень, градієнтного спуску. Джерело отримання даних – агрегатор оголошень нерухомості DIM.RIA API, Google Maps Plafortm. Платформа розробки - .NET, ASP.NET, ML.NET. Сховище подій – Kafka. Сховище даних - PostgreSQL. Мова програмування – C#.

4. Зміст текстової частини: аналіз сучасного стану питання та обґрунтування задачі; розробка архітектури програмного забезпечення системи; проектування методів і алгоритмів багатокритеріальної оцінки житлової нерухомості; розробка контрактів взаємодії компонентів системи; розробка публічного програмного інтерфейсу системи; проектування схем баз даних; розробка програмного забезпечення; розробка алгоритмів обробки вхідних даних; розгортання

програмного забезпечення системи; тестування програмного забезпечення системи; порівняння реалізованих методів оцінки з існуючими методами; економічна частина.

5. Перелік ілюстративного матеріалу: приклади роботи аналогів систем багатокритеріальної оцінки; архітектура програмного забезпечення системи; схеми бази даних; діаграма компонентів і розгортання.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-5	Катєльніков Д.І, к.т.н., доцент кафедри ПЗ	19.09.23	05.12.23
6	Кавецький В.В, к.т.н, доцент кафедри ЕПВМ	22.11.23	01.12.23

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану питання та обгрунтування задачі	20.09.23 – 02.10.23	Вик.
2	Розробка методів і алгоритмів багатокритеріальної оцінки житлової нерухомості	03.10.23 – 23.10.23	Вик.
3	Проектування архітектури програмного забезпечення системи	09.10.23 – 23.10.23	Вик.
4	Розробка програмного забезпечення системи	24.10.23 – 13.11.23	Вик.
5	Розгортання програмного забезпечення системи	14.11.23 – 21.11.23	Вик
6	Тестування програмного забезпечення системи	17.11.23 - 25.11.23	Вик
7	Економічна частина	22.11.23 – 01.12.23	Вик.

Студент _____ **Сєріков А. І.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Катєльніков Д. І.**

АНОТАЦІЯ

Атестаційна робота містить 107 сторінок, 32 рисунки, 20 таблиць, 40 джерел, 4 додатки.

Об’єктом дослідження є процес експертного багатокритеріального оцінювання житлової нерухомості.

В результаті виконання атестаційної роботи було розглянуто предметну галузь житлової нерухомості, існуючі аналоги експертної оцінки нерухомості. Були розроблені методи підвищення якості багатокритеріальної оцінки житлової нерухомості. Розроблено приклад архітектури інформаційної системи багатокритеріальної оцінки, що базується на розроблених методах підвищення якості оцінки. Розроблено і описано програмне забезпечення системи. Було виконано порівняльний аналіз використаних алгоритмів машинного навчання, що використовуються при експертному оцінюванні житлової нерухомості. Виконано розгортання та тестування інформаційної системи.

Ключові слова: розподілена експертна система, багатокритеріальна оцінка нерухомості, інкрементальне машинне навчання, ансамблі моделей машинного навчання, функціональне масштабування експертної системи, мікросервісна архітектура.

ANNOTATION

Master degree thesis has 107 pages, 32 images, 20 tables, 40 sources, 4 additions.

The object of the research is a process of expert multifactor residential property evaluation.

As a result of the certification work, the residential property sector has been researched. In addition, existing analogues of expert multifactor residential property have been investigated. The methods of improving the quality of multifactor residential property evaluation have been developed. The architecture of information system that uses the developed methods has been developed. Software of the information system has been developed and described. A comparative analysis of used machine learning algorithms is performed. Software has been deployed and tested.

Keywords: distributed expert system, multifactor residential property evaluation, increment machine learning, ensemble of machine learning models, functional scalability of expert system, microservices architecture.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Ринок житлової нерухомості традиційно зберігає велику долю в світовій економіці, по оцінці дослідницької компанії Grand View Research [1] капіталізація ринку дорівнює 3.8 трильйонів доларів в 2022 році, і очікується зріст до 5.85 трильйонів в 2030. Потреба в забезпеченні житлової нерухомості є однієї з базових потреб людини [2], потреба в комерційній нерухомості після епідемії COVID-19 також зростає [3]. Сучасні веб ресурси з оголошеннями вже є досить ефективним інструментом, у більшості вони підтримують обов'язковий набір атрибутів в веб-оголошені;можливість додати відео,фото, інтерактивну мапу з об'єктами, тощо.

Але під час пошуку необхідного об'єкту нерухомості також виникає потреба виконати оцінку якості нерухомості, її очікуваної вартості, ліквідності тощо. Більшість існуючих засобів передбачають участь спеціалістів-аналітиків в агенціях, хоча на ринку присутні декілька працюючих прикладів публічних програмних рішень, які будуть розглянуті далі. Але навіть серед існуючих рішень є проблема точності, покриття оцінки та функціонального масштабування, коли кількість критеріїв для оцінці нерухомості постійно зростає, а модель оцінки потребує через це суттєвих змін для оновлення.

Тому розробка методів і програмних засобів підвищення якості експертної багатокритеріальної оцінки житлової нерухомості є актуальною.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення якості експертної багатокритеріальної оцінки житлової нерухомості шляхом врахування більшої кількості критеріїв у комбінованій моделі прогнозування, яка об'єднує регресійний метод градієнтного бустінгу та градієнтний спуск.

Основними задачами дослідження є:

– виконати аналіз сучасного стану питання та існуючих аналогів;

– розробити розподілену комбіновану модель прогнозування, яка об'єднує регресійний метод градієнтного бустінгу та градієнтний спуск з інкрементальним навчанням;

– спроектувати архітектуру програмного забезпечення системи;

– розробити контракти взаємодії компонентів системи;

– розробити та розгорнути програмне забезпечення системи;

– виконати тестування програмного забезпечення системи;

– виконати порівняння реалізованих методів багатокритеріальної оцінки житлової нерухомості;

– виконати аналіз економічної доцільності проекту.

Об’єктом дослідження – процес експертного багатокритеріального оцінювання житлової нерухомості.

Предмет дослідження – методи та засоби розробки розподілених експертних систем багатокритеріальної оцінки житлової нерухомості на основі комбінованої моделі прогнозування, яка об'єднує регресійний метод градієнтного бустінгу та градієнтний спуск.

Методи дослідження. У процесі дослідження використовувались: теорія баз даних при організації багатопотокової обробки даних, теорія рішень та теорія розподілених систем для побудови розподіленої експертної системи; принципи реактивного програмування; методи прогнозування, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод експертної багатокритеріальної оцінки житлової нерухомості, у якому, на відміну від відомих методів прогнозування, використовується розподілена комбінована модель прогнозування, яка об'єднує регресійний метод градієнтного бустінгу та

градієнтний спуск з інкрементальним навчанням, що дозволяє підвищити точність оцінки на 7% і швидкість оцінки в 3 рази.

2. Подальшого розвитку отримав метод навчання експертних систем, який, на відміну від відомих методів, використовує інкрементальне навчання, що дозволяє зменшити навантаження системи при наступних ітераціях оновлення моделі та підвищити точність прогнозування.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що розроблено алгоритм і програмне забезпечення для інтеграції з сервісами роботи з нерухомістю для виконання якісного пошуку житла з боку покупця, і формування потенційних очікувань від продажу з боку продавця.

Особистий внесок здобувача. У наукових працях [4], [5], [6], опублікованих у співавторстві, автору належать відповідно такі результати: архітектура інформаційної системи оцінювання; модель та методи багатокритеріальної оцінки житлової нерухомості; контракти взаємодії компонентів системи оцінювання.

Апробація результатів роботи. Основні положення магістерської дипломної роботи доповідалися та обговорювалися на всеукраїнській конференціях:

- III Всеукраїнської науково-технічної конференції молодих вчених, аспірантів та студентів. «Комп'ютерні ігри та мультимедіа як інноваційний підхід до комунікації» (28-29 вересня 2023, м.Одеса)
- Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ» (20-21 листопада, 2023, м.Суми/Вінниця)
- LII Науково-технічна конференція факультету будівництва, цивільної та екологічної інженерії. (21-23 червня 2023, м.Вінниця).

Публікації. Основні результати досліджень опубліковано в 3 матеріалах конференцій.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ОБҐРУНТУВАННЯ ЗАДАЧ

1.1 Види оцінок житлової нерухомості

Об’єкт ринку нерухомості – це нерухомість кандидат на продаж-купівлю, традиційно її можно поділити на житлову та комерційну. Комерційна нерухомість – це нерухомість, що користуються попитом у бізнеса (офіси, склади, торгові центри, ринкові площадки). Житлова нерухомість може розділятися на квартиру в многоквартирному будинку, частні будинки та приміщення (кімнати). В цій роботі буде розглянута тільки житлова нерухомість. Первенний ринок – угоди купівлі-продажи нерухомості з компанією представником забудовника. Вторинний ринок – угоди купівлі-продажи нерухомості з компаніями-посередниками(агенціями) або фізичними особами.

Аналізуючи об’єкти на ринку нерухомості є можливість відокремити такі численні оцінки як актуальна ринкова вартість та очікувана вартість оренди. Треба відмітити, що на очікувану вартість впливають велика кількість факторів, вплив яких об’єктивно різний, але найбільш впливові фактори – це локація і тип нерухомості. Наприклад, середня ціна квартири за 1 кв.м на 4 квартал 2023 в Вінниці – 1000 доларів[7], в Києві – 1874 доларів[8], в Лос-Анджелесі – 7773 доларів[9].

Серед інвесторів та спеціалістів агенцій також найбільш поширеною оцінкою нерухомості є поняття ліквідності. В базовому розумінні, ліквідність – це здатність об’єкта бути швидком проданим з максимальним збережнням очікуванної вартості. Наприклад, об’єкт низької якості, але невеликого метражу в житловому районі крупного міста, може бути проданий швидше і без втрати вартості, ніж об’єкт високої якості з високим метражем за містом, через те, що для останнього об’єкту буде важче знайти потенційного покупця. Ринкова ситуація може відрізнятися від країни, але як правило локація, інфраструктура і ціна найбільше впливають на ліквідність через те, що в більшості на дешевле житло попит завжди вище через

доступність. Для оцінки ліквідності відокремлють наступний пріорітет факторів[10]:

- Локація, інфраструктура;
- Екологія;
- Планування, кількість кімнат;
- Ціна, якість житлового комплексу, клас житла;
- Рік введення в екслутацію;

Ліквідність є вважливою складовою, якщо покупець розглядає об’єкт з точки зору заробітку в майбутньому – інвестиції.

Іншим показником є так звана валатильність – властивість об’єкта до зміни вартості. У випадку нерухомості, різка зміни ціни (велика валатнльність) у більшості ситуацій залежить від економічних і політичних подій в крайні або регіоні. Впровадження працюючої іпотеки може суттєво підняти попит, і внаслідок ціни. Слабка монетарна політика держави може призвести загалом до інфляції або знецінення вартості грошеї, що також призведе до підвищення вартості нерухомості. Іпотечна або фінансова криза, військові дії в регіоні суттєво знижують наявність платоспроміжних клієнтів, що відповідно призводить до падіння попиту, і вартості нерухомості. Всі ці фактори майже не можливо включати в оцінку і передбачення ціни, тому що навіть в сучасному світі політичні рішення і кризи дуже часто непередбачувані.

Для особистого користування більш важливим оцінкою є рівень комфортності або індекс відповідності критеріям. В цілому очевидно, що критерії комортності схожі до критеріїв ціноутворення і ліквідності. Більшість клієнтів хотіли би мешкати в районі з розвиненою транспортною та соціальною інфрастрактурої (парки, спортивні та комерційні зони, навчальні заклади), з низькою злоченністю, з гарною екологією. Але сучасні міста містять мають багато районів з різним розвитком певних факторів, що вимагає від клієнтів робити пріорітизацію. Наприклад, для частини клієнтів більш вважливіше мати доступ до

транспорту, аеропортів через роботу; для людей що працюють дома вважливіше була б спортивна і комерційна інфраструктура поруч; для сімей – садочки та школи. Правильна стратегія – це виконувати оцінку комфортності відповідно до очікування клієнта. Для цього можливо зробити певне анкетування, де клієнту треба виконати пріорітизацію факторів.

Підсумовуючи відокремомо можливі оцінки (показники) житлової нерухомості:

- Очікувана ринкова ціна за кв.м;
- Очікувана орендна ціна за кв.м;
- Індекс ліквідності;
- Індекс валатильності;
- Індекс відповідності критеріям для певного клієнта.

1.2 Критерії оцінки житлової нерухомості

Виходячи з попереднього розліку критерії оцінки можно згрупувати в певні групи:

- Якість локації;
- Якість комплексу/будинку;
- Якість житла;
- Юридичні ризики.

В групі локації в першу чергу знаходиться критерії місцезнаходження ринку – країна, регіон, населений пункт, район населеного пункту. Найбільш універсальним та унікальним індетифікатором місця є його координати. Маючи координати є можливість більш точно враховувати відстань до транспортної та соціальної інфраструктури, зон відпочинку. Населенний пункт має базовий вплив в оцінці очікуванної ринкової та орендної ціни, інші більшість критеріїв має мультиплікативний ефект на ціну.

Але навіть в одному районі ціни можуть змінюватися в залежності від доступності до інфраструктури міста. Є можливість відокремити наступні категорії міської інфраструктури[11]:

- Транспортна інфраструктура;
- Соціальна інфраструктура;
- Комерційни зони;
- Культурні зони;
- Спортивні зони;
- Зелені зони;
- Індустріальні зони (пост-індустріальні).

Транспортна інфраструктура має наступні ввиди – станція метро, зупинка тролейбуса, трамвая, автобуса, автовокзал, аеропорт, порт, автомобільні дороги. Параметром доступності є відстань в метрах до міста, але також треба враховувати тип інфрастрактури: міський або міжміський, в тому сенсі що доступ до міського є більш цінним. В цьому випадку вплив критерію не лінійний, тому що зависока близкість до транспортної інфрастрактури у більшосі має негативний вплив через стороній шум.

Соціальна інфраструктура представлена у вигляді навчальних та медичних закладів. Комерційні зони представлені торговими центрами, ринками, павільонами, закладами харчування, офісами, тощо. Культурні зони – театри, кінотеатри, опери. Спортивні зони – стадіони, спортивні клуби, вілосіпенді та бігові доріжки. Зелені зони – парки, сади, лісопарки.

В групі комплексу/будинку розглядається показники якості забудови. Є можливість відокремити наступні критерії в групі якості комплексу[12]:

- Рік забудови;
- Рейтинг забудовника;
- Кількість домів в комплексі;

- Загальна площа комплексу;
- Загальна житлова площа комплексу;
- Вартість обслуговання комплексу;
- Клас комплексу;
- Тип паркінгу, якщо існує;
- Кількість паркоміст;
- Обмеженність території (закрита чи відкрита);
- Концепція двора;
- Наявність спортивних зон і зон відпочинку;
- Наявність комерційних зон;
- Наявність охорони.

Також відокремемо основні критерії для будинку:

- Висота стелі;
- Технологія побудови;
- Матеріал стін;
- Матеріал утіплення;
- Тип опалення;
- Поверховість забудови;
- Наявність укриття;
- Кількість квартир;
- Кількість підїздів (секцій);
- Можливі планування квартир;
- Архітектурні рішення.

На ринку нерухомості може бути представлений широкий діапозон пропозицій по критерію ціна-якість. На українському ринку забудовники відокремлють чотири категорії: економ, комфорт, бізнес і преміум. Нажаль, ці категорії більше несуть маркетинговий сегс, ніж стандартизацію, але частково

різницю в якості відображають. Економ має найменшу якість, але і найменшу ціну. Преміум має найвищу якість, але і найвищу ціну. Найбільш ліквідним частіше вважається[13] клас-комфорт через найкращу співвідношення ціни та якості. Повністю покладитися на формулювання класу від забудовника не варто через часті випадки, коли вказується вищий клас житла, ніж є насправді.

Якість житла можливо описати наступними критеріями:

- Кількість кімнат;
- Планування;
- Поверх;
- Загальна площа;
- Житлова площа;
- Площа кухні;
- Стан ремонту;
- Меблювання;
- Забезпеченність технікою.

На первиному ринку об’єкти як правило здаються без ремонту, найважливішом фактором в цьому випадку є планування квартири, площи кімнати. Також важливим факторм є поверховість будинку і поверх житла. Традиційно, першій і останій поверх житла мають низький попит. На вторинному ринку об’єкти у більшості квартири реалізуються вже з ремонтом, що може суттєво вплинути на вартість нерухомості. Оцінка ремонту квартири потребує глубокої експертези відповідних спеціаліства. В оголошеннях купівлі-продажу при наявності можуть вказувати категорію стану ремонту – «житловий», «косметичний», «євроремонт», «дизайнерський», «авторський», «преміум». Чіткої стандартизації в цих категоріях немає, продавець або представники продавця дуже часто завищують категорію якості ремонту для підвищення ціни і попиту. Використовувати цей параметр в оцінці треба косвено з мінімальним коефіціетном впливу.

1.3 Аналоги та їх основні характеристики

Одним із популярних сервісів нерухумості є DIM.RIA, це крупний агрегатор оголешень про купівлю-продаж нерухомості, а також крупна база не тільки об’єктів нерухомості, але і інформації щодо рейтингу районів, забудовників, продавців, тощо. Сервіс має безкоштовний калькулятор[14] оцінки вартості житла по параметрах (див.рис. 1.1)

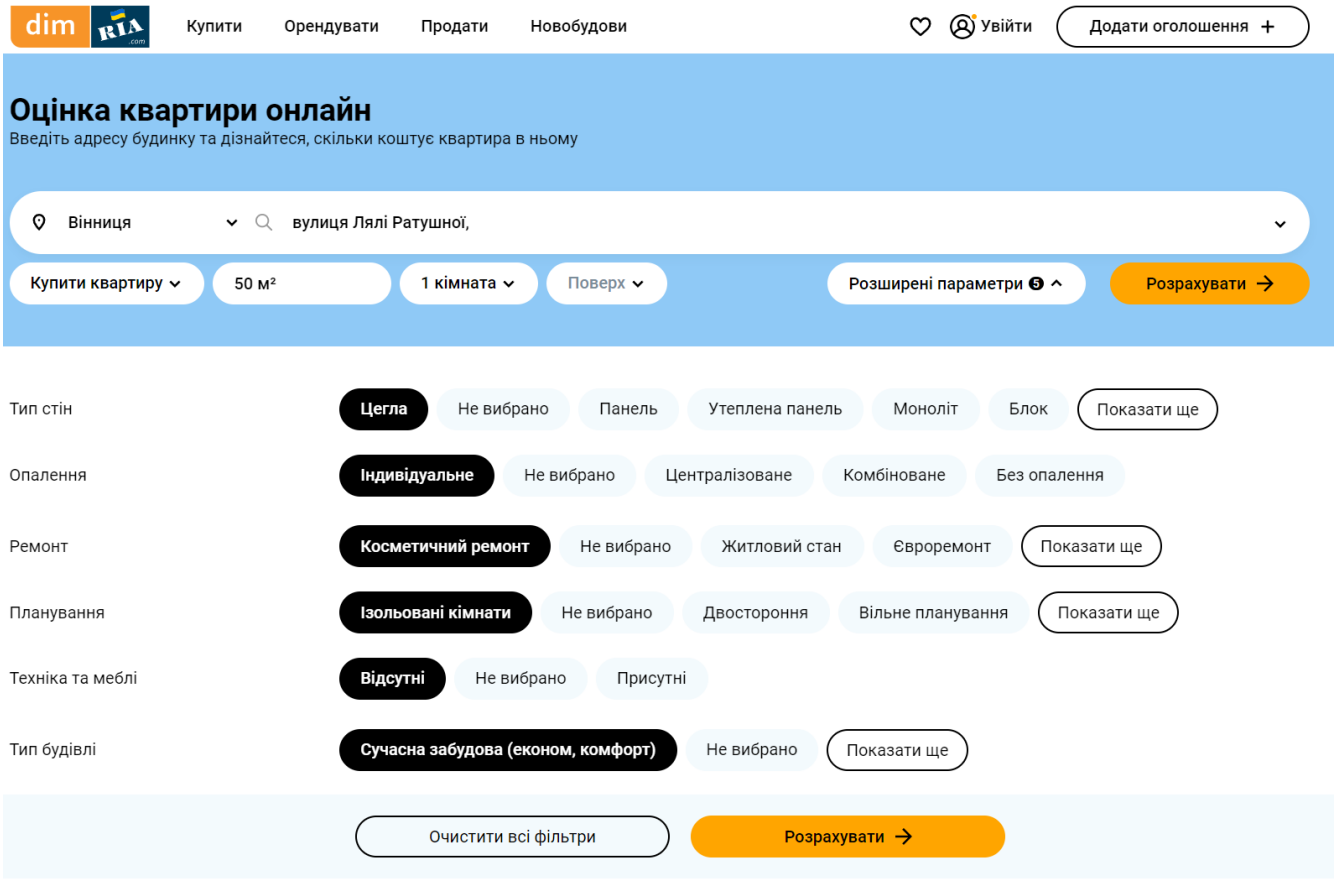

Рисунок 1.1 – Інтерфейс задання параметрів калькулятора оцінки квартири в DIM.RIA

Як можно побачити усі параметри калькулятора були розгялунті в попередніх розділах (див.розділ.1.2). Для розрахунку обов’язково потрібно вказати очікувану локацію та жилову площу об’єкта, кількість кімнат. Також обов’язковим є тип стін, усі інши параметри є опційними. Результати розрахунку сервісу відбуваються

досить швидко (200-300 мс), що очевидно свідчить про вже виконання прогнозування вже на побудованій моделі. Якщо виходити з калькулятора, усього в оцінці приймає участь приблизно 11 параметрів. В результаті розрахунку калькулятор виводить середню ціну за об’єкт, середню ціну за квадратий метр і також розподіл вартості об’єктів в залежності від кількості оголешень. Також сервіс пропонує передивитися список актуальних пропозицій, які найбільше підходять до вибраних фільтрів.

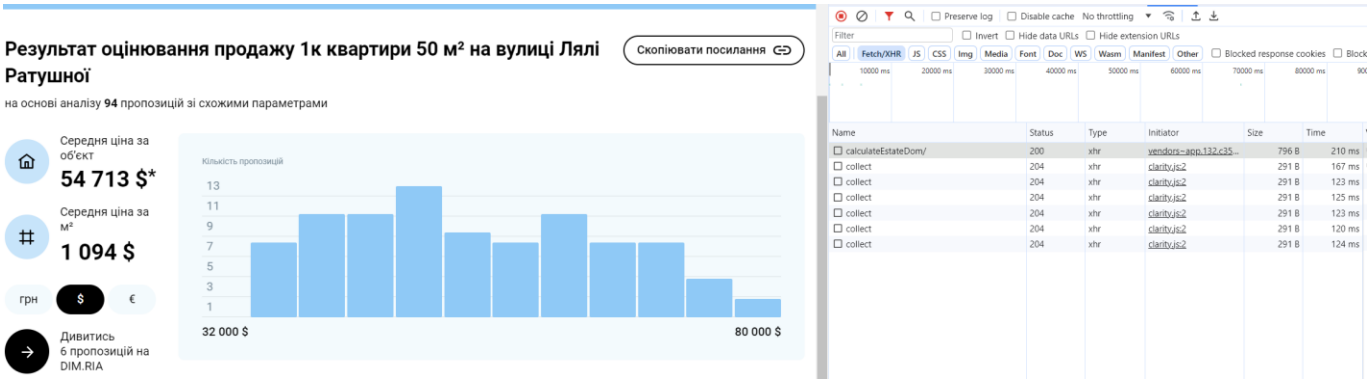

Рисунок 1.2 – Результат оцінки нерухомосоті по вказаним параметрам в DIM.RIA

Сутєвою властивістю роботи сервісу є те, що він працює з прогнозованою квартирою, не інтегрований з реальними оголошеннями. З одного боку це дає можливість працювати з гіпотетичними пропозиціями, але з другого боку відстутність ітеграції створює не зручність по ручному переносу фільтрів в калькутор, і по великому рахунку переносить частково відповідальність за помилку на клієнта.

Іншим сервісом, який підтримує можливість оцінки вартості житлової нерухомості є otodom від польської групи Grupa OLX[15]. Результати оцінки інтегровані з оголешенням, що є зручним рішенням. Звісно, метод оцінки є закритим, але сервіс стверджує, що виконує оцінку за допогою 140 критерії, в тому рахунку – опис оголешення, локація, інфраструктуру та оточення об’єкту (див.рис.1.3).

Рисунок 1.3 – Результат оцінки нерухомосоті по вказаним параметрам в otodom

Окремо треба відмітити те, що результат оцінки є орієнтованний діапозон значень, це дає можливість зрозуміти можливі коливання цін і точність розрахунку. Відображення вартості поточного об’єкту в діапозоні вказує наскільки об’єкт є переоціненим або недоціненим з точки зору сервіса.

1.4 Постановка завдання

Мета роботи є підвищення якості експертної багатокритеріальної оцінки житлової нерухомості. Що таке якість оцінки? Це точність результату і швидкість оціювання. У випадку представлені очікуванної вартості нерухомості в діапозоні, точність відображається співідношенням різниці в діапозоні до максимального значення. Швидкість оцінюваннння не є пріорітетною, якщо розглядати аналіз одного об’єкта нерухомості,але у випадку потреби зформувати очікувану вартість по багатьом об’єктам для порівняння вже суттєво впливає на очікування користувача.

Також є потреба враховувати швидкість побудування і оновлення моделі як

набір індексів, по яким вже буде виконуватися оцінювання. Потенційний ризик – це велика кількість факторів, і велика кількість оголешень щодо об’єктів нерухомості, і супутніх даних таких як інформація щодо оточення об’єкту, рейтинги компаній, статистика по використанню інфраструктури довколо об’єкту, тощо. По-перше, це потребує реалізації моделі і архітектури програмного забезпечення з високим рівнем функціонального масштабування – тобто можливості додати нову групу показників, що впливають на оцінювання, або новий тип оцінки з мінімальним ризиком до існуючої системи. По-друге, це потребує навантажульваного масштабування, тобто мати можливість додавати нові ресурси в систему з подальшим розподілом наватаження. Також треба враховувати, що для підтримки точності моделі оцінки, довидиться частіше виконувати її оновлення (особливо на початку її побудови), що вимагає більшого контролю над согласовоністю даних моделі, не допуску блокувань оновлень та перезапису.

У підсумку потрібно вирішити наступні завдання:

– розробити методи і алгоритми багатокритеріальної оцінки житлової нерухомості;
– спроектувати архітектуру програмного забезпечення системи;
– розробити контракти взаємодії компонентів системи;
– спроектувати схеми бази даних;
– розробити програмне забезпечення системи;
– виконати розгортання програмного забезпечення системи;
– виконати тестування програмного забезпечення системи;
– виконати порівняння реалізованих методів багатокритеріальної оцінки житлової нерухомості;
– виконати аналіз економічної доцільності проекту.

Система також підтримувати ряд нефункціональних вимог:

– функціональне масштабування;
– підтримка согласовоності даних;

– навантажувальне масштабування;
– швидкодія в оновленні та зчитані моделі оцінки.

1.5 Висновок

В цьому розділі було виконано аналіз предметної області. Серед можливих типів оцінок житлової нерухомості було відокремлено:

- Очікувана ринкова ціна за кв.м;
- Очікувана орендна ціна за кв.м;
- Індекс ліквідності;
- Індекс валатильності;
- Індекс відповідності критеріям для певного клієнта

Були проаналізовані потенційні фактори, що можуть впливати на вказані оцінки житлової нерухомості; було розглянуто можливу пріорітетність впливу певних факторів. Серед можливих груп критеріїв оцінки житлової нерухомості було відокремлено:

- Якість локації;
- Якість комплексу та будинку;
- Якість житла;
- Юридичні ризики.

Було виконано аналіз роботи систем-аналогів: DIM.RIA та otodom. Було виявлено ряд переваг і недоліків в поточних реалізаціях аналогів, зроблені відповідні висновки щодо функціональних та нефункціональних вимоги до системи оцінювання.

Розроблено постановку завдання з описом необхідних задач для вирішення та нефункціональних вимог, які необхідно підтримувати.

2 РОЗРОБКА МЕТОДІВ ПІДВИЩЕННЯ ЯКОСТІ БАГАТОКРИТЕРІАЛЬНОЇ ОЦІНКИ НЕРУХМОСТІ

2.1 Розробка методів багатокритеріальної оцінки нерухомості

Для побудови моделі навчання необхідно обрати зформувати групу критеріїв (атрибути), набір даних та необхідний алгоритм. Суттєва вимога-проблема – це необхідність підтримувати велику кількість факторів. Це призводить до складності моделі навчання, необхідності оперування великою кількістю даних, більш складне оновлення моделі на нову версію без простою. Одним з підходів рішення є ідея «model parallelism» - розподілення моделі навчання між декількома окремими вузлами (див.рис.2.1).

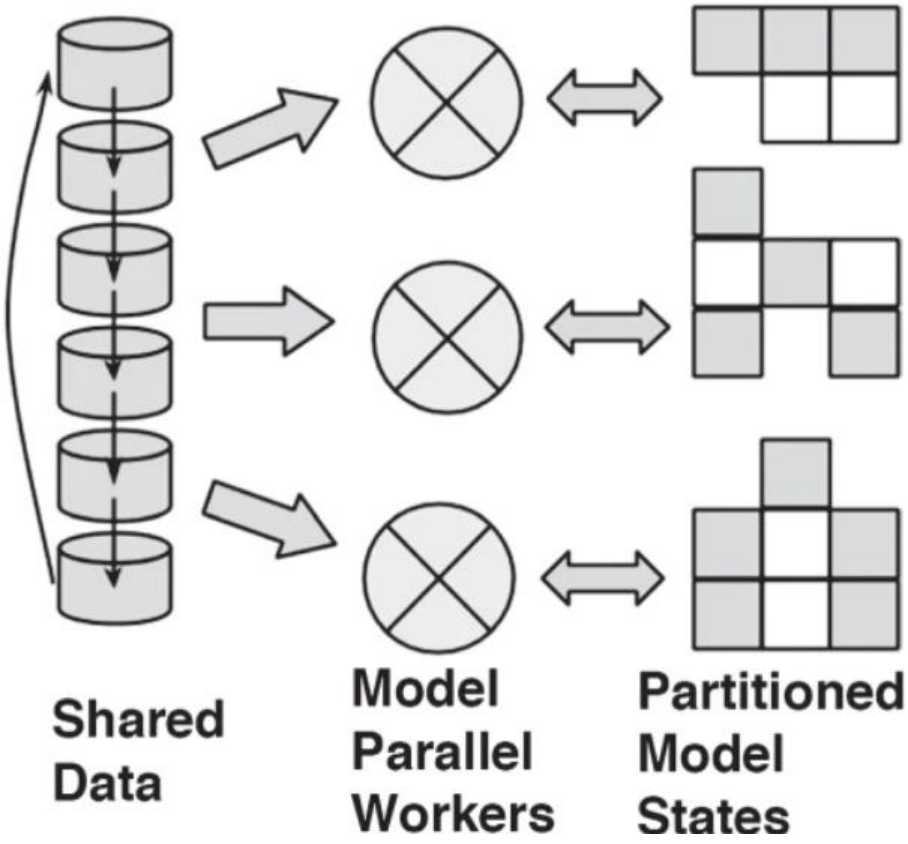

Рисунок 2.1 – Схема використання параллельної моделі навчання

Модель навчання розбивається на розділи (партиції) по певній групі атрибутів. В поточній задачі, ми можемо представити це групами, які були проаналізовані в розділі 1. Для кожної групи відокремелюється окремий(і) «worker», які відповідають за побудову власної моделі і аналіз певної частки моделі і зберігання відповідної статистики, коефіцієнтів атрибутів. У результаті маємо декілька часткових моделей, для яких потім виконується операція злиття в одну модель, яка вже готова до використання. Асоціативно це нагадує побудову пазлу окремими частками. Які переваги це надає у порівнянні з стандартною моделю навчання без резподілу? Можливість розбити велику групу факторів – розподілити відповідальність, більше можливостей до масштабування комп'ютерних ресурсів на вузлі «worker» (оперативна пам'ять, база даних, CPU). У випадку потреби додавання нового фактору потрібно оновити тільки відповідний «worker », а не всю систему. У випадку додавання нової групи факторів потрібно додати новий worker, але інші не мають бути зачеплені. Можливо потрібно буде оновити конфігурацію на певних вузлах для розуміння що розподіл змінився, але вже існують рішення, які не потребують зупинки вузла для виконання цих змін. Які недоліки ісують у порівнянні зі стандартною моделю навчання без розподілу? Більш ускладнена архітектура, необхідність підтримувати координацію розподілених worker. Підхід виправдовуються тільки у випадку моделі з великою кількістю аттрибутів, які можливо згрупувати та нормалізувати, та при використані великої кількості даних (екземплярів). Представемо що в оцінкі очікуваної вартості нерухомості ω приймає участь N критеріїв, які можно розподілити K груп:

$$K = \begin{cases} A = f(a_0, a_1, a_2, \dots a_n) \\ B = f(b_0, b_1, b_2, \dots b_n) \\ C = (c_0, c_1, c_2, \dots c_n) \\ \dots \\ Z = (z_0, z_1, z_2, \dots z_n) \end{cases} \quad (2.1)$$

де A, B, C – це відповідні оцінки за певну групу факторів; a_i, b_i, c_i – це відповідні атрибути в групі факторів.

Наприклад, група A групує фактори якості локації, група B – фактори якості комплексу та будинку, група C якості житала (квартири або будинку); група D – юридичні ризики.

Кінцева оцінка тоді формуються наступним чином, як функція над оцінками по кожній групі факторів A, B, C.

$$W = f(A, B, C \dots D) \tag{2.2}$$

Представити побудову моделі для оцінки можливо наступним чином (див. рис. 2.2):

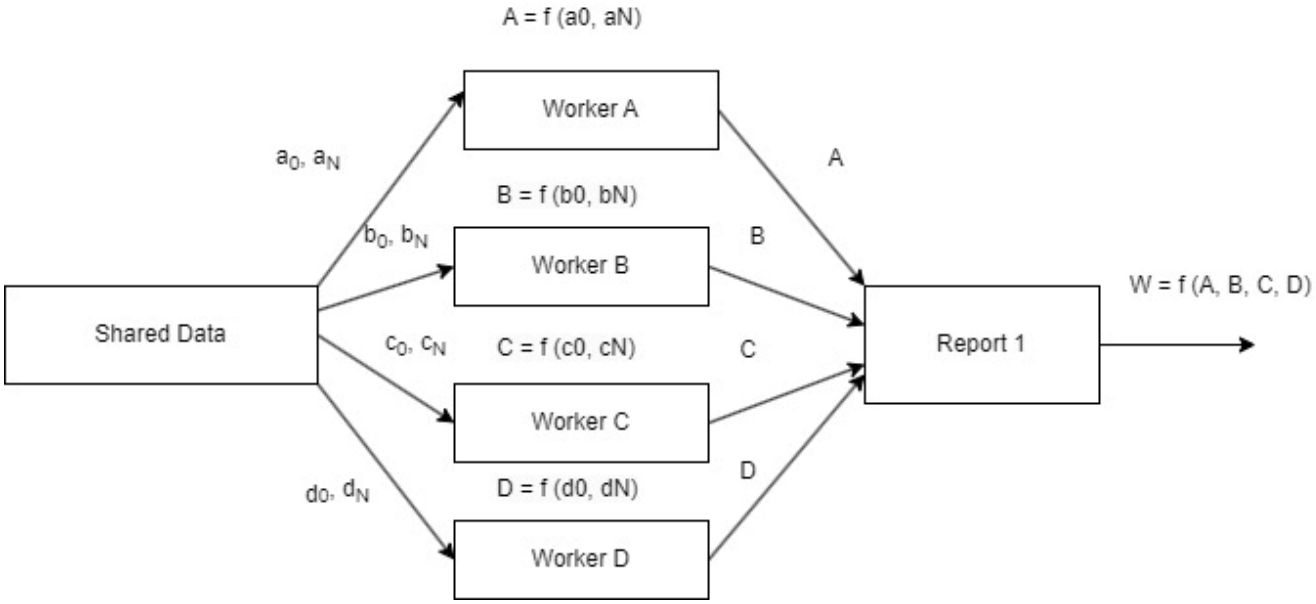

Рисунок 2.2 – Розподіл оцінювання між worker

Алгоритм багатокритеріальної експертної оцінки матиме наступну послідовність:

1. Отримання та збережння даних в загальному сховище;
2. Передача або читання даних з виборкою необхідних групових атрибутів кожним окремим «worker» - окремим вузлом (або кластером вузлів) відповідальним за оцінку групи факторів;

3. При обробці нових даних «worker» виконує створення/оновлення моделі передбачення, і також виконує оцінку, якщо існуюча версія моделі вже достатньо зформавана для оцінки;
4. Після формування оцінки для отриманного екземпляру даних кожен «worker» виконує публікацію або зберігання цієї оцінки для «reporter»;
5. Кожен «reporter» після отримання або читання необхідних оцінок від експертів формує власну модель оцінки, оцінює екзпемляр згідно неї, і зберігає кінцеву оцінку для передачі користувачу.

Кожен «worker» (експерт) та «reporter» може мати власне сховище даних, окремі алгоритми, які більше підходять для виконання відповідальної експертизи. Його відповідальність це отримати або зчитати загальні дані, виконати аналіз-передбачення за свою групу, та передати результат для оцінки. У випадку якості локації це отримання параметрів, щодо того як певна локація впливає на вартість нерухомості. Кожен «worker» (експерт) як навчається на отриманих даних об’єктів, так і виконує необхідну оцінку об’єкта. Деякі «worker» можуть залежити від експертизи других «worker», але вони не можуть бути залежними від результатів оцінювання «reporter».

2.2 Розробка архітектури програмного забезпечення

Оцінка вартості нерухомості по суті є передбачення ціни на поточний час виходячи з даних існуючих угод або оголошень. Першою складністю є кількість реальних факторів, що впливають на оцінку. Як було вказано в розіділі 2.1, їх можливо згрупувати в наступні первинні групи: локація (в тому числі і інфраструктура), якість будинку, якість житла (квартири), стан ремонту та обладнання (меблі, техніка), юридичні ризики. Це потребує від системи мати функціональне масштабування, тобто модель оцінки має мати можливість бути розширена аналізом нових факторів в наступних версіях без суттєвих проблем.

Одним із рішенням може бути розподіл системи на набір сервісів (мікросервісів) кожен з яких матиме відповідальність за певну групу факторів. Це також потенційно дозволить виконати розподіл[16] аналізу на декілька окремих процесів, але в той же час потребує більш детального вивчення питання координації роботи сервісів та контрактів взаємодії.

Другою складністю може бути об’єм даних по нерухомості, і враховуючи кількість факторів це створює потенційне питання в очікуваному часі на аналіз. Цілком очікувано, що модель аналізу має бути збережена як в постійній пам’яті, так і в кеші. В той же час ризикованим рішенням буде виносити процес в синхронне очікування результату аналізу (наприклад, HTTP виклик), велика ймовірність що час на оцінку буде більше, ніж стандартне очікування користувача на операції в інформаційних системах (при очікуванні більше 1 секунди користувач втрачає відчуття прямої роботи з даними[17]). Більш коректним рішенням є використанням асинхронних операцій[18] – обробка даних з шини і публікація результату аналізу також в шину, після якого користувач отримає нотифікацію про висновок експертизи. Для зменшення об’єму даних при передачі, який може бути великий через потокову обробку, варто виконати мінімізацію даних в контрактах (наприклад, використання нумерованих списків замість строк), а також використовувати більш «економні» протоколи передачі даних – бінарні замість текстовий форматів (наприклад, протокол Protobuf замість поширеного JSON).

Третьою складністю є потенційна велика кількість джерел даних, і потреба без зупинки роботи системи вводити нові джерела даних. Різні джерела даних мають різні протоколи та правила взаємодії (кількість запитів за хвилину, загальна кількість допустимих запитів, тощо). Це потребує створення окремих фонових процесів-виконавців, які будуть отримувати дані з джерел, приводити до одного прийнятого формату і публікувати на обробку в шину.

В кінцевому варіанті відобразити потенційну загальну систему можливо наступним чином(див.рис.2.3)

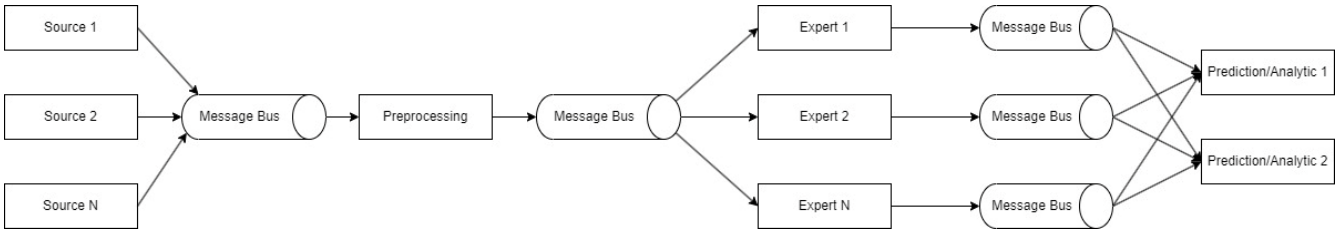

Рисунок 2.3 – Компонентна діаграма взаємодії в системі

Згідно з цією діаграми компонент Source представляє собою процес по роботі з джерелом даних, який отримує дані зі стороннього сервісу та надсилає дані у шину. На другому етапі дані проходять попередню обробку під час якої відбувається форматування та довантаження додаткових даній, отриманий результат даних надсилається у шину. На третьому етапі дані по певній групі факторам надсилаються сервісам-експертам, які виконують аналіз і навчання в відповідній зоні відповідальності. Цей процесс називається інкрементальне навчання. Треба відмітити, що малоймовірно що сервіси-експерти будуть діяти повністю паралельно та незалежно, існує ймовірність що для певної експертизи буде необхідна інша експертиза. Результат експертизи публікується в шину і вже використовується для виконання оцінок та звітів.

Чи можливо уникнути використання окремих компонентів – шин даних або сховища повідомлень? Можливо використання звичайної реляційної бази даних у цьому випадку, але це потребує реалізації додаткової відповідальності за координування процесу зчитання та балансування процесів-читачів, чого краще уникнути маючи можливість використання перевірених інструментів.

2.3 Обґрунтування використаних програмних залежностей

У якості розподіленого сховища повідомлень-подій буде використана система Apache Kafka. Apache Kafka – це розподілена система обміну повідомленнями, яка призначена для обробки потокових даних та розподілу повідомлень між різними компонентами системи. Повідомлення надсилаються в

систему через Producer, отримується за допомогою Consumer (див.рис.2.2). Розподіл навантаження відбувається за допомогою розділення топіку (абстрактний тип сховища) на партіції. Партіції розподіляються згідно з кількістю Consumer. Наприклад, при наявності топіку з 5 партіціями і два окремих Consumer, один з Consumer возьме дві партіції, інший три. Apache Kafka самостійно виконує балансування[19] партіцій, це потребує певного часу, яке залежить від кількості партіцій і Consumer.

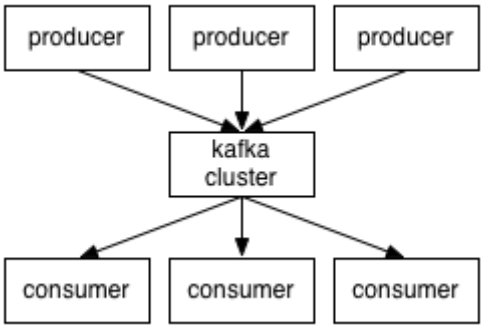

Рисунок 2.4 – Обробка подій в системі Apache Kafka

Apache Kafka має можливість реплікації даних, тобто при додавані повідомлення в систему, повідомлення зберігається як в лідері, так і додаткових репліках. У випадку відмови лідера, його місце займає одна із реплік. Таким чином відбувається підвищується надійність програмного забезпечення. Існують дві стратегії збереження даних в репілку – синхронно або асинхронно. У випадку синхронної реплікації, відбувається очікування з боку лідера, що дані успішно були збереженні в репліці. Асинхрона реплікація – лідер не очікує збереження даних в репліці. Відповідно, в системі вказується фактор реплікації – кількість реплік, і мінімальна кількість синхроних реплік. З одного боку наявність синхронної репліки матиме можливість у випадку збою лідера отримати більш актуальні дані, ніж при асинхронній репліці, але в той же час необхідність підтвердження забирає певний час при збережені даних. Звідси маємо класичну ситуацію балансу – підвищуюци

кількість синхронних реплік, ми підвищуємо надійність, але зменшуємо продуктивність системи. Стандартною конфігурацією вважається наявність одної синхронної репліки та декількох асинхроних.

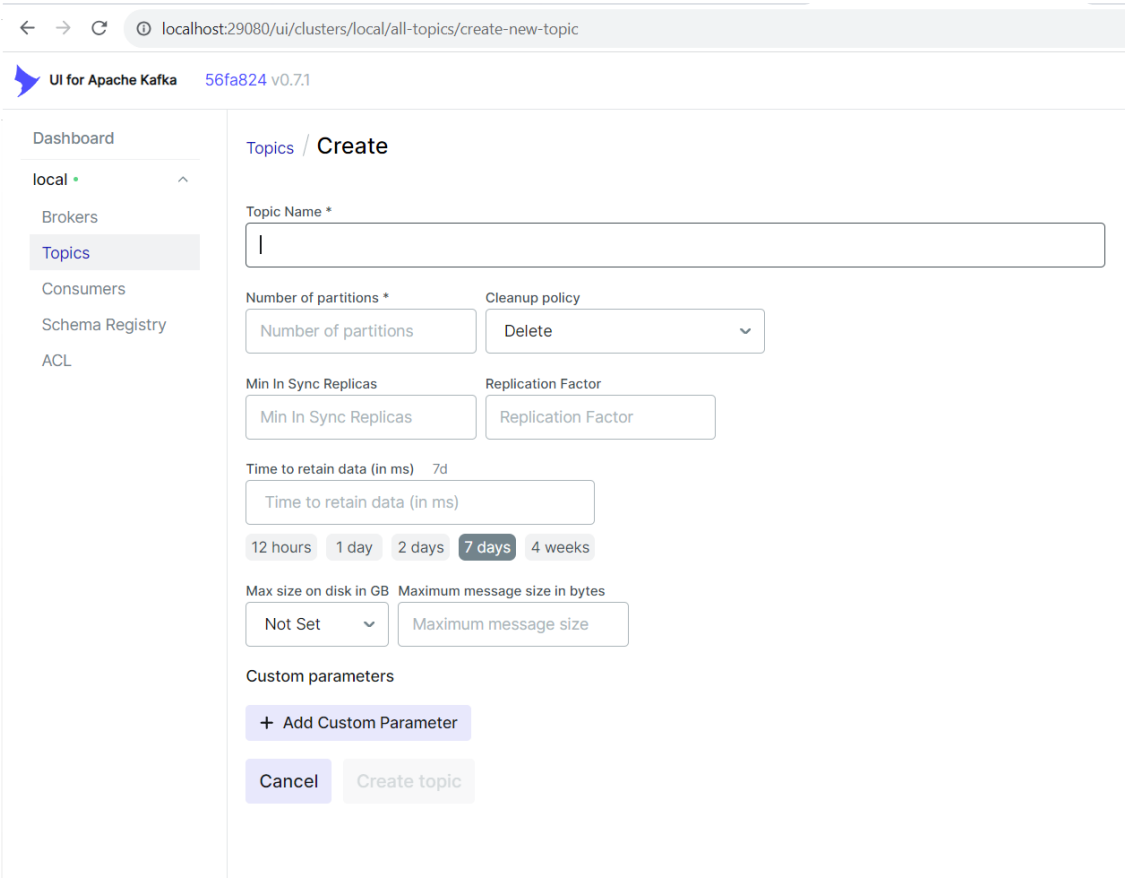

Рисунок 2.5 – Форма конфігурації топіка в Apache Kafka

Іншою перевагою є висока масштабованість Apache Kafka. Система дозволяє динамічно додавати як нові партіції, так і нові Consumer, але звісно це потребує виконання перебалансування Consumer між партіціями.

Альтернативним популярним рішенням у ролі шини даних є RabbitMQ, який також можно використовувати в розподіленній архітектурі, і RabbitMQ підтримує гарозинтальне масштабування. Але Apache Kafka має більш зручну стратегію балансування навантаження з підтримкою розподілу повідомлень до певних партіцій згідно ключа повідомлення. Тобто повідомлення оброблюється послідовно

і повідомлення з одним ключем потрапляють в одну партіцію, і відповідно будуть оброблені одним Consumer. Найбільш суттєвою різницією є те що Apache Kafka є сховищем повідомлень-подій, а RabbitMQ тільки брокер, тобто RabbitMQ видаляє повідомлення з черги після обробки, Apache Kafka залишає повідомлення згідно конфігурація. Для розуміння, що повідомлення вже оброблено Apache Kafka зберігає і оновлює offset (індекс від початку партіції). Службова інформація Apache Kafka зберігається в системі Zookeper.

У якості платформи розробки сервісів була обрана платформа .NET. .NET – відкрита платформа розробки від компанії Microsoft, як кросплатформене еволюційне продовження .NET Framework. Система .NET є провіренним часом рішенням з різноманіттям вже реалізованик бібліотек, пакетів та паттернів розробки. В тому рахунку існують вже бібліотке для інтеграції з Apache Kafka та з базами даних, HTTP клієнти. Кросплатформаність рішення надає можливість без проблем створювати Docker образи, і використовувати в таких хмарних рішеннях як AWS, Azure, Google Cloud, тощо. При розробці під платформу .NET можно використовувати різні мови, але найпоширенішою є C# - об’єктно-орієнтовна мова програмування з статичною типизіацію та автоматичною збіркою пам’яті. Також, .NET є відкритою системою розробки з MIT ліцензією, тобто дозволяється використовувати в закритом програмном забезпеченні.

У якості реляційної бази даних для «warehouse» і «transaction processing» буде використаний PostgreSQL. PostgreSQL – це вільна та відкрита система управління базами даних (СУБД). Основні переваги PostgreSQL – це надійність і розширюваність. PostgreSQL гарантує надійну роботу та збереження даних навіть після аварій. Також PostgreSQL має високу швидкість та оптимізацію обробки великих обсягів даних[20].

У якості розподіленного кеша використовуються Redis. Redis є дуже поширенним рішенням in-memory бази даних для зберігання ключ-значення. Основними перевагами Redis є швидкодія (через те, що дані зберігаються в ОЗУ),

підтримка різних типів даних (ключ-значення, масив значень, хеш-таблиця, тощо), масштабованість, можливість використання в розподіленній системі та обробки великої кількості запитів. Недоліки виходять з фізичною моделі зберігання даних – ОЗУ: це обмежений обсяг пам’яті, втрата даних при перевантаженні,і відсутність можливості обробки запитів до ієрархічних та реляційних структур даних, але таких вимог до кеш-па’яті і немає, складні запити мають оброблюватися в рамках патерну «data warehourse» в реляційних та документних базах даних.

2.4 Опис принципів взаємодії компонентів системи

Для між серверної взаємодії з використанням Apache Kafka можуть бути використані різні протоколи серіалізації даних – текстові та бінарні. Основна перевага текстових, це відсутність потреби підтримувати окремо схему даних тому що дані вже відображають схему, але недоліком є затратне кодування[21]. Наприклад, JSON досить лаконічний в текстову форматі в форматі ключ-значення, але для ідентифікації атрибут потребує кодувати і передавати ключ.

В бінарному протоколі Protobuf поля ідентифікується не текстовим ключом, а числовим номером. Серіалізація і десеріалізація відбуваються через адресацію по номеру та очікуваному розміру даних в атрибуті згідно типу даних. Protobuf є мовнонезалежним механізмом передачі даних і підтримує версіонування контрактів. Додавання опціонних нових полів не порушує зворотну та пряму сумності, але видалення поля порушує пряму сумісність, а додавання нового обов’язкового поля порушує зворотну сумісність. Логічно також, що зміна номера для існуючого є несумісною зміною.

Назва атрибуту (ключа) залишається, але не використовується в процесі в серіалізації, під час десеріалізації використовується для переводу в створенні об’єкта (див.рис 2.3).

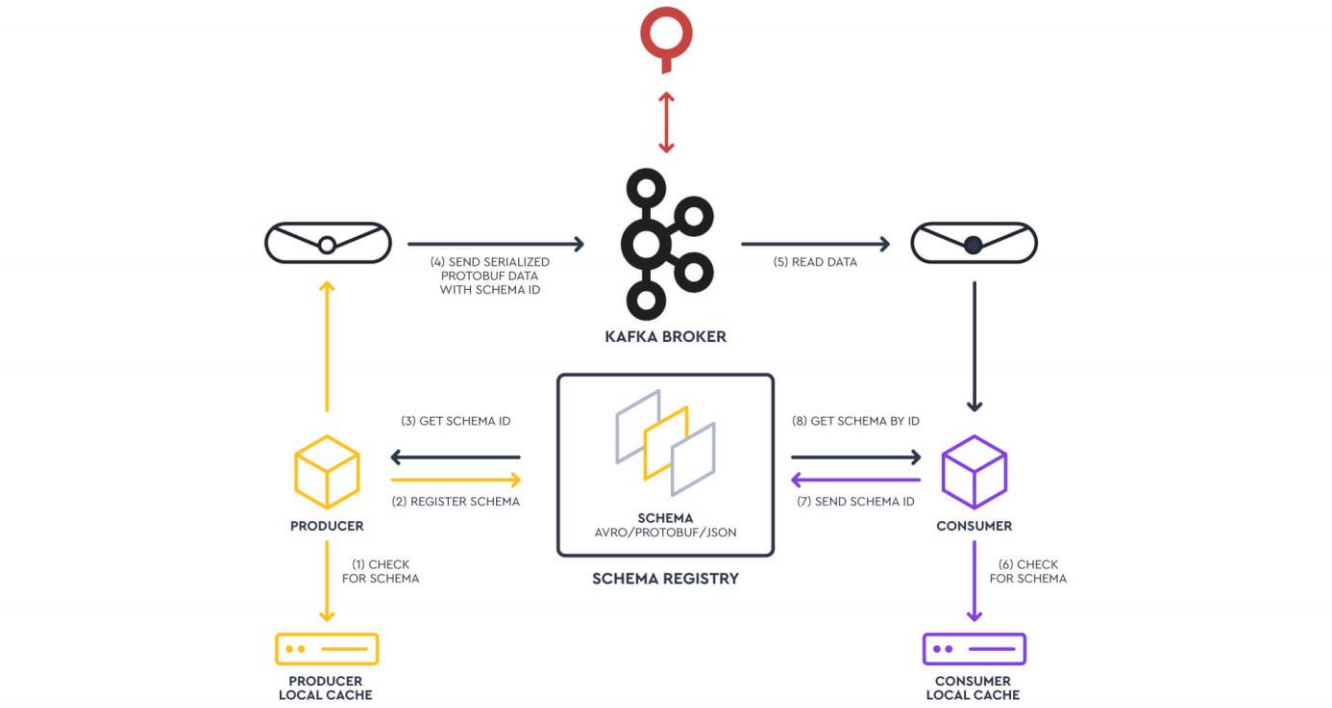

Рисунок 2.6 – Схема передачі даних з використанням протоколу Protobuf

Через відсутність потреби передавання назви атрибути відбувається значна економія на об’єму передачі даних, що також зменшує час на передачу даних. Ускладненням у випадку Protobuf є потреба в зберіганні і підтримки схеми окремо. Для Apache Kafka популярним рішенням сховища схем є Schema Registry.

2.5 Висновки

Розроблено метод багатокритеріальної оцінки нерухомості, який дозволить підвищити точність-швидкість оцінювання, більшу здатність до функціонального та навантажувального масштабування системи. Розроблена архітектура інформаційної системи експертної оцінки нерухомості. Були обгрунтовані необхідні програмні залежності. Описані принципи взаємодії компонентів системи.

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка сервісів завантаження даних

Дані які необхідні як для навчання моделі, так і для безпосередньо оцінки завантажуються в систему з сервісів оголешень, це може бути багато джерел даних з різних ринків нерухомості. У переважному кількості випадків взаємодія на отримання даних відбувається через REST API по протоколу HTTP. При роботі з REST API треба враховувати наступні обмеження від зовнішіх джерел даних:

- Часовий ліміт (максимальна кількість запитів до ресурсу за певний час);
- Місячний або добовий ліміт;
- Додаткова авторизація;
- Ліміт кількості параллельних запитів.

Найчастіше зовніші системи з даними накладають обмеження згідно з плану для того, щоб обезпечети власні бази даних від перевантаження на кількість операцій читання та з’єднаннь. У випадку перевантаження більшість баз даних відхиляють нові з’єднання і загалом підвищується час очікування відповіді, що можно негативно відобразится на внутрішніх процесах самої системи.

Аналізуючи декілька прикладів існуючих API – DIM.RIA, RentCast API, RapidAPI, дозволяє зформувати наступні інтерфейси взаємодії:

- Пошук масиву оголошень по параметрам;
- Отримання детальної щодо об’єкту/оголешення;
- Отримання додаткової інформації щодо міста, району, тощо.

Зрозуміло, що стратегією завантаження інформації буде пошук масиву оголешень па параметрам з наступним процесом завантаження детальної інформації по кожному оголешеню по індетифікатору з можливим параллельним режимом замість послідовного. Але тут є нюанс те, що пошук нових оголешень буде відбуватися постійно, а завантаження вже існуючої бази займе суттєвий час,

особливо якщо враховувати обмеження з боку джерела. Це потребує зберігання прогресу оновлення в базі даних та кеші, для розуміння які оголешення вже були оброблені, а які ще очікують обробки.

Деякі джерела даних обмежують кількість об’єктів, які повертаються при пошуку, наприклад REST API DIM RIA повертає максимально 100 елементів без підтримки пагінації (використання параметрів «offset» і «limit»), це накладає додаткові обмеження на пошук тому що тільки в одному районі Києва оголешенн може бути тисячи або десятки тисяч. В таких випадках для цього потрібно виконати сегментацію ринку на мінімальниї секції пошуку, наприклад використовуючі наступні параметри:

- Тип нерухомості (квартира, будинок);
- Типе операції (продаж);
- Країна (наприклад, Україна)
- Область (наприклад, Вінницька);
- Місто (наприклад, Вінниця);
- Дата публікації.

Завантаження відбувається від по даті публікації від найминулішої дати до поточного дня роботи сервісу, це необхідно для постійного отримання нових оголешень. Найминуліша дата визначетемиться налаштуванням, зазвичай актуальність оголошень втрачається після 1-2 кварталів.

Але як було вказано раніше цього може бути недостатньо для отримання повного результату, тому найбільш гнучкою стратегією буде використання більш розподіленного критерія - вартість нерухомості. Вартість нерухомості розподілена нелінійно, наприклад більшість об'єктів оголошень продажу в Києві знаходяться у діапозоні від 50 до 150 тис доларів. Для більш ефективного пошуку замість послідовної виборки з фіксованим діапозоном (наприклад, по 1 тис доларі) має сенс

використовувати рекурсивний алгоритм бінарного пошуку з логаріфмічною складністю, який можливо відобразити у вигляді бінарного дерева[22] (дис.рис.3.1):

1. Обирається мінімальна нижня та максимальна верхня межа вартості об’єкта (наприклад, від 0 до 10 000 000);
2. Виконується пошук по згідно межами
3. Якщо в результаті пошуку оголешень більше, ніж максимальний ліміт пошуку API, діапозон поділяється на дві частини; перший діапозон від нижньої межі до середини, і від середени до верхньої межі. Для обох діапозонів виконується логіка починаючи з кроку 2.
4. Якщо в результаті пошуку елементів дорівнює або менше ніж максимальний ліміт, зберігається масив елементів для відповідного фільтра в базу даних;
5. Виконується деталізований пошук по оголешенням з результату пошуку.

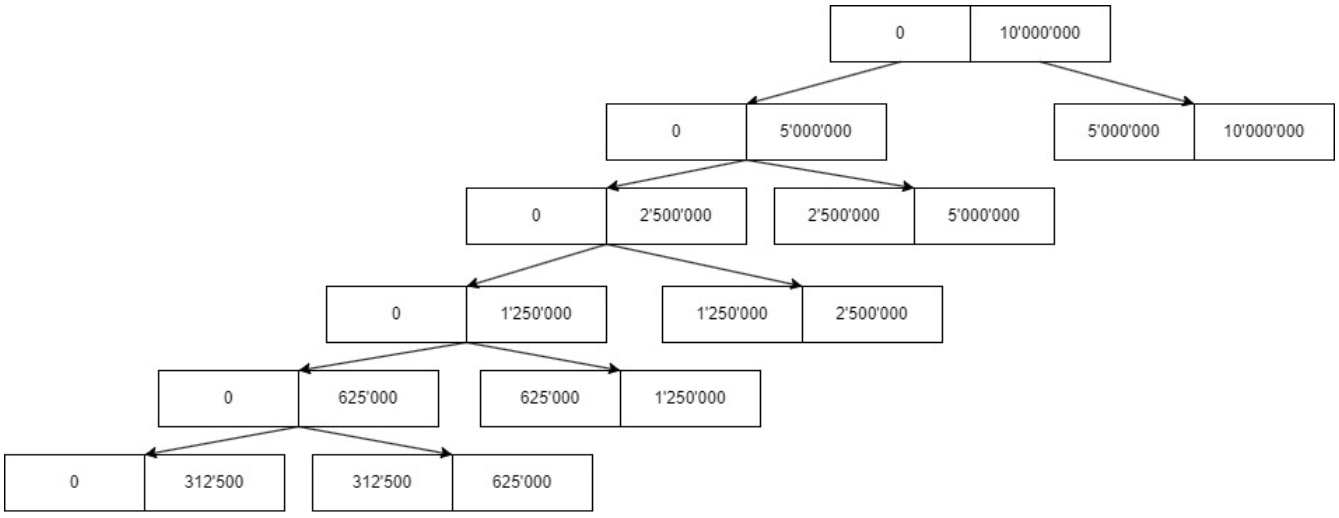

Рисунок 3.1 – Приклад розподілу значень в B-дереві

Як результати пошуку зі списком оголешення, так і деталізовано інформація оголешень зберігатемиться в базі даних для того, щоб у випадку переривання роботи завантаження при відновленні мати можливість повернутися. Структура сегментного пошуку можно представимо в таблиці 3.1.

Таблиця 3.1 – Структура сегментного запита пошуку

Атрибут	Тип даних C#	PostgreSQL тип
Індефікатор пошуку	Guid	Uuid
Тип нерухомості	PropertyType (enum)	property_type (enum)
Тип операції	OperationType (enum)	operation_type (enum)
Країна	int	integer
Місто	int	integer
Діапозон старту та закінчення дати публікації	DateTimeOffset[]	tstzrange
Діапозон вартості об’єкта нерухомості	decimal[]	Numrange
Пошук закінченний	bool	boolean

Для швидкого пошуку останнього запиту, варто створити додатковий індекс на атрибути: країна, місто та діапозон старту та закінчення дати публікації. Деталізована інформація по об’єкту зберігається в окремій таблиці з інформацією про запит пошук, який був використаний для отримання індифікатора оголешення. Зміст самого оголешення зберігається в оригінальному вигляді в JSON форматі.

Таблиця 3.2 – Стуруктура об’єкта з інформацією про оголошення

Атрибут	Тип даних C#	PostgreSQL тип
Зовнішній індефікатор оголешення	string/int/Guid	nvarchar/integer/uuid
Дані	string	jsonb
Дата додання	DateTimeOffset	timestamptz
Індефікатор пошуку	Guid	Uuid

Для коректного пошуку потрібно додати індекси (b-tree[23]) на наступні атрибути:

- Зовнішній індефікатор оголошення;
- Індефікатор пошуку;
- Дата додання.

Загальний алгоритм можно визначити наступним чином:

1. Отримання останього існуючого запиту пошуку до БД;
2. Якщо запит існує, то формується наступна ітерація. Якщо існуючий не був повністю завершенний (прервенний або відхилення від API через велику кількість запитів, або термін пошуку тоді ще не закінчився), то пошук виконується знов з аналогічними параметрами;
3. Якщо це перший запит, тоді виконується формування запиту згідно розглянутого вище алгоритму пошуку сегментів;
4. При отриманні списку виконується фільтрація між тими оголешеннями, які вже були збережені;
5. Виконується отримання оголошень згідно індефікторів отриманих в списку. Якщо список пустий, крок пропускається;
6. В одній транзакції зберігаються – запит пошуку до БД, оголешення. Якщо сьогоднішня дата по UTC попадає в діапозон старту та закінчення дати публікації, тоді запит зберігається як не завершений для того, щоб виконати пошук повторно на наступній ітерації для оголешень, які ще з’являться в майбутньому;
7. Після успішного збереження даних до БД, виконується конвертація інформації оголошення в об’єкт контракту, та публікується в Kafka;
8. Логується інформація з метриками: всього результатів пошуку, час витраченний на завантаження даних;
9. Перехід до наступної ітерації (крок 1) з можливою затримкою.

У вигляді діаграми алгоритм матиме наступний вигляд:

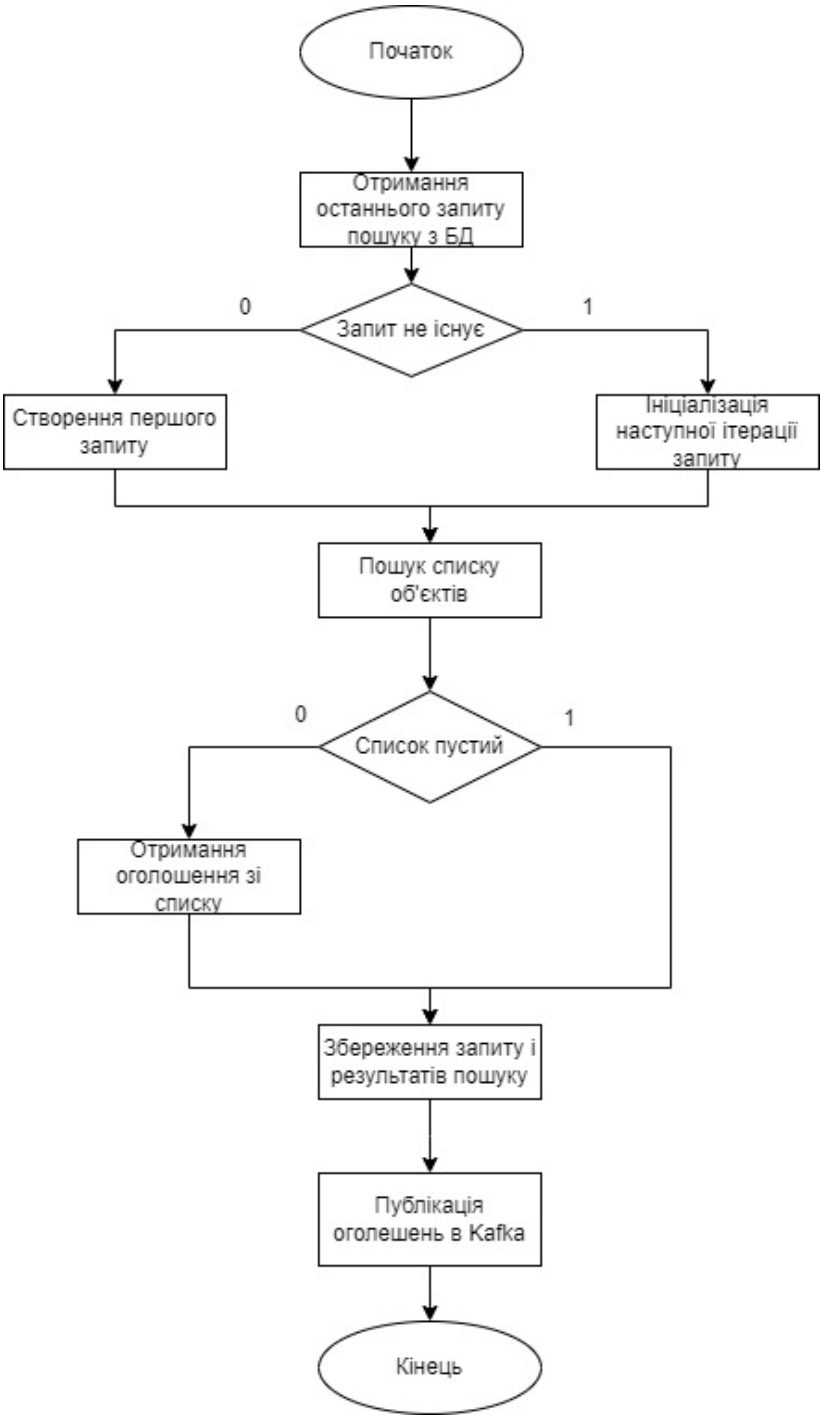

Рисунок 3.2 – Алгоритм пошуку та збереження оголошень нерухомості

У випадку перебільшення кількості запитів API може заблокувати ресурс, якщо не перестати виконувати запити. Це вимагає контролю кількості запитів до ресурсу і запобігання виконання нових запитів при вичерпані дозволених лімітів. Платформа .NET дозволяє виконати перевизначення HttpMessageHandler через

створення власного обробника наслідуючись від DelegatingHandler. Іншим варіантом, більш зручним, є використання існуючої бібліотеки Flurl, яка вже має реалізований додатковий функціонал в тому числі можливість додати підписку на дію після обробки відповіді на кожний запит до API. При обробці підписки на результат створюється/інкрементується лічильники в Redis відповідно від існуючих лімітів. Redis підтримує функціонал по автоматичному видаленню об'єкту після закінчення певного часу. Наприклад, в DIM.RIA є два ліміти – місячний і годиний[24]. Відповідно, при кожному запиті на кожний можливий ліміт інкрементується окремий лічильник, коли час на ліміт пройде – лічильник буде видалений і вже в наступний раз рохування почнеться знов з 1. Перед новою ітерації пошуку потрібно виконати перевірку на проходження лімітів, якщо ліміт вичарпаний, то виконання призупиняється на очікуванний час ліміту. В .NET є можливість створити асинхронну затримку на певний час.

Також, існує висока ймовірність, що перевищення лімітів відбудеться не новій ітерації пошуку, а під час отримання необхідного оголешення, або ліміт від API може змінитися, і тоді застусонок отримає HTTP код 429 (Too many requests). В цьому випадку потрібно явно перервати виконання ітерації. Для цього в .NET використовується механізм CancellationTokenSource та CancellationToken. CancellationTokenSource генерує певний токен - CancellationToken, який має прапор відміни. Якщо CancellationTokenSource сигналюзує, що є потреба виконати відміну, то в усіх згенерованих CancellationToken прапор відміни змінюється з false на true, і при навяності перевірки в коді є можливість згенерувати виключення або виконати вихід з функції. Після зупинки виконання ітерації пошуку по сегменту, вже на наступній ітерації при перевірці лімітів, стане зрозуміло що один із лімітів був вичерпаний, і виконання призупиниться на час ліміту. Хто має бути відповідальний за ініціативу відміни для CancellationTokenSource? Викликати відміну можливо в підписки на закінчення обробки запиту в Flurl після оновлення в лічильників в Redis.

Відобразити взаємодію компонентів можливо через наступну діаграму:

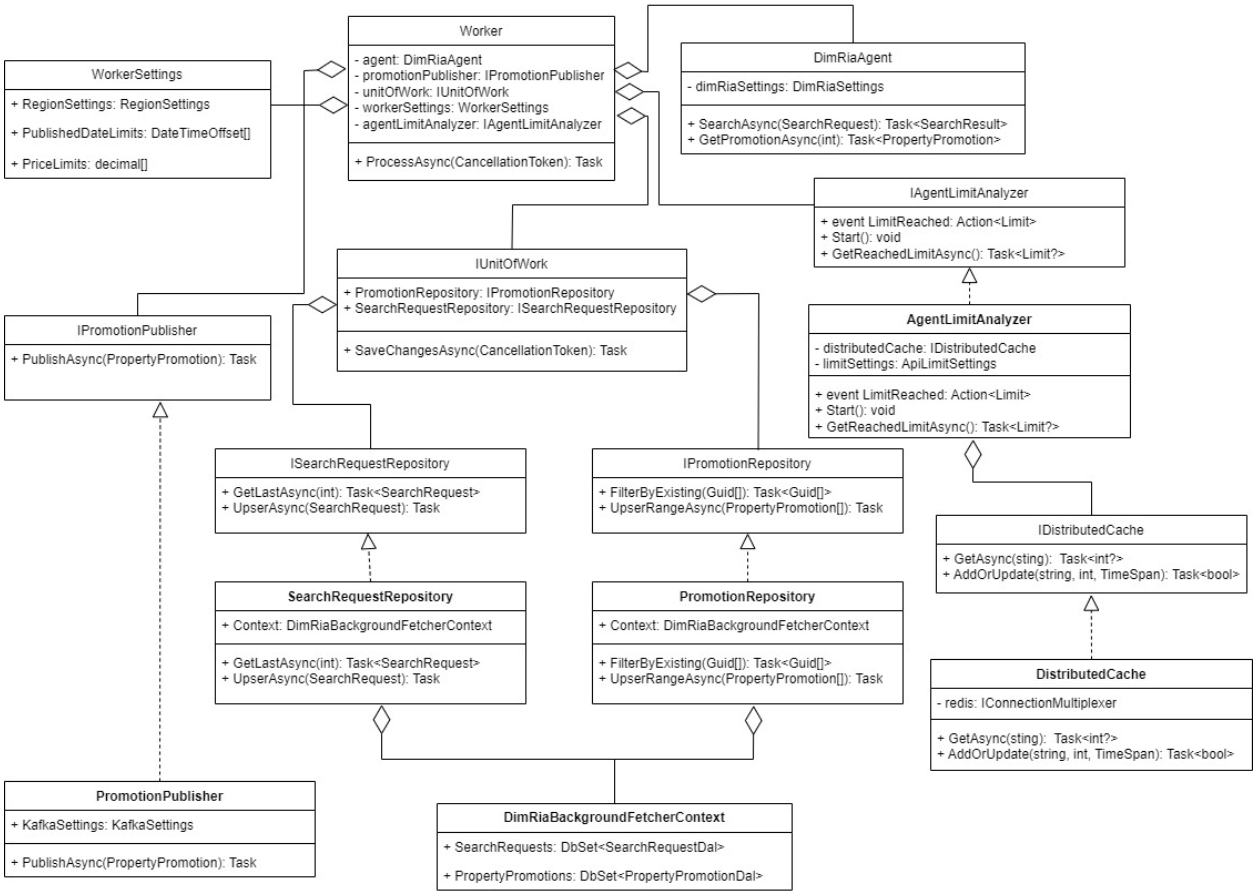

Рисунок 3.3 UML діаграма взаємодії компонентів в застосунку

- Клас Worker відповідає за координацію процесу пошуку та завантаження оголошень;
- Клас DimRiaAgent інкапсулює логіку по роботі API. В даному випадку, з REST API DimRia;
- Клас WorkerSettings містить налаштування процесу пошуку та завантаження оголошень;
- Клас AgentLimitAnalyzer контролює ліміти запитів до API. В класі виконується підписка на запити DimRiaAgent, при перевищіні кількості лімітів публікується подія LimitReached, також є метод GetReachedLimit, який повертає перевищений ліміт, якщо такий є;

- Клас PromotionPublisher відповідає за публікацію оголешення в Kafka;
- Клас UnitOfWork відповідає за координацію репозиторієв для роботи з БД;
- Клас SearchRequestRepository – репозиторій для роботи з БД сутностями запитами пошуку;
- Клас PromotionRepository – репозиторій для роботи з БД сутностями оголошеннями об’єктів нерухомості;
- Клас DimRiaBackgroundFetcherContext – клас з прямим доступом до БД: виконання операцій пошуку, читання та запису;
- Клас DimRiaSettings містить налаштування для роботи з API: базовий URI, токен авторизації, допустимий час очікування;
- Клас ApiLimitSettings – клас з налаштуваннями лімітів HTTP запитів по роботі з API;
- Клас DistributedCache відповідає за отримання та зберження кеша в розподіленному сховища – Redis;
- IUntOfWork, IPromotionPublisher, ISearchRequestRepository, IPromotionRepository, IDistributedCache, IAgentLimitAnalyzer – відповідні інтерфеси реалізацій.

3.2 Розробка сервісів експертної оцінки

В основі логіки оцінки є отримання аналітики за допомогую функції передбачення по моделі навчання, яка будується відповідно до отриманих даних з сервісів завантаження даних. Наступні сервіси мають наступні відповідальності по передбаченню:

- Оцінка локації - передбачення очікуванної середньої вартості за квадратний метр за локацію. Оцінка локації – має базовий коеффіцієнт в розрахунку кінцевої оцінки;

- Оцінка житлового комплексу/будинку - передбачення очікуванного впливу на середню вартість за квадратний мерт згідно з якості та благоустрію будинку;
- Оцінка квартири (житла) - передбачення впливу критеріїв житла (кількість кімнат, планування, ремонт, тощо) на середню вартість за квадратний метр;

Сервіси експерти відповідальні як за побудову моделі оцінки, так і безпосередньо виконання оцінки[25] (див.рис. 3.4).

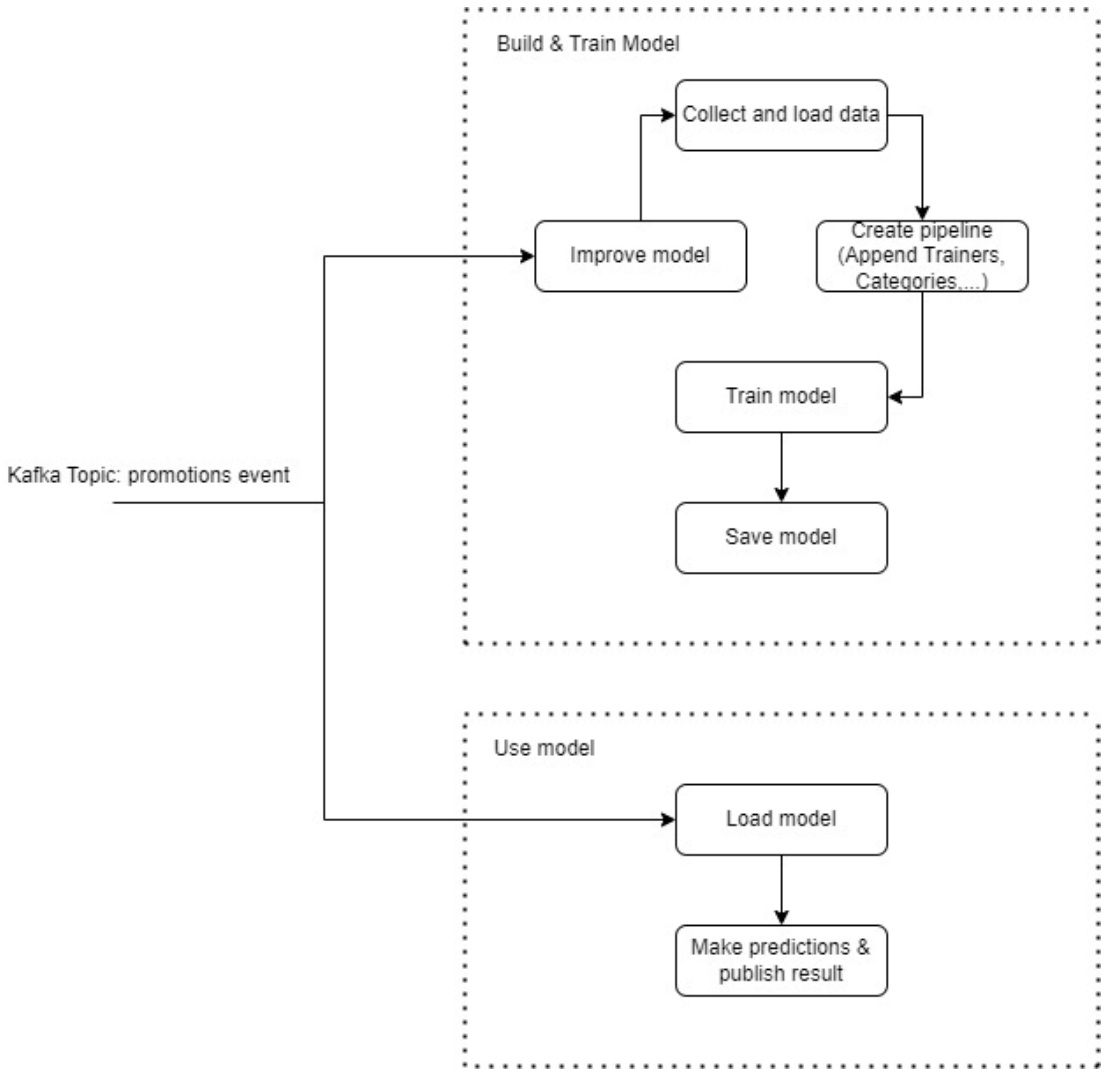

Рисунок 3.4 - Етапи побудови та навчання моделі машинного навчання, прогнозування результату

Сервіс експертизи має два окремих Kafka-consumer – один для побудови моделі оцінки, другий для оцінки. На етапі першої побудови моделі має сенс обмежети використання другого consumer за допомогою налаштування, і після того як модель вже більш менш стабілізується – другий consumer підключиться для оцінювання. При побудові моделі також треба враховувати динаміку отримання нових подій – кількість нових оголошень можуть суттєво відрізнятися від часу доби, дня тижня або наявності вихідних. Оновлення моделі оцінки на кожне нове оголешення, коли їх може бути наприклад за хвилини тисячи або десятки тисяч – це не ефективне використання обчислювальних ресурсів сервісу. Більш правильною стратегією є оновлення моделі при накопиченні пачки нових оголошень або при наявності в пачці хочаб одного оголошення і при цьому проходження певного часового таймеру. Відповідо до цієї стратегіє треба брати до уваги, що підтвердження offset consumer до Kafka має відбуватися після повної обробки пачки оголошень.

Підсумовуючи, стратегія процесу навчання моделі матиме наступний вигляд:

- В наслідок накопичення N повідомлень з новими оголошеннями або проходженням певного часового інтервалу відбувається ініціація побудови моделі навчаня;
- Вібдувається створення конвеєру для застосування функцій нормалізації параметрів, алгоритму машинного навчання;
- Навчання моделі;
- Оцінювання моделі, можливо покращення моделі після її оцінки;
- Збереження моделі в сховище та в кеші для використання надалі.

Стратегія оцінки за моделю відбувається тоді наступним чином:

- Перевіряться флаг з налаштуваннь чи дозволено використання моделі, якщо ні дозволено – оцінка не відбувається, повідомлення не комітиться для обробки в майбутньому;

- При отримані нового оголешення виконується завантаження моделі;
- Після завантаження виконується оцінка об’єкта нерухомості;
- Оцінка об’єкта нерухомості зберігається в сховище;
- Оцінка об’єкта нерухомості публікується в розподіленне сховище Kafka.

Для виконання більш складних маніпуляцій над потоками повідомлень ефективно використовуються парадигма реактивного програмування. Парадигма реактивного програмування базується на ідеї реагування на певні події по певним змінам. У більшості взаємодія асинхроних потоків описуються декларативно, а не імперативно через інструкцію по зміні стану. В парадигмі об’єктно-орієнтованого програмування push-модель реалізується за допомогою патерну «Спостерігач» (Observable) класичний варіант якого був опублікований в відомій книзі «Банди Чотирьох»[26]. Діаграму патерну «Спостерігач» можно відобразити наступним чином (див. рис.3.5):

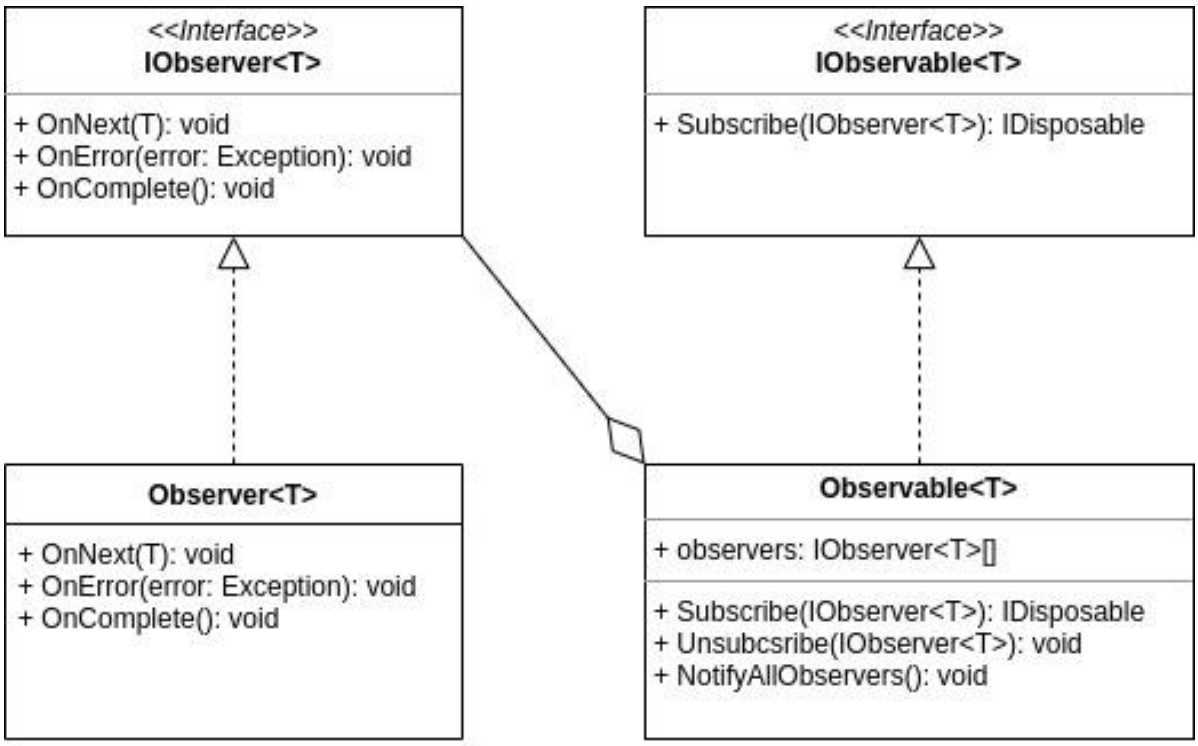

Рисунок 3.5 - Діаграма патерну «Спостерігач»

- Observable – «спостерігаймий» (або видавець), реалізую обов’язки спостерігача, містить список «спостерігачів». При зміні стану викликається метод NotifyAllObservers, який в циклі викликає обробники «спостерігачів»;
- Observer – реалізує «спостерігача», OnNext – обробка успішної події, OnError() – обробка помилки, OnComplete – обробка успішної завершення черги подій;
- IObserver, IObservable – відповідні інтерфейси до Observer та Observable;

Основна перевагу використання патерну – це реалізація слабозв’язанного дизайну, де видавець абстрагований від реалізацій спостерігачів, а спостерігачі не знають деталей реалізації видавця. Під більшість мов програмування вже існують реалізації бібілотек багатьма паттернів і методів для роботи з IObservable. Для .NET – це бібліотке Rx.Net. Для реалізації накоплення пачки подій використовується функція Buffer, для генерації потоку подій по таймеру - функція Interval. У результуючому потоку подій також маємо виконати перевірку через функцію Where про наявність повідомлень (див.рис.3.6).

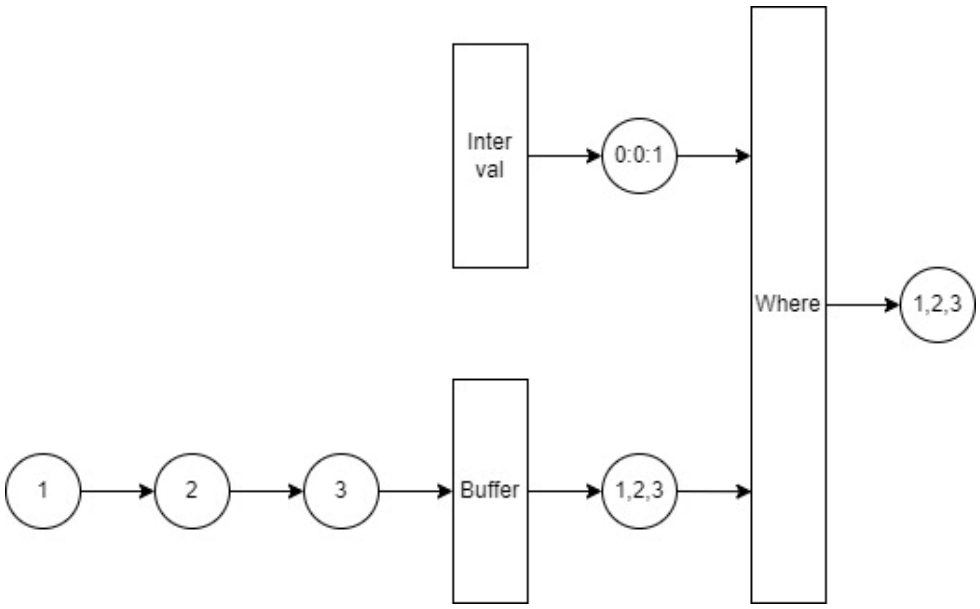

Рисунок 3.6 – Генерація потоку подій для оновлення моделі оцінки

Для кожної задачі машинного навчання може існувати декілька алгоритмів машинного навчання, вибір залежить від сутності проблеми, характеристики даних, обчислювальні можливості та ресурсів зберігання. Алгоритми найчастіше оперують функціями (features) – певними числовими аттрибутами, у результаті отримується модель. Загалом розділяють наступні задачи:

- Бінарна класифікація;
- Багатокласова класифікація;
- Ранжування;
- Регресія – передбачення певного числового значення.

Передбачення вартості нерухомості є регресійна задача. Поширенним підходом вирішення цієї задачі є лінійні алгоритми, які розраховують певний бал виходячи з вхідних даних і вагових коефіцієнтів. Кінцевим результатом лінійних алгоритмів є функція. Лінійні алгоритми мають можливість масштабуватися, відносно інших алгоритмів більш ефективні в начанні та передбаченні, але лінійний алгоритм потребує нормалізації параметрів для запобігання того, що деякі можуть мати більший вплив ніж інші. Параметри мають бути лінійно залежними для побудови моделі, тобто використання індефікаторів або перечислень скоріше матиме негативний ефект. Наприклад, індифікатор района не нормалізований для лінійних алгоритмів, замість використання індифікатора необхідно зформувати відстань до центру, кількість населення в районі, середній дохід жильців, тощо. Популярним лінійним регресійним алгоритмом для моделі з великою кількістю параметрів є алгоритм градієнтного спуску[27]. Головна задача алгоритму ітеративний рух в напрямку функції в точці поточної апроксимації мінімуму. Чим менше крок руху, тим більш точним буде кінцевий результат, але тим довше буде час навчання. Також використовуючи заданто малий крок існує ймовріність застряги в локальному мінімуму, при заданто великому можливо перестрибнути найбільш оптимальну точку функції. Приклад виконання градієтного спуску можливо у вигляді 3D діаграми (див.рис 3.7).

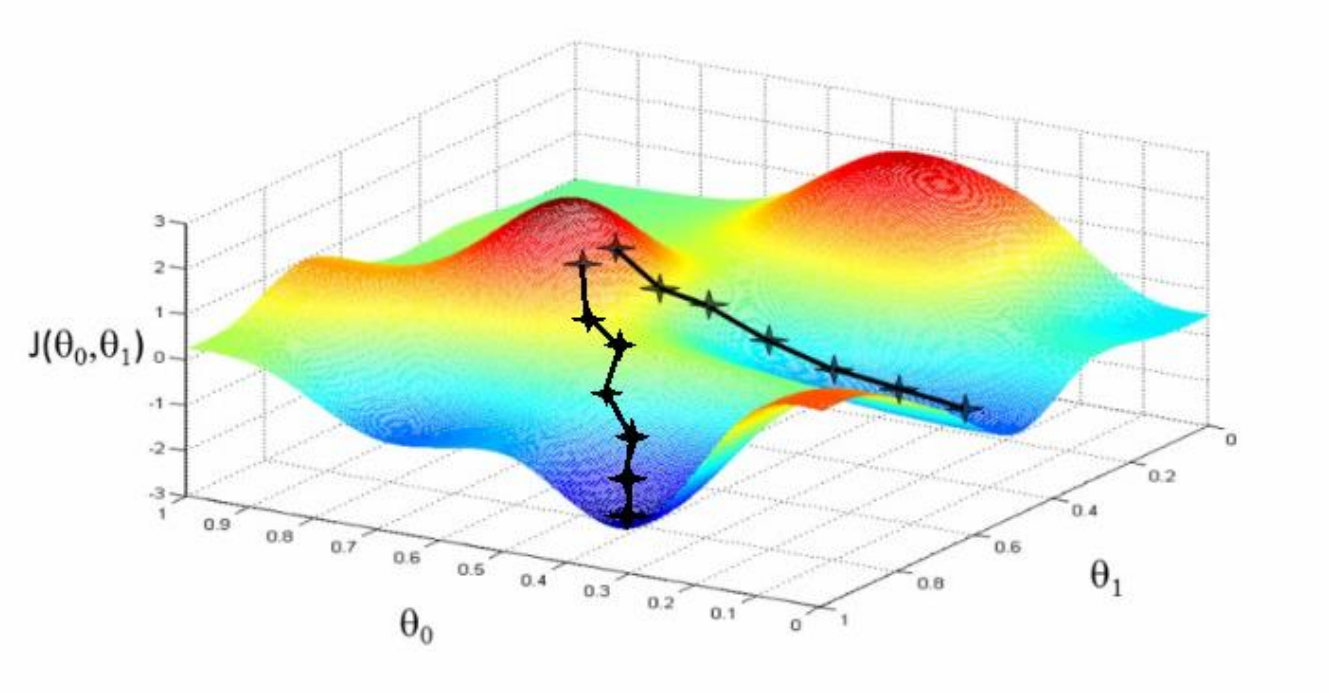

Рисунок 3.7 – Алгорим градієнтний спуску

Іншим напрямом рішення цієї задачі є використання алгоритмів підсилення або алгоритмів-бустінгу з ретроспективним навчанням, роблячи висновки з минулих помилок. Градієнтний бустінг об’єднує декілька слабких моделей для того, щоб створити більш сильну модель. Зазвичай в основі використовуються модель дерево рішень, у випадку регресіїної задачі – дерево регресії[28], де кожен вузол представляє собою умову, а кожен листок дерева містить прогнозованне значення або категорію. Передбачення приймається шляхом проходження з корня до вузлів, де для кожного вузла розраховується умова, і напрям переходить ліворуч чи праворуч в залежності від результатів.

Основна перевага використання дерева регресії у порівнняні з лінійної регресією, це гнучкість та ефективна робота з нелінійними залежностями, такими як категорії та індефікатори. Цілком логічно, що передбачення по моделі дереву регресії може буть дещо довшим, ніж передбачення по лінійній регесії, але нормалізація вхідних може зайняти ще більше часу.

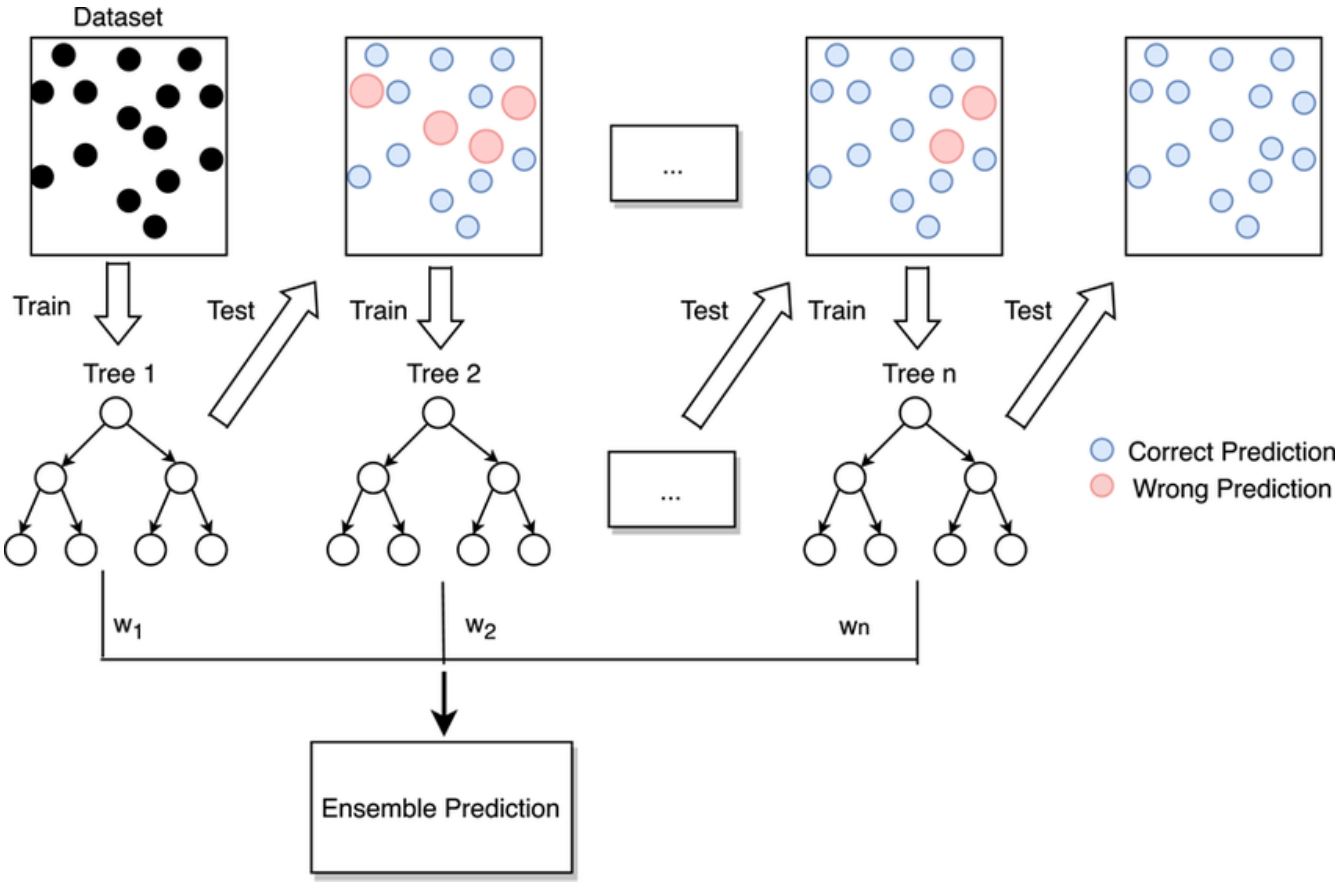

Рисунок 3.8 – Побудова моделі з використанням методу градієнтного бустінгу на основі дерева рішень

Алгоритм градієнтого бустінгу на основі дерева рішень можливо описати наступними етапи:

1. Побудова базового дерева з одного вузла або глибиною до 1;
2. Розрахунок помилок для кожного кортежу в навчальному наборі з актуальною побудованою моделі;
3. Визначення ваги для кожного кортежу відповідно до встановленою помилки;
4. Побудова нового дерева рішень з коррегуванням встановлених помилок, через врахування вагів;
5. Додаємо суму вагів нової моделі до попередньої моделі;

6. Повторення процесу – нові дерева додаються до загально ансамблю дерев, кожне дерево намагається скоригувати залишкові помилки;

Для управління процесом навчання контролюється глибина дереве, крок навчання і максимальна кількість дерев. Метод градієнтного бустінгу на основі дерева рішень, який використовує ретроспективне коригування помилок попередніх моделей в реалізації називають LightGbm – Light Gradient boosting machine[29]. Також існує інший метод градієнтного бустінгу FastTree, який базуються на алгоритмі CART – Classification and Regression Trees[30]. Цей алгоритм також будує дерево регресії за рахунок розбиття набору даних на дві підмножини за одної з ознак, процес повторується для кожного нового вузла дерева до досягнення визначенного критрею зупинки – максимальна глибина дерева або мінімальна кількість значень зразків у листі. FastTree алгоритм може буде швидше, ніж LightGBM, але при цьому буде менш точним і менш ефективним.

Виконаємо порівняння розглянутих алгоритмів в порівняльній таблиці:

Таблиця 3.3 - Порівняння алгоритмів LightGbm, FastTree, Graident Descent

Критерій/Алгоритм	LightGbm	FastTree	GradientDescent
Вирішення регресійної задачі	+	+	+
Швидкість навчання	1/3	2/3	2/3
Обробка категоріальних ознак	Не потребує попередньої обробки	Може вимагати попередньої обробки	Попередня обробка необхідна
Налаштування навчання	+	+	+
Використання пам’яті	2/3	3/3	1/3
Ітераційне навчання	Підтримується, але не в ML.NET	Підтримується, але не в ML.NET	Підримується

Вибір необхідного алгоритму з проаналізованих LightGbm, FastTree або GradientDescent буде відбуватися від очікуванного набору функцій (features), об’єму даних, потреби та вартості нормалізації, вимог щодо продуктивності.

Розглянемо детально, які атрибути (features) будуть використані для оцінки вартості нерухомості в категорії «Локація».

Таблиця 3.4– Атрибути оцінки категорії локація

Атрибут	Обов’язковість	Нормалізований тип даних	Приклад
Країна	+	int64	1; 2; 3
Область	+	int64	10; 12;
Місто	+	int64	15;
Район	+	int64	15187
Вулиця	-	int64	4505
Комплекс	-	int64	433819566
Будинок	-	int64	2342341
Довгота знаходження	-	double	30.48914807
Широта знаходження	-	double	50.5455052
Індекс соціальної інфраструктури	-	double	999
Індекс комерційної інфраструктури	-	double	998
Індекс транспортної інфраструктури	-	double	997
Індекс культурних та зелених зон	-	double	996
Індекс якості повітря	-	int32	78

З атрибутами: країна, область, місто, район, вулиця, комплекс, будинок, догота та широта знаходження зрозуміло – це категоріальні ознаки, які передаються з змісту оголешення, але інформація щодо соціальної, комерційної та транспортної інфраструктури у більшості випадків відсутня. Рішенням цього є окремий пошук і сортування об’єктів інфрастктури на мапі в радіусі в N метрів від місцезнаходження житла, і розрахунок оцінки окремо по кожній категорії. Одним із лідерів картографічних сервісів, який дає доступ до великої кількості інтерфейсів з великою базою даних - Google Maps платформа[31].

Один з цих інтерфейсів, Places API, дає доступ до пошуку місць по певному радіусу за координатами місця, також є можливість вказати додаткові фільтри. Пошук відбувається по REST API по методу POST за URL - https://places.googleapis.com/v1/places:searchNearby. Параметри для пошуку наступні:

- типи об’єктів (ресторани, школи, лікарні, тощо);
- максимальна кількість результатів;
- обмеження області по кругу – координати центру та радіус в метрах;
- сортування по дистанції або популярності.

У результаті API повертає масив об’єктів, де окрім інформації про місцезнаходження, можуть бути вказані контакти, рейтинг, кількість оцінок, категорії, тощо. Також API підтримує можливість обмежети поля, які потрібно повертати, через вказання атрибут X-Goog-FieldMask в заголовці HTTP запиту. Ключ доступу передається в заголовку запиту. В структурі Google Maps об’єкт має багато категорій і вони не повністю відповідні проаналізовані в цій роботі, тому конвертація категорій в тип з матиме наступний вигляд:

- Соціальна інфраструктура – hospital, doctor, university, school, athletic_field;
- Комерційна інфраструктура – reustaraunt, store, shopping mal, hotel, gym, café, bar; gas_station, car_repair, courier_service, market;

- Транспорта інфраструктура – transit_station, train_station, bus_stop, subway_station;
- Культурна та зелена зона – event_venue, park, tourist_attraction, aquarium, movie_theater.

Для розрахунку оцінки по інфраструктурі чи зоні пропонується використати атрибути – рейтинг та кількість оцінок, для розуміння наскільки об'єкт користується попитом. Розрахунок оцінки матиме наступний вигляд:

$$X = \frac{\sum_{i=0}^{n} r_i \, q_i}{K} \tag{3.1}$$

де r_i – середня оцінка за об'єкт інфраструктури;

q_i – кількість оцінок за об'єкт інфраструктури;

K – коефіцієнт щільності населення на 1 км^2, необхідний для нормалізації параметру, в Україні щільність залежить від міста, в Києві приблизно 3500 осіб на 1 км^2.

Для отримання деталізованої інформації по місцю існує окремий метод GET https://places.googleapis.com/v1/places/PLACE_ID, де у якості PLACE_ID передається індифікатор місця, який ми отримуємо при пошуку. API також підтримує обмеження списку атрибутів, що повертаються. У відповіді можливо отримати інформацію щодо:

- категорії об'єкту;
- контакти, веб-сайт;
- фізична адреса;
- години роботи;
- рейтинг і кількість оцінок;
- відгуки;
- фото.

Приклад пошуку місць по радіусу від вказаних координат можно відобразити візуально наступним чином (див.рис.3.9).

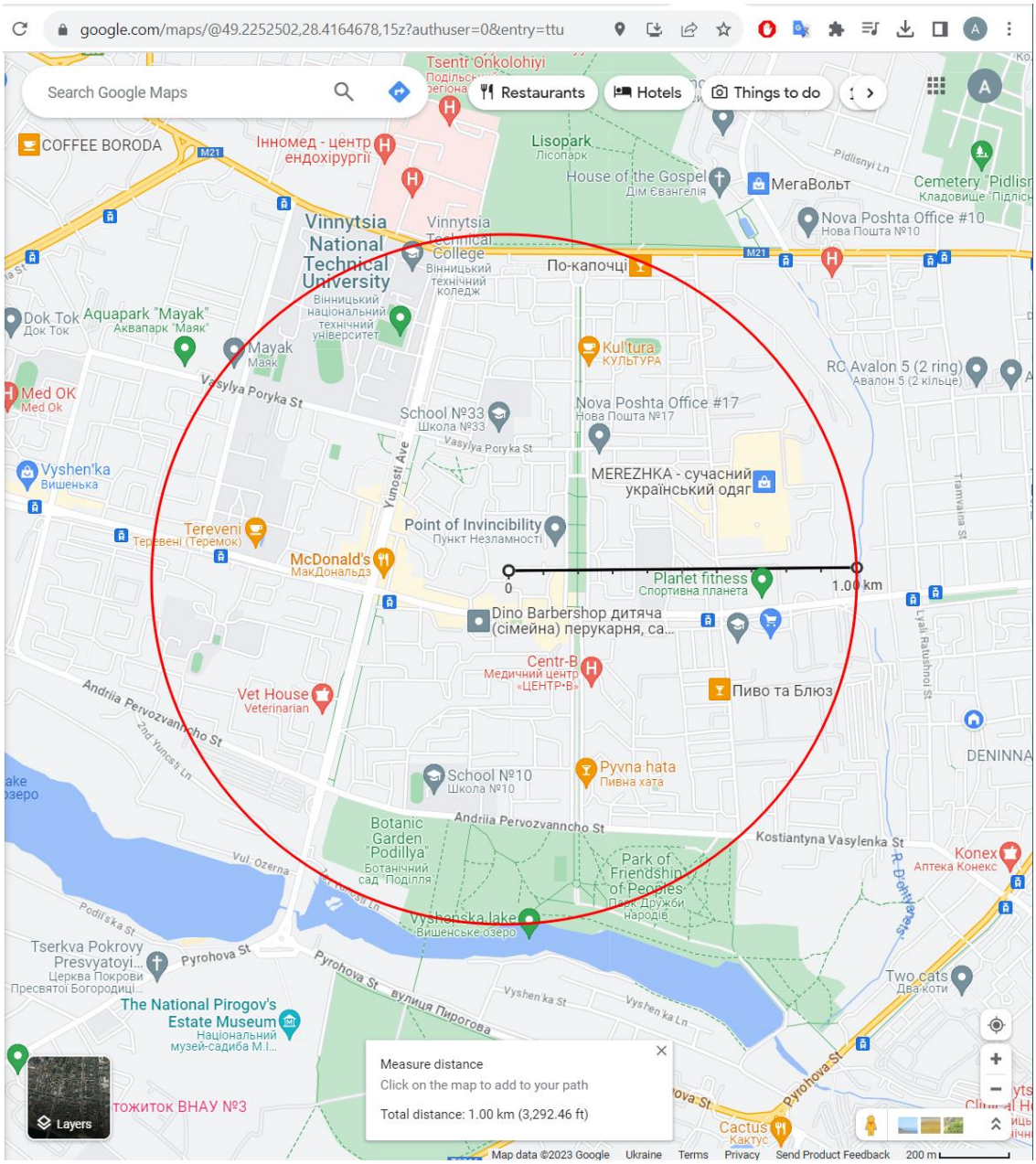

Рисунок 3.9 – Візуальний пошук найближчих місць в радіусі 1 кілометру

Також Google Maps Platform пропонує отримати рівень якості повітря по координатам використовуючи Air Quality API. Для отримання відповідного якості виконується метод POST по https://airquality.googleapis.com/v1/currentConditions:lookup. Координати передаються в тілі запиту, а ключ доступу як параметр в URL. У відповідь повертається статус 200 з тілом, яке містить дату та час створення індексу, індекс,

категорія ідексу (low, medium, high), кольоровий індефікатор індексу. Індекс якості повітря знаходиться в діапозоні від 0 до 100, де чим вище індекс, тим якість повітря вище. Цей індекс загальний для усіх країн, але на мапах вказують найчастіше національний індекс, які мають різні метод разрахунку. Наприклад, в Європі використовуються European AQI, в Індії AQI (див. рис. 3.10).

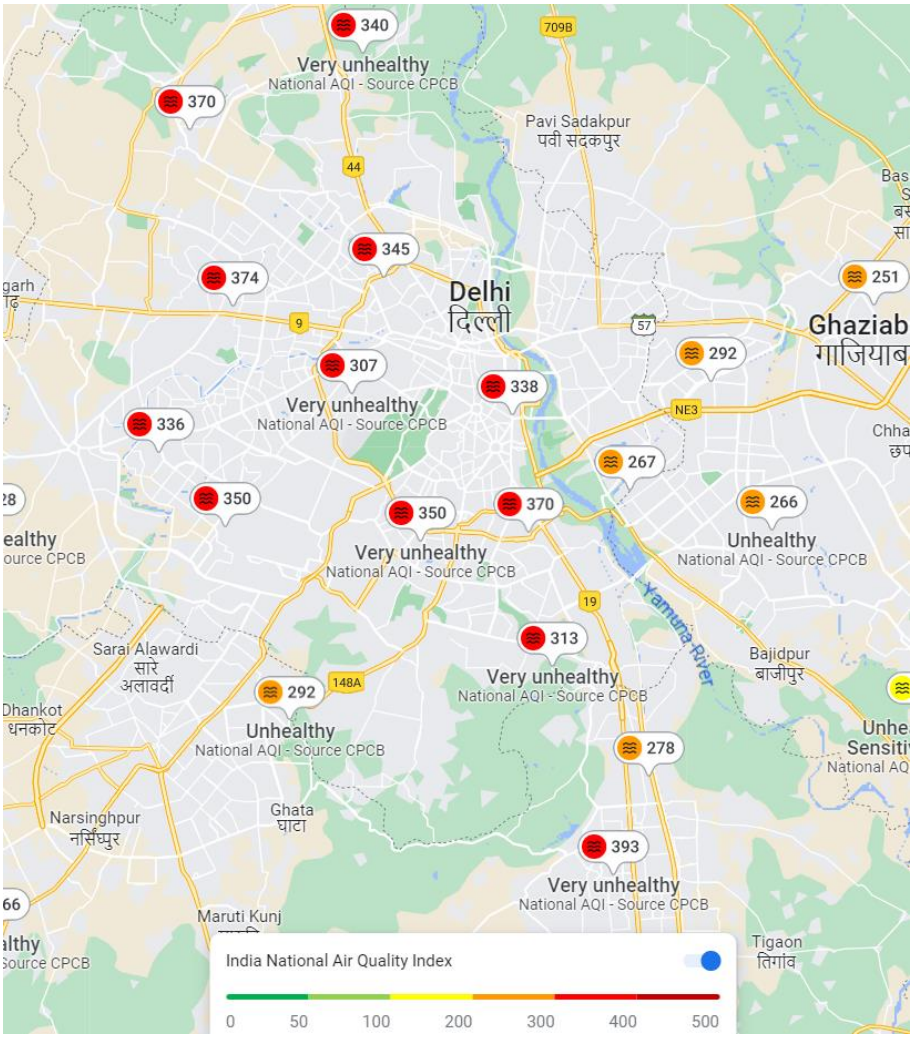

Рисунок 3.10 – Візуальне відображення індексу якості повітря в районах Делі, Індія

При розрахунку загального індексу якості повітря враховуються рівень змісту озона (O3), діоксиду азота (NO2), діоксид сіри (SO2), вуглицю (CO), твердих часток.

Враховуючи, що в аналізу локації використовується велика кількість категоріональних ознак – країна, область, місто, район, тощо, і також те що отримання додаткових даних та разрахунок додаткових лінійних атрибутів (таких як відстань до центру, статус міста, середній дохід мешканців в района) при навчанні та передбачанні займе більш часу, ніж ймовірна перевага лінійних алгоритмів, через це не розглядатимо використання градієнтного пуску для оцінки вартості нерухомості по локації. Розглянемо швидкість та точність моделей у порівянні алгоритмів FastTree і LightGbm. Для тренування використаємо 1 мільйон тествих екзепмлярів з розділенням вартості в залежності від критеріїв (дис.рис.3.3).

Таблиця 3.5 - Порівняння результатів використання алгоритмів LightGbm та FastTree в оцінці вартості нерухомості в локації

	LightGbm	FastTree
RSquared	0.4	0.63
RootMeanSquaredError	437	345
Час тренування, мс	224	983
Час передбачення, мс	67	64
Розмір моделі, КБайт	50.2	90.9

R-squared – статичний показник, який відображає наскільки змінна залежності пояснює зміну в залежній змінні при використані регресіній моделі. Значення знаходиться в діапозоні від 0 до 1, де 0 вказує, що вказана модель не відображає необхідну варіацію в залежній змінній, а 1 навпроти вказує на повну поясняюючу здатність моделі.

RMSE (Root Mean Squared Error) – показник, що відображає середнє квардратичне віхдилення між спостержуваними значеннями (тренувальними) та прогнозованними. Формує розрахунку має наступний вигляд:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(x_i - y_i)^2}{n}} \quad (3.2)$$

де n – кількість тренувальних кортежів (спостережень);

x_i – істинне значення залежної змінної для спостереження при і ітерації;

y_i – прогнозоване значення змінної для спостереження при і ітерації.

Враховуючи порівняльні результати найбільш ефективним алгоритмом для вирішення задачі передбачення є LightGbm – Light Gradient boosting machine.

Для оптимізації взаємодії з Google Maps API, результати пошуку місць по радіусу і якості повітря будуть кешуватися в Redis на певний часовий інтервал по ключу – координатам. Наприклад, не має сенс виконувати пошук, якщо за останні декілька днів на відстані до 50 метрів сходих координат вже виконувся пошук. В Redis існує окрема структура – геопросторовий індекс, яка дозволяє зберігати і роботи пошук координатом у якості ключа використовуючи радіус.

Після отримання оголошеня з боку Kafka consumer виконується заповнення додаткових даних щодо інфраструктури, оточення, якості повітря за допомогою Google Maps API зі збереженням отриманих даних в Redis-кеші. Перед навчанням моделі та передбаченням заповненне оголошення зберігається в БД в JSON.

Повний алгоритм обробки повідомлень для навчання матиме наступний вигляд:

1. Сервіс отримує оголешення з інформацією про об’єкт нерухомості;
2. Відповідно до конфігурації можливе виконання накопичення буфера оголошень через ліміт або часовий інтервал;
3. Після цього виконується навчання моделі за алгоритмом LightGbm;
4. Модель зберігається в БД і в кеш (зі автоматичним видаленням по часу) у вигляді байтового масиву;
5. Час навчання і об’єм моделі записуються як метрики;

Алгоритм обробки повідомлень для передбачення оцінки матиме наступний вигляд:

1. Якщо модель існує в кешу, виконується завантаження моделі з кеша;
2. Інакше модель завантажується з БД;
3. Якщо модель відсутня - обробка зупиняється, система записує лог з рівнем warning;
4. Виконується передбачення очікуваної вартості;
5. Публікація результату оцінки в Kafka;

Чи можлива проблема гонки в багатопотокому середовищі при обробці повідомлень? Kafka-consumer гарантує можливість синхронної обробки повідомлень, через що з накопиченням буферу проблем не має бути. Але чи можливо те, що новий буфер накопичеться раніше, ніж по передньому буферу виконається навчання моделі, через що можливо параллельне навчання і збереженння в БД. Загалом, буфер і алгоритм мають бути сконфігуровані таким, чином щоб ця ситуація була б не можливо, але навіть низький ризик варто уникнути. Також, при горизонтальному масштабовані, коли будуть додані нові вузли з новими сервісами-експертизи, то також будуть додані і нові Kafka-consumer. Але модель буде зберігатися одна, а оновлюватися з декількох окремих сервісів – ризик конфілктів зростає. Один із рішенням є розподіленне блокування, коли в кеш-пам’ять Redis (обробка в Redis однопотокова) зберігається унікальна строка (токен) наявність якої дає доступ тільки для одного процесу і запобігає для інших. Блокування ресурсу відбувається в процесі читання, разблокування після збереження. Основна перевага розподіленного блокування – це створенна абстракція, яка дозволяє блокувати будь який процес, але недоліків насправді значно більше. Головний недолік – це небезпека в відсутності пророблених транзакцій. Що відбудеться, якщо ресурсу не буде разблокованний через помилку в клієнті або проблеми з мережою? Це потребує ручного видалення або видалення по таймауту. Але що буде якщо клієнт відновить роботу, і буде намагатися зберегти

заблокуванний ресурс, яким він вже насправді не володії? Є ризик перезаписати зміни з боку іншого процесу. У висновку ми приходимо, що зберіганя токена блокування окремо від ресурса, який потрібно заблокувати, є витратним і ризикованим методом. Якщо потрібно заблокувати ресурс в БД на одну транзакцію від читання в інших транзакціях, то найкращий засіб – це використання блокуваннь кортежів (рядків) на рівні таблиці. В PostgreSQL це досягається за допомогою додавання до SELECT запиту інструкції FOR UPDATE[32], блокування можливо в двох режимах - WAIT (з очікуванням) і NO WAIT (без очікування). З очікуванням інші транзакції будуть очікувати розблокування ресурсу, без очікування відбуватиметься генерування виключення та відміна транзакцій. Після очікування транзакції вже зчитають оновлену версію (див.рис. 3.11).

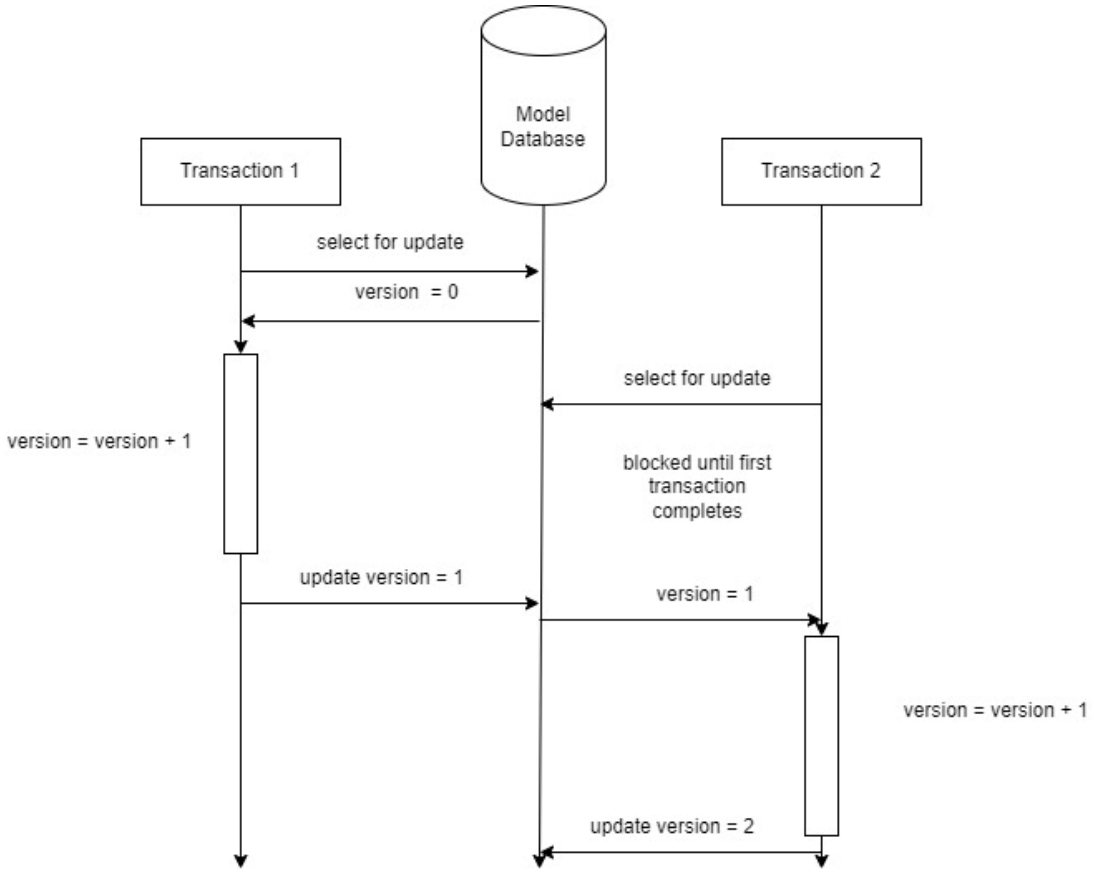

Рисунок 3.11 – Оновлення моделі навчання з блокуванням кортежу в БД

Інтеграція з кешом буде реалізована за допомогою паттерну декоратор[33]. Це структурний патерн, який пропонує додавання нового функціоналу за допомогою реалізації певного інтерфейса, і перевикористання іншого компоненту, який також реалізує згаданий інтерфейс, через композицію. Наслідування є одною з найсильніших зв'язків для перевикористання логіки, розірвати побудовану ієрархію типів може бути надскладною задачою. Декорація робота з кеш-пам'ятью над інтеграцією з GoogleMaps API матиме наступний вигляд:

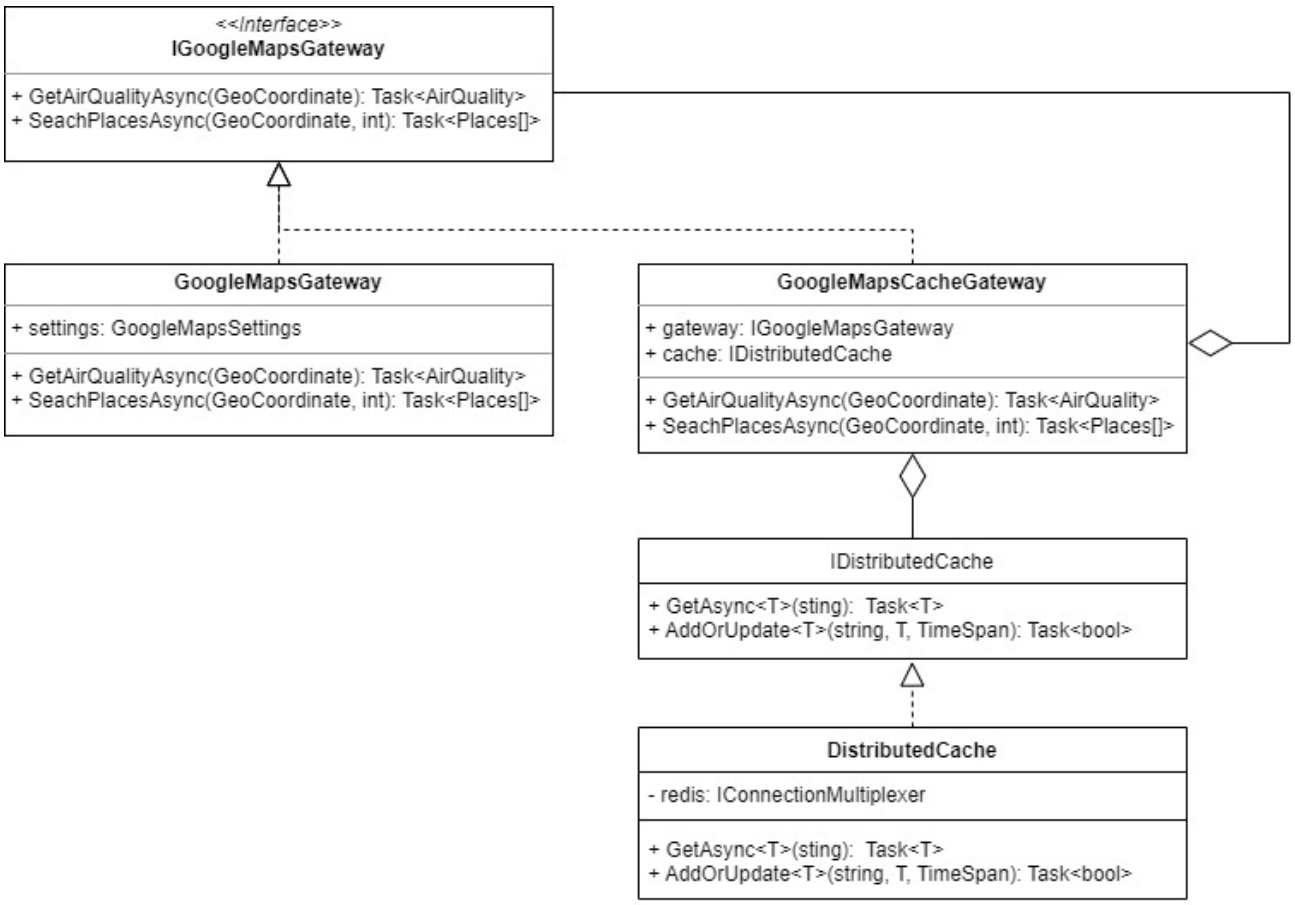

Рисунок 3.12 – Використання паттерну декоратор для додавання функціональності кешування до взаємодії з Google Maps API

- DistributedCache відповідає за зберігання та отримання даних з кеш-пам'яті в Redis;

- GoogleMapsGateway відповідає за пряму інтеграцію з GoogleMaps API. За допомого методу GetAirQualityAsync отримається інформація щодо якості повітря, за методом SearchPlacesAsync виконується пошук місць по радіусу від координат;
- GoogleMapsCacheGateway відповідає за інтеграцію кеш-пам’яті з GoogleMapsGateway, в компонент передається фактично реалізації GoogleMapsGateway та DistributedCache, якщо значення існує в кеш-пам’яті, то повертається з кеша, інакше виконується запит до Google Maps API, після отримання нового значення з API, дані зберігається в кеш.

Діаграма класів для сервісу матиме наступний вигляд(див.рис.3.13)

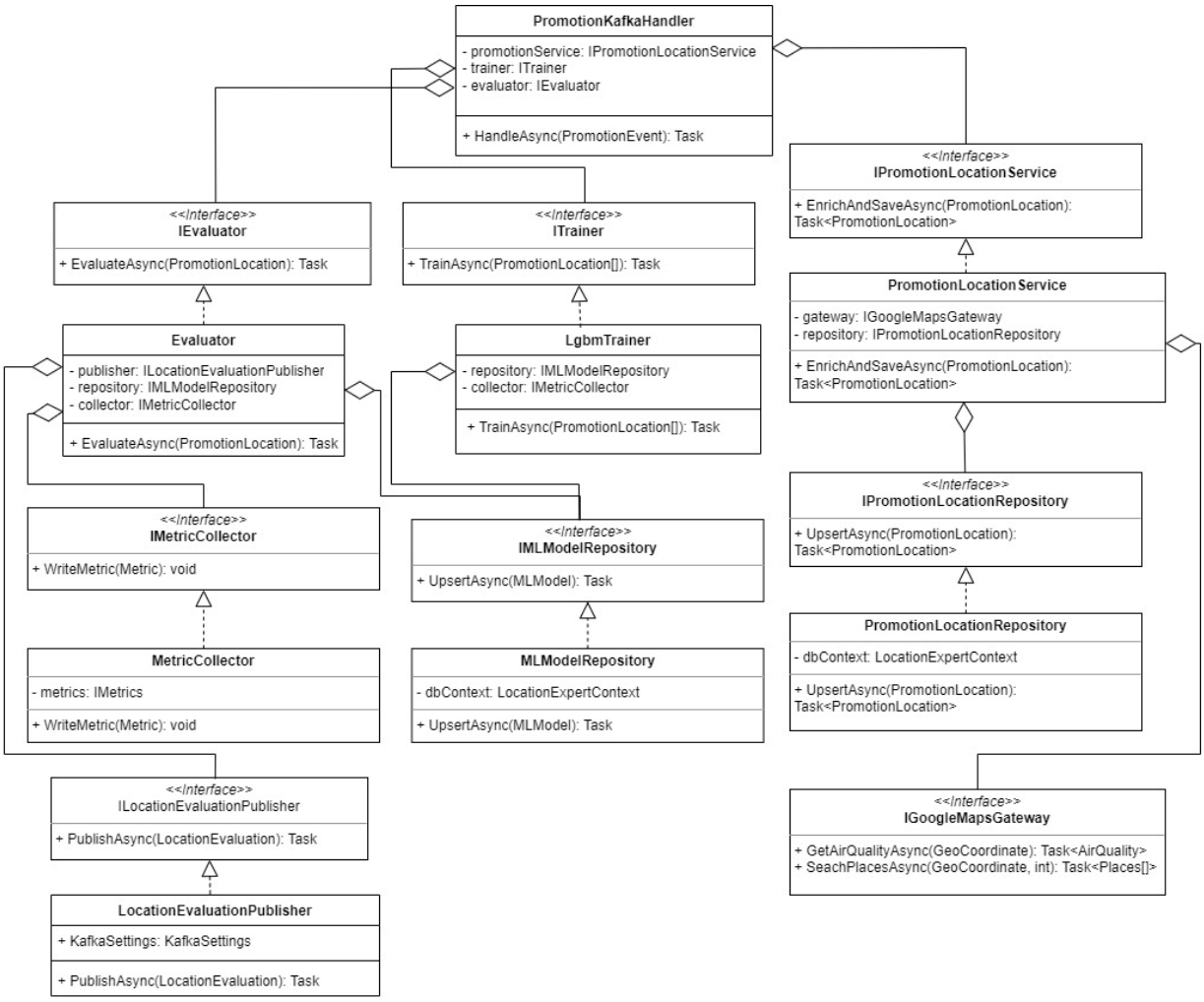

Рисунок 3.12 – Діаграма класів сервісу експертизи локації

- PromotionKafkaHandler – Kafka consumer, обробник подій з топіку promotion. Координує процес зберігання даних використаних для оцінки, тренування моделі, передбачення оцінки;
- PromotionLocationService відповідає за дозаповнення та зберігання даних необхідних для оцінки;
- PromotionLocationRepository інкапсулює логіку по роботі з БД по даних необхідних для оцінки;
- MLModeRepository інкапсулює логіку по роботі з БД по моделі навчання;
- LgmbTrainer відповідає за тренування моделі;
- Evaluator відповідає за оцінку нових оголошень;
- MetricCollector відповідає за збір метрік;
- LocationEvaluationPublisher відповідає за публікацію оцінки в Kafka.

Розглянемо детально, які атрибути (features) будуть використані для оцінки вартості нерухомості в розрізі житлового комплексу.

Таблиця 3.6 - Атрибути оцінки категорії житлового комплексу

Атрибут	Обов’язковість	Нормалізований тип даних	Категорія
Кількість поверхів	+	int64	-
Житловий комплекс	-	int64	+
Матеріал стін	+	int64	+
Кількість місць в паркінгу	-	int64	-
Кількість квартир	-	int64	-
Кількість ліфтів	-	int64	-
Матеріал утіплення	-	int64	+

Продовження таблиці 3.6

Атрибут	Обов’язковість	Нормалізований тип даних	Категорія
Рік побудови		int64	-
Клас ЖК	-	int64	+
Оцінка ЖК	-	int64	-
Кількість оцінок ЖК	-	int64	-
Оцінка вартості нерухомості в локації	+	float	-

Реалізація сервісу оцінки житлового комплексу матиме схожу реалізацію як до сервісу оцінки локації: при отриманні повідомлення з топіку promotion виконується заповнення необхідної інформації, якої може не вистачати для оцінки, після чого окремо виконується дотренування існуючої моделі та оцінка по цій моделі. Для побудови моделі має сенс використовувати також алгоритм LightGbm через достатню кількість атрибутів-категорій (матеріал стін, матеріал утіплення, клас ЖК).

Для отримання оцінки ЖК можливо використати Google Maps Platform через Places API. Використовуючи пошук через POST https://places.googleapis.com/v1/places:searchText Google Maps повертає список знайдених об’єктів зі списком рецензій користувачів. Треба розуміти, що вказана оцінка має косвенний вплив через те, що оцінки можуть залушати будь які користувачі з Google аккаунтом, і навіть деякі запрограмовані боти. Для пошуку в тіло запиту передається параметр textQuery з пошуковою адресою. Пошук є багатофункціональним, і дозволяє виконувати пошук по адресі або назві ЖК, навіть

в різних мовах. Результати відповіді також має сенс зберігти в кеш-пам’яті з автовидаленням через добу для економії трафіку HTTP.

При оцінці нерухомості в розрізі житлового комплекса вже необхідно враховувати оцінку в розрізі локації, тому що фактори локації базово мають більший вплив при моделі, яка будується для багатьох міст і країн. Для цього сервіс оцінки нерухомості в розрізі ЖК виконує запит на оцінку до сервісу в розрізі локації – POST /api/v1/locations:estimate, враховуючи що отримання оцінки відбувається зі збереженої та закешенової моделі, запит не матиме велику затримку. Результатом оцінки є середня вартість нерухомості в житловому комплексі, яка також публікується в Kafka. Діаграму класів можливо відобразити наступним чином (див.рис.3.13):

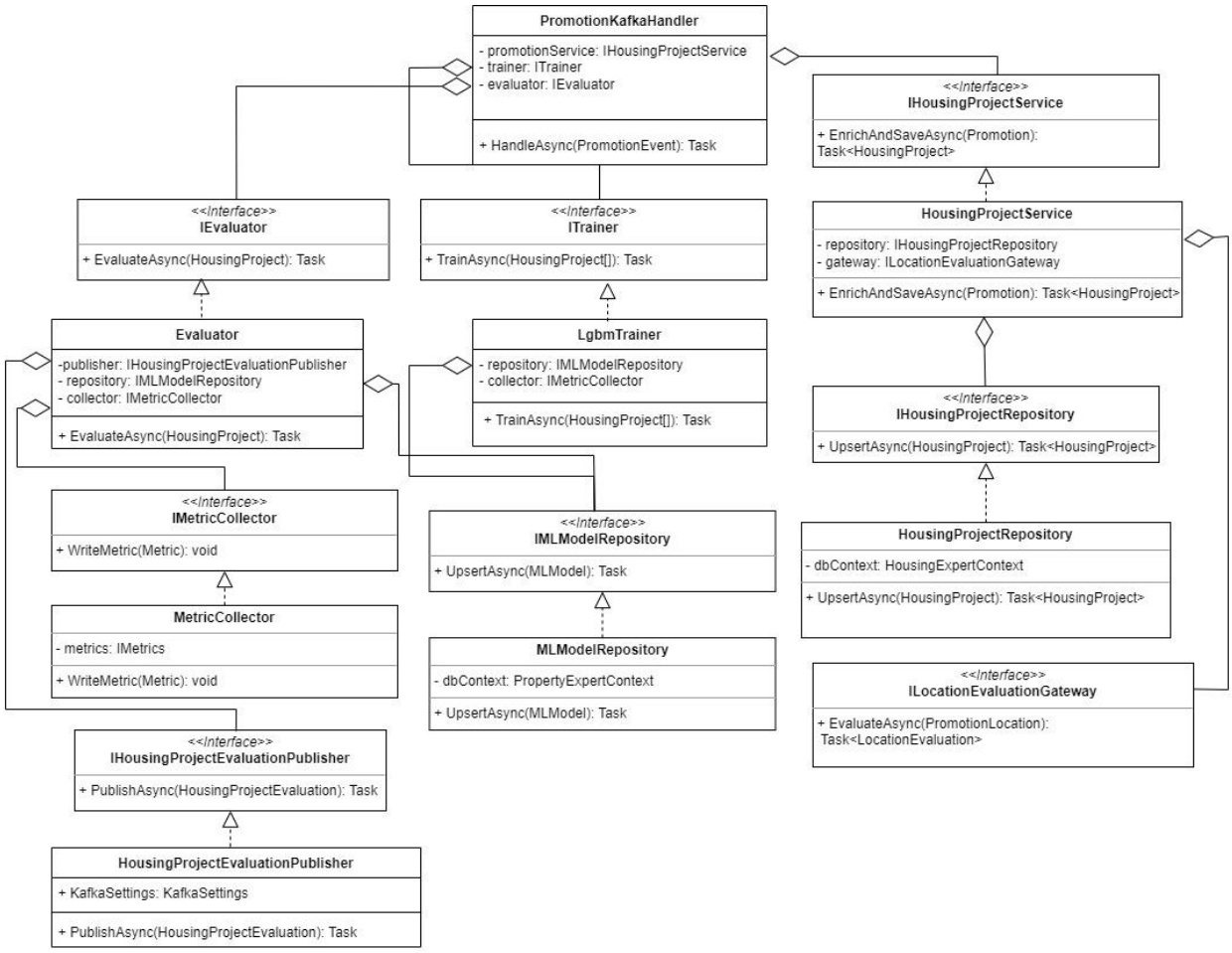

Рисунок 3.13 - Діаграма класів сервісу експертизи житлового комплексу

- PromotionKafkaHandler – Kafka consumer, обробник подій з топіку promotion. Координує процес зберігання даних використаних для оцінки житлового комплексу, тренування моделі, передбачення оцінки;
- HousingProjectService відповідає за дозаповнення та зберігання даних необхідних для оцінки житлового комплексу;
- HousingProjectRepository інкапсулює логіку по роботі з БД по даних необхідних для оцінки житлового комплексу;
- MLModeRepository інкапсулює логіку по роботі з БД по моделі навчання;
- LgmbTrainer відповідає за тренування моделі згідно алгоритму Lgmb (градієнтного бустінгу);
- Evaluator відповідає за оцінку даних з нових оголошень;
- MetricCollector відповідає за збір метрік;
- ILocationEvaluationGateway відповідає за взаємодію з сервісом експертизи локакції;
- HousingProjectEvaluationPublisher відповідає за публікацію оцінки в Kafka.

Розглянемо детально, які атрибути (features) будуть використані для оцінки вартості нерухомості в розрізі житла.

Таблиця 3.7 - Атрибути оцінки категорії житла

Атрибут	Обов’язковість	Нормалізований тип даних	Категорія
Площа	+	int64	-
Кількість кімнат	+	int64	-
Поверх	+	int64	-
Площа кухні	-	int64	-
Площа балкону	-	int64	-

Продовження таблиці 3.7

Індекс поверховості (поточний поверх/всього поверхів)	+	float	-
Кількість санвузлів	-	int64	-
Індекс якості ремонту	+	float	-
Оцінка вартості нерухомості в ЖК	+	float	-

Індекс поверховості розраховується як відношення номеру поточного поверху до усієї кількості поверхів. Індекс якості ремонту в поточній реалізації буду розраховуватися згідно з класу та стану ремонту:

- Чорнова обробка - 1;
- Косметичний ремонт – 2;
- Капітальний ремонт («євроремонт») – 3;
- Дизайнерський ремонт – 4.

Оцінка вартості нерухомості по умовам житлового комплексу отримується через запит до сервісу оцінки ЖК по POST /api/v1/housing-projects:estimate, вказана оцінка є необхідною для включення оцінки самого будинку. В тілі запиту передаються необхідні параметри для оцінки нерухомості в розрізі житлового комплексу (див. табл. 3.4).

При отримані повідомлення з даними оголошення зі схожим алгоритмом до обробки в сервісі оцінки житлового комплексу – виконується заповнення додаткових атрибутів(оцінка вартості нерухомості в ЖК), тренування моделі та

оцінка об’єкта з даних в отриманному оголошенні. Отримана модель зберігається в базі даних у вигляді масива байтів, також зберігається окремо в кеш-пам’яті. Дані, які використовуються при тренуванні також зберігаються в базі даних, це необхідно для відслідковування проблем та можливості виконати тренування моделі з певного часу.

При оцінюванні модель зчитується з кеша, виконується передбачення і результат публікується в Kafka. Парамети тренування, взаємодії з іншими залежностями (Kafka, HTTP API) визначається в окремому конфігураційному файлі, які зберігаються разом з усім проектом.

Виходячи з того, що в поточній моделі відсутні категоріальні ознаки, має сенс розглянути використання лінійного алгоритму – градієнтного пуску, порівняючи точність та продуктивність з градієнтним бустінгом на основі дерева рішень (див.табл.3.6)

Таблиця 3.8 - Порівняння результатів використання алгоритмів LightGbm та GradientDescent для оцінки вартості житла

	LightGbm	GradientDescent
RSquared	0.4	0.73
RootMeanSquaredError	389	125
Час тренування, мс	408	512
Час передбачення, мс	66	47
Розмір моделі, КБайт	45	30.6

Виходячи з результатів порівняння алгоритмів LightGbm та GradientDescent, отримуємо висновок що GradientDescent є більш ефективним алгоритмом для оцінки вартості житла при вказаній конфігруації функцій. Виконувати додаткову нормалізацію параметрів немає потреби через те, що усі параметри повинні мати значення в діапозпоні від 1 – 10000.

Діаграма класів має наступний вигляд (див.рис. 3.14):

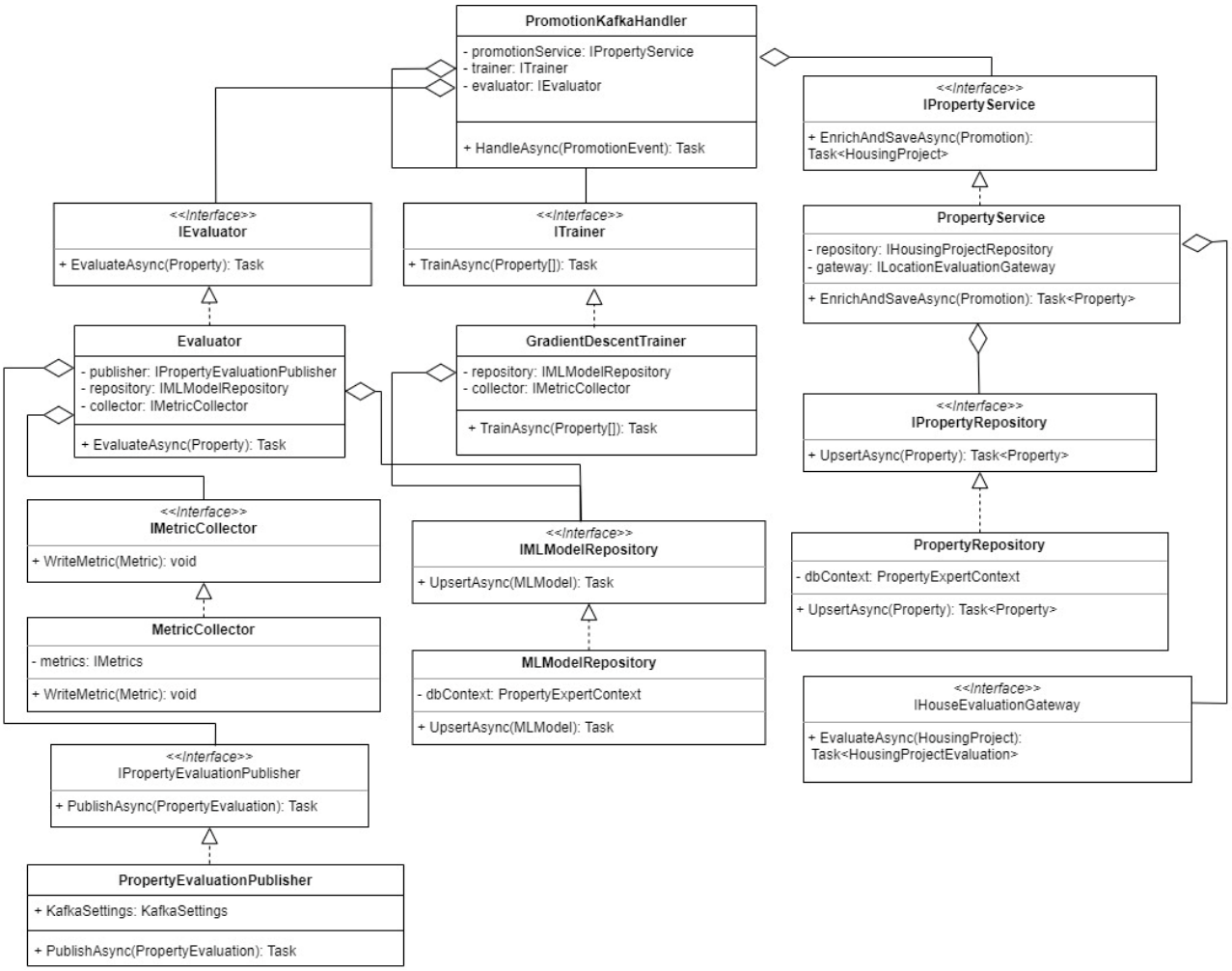

Рисунок 3.14 - Діаграма класів сервісу експертизи житла

- PromotionKafkaHandler – Kafka consumer, обробник подій з топіку promotion. Координує процес зберігання даних використаних для оцінки житла, тренування моделі, передбачення оцінки;
- PropertyService відповідає за дозаповнення та зберігання даних необхідних для оцінки житла;
- PropertyRepository інкапсулює логіку по роботі з БД по даних необхідних для оцінки житла;
- MLModeRepository інкапсулює логіку по роботі з БД по моделі навчання;

- GradientDescentTrainer відповідає за тренування моделі згідно алгоритму градієнтного пуску;
- Evaluator відповідає за оцінку житла з даних нових оголошень;
- MetricCollector відповідає за збір метрік;
- IHouseEvaluationGateway відповідає за взаємодію з сервісом експертизи житлового комплексу;
- PropertyEvaluationPublisher відповідає за публікацію оцінки в Kafka.

3.3 Розробка сервісу формування звітів

Сервіс формує звіт щодо очікуванної вартості об’єкту по певному оголошенню, де вказується:

- Ідентифікатор оголошення;
- Головні атрибути оголошення (країна, місто, житловий комплекс, кількість кімнат, житлова площа);
- Ресурс отримання оголошення;
- Дата створення оголололешення;
- Дата оновлення звіту;
- Вартість нерухомості за квадратний метр в оголошенні;
- Експертна оцінка вартості нерухомості за квадратний метр;
- Процент відхилення оціночної вартості від заявленної вартості за квадратний метр в оголошені;
- Оцінка очікуванної середньої вартості нерухомості за квадратний метр в локації оголошення;
- Оцінка очікуванної середньої вартості нерухомості за квадратний метр в житловому комплексі оголошення, якщо він явно зазначений.

Виходячи з цього сервіс акумулює інформацію отриману через Kafka повідомлення, зберігає результати в реляційній структурі (таблиці) в базі даних.

Звіт частково заповнюється з повідомлень від топіку promotions (загальні атрибути оголошення), і також містить результати оцінок від сервісів експертів – топік evaluations.

Після збереження результату звіт публікується в Kafka. Доступ до звітів також отримується за допомогою REST API. Для отримання списку звітів необхідно реалізувати ендпоінт – GET /api/v1/reports. У якості фільтрів використовується дата створення (оновлення) звіту, країна та місто. Сортування записів може виконуватися або по даті створення (оновлення) звіту, або по проценту відхилення оціночної вартості від заявленної вартості. Маючи фільтри і сортування відповідно потрібно підтримувати пагінацію через параметри offset та limit, де offset – визначає кількість звітів що треба пропусти, а параметр limit – кількість звітів, які беруться після offset. Використання пагінації можливо тільки при наявності сортуванні, тому в реалізації обов'язково має бути сортування по замовчунню – сортування по даті створення звіту.

Для більш ефективного пошуку необхідно використовувати додаткові індекси до таблиці звітів. Найбільш ефективною структурою для індексу в цьому випадку буде – btree, цей тип індексу покращує використання як фільтрації, так і сортування. Індекс hashindex[34] може бути більш швидкий в певних умовах, ніж btree індекс (O(log n) проти O(1)), але hashindex не підтримує оптимізацію сортування через те, що основа структури це хеш-таблиця, а не дерево. Також у більшості випадків, хеш-таблиця через наявну дефрегмантацію в масивах потребує більше пам'яті, ніж B-дерево. Повний список необхідних індексів:

- IX_updated_at_city – дата створення та місто;
- IX_city – місто.

Доступ до одного звіту відбувається через ендпоінт – GET /api/v1/reports/{reportId}. Виходячи з того, ідентифікатор звіту береться з ідентифікатору оголошення і виступає в ролі первинного ключа, індекс генерується автоматично.

Діаграму класів сервісу генерації звіту щодо очікуванної вартості об’єкту по оголошенню можно відобразити наступним чином(див.рис.3.15):

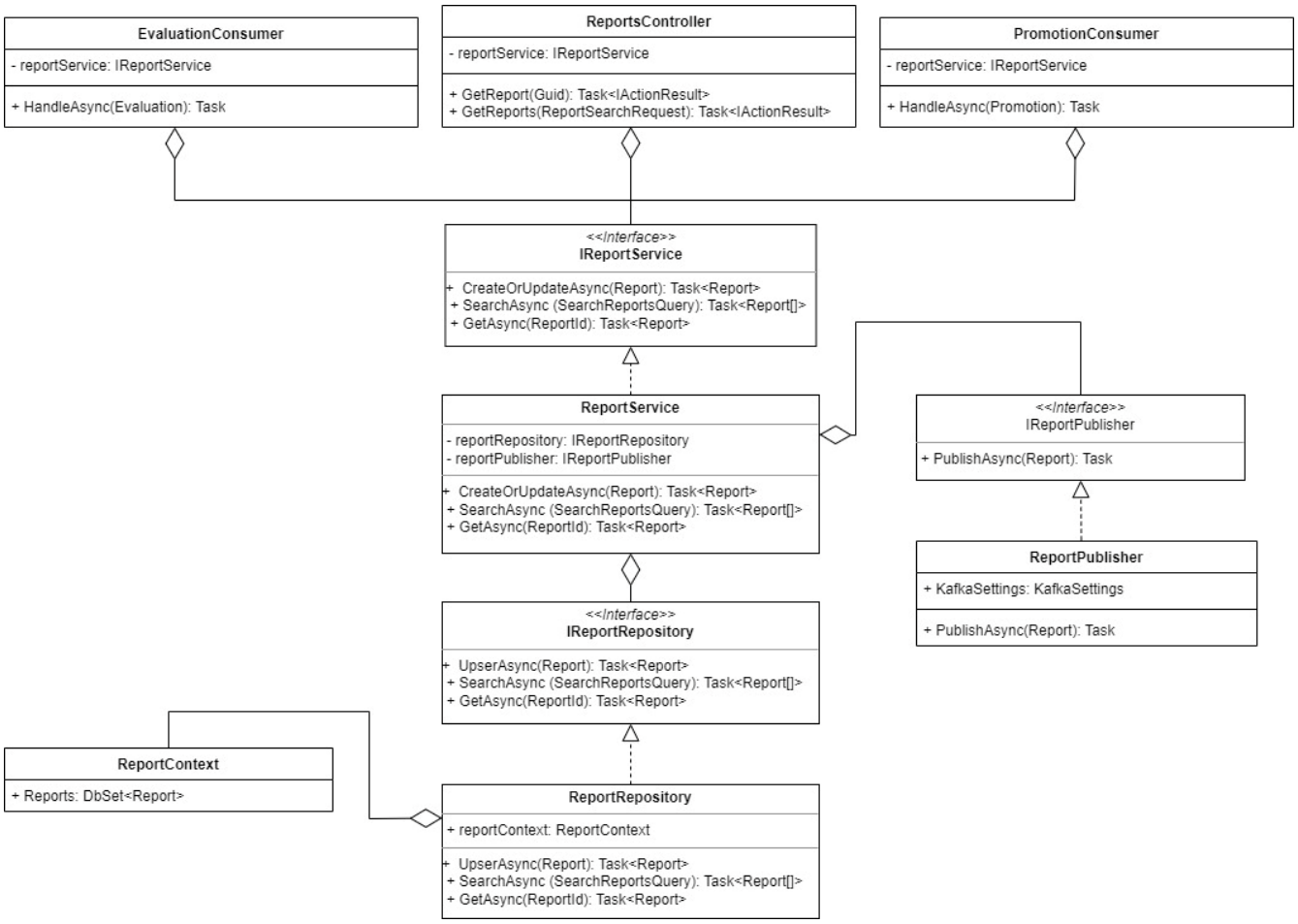

Рисунок 3.15 - Діаграму класів сервісу генерації звіту очікуванної вартості

- EvaluationConsumer – Kafka-consumer відповідальний за вичитування повідомлень з топіку evaluations та оновлення (або створення) звіту отриманими оцінками;
- PromotionConsumer - Kafka-consumer відповідальний за вичитування повідомлень з топіку promotions та оновлення (або створення) звіту головними атрибутами оголошення;
- ReportsController – реалізує REST API по пошуку та перегляду звітів;
- ReportService – відповідає за створення (або оноовлення), пошук звітів;

- ReportRepository – відповідно до ReportService безпосередньо інкапсулює логіку роботи з БД через використання ReportContext;
- ReportPublisher – відповідає за публікацію звітів в Kafka топік reports.

3.4 Висновок

В розділі розробка програмного забезпечення було описана реалізація сервісів відповідальних за завантаження даних, проведення експертної оцінки, генерування звітів. Для кожного сервісу була описана архітектура, діаграма класів, принципи взаємодії з іншими сервісами, алгоритми обробки даних. Для кожного сервісу експертної оцінки нерухомості було виконанно порівняльний аналіз між потенційними алгоритмами машинного навчання і обрано найбільш ефективний. Зформано список необхідних зовнішніх API, описано принцип взаємодії з ними, потенційні обмеження та ризики роботи.

4 РОЗГОРТАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Обгрунтування необхідних обчислювальних ресурсів

В процесі кваліфікаційної роботи було розроблено 5 окремих сервіса – сервіс завантаження даних з інтеграцією з DIM.RIA, сервіс оцінки локації нерухомості, сервіс оцінки житлового комплексу, сервіс оцінки житла, сервіс формування звітів. Кожен з сервісів має потенційну потребу в горизонтальному масштабованні – додаванні нових екземплярів серісу (формувати кластер). Кожен з серсів має потребу в реляційній базі. Сервіси завантаження даних та експертизи мають потребу в децентралізованій кеш-пам'яті. Окрім цього потрібен ще кластер з розподіленного сховища подій-повідомлень. Необхідні платформи/типи обчилювальних ресурсів:

- Реляційна БД – PostgreSQL;
- Розподіленне сховище даних повідомлень – Kafka;
- Розподілений кеш – Redis;
- Платформа розробка - .NET.

Серед основних гравців на ринку хмарних обчислень представлені AWS[35] (Amazon Web Services), Azure[36] (Microsoft) та GCP[37](Google Cloud platform). Усі вони є надають широкі можливості по роботі з потрібними обчислювальними ресурсами, але найбільшу частку ринку контролює AWS. AWS також має найбільший регіональний охват центрів даних по всьому світу. Загалом, AWS та Azure мають дуже схожі цінові політикі, які залежить від багатьох факторів – регіону розміщення серверу, рівню потужності, моделі розрахунку. Моделі розрахунку по суті розділяється на запланований час та обсяг, якщо використання розраховується за години або хвилини використання («Pay-as-you-go») без чіткого запланованого обсягу, що розглядається як оплата по факту використання, по результату це найдорожчя, але і найбільш гнучка модель. Найбільш ефективна

модель для довгострокових проектів, це «save when you commit» - оплата за сервіс з плановими зобов'язаннями на термін від одного до декількох років.

Для розгортання реалізованого програмного забезпечення буде використаний AWS через наявну можливість використання усіх необхідних залежностей, а також через наявність деталізованої документації та великої спільноти клієнтів. Необхідні обчислювані ресурси AWS представимо у вигляді таблиці 4.1.

Таблиця 4.1 – Необхідні обчислювальні ресурси AWS

Тип ресурсу	Тип AWS ресурсу	Кількість екземплярів
Реляційна БД	Amazon RDS	1 кластер, 5 баз даних
.NET Web Service	Amazon EC2	5 сервісів
Redis	Amazon ElastiCache	1 база даних
Kafka	Amazon MSK	1 кластер

Створення баз даних відбувається через AWS Console, де вказується необхідні налаштування:

- Ідентифікатор бази даних – має бути унікальним серед усіх баз даних в кластері;
- Права доступу – пароль, також можлива додаткова IAM аутенфікація;
- Клас екземпляру базу даних - кількість ядер ЦП, ОЗУ, пропускна спроміжність мережі;
- Розмір БД – вказується в ГБ, обмеження залежить від класу екземпляру, доступні RAM та CPU, права доступу;
- Тип сховища – SSD, HDD;
- Конфігурація автоматичного мастшабування;
- Конфігурація автоматичного резервування даних;
- Конфігурація автоматичного обслуговування серверу.

Для розгортання першої версії застосунку достатньо буде використовувати БД з класом db.t3.medium (2 ядра ЦП, 2 Гб ОЗУ, 2085 Мбіт пропускна здатність мережі).

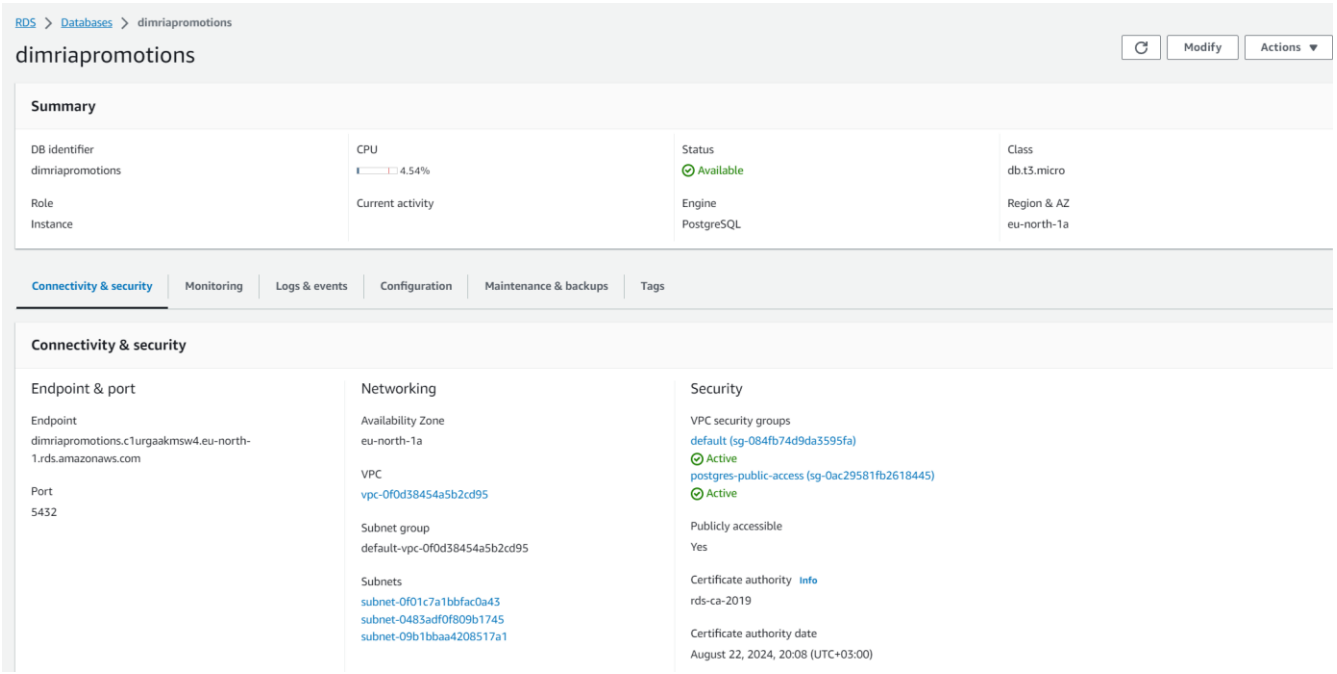

Рисунок 4.1 – Конфіграція бази даних DimRiaPromotions в AWS

Amazon EC2 надає можливості по створенню віртуальної машини, яка працює на основі AWS Cloud, що дає можливість розгортати серверні застосунки. При створенні віртуальної машини обов’язково треба враховувати тип операційної системи, розрядність архітектури. .NET кросплатформений фреймворк, що дає можливість використовувати і дистрибутиви Linux, і Windows. AWS має власний дистрибутив Linux - Amazon Linux, який претендує на більш кращу оптимізацію серверної обробки. Окрім класу і операційної системи екземпляру EC2, AWS також надає можливості налаштування:

- Мережеві налаштування (підмережа, міжмережевий екран, додавання публічної IP- адреси);
- Налаштування файлової системи;

- Кількість екзпемлярів.

Для розгортання першої версії віртуальної машіни достатньо використати екземпляр класу t3.medium (2 ядра ЦП, 4 ГБ ОЗУ). Приклад конфігурації створенної віртуальної машини для сервісу завантаження даних з інтеграцією DimRia (див.рис.4.2).

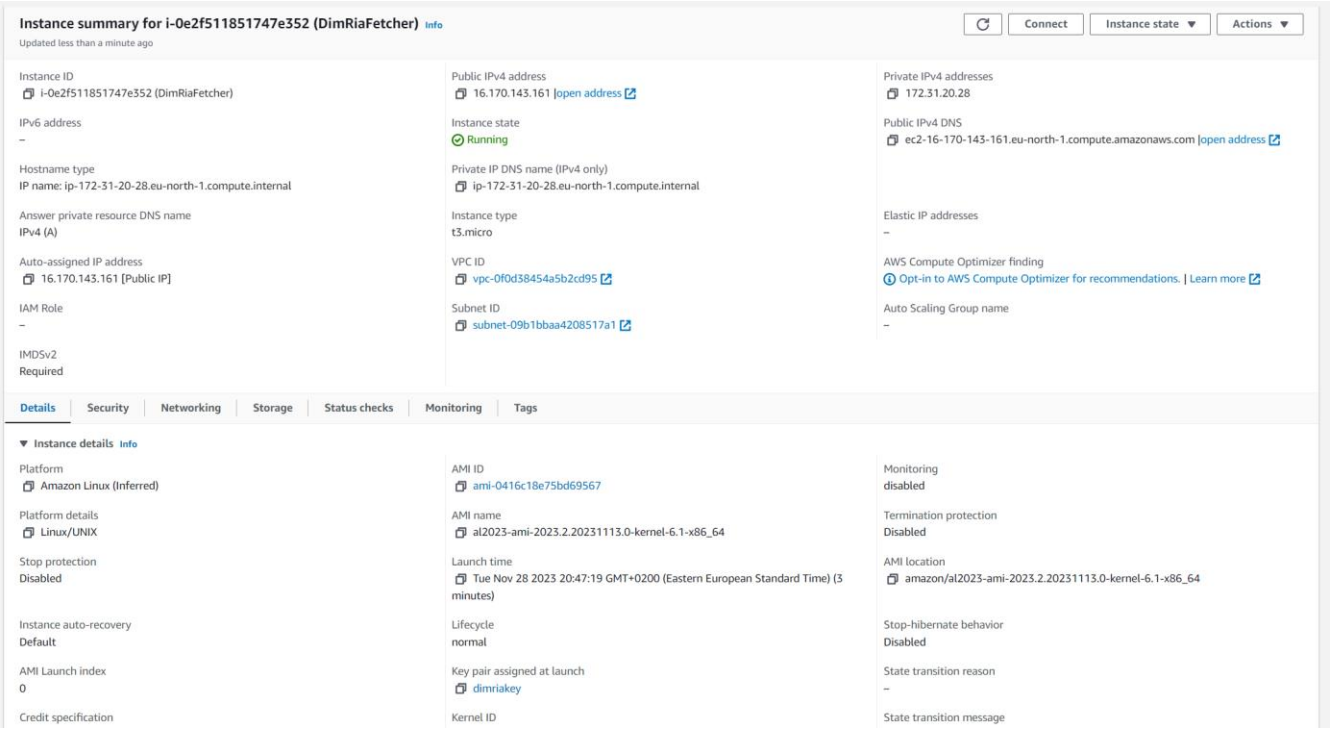

Рисунок 4.2 – Конфіграція віртуальної машини для сервісу завантаження даних

Amazon ElastiCache дозволяє створити «in-memory» базу даних в тому числі з Redis. AWS дозволяє обирати потрібну конфігурація з необхідним обсягом ОЗУ та пропускною спроміжністю. AWS пропонує наступні типи конфігурацій:

- Production – до 26.32 Гбайт та 10 Гбайт/с пропускної спроміжності мережі
- Dev – до 13.07 Гбайт та 10 Гбайт/с пропускної спроміжності мережі;
- Demo – до 0.5 Гбайт та 5 Гбайт/с пропускної спроміжності мережі.

Також є можливість налаштувати мережеву конфігурація – необхідна підмережа, необхідний IP протокол, тощо. Для розгортання «in-memory» БД

оберемо клас конфігурації - cache.r6g.large, яка дозволяє використовувати до 26 Гбайт ОЗУ та 5 Гбіт/с пропускної здатності мережі (див.рис.4.3).

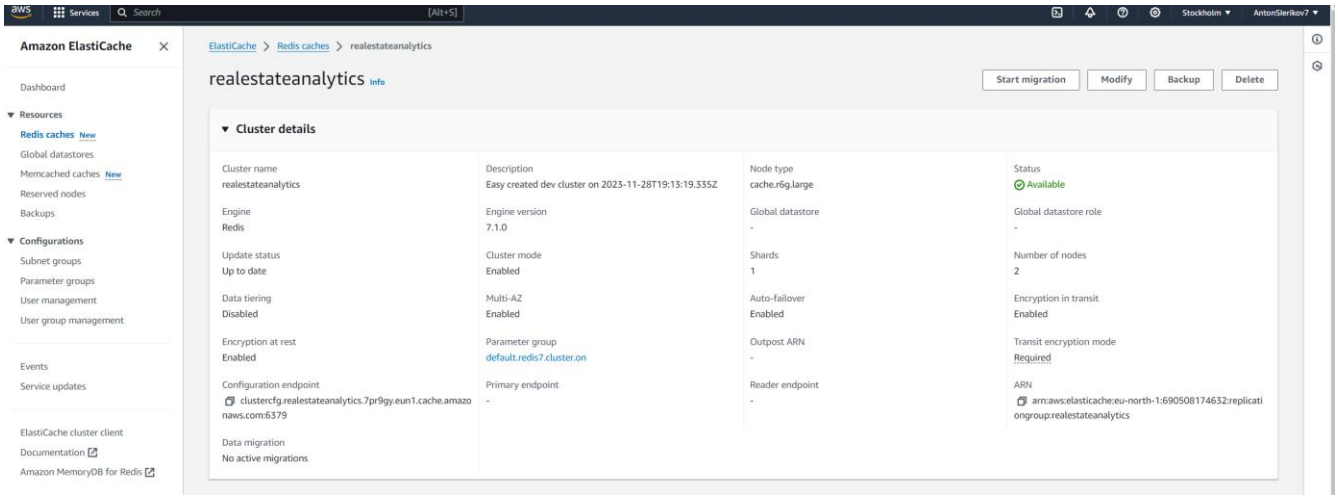

Рисунок 4.3 – Конфігурація Redis БД в Amazon ElastiCache

Amazon MSK дозволяє створити кластер Kafka, де можливо вказати конфігурацію необхідного класу брокеру повідомлень. Властивості конфігурації схожі до конфігруації бази даних - кількість ядер процесора, об’єм ОЗУ (в Гбайт), пропускна здатність мережі (Гбіт/с), об’єм сховища (в Гбайт). Додатково також можливо вказати версію Kafka. Вибір необхідного класу брокеру через конфігурацію також впливає на рекомендованну максимальну кількість можливих партіцій:

- t3.small (тільки для розробки) - 300 партіцій
- m7g.large – 1000 партіцій;
- m7g.2xlarge – 2000 партіцій;
- m7g.4xlarge – 4000 партіцій;

Для розгортання першої версії брокеру достатньо буде використати kafka.t3.small зі сховищем в 100 Гбайт, 2 ядрами процесора, 2 Гб ОЗУ, 5 Гбіт/с

пропускої здатності. Версію Apache Kafka рекомендується обрати останю стабільну (див.рис.4.4).

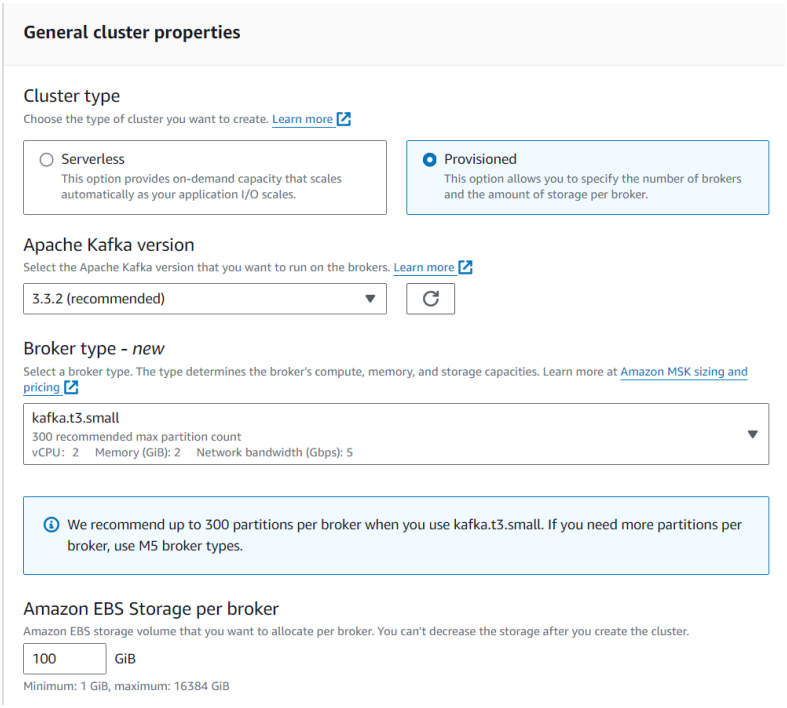

Рисунок 4.4 – Конфігурація Apache Kafka в Amazon MKS

4.2 Висновки

Були обгрунтовані необхідні обчислювальні ресурси, проаналізовані можливі провайдери хмарних обчислень. Для розгортання системи багатокритеріальної оцінки житлової нерухомості було обрано платформу Amazon AWS. Описана необхідна конфігурація і класи для розгортання RDS, EC2, ElastiCache, MKS.

5 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Тестування сервісу завантаження даних

Окрім написання модульних та інтеграційних тестів під час розробки, необхідно виконати мануальне тестування з використанням необхідних інтеграцій. Автоматизоване тестування є більш ефективним з точки зору часу та точності, ніж мануальне тестування, бо людина менш повільна, ніж машина, а також частіше робить помилки. Але коли мова їде про роботу з реальними даними та інтеграціями, в першу чергу виникає проблема перевищення допустимих лімітів при запуску автоматизованих тестів, створення конфліктів доступу до даних, також людина швидше реагує на розбіжності в реальних даних.

Для швидкого комплекстного тестування сервісу завантаження даних дуже ефективно використовується програми для відслідковування мережового трафіку (в тому числі і HTTP). Одним з таких інструментів є Fiddler[38], від дозволяє додавати фільтри на запити, які необхідно перехоплювати, і робити аналіз як логи запиту, так і логу відповіді. При завантаженні даних логи сервісу відображають роботу алгоритму описаного в пункті 3.1 (див.рис.5.1).

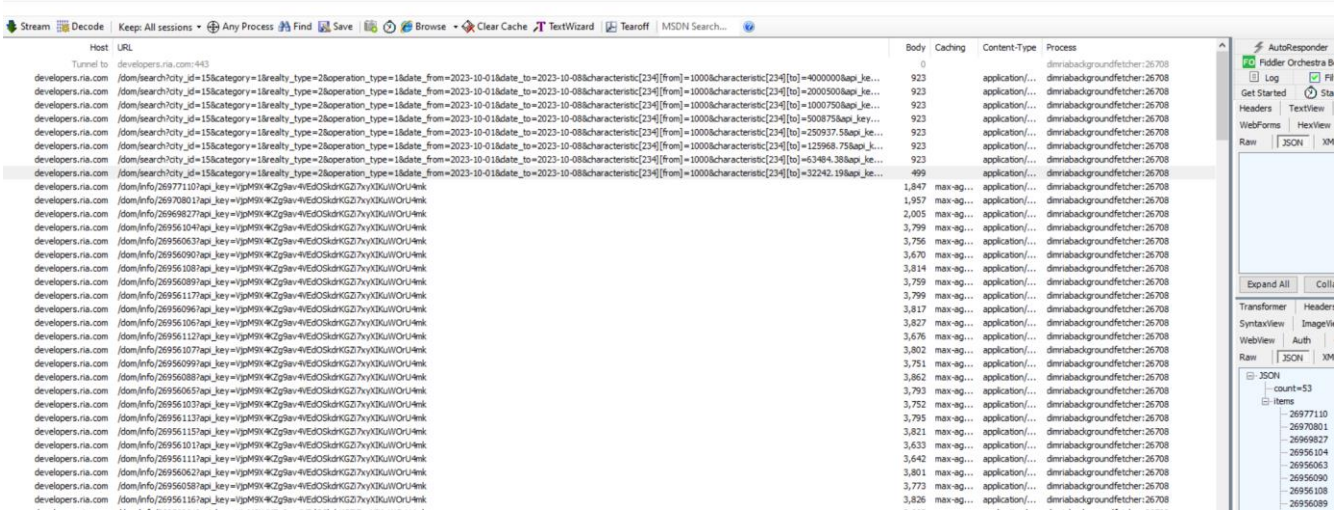

Рисунок 5.1 – Fiddler логи роботи сервісу завантаження даних

На восьмому запиту пошуку виконуються формування необхідного сегменту оголошень через те, що перші сім запитів пошуку містили в собі більше 100 елементів. По параметрам запиту можливо зробити висновок, що інтервал пошуку оголошень – тиждень, а фрагментація пошуку виконується через зміни лімітів повної вартості нерухомості, починаючи від інтервалу від 1000 до 4 млн, і закінчуючи від 1000 до 32242. Після отримання повного списку оголошень (53 елемента), виконується почергове завантаження оголошень з наступним збереженням даних в БД та Kafka.

5.2 Тестування сервісів оцінки нерухомості

Після того, як дані були отримані і заповнені сервісом оцінки локації нерухомості є змога перевірити корректність розрахунку інфраструктурних індексів. Розуміючи, що найвищі індекси мають тендецію належати до об’єктів в центральних районах міста, а найнижчі індекси спальним районам на околоцях. Для початку використаємо обрахуємо комплексну інфраструктурний індекс, як суму всіх інфраструктурих індексів:

$$I = I_c + I_s + I_t + I_l$$

де I – комплексний інфраструктурий індекс;

I_c – комерційний інфраструктурний індекс;

I_s - соціальний інфраструктурний індекс;

I_t - транспортний інфраструктурний індекс;

I_l – індекс культурних та зелених зон.

Маючи комплексний інфрастрактурний індекс виконаємо сортування локацій в порядку від найбільшого до найменшого. У результаті перші локації мають вказувати на більш відвідувані локації в містах, а останні навпроти менш відвідувані. Виконаємо перегляд локацій для таких міст як Львів, Вінниця, Тернопіль (див.рис.5.2):

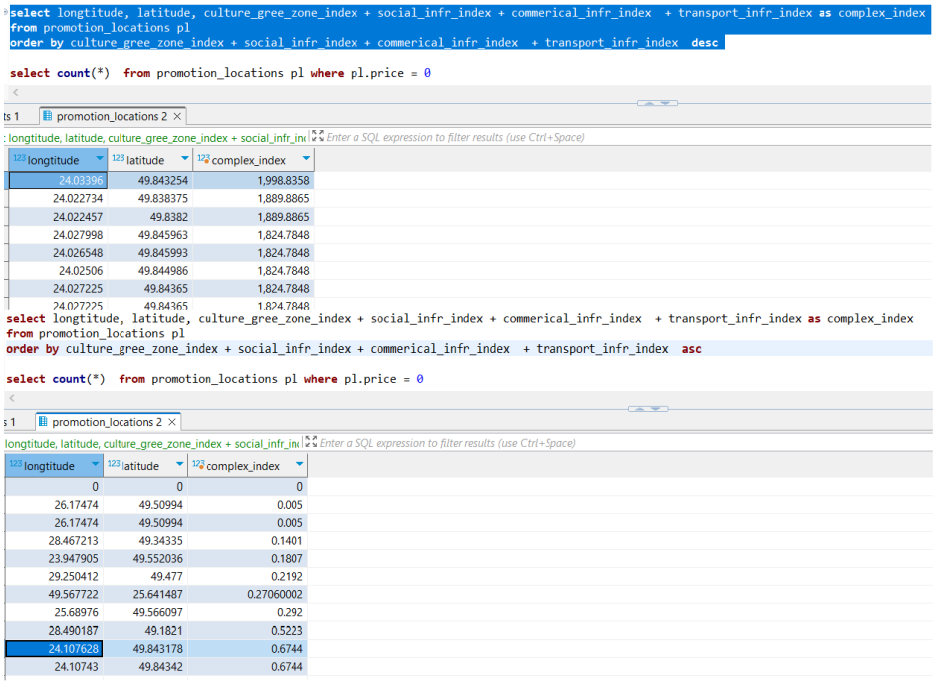

Рисунок 5.2 – Найбільш та найменш відвідувані локації за комплексним індексом у Львові, Вінниці, Тернополі

Відповідно до очікування найбільший індекс мають центральні вулиці. Наприклад, перші індекси належать до об’єктів нерухомості біля «Ринкової площі» у Львові, біля Європейської площі у Вінниці та біля Академічного обласного українського драматичного театру імені Т.Г.Шевченка. Ці локації – це центри містів, де є велика кількість в першу чергу культурних та комерційних зон. Найнижчий індекс очікувано належить до околиць міст – це район «Старе місто» в Вінниці, район «Скнилів» у Львові. Беручи до уваги інфрастрактурні індекси окремо один від одного можливо перевірити коректність розрахунку, так наприклад найблільший індекс культурних та зелених зон належить до об’єктів, які знаходяться поруч парку ім.Івана Фрака у Львові, району між центральним міськими парком ім.Леонтовича і річкою Південнний Буг у Вінниці.

Маючи вже натреновану модель передбачення вартості в розрізі локації для декількох міст виконаємо виконаємо тестові оцінювання через реалізованне HTTP API POST /api/v1/locations/estimate (див.рис.5.3)

Рисунок 5.3 – Оцінка вартості нерухомості в розрізі локації

Результати тестування передбачення вартості нерухомості в розрізі локації в декілької ітераціях відобразимо в таблиці 5.1.

Таблиця 5.1 - Результати тестування передбачення вартості нерухомості в розрізі локації

Місто	Район	Соц. індекс	Комер. індекс	Тран. індек	Культур. індекс	Індекс повітря	Вартість за кв.метр
Львів	Галицький	25	1200	0	700	84	1899.7
Львів	Сихівський	2	25	5	10	85	1473
Вінниця	Вишенька	4	79	5	33	83	1107
Вінниця	Старе місто	5	1	0	0	84	947
Тернопіль	Східний	1	39	0	12	85	835

Маючи перевірену модель передбачення вартості в розрізі локації та натреновану модель передбачення вартості житла виконаємо тестові передбачення (див. рис. 5.4).

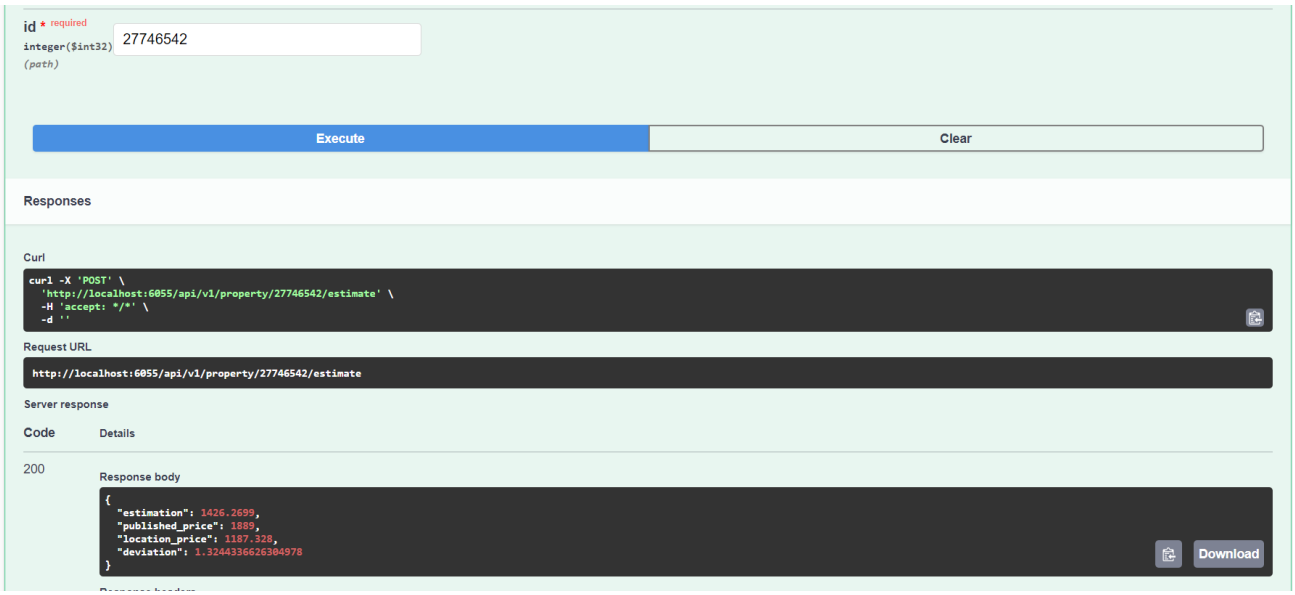

Рисунок 5.4 – Синхроне передбачення вартості житла по номеру оголошення

Результати тестування передбачення вартості житла з порівнням до калькутора DimRia в декількох ітераціях відобразимо в таблиці 5.2.

Таблиця 5.2 - Результати тестування передбачення вартості нерухомості в розрізі локації

Номер оголошення	Оголошенна ціна	Сервіс оцінки		DIM.RIA калькулятор	
		Вартість	Відхилення	Вартість	Відхилення
27746542	1889	1426	+32%	1314	+44%
26559557	1000	1260	-21%	1046	-5%
26981889	1353	1442	+6%	1573	-14%
27853586	1605	1826	-12%	1242	+29%

Продовження таблиці 5.2

Номер оголошення	Оголошена ціна	Сервіс оцінки		DIM.RIA калькулятор	
		Вартість	Відхилення	Вартість	Відхилення
25587772	1570	1537	+2%	1582	-1%
26735572	1349	1182	+14%	1287	+5%
26319599	760	920	-18%	814	-7%
27165019	1188	1127	+5%	864	+37%

Відповідно до результатів тестування можно зробити висновок, що середній відхил оцінки від оголошенної ціни з боку розробленого сервісу нижче на 7 відсотків, ніж у аналога – калькулятора DIM.RIA. Це може пояснити, тим що калькулятор DIM.RIA враховує меншу кількість факторів, використовує менш точну геолокацію. Також, DIM.RIA має більшу затримку на оцінювання, ніж розроблений сервіс оцінки: 200-300 мс проти 40-60 мс. Але в обох сервісах час затримки є прийнятним[39], і велика частина різниці у більшості формується через мережеві затримки.

5.3 Висновок

Були проведено тестування сервісів завантаження даних, сервісів оцінки нерухомості. Викано аналіз результатів тестування. Виконано порівняння оцінки вартості нерухомості між розробленим сервісом оцінки та калькулятором DIM.RIA.

6 ЕКОНОМІЧНА ЧАСТИНА

6.1 Оцінювання комерційного потенціалу розробки

Метою проведення комерційного та технологічного аудиту є оцінювання комерційного потенціалу впровадження методів та програмного забезпечення експертної багатокритеріальної оцінки житлової нерухомості.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету к.т.н., доцента Бабюк Н. П., к.т.н., доцента Ракитянську Г. Б., к.т.н., доцента Рейди О.М з кафедри програмного забезпечення.

Технологічний аудит проведемо з використанням таблиці 6.1 [40], де за п’ятибальною шкалою використовуючи 12 критеріїв оцінимо комерційний потенціал

Таблиця 6.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
#	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження табл. 6.1

4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років

Продовження табл. 6.1

12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 6.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Таблиця 6.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1. Бабюк Н. П.	2. Ракитянська Г. Б.	3. Рейда О. М.
	Бали, виставлені експертами:		
1	3	4	3
2	2	2	2
3	3	3	2
4	3	3	3
5	2	3	3
6	2	3	3
7	1	2	2
8	1	3	2
9	3	3	2
10	3	4	4
11	3	3	4
12	4	4	4
Сума балів	$СБ_1$=30	$СБ_2$=37	$СБ_3$=34
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{30+37+34}{3} = 34$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 34 бали, що згідно таблиці 6.3 вважається, що рівень комерційного потенціалу проведених досліджень є вище середнього.

Основними клієнтами системи експертної багатокритеріальної оцінки житлової нерухомості є як кінцеві продавці та покупці житла, так і агенції нерухомості, платформи з пошуку житла онлайн, які можуть інтегрувати продукт з їх системою. Сервіс буде надаватися як SaaS – «Software as a service» у вигляді веб-додатку або веб-служби як форму хмарних обчислень. Відповідно це вимагає враховувати в вартість підтримку інфраструктури – бази даних, сервера обчислень, шини даних, тощо.

На українському ринку не існує повноцінного аналога, який би виконував оцінку вартості існуючих об’єктів житлової нерухомості на оголошеннях автоматизовано. Існує калькулятор від компанії DIM.RIA, який за обмеженим переліком основних параметрів нерухомості виконає оцінку вартості гіпотетичного об’єкту, але калькулятор не інтегрований з існуючими оголошеннями, де присутня більша кількість критеріїв, і реальні дані.

В якості аналога для розробки було обрано рішення від польського сервісу otodom. Основними недоліками аналога є затримка в оцінці об’єктів, суттєва частина об’єктів на ресурсі не оцінені. Діапазон оцінки вартості нерухомості може значно коливатися – до 20%, але це може бути викликано рідкістю певних об’єктів. Перевагами аналогу є деталізація аналізу в більше ніж 140 факторів, швидкість завантаження інформації після першої оцінки, і суттєвою ознакою є автоматичне використання сервісу оцінки на більшості оголошень, це означає що сервіс більш менш впевнений в точності і якості свого рішення.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом. В таблиці 6.4 наведені основні техніко-економічні показники аналога і нової розробки.

Таблиця 6.4 – Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
1	2	3	4	5
Максимальна межа розходження діапазону оцінки вартості, %	20	15	1.33	25%
Час оцінки вартості, мс	1500	800	1.875	25%
Час завантаження результату після оцінки, мс	100	80	1.25	10%
Відмовістійкість, %	95	95	1	20%
Кількість факторів в моделі оцінки	146	89	0.6	20%

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Визначимо відносні одиничні показники якості по кожному параметру за формулами (6.1) та (6.2) і занесемо їх у відповідну колонку табл. 6.5.

$$q_i = \frac{P_{Hi}}{P_{Бi}} \tag{6.1}$$

або

$$q_i = \frac{P_{Бi}}{P_{Hi}} \tag{6.2}$$

де P_{Hi}, $P_{Бi}$ – числові значення *i*-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{20}{15} = 1.33;$$

$$q_2 = \frac{1500}{500} = 3;$$

$$q_3 = \frac{100}{80} = 1.25;$$

$$q_4 = \frac{95}{95} = 1;$$

$$q_5 = \frac{89}{146} = 0.6;$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \alpha_i , \quad (6.3)$$

$$К_{я.в.} = 1.33 * 0.25 + 1.875 * 0.25 + 1.25 * 0.1 + 1 * 0.2 + 0.6 * 0.2 = 1.246$$

Відносний коефіцієнт показника якості нової розробки більший одиниці, отже нова розробка незначно якісніше базового товару-конкурента, але враховуючи що на українському ринку аналогу взагалі немає, то товар рекомендовано реалізувати на українському ринку, і після виконання нових покращень і залагоджень розглядати вихід на інші європейські ринки. В першу чергу треба розглядати ближчі ринки нерухомості, де присутні велика діаспора українців. Це ринки Польщі, Естонії, Латвії, Литви та Молдови.

Наступним кроком є визначення конкурентоспроможності товару. Конкурентоспроможність товару є головною умовою конкурентоспроможності підприємства на ринку і важливою основою прибутковості його діяльності.

Однією із умов вибору товару споживачем є збіг основних ринкових характеристик виробу з умовними характеристиками конкретної потреби покупця. Такими характеристиками найчастіше вважають нормативні та технічні параметри, а також ціну придбання та вартість споживання товару.

В табл. 6.5 наведено технічні та економічні показники для розрахунку конкурентоспроможності нової розробки відносно товару-аналога, технічні дані взяті з попередніх розрахунків.

Таблиця 6.5 – Нормативні, технічні та економічні параметри нової розробки і товару-виробника

Показники	Варіанти	
	Базовий (товар-конкурент)	Новий (інноваційне рішення)
1	2	3
1. Нормативно-технічні показники		
Максимальна межа розходження діапазону оцінки вартості	20	15
Час оцінки вартості	1500	800
Час завантаження результату після оцінки	100	80
Відмовістійкість	95	95
Кількість факторів в моделі оцінки	146	89
2. Економічні показники		
Ціна придбання, млн грн.	7	3

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{6.4}$$

де $I_{m.n.}$ – індекс технічних параметрів; $I_{e.n.}$ – індекс економічних параметрів. Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (6.5)

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{Бei}}, \tag{6.5}$$

де P_{Hei}, $P_{Бei}$ – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

$$I_{е.п.} = \frac{3}{7} = 0.43$$

$$К = \frac{1.246}{0.43} = 2.9$$

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде більш конкурентоспроможне, ніж конкурентний товар.

6.2 Прогнозування витрат на виконання науково-дослідної роботи

1. Витрати пов’язані на проведення науково-дослідних робіт можливо скласти в наступні статті

- Витрати на оплату праці;
- Витрати на матеріальні засоби (канцтовари, обладнання);
- Витрати на інструментальне та інфраструктурне програмне забезпечення.

Основна заробітна плата кожного із залучених осіб визначається за формулою (6.6)

$$З_0 = \sum_{i=1}^{K} \frac{M_{ni} * t_i}{T_p} \text{ (грн)} \qquad (6.6)$$

де M – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$T_р$– число робочих днів в місяці; оцінимо середню кількість робочих днів приблизно $T_р = 22$;

t - кількість робочих днів роботи розробника.

Для розробки системи багатокритеріальної оцінки житлової нерухомості необхідно залучити – Engineering Manager, Business Analyst та Software Engineer. Посадові оклади, число днів роботи та витрати на компенсацію додаємо в таблицю 6.6.

Таблиця 6.6 - Компенсація спеціаліста в дослідницькій установі

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Engineering Manager	36000	1636	35	57260
Business Analyst	25000	1136	15	17040
Software Engineer	28000	1273	35	44555
Всього				118855

2. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх робітників, які приймали участь в розробці нового технічного рішення розраховується як 10 - 15 % від основної заробітної плати робітників як премія.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 10% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{дод}}{100\%} \quad (6.7)$$

$$З_д = 0{,}1 * 118855 = 11886 \text{ грн}$$

3. Нарахування на заробітну плату $Н_{ЗП}$ робітників, які брали участь у виконанні роботи, розраховуються за формулою (6.9):

$$З_н = (З_о + З_р + З_д) * \frac{Н_{ЗП}}{100} \text{ (грн)} \quad (6.8)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

$Н_{ЗП}$ – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Основна ставка єдиного внеску на загальнообов'язкове державне соціальне страхування на 2023 рік – 22 %, тоді:

$$З_н = (118855 + 11886) * 0.22 = 28763 \text{ (грн)}$$

4. Витрати на комплектуючі вироби, які використовують при виготовленні одиниці продукції, розраховуються, згідно їх номенклатури, за формулою:

$$К=\sum_{i=1}^{n} Н_i \cdot Ц_i \cdot К_i \quad , \tag{6.9}$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.;

$Ц_i$ – покупна ціна комплектуючих *i*-го найменування, грн.;

$К_i$ – коефіцієнт транспортних витрат (1,1…1,15).

Потрібно закладати витрати на доставку у вигляді коефіцієнту транспортних витрат – 1.1. Інформацію про використані матеріали та комплектуючі подаємо у вигляді табл. 6.7.

Таблиця 6.7 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Дошка магнітно-маркерна Nota Bene 90x60 см	843	1	843
Набір маркерів Centropen для білої дошки	175	1	175
Папір офісний Maestro A3 160 г/м білий 250 аркушів	928	1	928
Накопичувач Kingston DT KYSON 128 ГБ	575	1	150
Степлер Exakt-2 Axent	112	1	112
Всього			2208
З врахуванням коефіцієнта транспортування			2429

5. Використані засоби програмного забезпечення включає в себе витрати на інструментальне програмне забезпечення – середа розробки, клієнти взаємодії, тощо, так і інфраструктурні інструменти. Розгортання і підтримка повноцінної власної інфраструктури дуже коштовна справа, на перших етапах розвитку продукту більш доцільно обрати платформи хмарних обчислень – AWS. Балансову вартість програмного забезпечення розраховують за формулою:

$$В_{прг} = \sum_{i=1}^{k} Ц_{іпрг} \cdot С_{пргі} \cdot К_і, \quad (6.10)$$

де $Ц_{іпрг}$ – ціна придбання/використання одиниці програмного засобу цього виду, грн;

$С_{пргі}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

$К_і$ – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо ($К_і$ = 1,10…1,12).

k – кількість найменувань програмних засобів.

Отримані результати занесемо в таблицю 6.8.

Таблиця 6.8 – Витрати на використання програмних засобів по кожному виду

Найменування устаткування	Час використання, місяців	Ціна за місяць, грн	Вартість
Підписка JetBrains Rider	3	548	1644
AWS MSK Kafka	3	2668	8004
AWS EKS	3	2632	7896
Всього			17544

6. Амортизація обладнання, комп’ютерів та приміщень, які використовувались під час виконання роботи

Дані відрахування розраховують по кожному виду обладнання, приміщенням тощо.

$$А_{обл} = \frac{Ц_б}{Т_в} * \frac{t_{вик}}{12} \quad (6.11)$$

де $Ц_б$ – балансова вартість даного виду обладнання (приміщень), грн.;

$t_{вик}$ – час користування;

$Т_в$ – термін використання обладнання (приміщень), цілі місяці.

Для розробки продукту використовувався персональний комп’ютер вартістю 30000 грн

Таблиця 6.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Ноутбук (3)	90000	4	2	3750
Офісне приміщення	1 500 000	25	2	10000
Всього				13750

7. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \quad (6.12)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

При використані 3 комп'ютерів з сумарною потужністю 1.5 кВт витрачалася електроенергія з ціною 7.6 грн за кВт в 2023 році.

$$B_e = \frac{1.5 \cdot 480 \cdot 7.6 \cdot 0.45}{0.76} = 3240$$

До статті «Службові відрядження» належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень. Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = \left(З_о + З_р\right) \cdot \frac{Н_{св}}{100} \quad (6.13)$$

де $Н_{св}$ – норма нарахування за статтею «Інші витрати».

$$B_{св} = 118855 * 0.22 = 26148 \text{ грн}$$

До статті «Витрати на роботи, які виконують сторонні підприємства, установи і організації» належать витрати на проведення досліджень, що не можуть бути виконані штатними працівниками або наявним обладнанням організації, а виконуються на договірній основі іншими підприємствами, установами і організаціями незалежно від форм власності та позаштатними працівниками.

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуються як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = \left(З_о + З_р\right) \cdot \frac{Н_{сп}}{100} \quad (6.14)$$

де $Н_{пв}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації».

$$В_{сп} = 118855 * 0.35 = 41599 \text{ грн}$$

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100} \quad (6.15)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати».

$$І_в = 118855 * 0.6 = 71313 \text{ грн}$$

Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як 110% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100}, \quad (6.16)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$В_{нзв} = 118855 * 1.1 = 130741 \text{ грн}$$

Сума всіх попередніх статей витрат дає витрати, які необхідні для повноцінної розробки системи:

$$В = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв} \quad (6.17)$$

$$В = 118855 + 11886 + 28763 + 2429 + 17544 + 13750 + 3240 + 26148 + 41599 + 71313 + 130741 = 466268 \text{ грн}$$

Прогнозування загальних втрат ЗВ на виконання та впровадження комерційної розробки системи багатокритеріальної оцінки житлової нерухомості здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (6.18)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії дослідного зразка, то коефіцієнт β = 0.5.

Звідси:

$$ЗВ = \frac{466268}{0.5} = 932536 \text{ грн.}$$

6.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою (6.19):

$$\Delta П_i = \sum_1^n \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right) \quad (6.19)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

Цо – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

л – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333.

р – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

х – ставка податку на прибуток. У 2023 році – 18%.

Впровадження результатів дослідження підвищення якості багатокритеріальної оцінки в агенціях з нерухомістю дозволить більш точно і швидше виконувати оцінку. Припустимо, очікувана середня стандартна комісія від реалізації нерухомості агенції – 2%, середня вартість квартири в Києві – ~120 000 $ або 4340400 грн по курсу 36.56. Тобто середня комісія за квартиру дорівнює 86 тис грн. Згідно з Мінюста, кількість угод купівлі-продажу житла в 2021 году в Києві приблизно 41000. Допустимо, що агенція реалізує 3% усього ринку нерухомості, тобто приблизно 1230 об'єктів за рік.

Припустимо, що більша точність дозволить реалізувати якісні об'єкти нерухомості за більшою ціною і середня комісія за квартиру зросте на 4 тис грн до 90 тис.грн, а кількість реалізованих об'єктів в перший рік зросте на 20, в другий на 50, і в третій на 50.

$$\Delta П_1 = [4000 \cdot 1230 + (86000 + 4000) \cdot 20] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right)$$

$$= 6720000 * 0.2458 = 1\,651\,776 = 1.65 \text{ млн}$$

$$\Delta П_2 = [4000 \cdot 1230 + (86000 + 4000) \cdot (20 + 50)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right)$$

$$= 11\,220\,000 * 0.2458 = 2\,757\,876 = 2.75 \text{ млн}$$

$$\Delta П_3 = [4000 \cdot 1230 + (86000 + 4000) \cdot (20 + 50 + 50)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right)$$

$$= 15\,720\,000 * 0.2458 = 3\,863\,976 = 3.86 \text{ млн}$$

6.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ} \qquad (6.20)$$

$k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}} = 2…5$).

$$PV = 2 \cdot 932536 \ = 1\,865\,072$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$E_{абс} = (ПП - PV) \qquad (6.21)$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (6.22)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$ПП = \frac{1651776}{(1+0{,}2)^1} + \frac{2757876}{(1+0{,}2)^2} + \frac{3863976}{(1+0{,}2)^3} = 1376480 + 1915192 + 2236097$$

$$= 5\ 527\ 769 \text{ грн.}$$

$$Е_{абс} = (5\ 527\ 769 - 1\ 865\ 072\ ) = 3\ 662\ 697 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{6.23}$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$$Е_в = \sqrt[3]{1 + \frac{3\ 662\ 697}{1\ 865\ 072}} - 1 = \sqrt[3]{1 + 1.96} - 1 = 1.436 - 1$$

$$Е_в = 0.436 = 44\ \%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{6.24}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d= (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau = 0.16 + 0.05 = 0.21$$

Так як $E_в > \tau_{min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в} \tag{6.25}$$

$$\mathrm{T_{ок}} = \frac{1}{0.436} = 2.3 \text{ роки} = 27.6 \text{ міс} = 2 \text{ роки та 4 міс}$$

Так як $T_{ок} \leq$ 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

6.5 Висновки до економічного розділу

Було проведено оцінку комерційного потенціалу розробки програмного забезпечення системи багатокритеріальної оцінки житлової нерухомості, яка є на вище середньому рівні. При порівнянні нової розробки з аналогом виявлено, що вона є якіснішою і конкурентоспроможною відносно аналога, а також краще по частці технічних і економічних показників.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 466 268 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 932536 грн.

Вкладені інвестиції в даний проект окупляться через 2 роки та 4 місяці при прогнозованому прибутку 5 527 769 грн. за три роки.

ВИСНОВКИ

Були проаналізовані потенційні фактори, що можуть впливати на оцінки житлової нерухомості; було розглянуто можливу пріоритетність впливу певних факторів. Було сформовано можливі групи критеріїв оцінок житлової нерухомості. Було виконано аналіз роботи систем-аналогів: DIM.RIA та otodom. Було виявлено ряд переваг і недоліків в поточних реалізаціях аналогів, зроблені відповідні висновки щодо функціональних та нефункціональних вимоги до системи оцінювання. Розроблено постановку завдання з описом необхідних задач для вирішення та нефункціональних вимог, які необхідно підтримувати.

Був розроблений метод багатокритеріальної оцінки нерухомості, який дозволить підвищити точність-швидкість оцінювання, більшу здатність до функціонального та навантажувального масштабування системи. Розроблена архітектура інформаційної системи експертної оцінки нерухомості.

Була описана реалізація сервісів відповідальних за завантаження даних, проведення експертної оцінки, генерування звітів. Для кожного сервісу була описана архітектура, діаграма класів, принципи взаємодії з іншими сервісами, алгоритми обробки даних. Для кожного сервісу експертної оцінки нерухомості було виконано порівняльний аналіз між потенційними алгоритмами машинного навчання і обрано найбільш ефективний. Складений список необхідних зовнішніх API, описано принцип взаємодії з ними, потенційні обмеження та ризики роботи.

Виконано розгортання та тестування системи. Виконано порівняння оцінки вартості нерухомості між розробленим сервісом оцінки та калькулятором DIM.RIA.

Було проведено оцінку комерційного потенціалу розробки програмного забезпечення системи багатокритеріальної оцінки житлової нерухомості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Commercial Real Estate. United States. [Electronic resource] – Access Mode: https://www.statista.com/outlook/fmo/real-estate/commercial-real-estate/united-states.
2. Denton, John A. Society and the official world: a reintroduction to sociology. Dix Hills, N.Y: General Hall. 1990 - c. 17.
3. How Has COVID Affected the Housing Market. [Electronic resource] – Access Mode: https://www.fastexpert.com/blog/housing-market-after-covid/
4. Сєріков А.І, Катєльніков Д.І. Розробка експертної системи багатокритеріальної оцінки житлової нерухомості в ділових іграх. Доступ: Збірник матеріалів Міжнародної науково-практичної Інтернет-конференції «Електронні інформаційні ресурси: створення, використання, доступ», 20-21 листопада 2023 р - с.258-259.
5. Сєріков А.І, Катєльніков Д.І. Розробка експертної системи багатокритеріальної оцінки житлової нерухомості в ділових іграх. Доступ: Збірник матеріалів Міжнародної науково-технічної конференції молодих вчених, аспірантів та студенті «Комп'ютерні ігри і мультимедіа як інноваційний підхід до комунікацій», 28-29 вересня 2023 р - с.122-123.
6. Сєріков А.І. Solving problems with data updates in hold load systems. Доступ: Збірник матеріалів LII Науково-технічної конференції факультету будівництва, цивільної та екологічної інженерії. м.Вінниця – 2023 р.
7. Ціни на квартири в Вінниці. [Електронний ресурс] – Режим доступу: https://dom.ria.com/uk/prodazha-kvartir/vinnitsa/ceny/
8. Ціни на квартири в Києві. [Електронний ресурс] – Режим доступу: https://dom.ria.com/uk/prodazha-kvartir/kiev/ceny/
9. Los Angeles: housing price and price/m. [Electronic resource] – Access Mode: https://www.properstar.com/united-states/los-angeles/house-price

10. Ліквідність квартири. [Електронний ресурс] – Режим доступу: https://dom.ria.com/uk/articles/likvidnaya-kvartira-kak-vybrat-i-osnovnye-kharakteristiki-173773.html
11. Zoning. [Electronic resource] – Access Mode: https://www.townandcountryplanninginfo.com/2020/11/zoning.html
12. Ivana Branic. Housing Quality Assessment Criteria - University of Osijek, Faculty of Civil Engineering Osijek, 2017. p.38.
13. Класи нерухомості. [Електронний ресурс] – Режим доступу: https://riel.ua/blogs/u-chomu-riznicya-mizh-zhitlom-komfort-klasu-i-ekonomom
14. Калькулятор нерухомості DIM.RIA. [Електронний ресурс] – Режим доступу:: https://dom.ria.com/uk/kalkuljator-stoimosti/prodazha-kvartir/
15. Otodom. [Electronic resource] – Access Mode: https://www.otodom.pl/
16. Distributed Parallel Training: Data Parallelism and Model Parallelism [Electronic resource]. – Access mode: https://towardsdatascience.com/distributed-parallel-training-data-parallelism-and-model-parallelism-ec2d234e3214
17. Speed Still Maters [Electronic resource] – Access Mode: - https://blog.codinghorror.com/speed-still-matters/
18. Martin Kleppmann. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems – Sebastopol, O`Reilly Media Inc. 2017. p.136-139.
19. Rebalance & Partition Assignment Strategies in Kafka. [Electronic resource] – Access Mode: - https://blog.codinghorror.com/speed-still-matters/https://medium.com/trendyol-tech/rebalance-and-partition-assignment-strategies-for-kafka-consumers-f50573e49609
20. Postgres vs. MySQL: a Complete Comparison in 2023. [Electronic resource] – Access Mode: https://www.bytebase.com/blog/postgres-vs-mysql/
21. Key differences between Protobuf vs JSON. [Electronic resource] – Access Mode: https://www.educba.com/protobuf-vs-json/

22. Erik Demaine, Srini Devadas and Nancy Lynch. Binary Trees. 6.006 Introduction to Algorithms. – MIT OpenCourseWare. 2020. p 1- 5.
23. Erik Demaine, Srini Devadas and Nancy Lynch. B-Trees. 6.006 Introduction to Algorithms. – MIT OpenCourseWare. 2020. p 5 -7
24. DIM RIA. Ліміти Веб-Сервісів. [Електронний ресурс] – Режим доступу: https://api-docs-v2.readthedocs.io/ru/latest/general-documentation/rate_limits.html
25. Train and evaluate a model. [Electronic resource] – Access Mode: - https://learn.microsoft.com/en-us/dotnet/machine-learning/how-to-guides/train-machine-learning-model-ml-net
26. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional. 1995. - p.185
27. Gradient Descent Algorithm. [Electronic resource] – Access Mode: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21
28. Gradient Boosting Algorithm − Part 1. Regression. [Electronic resource] – Access Mode: https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502
29. Guolin Ke, , Qi Meng, Thomas Finle, Taifeng Wang. LightGBM: A Highly Efficient Gradient Boosting Decision Tree - 31st Conference on Neural Information . – Processing Systems Long Beach, CA, USA. 2017
30. Leo Breiman. Classification and Regression Trees. Routledge. 1984 – p.130
31. Google Maps Platform Documentation. [Electronic resource] – Access Mode: https://developers.google.com/maps/documentation
32. PostgreSQL. Select for update. [Electronic resource] – Access Mode: https://www.postgresql.org/docs/current/sql-select.html

33. Pattern decorator. [Electronic resource] – Access Mode: https://refactoring.guru/design-patterns/decorator
34. PostgtreSQL. Index Types. [Electronic resource] – Access Mode: https://www.postgresql.org/docs/current/indexes-types.html
35. AWS Documentation. [Electronic resource] – Access Mode: https://docs.aws.amazon.com/
36. Azure Documentation. [Electronic resource] – Access Mode: https://learn.microsoft.com/en-us/azure
37. GCP Documentation. [Electronic resource] – Access Mode: https://cloud.google.com/docs
38. Fiddler Filters. [Electronic resource] – Access Mode: https://docs.telerik.com/fiddler/knowledge-base/filters#filters-reference
39. Fiona Fui-Hoon Nah. A Study on Tolerable Waiting Time: How Long Are Web Users Willing to Wait? 9th Americas Conference on Information Systems, AMCIS 2003 - Tampa, FL, USA, August 4-6, 2003
40. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

ДОДАТКИ

Додаток А

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"19" вересня 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу «Розробка методів і програмних засобів підвищення якості експертної багатокритеріальної оцінки житлової нерухомості» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

_____ к.т.н., доцент Д.І.Катєльніков

"19" 09 2023 р.

Виконав:

_____студент гр.2ПІ-22м А.І. Сєріков

"19" 09 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмних засобів підвищення якості експертної багатокритеріальної оцінки житлової нерухомості».

Галузь застосування – агрегати веб-оголошень та системи житлової нерухомості.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення якості експертної багатокритеріальної оцінки житлової нерухомості за рахунок розподілення експертизи між декількома окремими моделями.

Призначення роботи – розробка методів і програмних засобів підвищення якості експертної багатокритеріальної оцінки житлової нерухомості.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Guolin Ke, , Qi Meng, Thomas Finle, Taifeng Wang. LightGBM: A Highly Efficient Gradient Boosting Decision Tree - 31st Conference on Neural Information . – Processing Systems Long Beach, CA, USA. 2017;
2. Leo Breiman. Classification and Regression Trees. Routledge – 1984. – p.86;
3. .Martin Kleppmann. Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems – Sebastopol, O`Reilly Media Inc. 2017. p.136-139;

4. Ivana Branic. Housing Quality Assessment Criteria - University of Osijek, Faculty of Civil Engineering Osijek, 2017. p.38;
5. Train and evaluate a model. [Electronic resource] – Access Mode: - https://learn.microsoft.com/en-us/dotnet/machine-learning/how-to-guides/trainmachine-learning-model-ml-net

5. Технічні вимоги

Методи регресійного прогнозування даних – методи градієнтного бустінгу на основі дерева рішень, градієнтного спуску. Джерело отримання даних – агрегатор оголошень нерухомості DIM.RIA API, Google Maps Plafortm. Платформа розробки - .NET, ASP.NET, ML.NET. Сховище подій – Kafka. Сховище даних - PostgreSQL. Мова програмування – C#.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної Роботи	Строк виконання етапів роботи	Примітка
1	Аналіз сучасного стану питання та обґрунтування задачі	20.09.23 – 02.10.23	Вик.
2	Розробка методів і алгоритмів багатокритеріальної оцінки житлової нерухомості	03.10.23 – 23.10.23	Вик.
3	Проектування архітектури програмного забезпечення системи	09.10.23 – 23.10.23	Вик.
4	Розробка програмного забезпечення системи	24.10.23 – 13.11.23	Вик.
5	Розгортання програмного забезпечення системи	14.11.23 – 21.11.23	Вик
6	Тестування програмного забезпечення системи	17.11.23 - 25.11.23	Вик
7	Економічна частина	22.11.23 – 01.12.23	Вик.

9. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б. Протокол перевірки роботи

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Розробка методів і програмних засобів підвищення якості експертної багатокритеріальної оцінки житлової нерухомості

Тип роботи: магістерська кваліфікаційна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ, 2ПІ – 22м

Науковий керівник: Катєльніков Д. І.

Unicheck	
Оригінальність	**97,6**
Схожість	2,4

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Серіков А.І

Керівник роботи _____ Катєльніков Д. І.

Додаток В. Лістинг програми

Сервіс завантаження даних

DistributedCache.cs

```
using StackExchange.Redis;

namespace DimRiaBackgroundFetcher.Cache;

public class DistributedCache : IDistributedCache
{
    private readonly IConnectionMultiplexer _redis;
    public DistributedCache(IConnectionMultiplexer redis)
    {
        _redis = redis;
    }
    public async Task<int?> GetAsync(string key)
    {
        var    value    =    await    _redis.GetDatabase().StringGetAsync(new
RedisKey(key));
        return value.HasValue ? int.Parse(value!) : null;
    }

    public async Task<bool> AddOrUpdateAsync(string key, int value, TimeSpan
absoluteExpiration)
    {
        var database = _redis.GetDatabase();
        var result = await database.StringSetAsync(
            new RedisKey(key),
            new RedisValue(value.ToString()),
            absoluteExpiration);
        return result;
    }
}
```

PropertyPromotion.cs

```
public record PropertyPromotion(Guid RequestId, int ExternalId, string Data);
```

SearchRequest.cs

```
namespace DimRiaBackgroundFetcher.Domain;

public class SearchRequest
{
    public Guid Id { get; }

    public int CountryCode { get; }
```

```
    public int CityId { get; }

    public int Category { get; } = 1;

    public int RealtyType { get; } = 2;

    public int OperationType { get; } = 1;

    public (DateTime From, DateTime To) PublishedRange { get; }

    public (decimal PriceFrom, decimal PriceTo) TotalPricesRange { get; }

    public (decimal PriceFrom, decimal PriceTo) SubPriceRange { get; private
set; }

    public bool Completed { get; private set; }

    private SearchRequest(
        Guid id,
        int countryCode,
        int cityId,
        (DateTime From, DateTime To) publishedRange,
        (decimal PriceFrom, decimal PriceTo) totalPricesRange,
        (decimal PriceFrom, decimal PriceTo) subPriceRange,
        bool completed)
    {
        Id = id;
        CountryCode = countryCode;
        CityId = cityId;
        PublishedRange = publishedRange;
        TotalPricesRange = totalPricesRange;
        SubPriceRange = subPriceRange;
        Completed = completed;
    }

    public static SearchRequest Create(
        int countryCode,
        int cityId,
        (DateTime From, DateTime To) publishedRange,
        (decimal PriceFrom, decimal PriceTo) pricesRange,
        (decimal PriceFrom, decimal PriceTo) subPricesRanges) =>
        new     SearchRequest(Guid.NewGuid(),     countryCode,     cityId,
publishedRange, pricesRange, subPricesRanges, false);

    public static SearchRequest From(
        Guid id,
        int countryCode,
        int cityId,
        (DateTime From, DateTime To) publishedRange,
        (decimal PriceFrom, decimal PriceTo) totalPricesRange,
        (decimal PriceFrom, decimal PriceTo) subPricesRange,
        bool  completed)  =>  new  SearchRequest(id,  countryCode,  cityId,
publishedRange, totalPricesRange, subPricesRange, completed);
```

```
    public void LimitRange()
    {
        var    priceToMedian    =    Math.Round((SubPriceRange.PriceTo    -
SubPriceRange.PriceFrom) / 2, 2);
        SubPriceRange = (
            SubPriceRange.PriceFrom,
            SubPriceRange.PriceFrom + priceToMedian);
    }

    public void MoveToNextRangeOrComplete()
    {
        if (SubPriceRange.PriceTo == TotalPricesRange.PriceTo)
        {
            Completed = true;
            return;
        }

        var priceTo = SubPriceRange.PriceTo * 2;
        SubPriceRange = (
            SubPriceRange.PriceTo,
            Math.Min(priceTo, TotalPricesRange.PriceTo));
    }

    public override string ToString()
    {
        return  $"Country:{CountryCode},  Date:{PublishedRange},  SubPrices:
{SubPriceRange}; TotalPricesRange: {TotalPricesRange}";
    }
}


public record SearchResult(

    [property: JsonPropertyName("items")] IReadOnlyList<int> Items,

    [property: JsonPropertyName("count")] int Count

);
```

PromotionPublisher.cs

```
using Confluent.Kafka;
using DimRiaBackgroundFetcher.Domain;
using DimRiaBackgroundFetcher.Settings;

namespace DimRiaBackgroundFetcher.Kafka;

public class PromotionPublisher : IPromotionPublisher
{
    private readonly KafkaSettings _kafkaSettings;
    public PromotionPublisher(
        KafkaSettings kafkaSettings)
    {
        _kafkaSettings = kafkaSettings;
```

```
    }

    public async Task PublishAsync(PropertyPromotion promotionData,
CancellationToken cancellationToken)
    {
        var config = new ProducerConfig
        {
            BootstrapServers = _kafkaSettings.BootstrapServers,
        };
        using var producer = new ProducerBuilder<Null,
string>(config).Build();
        var message = new Message<Null, string>
        {
            Value = promotionData.Data
        };
        var result = await producer.ProduceAsync(
            new
TopicPartition(_kafkaSettings.PromotionProducerSettings.Topic,
Partition.Any),
            message,
            cancellationToken);
    }
}
```

PromotionRepository.cs

```
using DimRiaBackgroundFetcher.Domain;
using DimRiaBackgroundFetcher.Persistance.Entities;
using Microsoft.EntityFrameworkCore;

namespace DimRiaBackgroundFetcher.Persistance.Repositories;

public class PromotionRepository : IPromotionRepository
{
    private readonly DimRiaBackgroundFetcherContext _context;

    public PromotionRepository(DimRiaBackgroundFetcherContext context)
    {
        _context = context;
    }

    public async Task UpsertRangeAsync(PropertyPromotion[]
propertyPromotions, CancellationToken cancellationToken)
    {
        var entities = propertyPromotions
            .Select(p => ToDal(p, DateTimeOffset.UtcNow,
DateTimeOffset.UtcNow))
            .ToArray();
        foreach (var entity in entities)
        {
            await _context.PropertyPromotions
                .Upsert(entity)
                .On(p => p.ExternalId)
```

```
                .WhenMatched((dbEntity,      updatedEntity)      =>      new
PropertyPromotionDal()
                {
                    Data = updatedEntity.Data,
                    SearchRequestId = updatedEntity.SearchRequestId,
                    UpdatedAt = DateTimeOffset.UtcNow,
                })
                .RunAsync(cancellationToken);
        }
    }

    private PropertyPromotionDal ToDal(
        PropertyPromotion promotion,
        DateTimeOffset createdAt,
        DateTimeOffset updatedAt) =>
        new PropertyPromotionDal()
        {
            ExternalId = promotion.ExternalId,
            CreatedAt = createdAt,
            UpdatedAt = updatedAt,
            SearchRequestId = promotion.RequestId,
            Data = promotion.Data,
        };
}
```

SearchRequestRepository.cs

```
using DimRiaBackgroundFetcher.Domain;
using DimRiaBackgroundFetcher.Persistance.Entities;
using Microsoft.EntityFrameworkCore;
using NpgsqlTypes;

namespace DimRiaBackgroundFetcher.Persistance.Repositories;

public class SearchRequestRepository : ISearchRequestRepository
{
    private readonly DimRiaBackgroundFetcherContext _context;

    public            SearchRequestRepository(DimRiaBackgroundFetcherContext
backgroundFetcherContext)
    {
        _context = backgroundFetcherContext;
    }

    public    async    Task<SearchRequest?>    GetLastAsync(int    cityId,
CancellationToken cancellationToken)
    {
        var lastRequest = await _context.SearchRequests
            .Where(r => r.CityId == cityId)
            .OrderByDescending(r => r.PublishedAtRange.UpperBound)
            .ThenByDescending(r => r.UpdatedAt)
            .FirstOrDefaultAsync(cancellationToken);
        return lastRequest is null ? null : ToDomain(lastRequest);
    }
```

```
    public    async    Task    UpsertAsync(SearchRequest    searchRequest,
CancellationToken cancellationToken)
    {
        var entity = ToDal(searchRequest);

        await _context.SearchRequests
            .Upsert(entity)
            .On(p => p.Id)
            .WhenMatched((dbEntity, updatedEntity) => new SearchRequestDal()
            {
                UpdatedAt = updatedEntity.UpdatedAt,
                SubPriceRange = updatedEntity.SubPriceRange,
                Completed = updatedEntity.Completed,
            })
            .RunAsync(cancellationToken);
    }

    private SearchRequest ToDomain(SearchRequestDal requestDal) =>
        SearchRequest.From(
            requestDal.Id,
            requestDal.CountryId,
            requestDal.CityId,
            (requestDal.PublishedAtRange.LowerBound,
requestDal.PublishedAtRange.UpperBound),
            (requestDal.TotalPriceRange.LowerBound,
requestDal.TotalPriceRange.UpperBound),
            (requestDal.SubPriceRange.LowerBound,
requestDal.SubPriceRange.UpperBound),
            requestDal.Completed);

    private SearchRequestDal ToDal(SearchRequest request) =>
        new SearchRequestDal()
        {
            Id = request.Id,
            PropertyType = request.RealtyType,
            OperationType = request.OperationType,
            CountryId = request.CountryCode,
            CityId = request.CityId,
            PublishedAtRange = ToDal(request.PublishedRange),
            UpdatedAt = DateTimeOffset.UtcNow,
            TotalPriceRange                    =                    new
NpgsqlRange<decimal>(request.TotalPricesRange.PriceFrom,
request.TotalPricesRange.PriceTo),
            SubPriceRange                    =                    new
NpgsqlRange<decimal>(request.SubPriceRange.PriceFrom,
request.SubPriceRange.PriceTo),
            Completed = request.Completed,
        };

    public static NpgsqlRange<DateTime> ToDal((DateTime From, DateTime To)
domain)
        => new NpgsqlRange<DateTime>(
            DateTime.SpecifyKind(domain.From, DateTimeKind.Utc),
```

```
            true,
            DateTime.SpecifyKind(domain.To, DateTimeKind.Utc),
            false);
}
```

DimRiaBackgroundFetcherContext.cs

```
using DimRiaBackgroundFetcher.Persistance.Entities;
using Microsoft.EntityFrameworkCore;

namespace DimRiaBackgroundFetcher.Persistance;

public class DimRiaBackgroundFetcherContext : DbContext
{
    public
DimRiaBackgroundFetcherContext(DbContextOptions<DimRiaBackgroundFetcherCont
ext> options)
        : base(options)
    {
    }

    public DbSet<SearchRequestDal> SearchRequests { get; set; }

    public DbSet<PropertyPromotionDal> PropertyPromotions { get; set; }

    protected    override    void    OnConfiguring(DbContextOptionsBuilder
optionsBuilder)
    {

    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<PropertyPromotionDal>()
            .Property(x => x.ExternalId)
            .ValueGeneratedNever();
        modelBuilder.Entity<SearchRequestDal>()
            .Property(x => x.Id)
            .ValueGeneratedNever();
    }
}
```

UnitOfWork.cs

```
using DimRiaBackgroundFetcher.Persistance.Repositories;
using Microsoft.EntityFrameworkCore.Storage;

namespace DimRiaBackgroundFetcher.Persistance;

public class UnitOfWork : IUnitOfWork
{
    private readonly DimRiaBackgroundFetcherContext _context;
    private readonly Lazy<IPromotionRepository> _promotionRepositoryLazy;
```

```
    private             readonly             Lazy<ISearchRequestRepository>
_searchRequestRepositoryLazy;

    public        IPromotionRepository        PromotionRepository        =>
_promotionRepositoryLazy.Value;
    public     ISearchRequestRepository     SearchRequestRepository     =>
_searchRequestRepositoryLazy.Value;

    public                          UnitOfWork(DimRiaBackgroundFetcherContext
dimRiaBackgroundFetcherContext)
    {
        _context = dimRiaBackgroundFetcherContext;
        _promotionRepositoryLazy = new Lazy<IPromotionRepository>(() => new
PromotionRepository(dimRiaBackgroundFetcherContext));
        _searchRequestRepositoryLazy = new Lazy<ISearchRequestRepository>(()
=> new SearchRequestRepository(dimRiaBackgroundFetcherContext));
    }

    public                                        Task<IDbContextTransaction>
StartDbTransactionAsync(CancellationToken cancellationToken)
    {
       return _context.Database.BeginTransactionAsync(cancellationToken);
    }

    public async Task SaveChangesAsync(CancellationToken cancellationToken)
    {
        await _context.SaveChangesAsync(cancellationToken);
    }
}
```

AgentLimitAnalyzer.cs

```
using DimRiaBackgroundFetcher.Cache;
using DimRiaBackgroundFetcher.Settings;
using Flurl.Http;

namespace DimRiaBackgroundFetcher;

public class AgentLimitAnalyzer : BackgroundService, IAgentLimitAnalyzer
{
    private readonly IDistributedCache _distributedCache;
    private readonly ApiLimitSettings _limitSettings;
    private readonly DimRiaSettings _dimRiaSettings;

    public event Action<Limit> LimitReached;

    public      AgentLimitAnalyzer(IDistributedCache      distributedCache,
ApiLimitSettings limitSettings, DimRiaSettings dimRiaSettings)
    {
        _distributedCache = distributedCache;
        _limitSettings = limitSettings;
        _dimRiaSettings = dimRiaSettings;
```

```
    }

    private async Task OnCall(FlurlCall call)
    {
        foreach (var limit in _limitSettings.Limits)
        {
            var              currentValue              =              await
_distributedCache.GetAsync(Key(limit.Duration)) ?? 0;
            var newValue = ++currentValue;
            await    _distributedCache.AddOrUpdateAsync(Key(limit.Duration),
newValue, limit.Duration);

            if (newValue >= limit.MaxRequestCount)
            {
                LimitReached?.Invoke(limit);
            }
        }
    }

    public async Task<Limit?> GetReachedLimitAsync()
    {
        foreach (var limit in _limitSettings.Limits)
        {
            var              currentValue              =              await
_distributedCache.GetAsync(Key(limit.Duration)) ?? 0;

            if (currentValue >= limit.MaxRequestCount)
                return limit;
        }

        return null;
    }

    public override void Dispose()
    {
        FlurlHttp.ConfigureClient(_dimRiaSettings.BaseApiUrl, client =>
        {
            client.Settings.AfterCallAsync -= OnCall;
        });
        base.Dispose();
    }

    protected override Task ExecuteAsync(CancellationToken stoppingToken)
    {
        FlurlHttp.ConfigureClient(_dimRiaSettings.BaseApiUrl, client =>
        {
            client.Settings.AfterCallAsync += OnCall;
        });

        return Task.CompletedTask;
    }

    private string Key(TimeSpan key) => $"API_LIMIT_{key}";
}
```

DimRiaAgent.cs

```
using DimRiaBackgroundFetcher.Domain;
using DimRiaBackgroundFetcher.Settings;
using Flurl;
using Flurl.Http;

namespace DimRiaBackgroundFetcher;

public class DimRiaAgent
{
    private readonly DimRiaSettings _dimRiaSettings;

    public DimRiaAgent(DimRiaSettings dimRiaSettings)
    {
        _dimRiaSettings = dimRiaSettings;
    }

    public Task<SearchResult> SearchAsync(SearchRequest searchRequest,
CancellationToken cancellationToken)
    {
        return _dimRiaSettings.BaseApiUrl
            .AppendPathSegment("search")
            .SetQueryParam("city_id", searchRequest.CityId)
            .SetQueryParam("category", searchRequest.Category)
            .SetQueryParam("realty_type", searchRequest.RealtyType)
            .SetQueryParam("operation_type", searchRequest.OperationType)
            .SetQueryParam("date_from",
searchRequest.PublishedRange.From.ToString("yyyy-MM-dd"))
            .SetQueryParam("date_to",
searchRequest.PublishedRange.To.ToString("yyyy-MM-dd"))
            .SetQueryParam("characteristic[234][from]",
searchRequest.SubPriceRange.PriceFrom)
            .SetQueryParam("characteristic[234][to]",
searchRequest.SubPriceRange.PriceTo)
            .SetQueryParam("api_key", _dimRiaSettings.AccessToken)
            .WithOAuthBearerToken(_dimRiaSettings.AccessToken)
            .GetJsonAsync<SearchResult>(cancellationToken);
    }

    public Task<string> GetPromotionAsync(int promotionId, CancellationToken
cancellationToken)
    {
        return _dimRiaSettings.BaseApiUrl
            .AppendPathSegment("info")
            .AppendPathSegment(promotionId.ToString())
            .SetQueryParam("api_key", _dimRiaSettings.AccessToken)
            .WithOAuthBearerToken(_dimRiaSettings.AccessToken)
            .GetStringAsync(cancellationToken);
    }
}
```

Worker.cs

```
using DimRiaBackgroundFetcher.Domain;
using DimRiaBackgroundFetcher.Kafka;
using DimRiaBackgroundFetcher.Persistance;
using DimRiaBackgroundFetcher.Settings;

namespace DimRiaBackgroundFetcher;

public class Worker : BackgroundService
{
    private readonly DimRiaAgent _agent;
    private readonly IPromotionPublisher _promotionPublisher;
    private readonly IServiceScopeFactory _serviceScopeFactory;
    private readonly WorkerSettings _workerSettings;
    private readonly ILogger<Worker> _logger;
    private readonly IAgentLimitAnalyzer _agentLimitAnalyzer;

    public Worker(
        DimRiaAgent agent,
        IPromotionPublisher promotionPublisher,
        IServiceScopeFactory serviceScopeFactory,
        WorkerSettings workerSettings,
        IAgentLimitAnalyzer agentLimitAnalyzer,
        ILogger<Worker> logger)
    {
        _agent = agent;
        _promotionPublisher = promotionPublisher;
        _serviceScopeFactory = serviceScopeFactory;
        _workerSettings = workerSettings;
        _logger = logger;
        _agentLimitAnalyzer = agentLimitAnalyzer;
    }

    protected override async Task ExecuteAsync(CancellationToken
stoppingToken)
    {
        var limitReachedTokenSource = new CancellationTokenSource();
        Action<Limit> cancelToken = _ => limitReachedTokenSource.Cancel();
        _agentLimitAnalyzer.LimitReached += cancelToken;

        var limitReachedCancellationToken = limitReachedTokenSource.Token;

        while (!stoppingToken.IsCancellationRequested)
        {
            try
            {
                var reachedLimit = await
_agentLimitAnalyzer.GetReachedLimitAsync();
                if (reachedLimit is not null)
                {
                    await Task.Delay(reachedLimit.Duration, stoppingToken);
                }
```

```
                using              var              linkedCts              =
CancellationTokenSource.CreateLinkedTokenSource(
                    stoppingToken,
                    limitReachedCancellationToken);
                await ProcessInternalAsync(linkedCts.Token);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error, while load promotions");
                _agentLimitAnalyzer.LimitReached -= cancelToken;
            }
        }
    }

    public      async      Task      ProcessInternalAsync(CancellationToken
cancellationToken)
    {
        var cities = _workerSettings.RegionSettings.Countries
            .SelectMany(c   =>   c.CityIds.Select(city   =>   (c.CountryCode,
city)))
            .ToArray();

        foreach (var (CountryCode, CityId) in cities)
        {
            cancellationToken.ThrowIfCancellationRequested();
            var    lastRequest    =    await    GetLastRequestAsync(CityId,
cancellationToken);
            SearchRequest newRequest;

            if (lastRequest is null || lastRequest.Completed)
            {
                if (DateTimeOffset.UtcNow < lastRequest?.PublishedRange.To)
                    return;

                var completed = lastRequest?.Completed ?? false;
                var startSearchFrom = completed
                    ? lastRequest!.PublishedRange.To
                    : _workerSettings.StartSearchFrom;
                newRequest = SearchRequest.Create(
                    CountryCode,
                    CityId,
                    (startSearchFrom,
startSearchFrom.AddDays(_workerSettings.SearchDateIntervalInDays)),
                    (_workerSettings.MinPriceSearch,
_workerSettings.MaxPriceSearch),
                    (_workerSettings.MinPriceSearch,
_workerSettings.MaxPriceSearch));
            }
            else
            {
                newRequest = lastRequest;
            }
```

```
            await ProcessAsync(newRequest, cancellationToken);
        }
    }

    public async Task ProcessAsync(SearchRequest request, CancellationToken
cancellationToken)
    {
        using var scope = _serviceScopeFactory.CreateScope();
        var                              unitOfWork                              =
scope.ServiceProvider.GetRequiredService<IUnitOfWork>();
        var     searchResult     =     await     _agent.SearchAsync(request,
cancellationToken);

        if (searchResult.Count > searchResult.Items.Count)
        {
            _logger.LogInformation(
                "Search   segment   too   big.   Request   -   {SearchRequest};
Promotions   count:   {PromotionsCount};   Total   promotions   in   segment:
{TotalPromotionsCount}",
                request,
                searchResult.Items.Count,
                searchResult.Count);
            request.LimitRange();
            await   unitOfWork.SearchRequestRepository.UpsertAsync(request,
cancellationToken);
            return;
        }

        if (searchResult.Count == 0)
        {
            request.MoveToNextRangeOrComplete();
            await   unitOfWork.SearchRequestRepository.UpsertAsync(request,
cancellationToken);
            return;
        }

        var promotions = new List<PropertyPromotion>();
        foreach (var promotionId in searchResult.Items)
        {
            var  promotion  =  await  _agent.GetPromotionAsync(promotionId,
cancellationToken);
            promotions.Add(new  PropertyPromotion(request.Id,  promotionId,
promotion));
            await             Task.Delay(_workerSettings.GetPromotionDelay,
cancellationToken);
        }

        request.MoveToNextRangeOrComplete();

        await       using       var       transaction       =       await
unitOfWork.StartDbTransactionAsync(cancellationToken);
        await       unitOfWork.SearchRequestRepository.UpsertAsync(request,
cancellationToken);
```

```
        await
unitOfWork.PromotionRepository.UpsertRangeAsync(promotions.ToArray(),
cancellationToken);
        await unitOfWork.SaveChangesAsync(CancellationToken.None);
        await transaction.CommitAsync(CancellationToken.None);

        foreach (var promotion in promotions)
        {
            await                _promotionPublisher.PublishAsync(promotion,
cancellationToken);
        }
    }

    private  async  Task<SearchRequest?>  GetLastRequestAsync(int  cityId,
CancellationToken cancellationToken)
    {
        using var scope = _serviceScopeFactory.CreateScope();
        var                lastRequest                =                await
scope.ServiceProvider.GetRequiredService<IUnitOfWork>()
            .SearchRequestRepository.GetLastAsync(cityId,
cancellationToken);
        return lastRequest;
    }
}
```

Program.cs

```
using DimRiaBackgroundFetcher;
using DimRiaBackgroundFetcher.Kafka;
using DimRiaBackgroundFetcher.Persistance;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

IConfiguration configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("appSettings.json", optional: false)
    .Build();
var serviceCollection = builder.Services;
serviceCollection.AddRedis(configuration);
serviceCollection.AddDatabaseContext(configuration);
serviceCollection.AddConfiguration(configuration);
serviceCollection.AddSingleton<IAgentLimitAnalyzer, AgentLimitAnalyzer>();
serviceCollection.AddTransient<DimRiaAgent>();
serviceCollection.AddHostedService<Worker>();
serviceCollection.AddHostedService<AgentLimitAnalyzer>();
serviceCollection.AddControllers();
serviceCollection.AddEndpointsApiExplorer();
serviceCollection.AddTransient<IPromotionPublisher, PromotionPublisher>();
serviceCollection.AddScoped<IUnitOfWork, UnitOfWork>();
builder.Configuration.AddConfiguration(configuration);

var app = builder.Build();
app.MapControllers();
```

```
RunMigrations(app);
app.Run();

static void RunMigrations(IHost host)
{
    using var scope = host.Services.CreateScope();
    var scopedServiceProvider = scope.ServiceProvider;
    var                                logger                                =
scopedServiceProvider.GetRequiredService<ILogger<Program>>();

    try
    {
        var                                context                                =
scopedServiceProvider.GetRequiredService<DimRiaBackgroundFetcherContext>();
        context.Database.Migrate();
        logger.LogInformation("Database  migrations  has  been  successfully
run");
    }
    catch (Exception e)
    {
        logger.LogError(e, "Migrations failed");
        throw;
    }
}
```

Сервіс оцінку локації

PromotionConsumer.cs

```
using System.Text.Json;
using Confluent.Kafka;
using LocationAnalyzer.Domain;
using LocationAnalyzer.Persistance.Repository;
using LocationAnalyzer.Services;
using LocationAnalyzer.Settings;
using LocationAnalyzer.Trainers;

namespace LocationAnalyzer.ConsumerHandlers;

public class PromotionConsumer : BackgroundService
{
    private readonly KafkaSettings _kafkaSettings;
    private readonly ILogger<PromotionConsumer> _logger;
    private readonly IServiceScopeFactory _serviceScopeFactory;

    public PromotionConsumer(
        KafkaSettings kafkaSettings,
        ILogger<PromotionConsumer> logger,
        IServiceScopeFactory serviceScopeFactory)
    {
        _kafkaSettings = kafkaSettings;
        _logger = logger;
        _serviceScopeFactory = serviceScopeFactory;
    }
```

```
    protected override async Task ExecuteAsync(CancellationToken
stoppingToken)
    {
        var config = new ConsumerConfig
        {
            BootstrapServers = _kafkaSettings.BootstrapServers,
            GroupId = _kafkaSettings.Group,
            AutoOffsetReset = AutoOffsetReset.Earliest
        };
        using var consumer = new ConsumerBuilder<Ignore,
string>(config).Build();
        consumer.Subscribe(_kafkaSettings.PromotionConsumerSettings.Topic);
        while (!stoppingToken.IsCancellationRequested)
        {
            try
            {
                var consumeResult = consumer.Consume(stoppingToken);
                var promotionData =
JsonSerializer.Deserialize<PropertyPromotionData>(consumeResult.Message.Val
ue)!;
                await Task.WhenAll(
                    TrainAsync(promotionData));
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, ex.Message);
            }
        }
    }

    private async Task TrainAsync(PropertyPromotionData promotionData)
    {
        using var scope = _serviceScopeFactory.CreateScope();
        var location = await
scope.ServiceProvider.GetRequiredService<IPromotionLocationService>()
            .EnrichAndSaveAsync(ToDomain(promotionData),
CancellationToken.None);
        var locations = await
scope.ServiceProvider.GetRequiredService<IPromotionLocationRepository>().Ge
tAllAsync();
        await
scope.ServiceProvider.GetRequiredService<ITrainer>().Train(locations.Concat
(new[] { location }).ToArray());
    }

    private PromotionLocation ToDomain(PropertyPromotionData
propertyLocation) =>
        new PromotionLocation(
            propertyLocation.RealtyId,
            380,
            propertyLocation.StateId,
            propertyLocation.CityId,
            propertyLocation.DistrictId,
            propertyLocation.UserNewbuildName,
```

```
            propertyLocation.Longitude,
            propertyLocation.Latitude,
            0,
            0,
            0,
            0,
            0,
            propertyLocation.CurrencyType switch
            {
                "грн" => propertyLocation.PriceItem / 36.6f,
                _ => propertyLocation.PriceItem
            });
}
```

LocationEvaluationController.cs

```
using LocationAnalyzer.Dto;
using LocationAnalyzer.Evaluators;
using LocationAnalyzer.Services;
using LocationAnalyzer.Trainers;
using Microsoft.AspNetCore.Mvc;

namespace LocationAnalyzer.Controllers;

[Route("/api/v1/locations")]
public class LocationEvaluationController : ControllerBase
{
    private readonly IEvaluator _evaluator;
    private readonly IPromotionLocationService _locationService;

    public        LocationEvaluationController(IEvaluator        evaluator,
IPromotionLocationService locationService)
    {
        _evaluator = evaluator;
        _locationService = locationService;
    }

    [HttpPost("estimate")]
    public                    async                    Task<IActionResult>
EstimateLocation([FromBody]LocationEstimateInput input)
    {
        var locationTrainerData = new Trainer.LocationTrainerData()
        {
            CountryId = input.CountryId,
            StateId = input.StateId,
            CityId = input.CityId,
            DistrictId = input.DistrictId,
            SocialInfrastructureIndex = input.SocialInfrastractureIndex,
            CommercialInfrastuctureIndex                                    =
input.CommercialInfrastructureIndex,
            TransportInfrastructureIndex                                    =
input.TransportInfrastructureIndex,
            CultureAndGreenZoneIndex = input.CultureAndGreenZoneIndex,
```

```
            AirQuality = input.AirQualityIndex,
        };
        return    new    OkObjectResult(new    {    estimate    =    await
_evaluator.PredictAsync(locationTrainerData) });
    }

    [HttpPost("/api/v2/locations/estimate")]
    public                    async                    Task<IActionResult>
EstimateLocation([FromBody]LocationEstimateInputV2 input, CancellationToken
cancellationToken)
    {
        var                 indexes                 =                 await
_locationService.GetLocationIndexesAsync(input.Longitude,   input.Latitude,
cancellationToken);
        var locationTrainerData = new Trainer.LocationTrainerData()
        {
            CountryId = input.CountryId,
            StateId = input.StateId,
            CityId = input.CityId,
            DistrictId = input.DistrictId,
            SocialInfrastructureIndex = indexes.SocialInfrastractureIndex,
            CommercialInfrastuctureIndex                                    =
indexes.CommercialInfrastructureIndex,
            TransportInfrastructureIndex                                    =
indexes.TransportInfrastructureIndex,
            CultureAndGreenZoneIndex = indexes.CultureAndGreenZoneIndex,
            AirQuality = indexes.AirQualityIndex,
        };
        return    new    OkObjectResult(new    {    estimate    =    await
_evaluator.PredictAsync(locationTrainerData) });
    }

    [HttpPost("evaluation")]
    public    async    Task<IActionResult>    EvaludateLocation([FromBody]
LocationEvaluationInput[] input)
    {
        var data = input.Select(i => new Trainer.LocationTrainerData()
            {
                CountryId = i.CountryId,
                StateId = i.StateId,
                CityId = i.CityId,
                DistrictId = i.DistrictId,
                SocialInfrastructureIndex = i.SocialInfrastractureIndex,
                CommercialInfrastuctureIndex                                =
i.CommercialInfrastructureIndex,
                TransportInfrastructureIndex                                =
i.TransportInfrastructureIndex,
                CultureAndGreenZoneIndex = i.CultureAndGreenZoneIndex,
                AirQuality = i.AirQualityIndex,
                Price = i.Price,
            })
            .ToArray();
        var evaluationResult = await _evaluator.EvaluateAsync(data);
```

```
        return      new      OkObjectResult(new      {      r_squared      =
evaluationResult.RSquared,             root_mean_squared_error             =
evaluationResult.RootMeanSquaredError });
    }
}
```

AirQuality.cs

```
using System.Text.Json.Serialization;
using Newtonsoft.Json;

namespace LocationAnalyzer.Domain;

    public class Color
    {
        [JsonProperty("red", NullValueHandling = NullValueHandling.Ignore)]
        [JsonPropertyName("red")]
        public double Red { get; set; }

        [JsonProperty("green",              NullValueHandling             =
NullValueHandling.Ignore)]
        [JsonPropertyName("green")]
        public double Green { get; set; }
    }

    public class Index
    {
        [JsonProperty("code", NullValueHandling = NullValueHandling.Ignore)]
        [JsonPropertyName("code")]
        public string Code { get; set; }

        [JsonProperty("displayName",           NullValueHandling           =
NullValueHandling.Ignore)]
        [JsonPropertyName("displayName")]
        public string DisplayName { get; set; }

        [JsonProperty("aqi", NullValueHandling = NullValueHandling.Ignore)]
        [JsonPropertyName("aqi")]
        public int Aqi { get; set; }

        [JsonProperty("aqiDisplay",           NullValueHandling           =
NullValueHandling.Ignore)]
        [JsonPropertyName("aqiDisplay")]
        public string AqiDisplay { get; set; }

        [JsonProperty("color",              NullValueHandling             =
NullValueHandling.Ignore)]
        [JsonPropertyName("color")]
        public Color Color { get; set; }

        [JsonProperty("category",            NullValueHandling            =
NullValueHandling.Ignore)]
        [JsonPropertyName("category")]
        public string Category { get; set; }
```

```
        [JsonProperty("dominantPollutant",        NullValueHandling        =
NullValueHandling.Ignore)]
        [JsonPropertyName("dominantPollutant")]
        public string DominantPollutant { get; set; }
    }

    public class AirQuality
    {
        [JsonProperty("dateTime",            NullValueHandling            =
NullValueHandling.Ignore)]
        [JsonPropertyName("dateTime")]
        public DateTime DateTime { get; set; }

        [JsonProperty("regionCode",           NullValueHandling           =
NullValueHandling.Ignore)]
        [JsonPropertyName("regionCode")]
        public string RegionCode { get; set; }

        [JsonProperty("indexes",             NullValueHandling            =
NullValueHandling.Ignore)]
        [JsonPropertyName("indexes")]
        public List<Index> Indexes { get; set; }
    }
```

IndexesWrapper.cs

```
namespace LocationAnalyzer.Domain;

public record IndexesWrapper(
    float SocialInfrastractureIndex,
    float CommercialInfrastructureIndex,
    float TransportInfrastructureIndex,
    float CultureAndGreenZoneIndex,
    float AirQualityIndex);
```

Place.cs

```
using System.Text.Json.Serialization;
using Newtonsoft.Json;

namespace LocationAnalyzer.Domain;

public class DisplayName
{
    [JsonProperty("text", NullValueHandling = NullValueHandling.Ignore)]
    [JsonPropertyName("text")]
    public string Text { get; set; }
```

```
    [JsonProperty("languageCode",            NullValueHandling            =
NullValueHandling.Ignore)]
    [JsonPropertyName("languageCode")]
    public string LanguageCode { get; set; }
}

public class Location
{
    [JsonProperty("latitude", NullValueHandling = NullValueHandling.Ignore)]
    [JsonPropertyName("latitude")]
    public double Latitude { get; set; }

    [JsonProperty("longitude",              NullValueHandling            =
NullValueHandling.Ignore)]
    [JsonPropertyName("longitude")]
    public double Longitude { get; set; }
}

public class Place
{
    [JsonProperty("id", NullValueHandling = NullValueHandling.Ignore)]
    [JsonPropertyName("id")]
    public string Id { get; set; }

    [JsonProperty("types", NullValueHandling = NullValueHandling.Ignore)]
    [JsonPropertyName("types")]
    public List<string> Types { get; set; }

    [JsonProperty("location", NullValueHandling = NullValueHandling.Ignore)]
    [JsonPropertyName("location")]
    public Location Location { get; set; }

    [JsonProperty("rating", NullValueHandling = NullValueHandling.Ignore)]
    [JsonPropertyName("rating")]
    public double Rating { get; set; }

    [JsonProperty("userRatingCount",           NullValueHandling          =
NullValueHandling.Ignore)]
    [JsonPropertyName("userRatingCount")]
    public int UserRatingCount { get; set; }

    [JsonProperty("displayName",             NullValueHandling            =
NullValueHandling.Ignore)]
    [JsonPropertyName("displayName")]
    public DisplayName? DisplayName { get; set; }

    [JsonProperty("formattedAddress",          NullValueHandling          =
NullValueHandling.Ignore)]
    [JsonPropertyName("formattedAddress")]
    public string? FormattedAddress { get; set; }
}

public class PlaceCollection
{
```

```
    [JsonProperty("places", NullValueHandling = NullValueHandling.Ignore)]
    [JsonPropertyName("places")]
    public Place[] Places { get; set; }
}
```

PromotionLocation.cs

```
public record PromotionLocation(
    int ExtraId,
    int CountryId,
    int StateId,
    int CityId,
    int DistrictId,
    string HousingProject,
    double Longtitude,
    double Latitude,
    float SocialInfrastractureIndex,
    float CommercialInfrastructureIndex,
    float TransportInfrastructureIndex,
    float CultureAndGreenZoneIndex,
    float AirQualityIndex,
    float Price);
```

Evaluator.cs

```
using LocationAnalyzer.Persistance.Repository;
using LocationAnalyzer.Trainers;
using Microsoft.ML;
using Microsoft.ML.Data;

namespace LocationAnalyzer.Evaluators;

public class Evaluator : IEvaluator
{
    private readonly IMLModelRepository _mlModelRepository;

    public Evaluator(IMLModelRepository mlModelRepository)
    {
        _mlModelRepository = mlModelRepository;
    }

    public   async   Task<float>   PredictAsync(Trainer.LocationTrainerData
trainerData)
    {
        var mlModel = await _mlModelRepository.GetLastVersionAsync();
        using var memoryStream = new MemoryStream(mlModel);

        MLContext mlContext = new MLContext(0);
        var transformer = mlContext.Model.Load(memoryStream, out var schema);
```

```
        var predictionFunction =
mlContext.Model.CreatePredictionEngine<Trainer.LocationTrainerData,
LocationPrediction>(transformer);
        return predictionFunction.Predict(trainerData).Price;
    }

    public async Task<(double RSquared, double RootMeanSquaredError)>
EvaluateAsync(Trainer.LocationTrainerData[] evaluationData)
    {
        var mlModel = await _mlModelRepository.GetLastVersionAsync();
        using var memoryStream = new MemoryStream(mlModel);

        MLContext mlContext = new MLContext(0);
        var transformer = mlContext.Model.Load(memoryStream, out var schema);
        IDataView dataView =
mlContext.Data.LoadFromEnumerable(evaluationData);
        var predictions = transformer.Transform(dataView);
        var metrics = mlContext.Regression.Evaluate(predictions, "Label",
"Score");
        return (metrics.RSquared, metrics.RootMeanSquaredError);
    }
}

public class LocationPrediction
{
    [ColumnName("Score")]
    public float Price;
}
```

GoogleMapCacheGateway.cs

```
using LocationAnalyzer.Cache;
using LocationAnalyzer.Domain;

namespace LocationAnalyzer.Gateway;

public class GoogleMapCacheGateway : IGoogleMapGateway
{
    private const int MaxPlaceCount = 20;
    private const double AirQualityRaidus = 2000;
    private readonly IGoogleMapGateway _googleMapGateway;
    private readonly IDistributedCache _distributedCache;

    public GoogleMapCacheGateway(
        IGoogleMapGateway googleMapGateway,
        IDistributedCache distributedCache)
    {
        _googleMapGateway = googleMapGateway;
        _distributedCache = distributedCache;
    }

    public async Task<PlaceCollection> SearchPlacesAsync(double radius,
double longitude, double latitude, CancellationToken cancellationToken)
    {
```

```
        const string searchPlacesKey = "SearchPlaces";
        var                  places                  =                  await
_distributedCache.SearchCoordinateDataAsync<Place>(longitude,     latitude,
radius, searchPlacesKey);

        if (places.Length < MaxPlaceCount)
        {
            var               newPlaces               =               (await
_googleMapGateway.SearchPlacesAsync(radius,       longitude,       latitude,
cancellationToken))?.Places ?? Array.Empty<Place>();
            foreach (var newPlace in newPlaces)
            {
                await    _distributedCache.AddCoordinateDataAsync(longitude,
latitude, searchPlacesKey, newPlace);
            }
            places = newPlaces
                .UnionBy(places, p => p.Id)
                .ToArray();
        }

        return new PlaceCollection() { Places = places.ToArray() };
    }

    public  async  Task<AirQuality>  GetAirQualityAsync(double  longtitude,
double latitude, CancellationToken cancellationToken)
    {
        var key = "AirQuality";
        var              storedAirQuality              =              await
_distributedCache.SearchCoordinateDataAsync<AirQuality>(longtitude,
latitude, AirQualityRaidus, key);

        if (storedAirQuality.Any())
        {
            return storedAirQuality.First();
        }

        var                airQuality                =                await
_googleMapGateway.GetAirQualityAsync(longtitude,                  latitude,
cancellationToken);
        await _distributedCache.AddCoordinateDataAsync(longtitude, latitude,
key, airQuality);
        return airQuality;
    }
}
```

GoogleMapGateway.cs

```
using Flurl;
using Flurl.Http;
using LocationAnalyzer.Domain;
using LocationAnalyzer.Settings;

namespace LocationAnalyzer.Gateway;
```

```
public class GoogleMapGateway : IGoogleMapGateway
{
    private readonly GoogleMapsSettings _googleMapsSettings;

    public GoogleMapGateway(GoogleMapsSettings googleMapsSettings)
    {
        _googleMapsSettings = googleMapsSettings;
    }

    public async Task<PlaceCollection> SearchPlacesAsync(double radius,
double longitude, double latitude, CancellationToken cancellationToken)
    {
        var result = await _googleMapsSettings.PlacesBaseApiUrl
            .AppendPathSegment("places:searchNearby")
            .WithHeader("Content-Type", "application/json")
            .WithHeader("X-Goog-FieldMask",
"places.types,places.location,places.rating,places.userRatingCount,places.d
isplayName,places.id,places.formattedAddress")
            .WithHeader("X-Goog-Api-Key", _googleMapsSettings.AccessToken)
            .PostJsonAsync(
                new
                {
                    maxResultCount = 20,
                    locationRestriction = new
                    {
                        circle = new
                        {
                            center = new
                            {
                                latitude = latitude,
                                longitude = longitude,
                            },
                            radius = radius,
                        },
                    },
                    rankPreference = "POPULARITY",
                },
                cancellationToken)
            .ReceiveJson<PlaceCollection>();
        return    result    ??    new    PlaceCollection()    {Places    =
Array.Empty<Place>()};
    }

    public Task<AirQuality> GetAirQualityAsync(double longitude, double
latitude, CancellationToken cancellationToken)
    {
        return _googleMapsSettings.AirqualityBaseApiUrl
            .AppendPathSegment("currentConditions:lookup")
            .WithHeader("Content-Type", "application/json")
            .SetQueryParam("key", _googleMapsSettings.AccessToken)
            .PostJsonAsync(
                new
                {
                    location = new
```

```
                    {
                        latitude = latitude,
                        longitude = longitude,
                    }
                },
                cancellationToken)
            .ReceiveJson<AirQuality>();
    }
}
```

LocationAnalyzerContext.cs

```
using LocationAnalyzer.Persistance.Entities;
using Microsoft.EntityFrameworkCore;

namespace LocationAnalyzer.Persistance;

public class LocationAnalyzerContext : DbContext
{
    public
LocationAnalyzerContext(DbContextOptions<LocationAnalyzerContext> options)
        : base(options)
    {
    }

    public DbSet<PromotionLocationDal> Locations { get; set; }

    public DbSet<TrainModelDal> TrainModels { get; set; }

    public DbSet<PlaceDal> Places { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<PromotionLocationDal>()
            .Property(x => x.ExtraId)
            .ValueGeneratedNever();
        modelBuilder.Entity<TrainModelDal>()
            .Property(x => x.Id)
            .ValueGeneratedNever();
    }
}
```

PromotionLocationService.cs

```
using LocationAnalyzer.Domain;
using LocationAnalyzer.Gateway;
using LocationAnalyzer.Persistance.Repository;

namespace LocationAnalyzer.Services;

public class PromotionLocationService : IPromotionLocationService
{
    private const double Radius = 1000;

    private readonly HashSet<string> _socialInfrastructure = new(
```

```
        new[]     {"hospital",     "doctor",     "university",     "school",
"athletic_field"});

    private readonly HashSet<string> _commercialInfrastucture = new(
        new[]
        {
            "reustaraunt", "store", "shopping mal", "hotel", "gym", "café",
"bar", "gas_station", "car_repair",
            "courier_service", "market"
        });

    private readonly HashSet<string> _transportInfrastructure = new(
        new[]
        {
            "transit_station", "train_station", "bus_stop", "subway_station"
        });

    private readonly HashSet<string> _cultureZones = new(
        new[]
        {
            "event_venue",    "park",    "tourist_attraction",    "aquarium",
"movie_theater"
        });

    private readonly IGoogleMapGateway _googleMapGateway;
    private               readonly               IPromotionLocationRepository
_promotionLocationRepository;
    private readonly IPlaceRepository _placeRepository;
    private readonly ILogger<PromotionLocationService> _logger;

    public PromotionLocationService(
        IGoogleMapGateway googleMapGateway,
        IPromotionLocationRepository promotionLocationRepository,
        IPlaceRepository placeRepository,
        ILogger<PromotionLocationService> logger)
    {
        _googleMapGateway = googleMapGateway;
        _promotionLocationRepository = promotionLocationRepository;
        _placeRepository = placeRepository;
        _logger = logger;
    }

    public                    async                    Task<PromotionLocation>
EnrichAndSaveAsync(PromotionLocation  promotionLocation,  CancellationToken
cancellationToken)
    {
        try
        {
            var             existingLocation             =             await
_promotionLocationRepository.FindAsync(promotionLocation.ExtraId,
cancellationToken);
            if (existingLocation is not null)
            {
                if (existingLocation.Price is 0)
```

```
                {
                    existingLocation = existingLocation with
                    {
                        Price = promotionLocation.Price,
                    };
                    await
_promotionLocationRepository.UpsertRangeAsync(new[]     {existingLocation},
cancellationToken);
                }
                return existingLocation;
            }

            promotionLocation   =   await   EnrishAsync(promotionLocation,
cancellationToken);
            await       _promotionLocationRepository.UpsertRangeAsync(new[]
{promotionLocation}, cancellationToken);
            return promotionLocation;
        }
        catch (Exception exception)
        {
            _logger.LogError(exception, "Error occurred while try to enrich
location='{Id}'", promotionLocation.ExtraId);
            return promotionLocation;
        }
    }

    public   async   Task<IndexesWrapper>   GetLocationIndexesAsync(double
longitude, double latitude, CancellationToken cancellationToken)
    {
        if (longitude == 0 || latitude == 0)
            return new IndexesWrapper(0, 0, 0, 0, 0);

        var  places  =  await  _googleMapGateway.SearchPlacesAsync(Radius,
longitude, latitude, cancellationToken);
        var                airQuality                =                await
_googleMapGateway.GetAirQualityAsync(longitude,                   latitude,
cancellationToken);
        return new IndexesWrapper(
            CalculateIndex(places, _socialInfrastructure),
            CalculateIndex(places, _commercialInfrastucture),
            CalculateIndex(places, _transportInfrastructure),
            CalculateIndex(places, _cultureZones),
            airQuality?.Indexes.FirstOrDefault(p => p.Code == "uaqi")?.Aqi
?? 0f);
    }

    private  async  Task<PromotionLocation>  EnrishAsync(PromotionLocation
promotionLocation, CancellationToken cancellationToken)
    {
        if (promotionLocation.Longtitude == 0 || promotionLocation.Latitude
== 0)
            return promotionLocation;
```

```
        var places = await _googleMapGateway.SearchPlacesAsync(Radius,
promotionLocation.Longtitude, promotionLocation.Latitude,
cancellationToken);
        await _placeRepository.UpsertAsync(places.Places,
cancellationToken);
        var airQuality = await
_googleMapGateway.GetAirQualityAsync(promotionLocation.Longtitude,
promotionLocation.Latitude, cancellationToken);
        return promotionLocation with
        {
            SocialInfrastractureIndex = CalculateIndex(places,
_socialInfrastructure),
            CommercialInfrastructureIndex = CalculateIndex(places,
_commercialInfrastucture),
            TransportInfrastructureIndex = CalculateIndex(places,
_transportInfrastructure),
            CultureAndGreenZoneIndex = CalculateIndex(places,
_cultureZones),
            AirQualityIndex = airQuality?.Indexes.FirstOrDefault(p => p.Code
== "uaqi")?.Aqi ?? 0f,
        };
    }

    private float CalculateIndex(PlaceCollection placeCollection,
HashSet<string> categories)
    {
        const int peopleFactor = 1000;
        var places = placeCollection.Places
            .Where(p => p.Types.Any(categories.Contains))
            .ToArray();
        return (float)places.Sum(e => e.Rating * e.UserRatingCount) /
peopleFactor;
    }
}
```

Trainer.cs

```
using System.Diagnostics;
using LocationAnalyzer.Domain;
using LocationAnalyzer.Persistance.Repository;
using Microsoft.ML;
using Microsoft.ML.Data;

namespace LocationAnalyzer.Trainers;

public class Trainer : ITrainer
{
    private readonly IMLModelRepository _mlModelRepository;

    public Trainer(IMLModelRepository mlModelRepository)
    {
        _mlModelRepository = mlModelRepository;
    }
```

```
    public async Task Train(PromotionLocation[] promotionLocations)
    {
        // <Snippet3>
        MLContext mlContext = new MLContext(seed: 0);
        // </Snippet3>

        // <Snippet5>
        Stopwatch stopwatch = Stopwatch.StartNew();
        IDataView                          dataView                          =
mlContext.Data.LoadFromEnumerable(promotionLocations.Select(ToLocationTrain
erData));
        var model = Train(mlContext, dataView);
        Console.WriteLine($"Train time - {stopwatch.ElapsedMilliseconds}");
        // </Snippet5>

        using var memoryStream = new MemoryStream();
        mlContext.Model.Save(model, dataView.Schema, memoryStream);
        await _mlModelRepository.InsertAsync(memoryStream.ToArray());
    }

    public static ITransformer Train(MLContext mlContext, IDataView dataView)
    {
        // <Snippet6>
        // </Snippet6>

        // <Snippet7>
        var  pipeline  =  mlContext.Transforms.CopyColumns(outputColumnName:
"Label", inputColumnName: "Price")

.Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName:
"CountryIdEncoded", inputColumnName: "CountryId"))

.Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName:
"StateIdEncoded", inputColumnName: "StateId"))

.Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName:
"CityIdEncoded", inputColumnName: "CityId"))

.Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName:
"DistrictIdEncoded", inputColumnName: "DistrictId"))
            .Append(mlContext.Transforms.Concatenate("Features",
"CountryIdEncoded", "StateIdEncoded", "CityIdEncoded", "DistrictIdEncoded",
"SocialInfrastructureIndex",                "CommercialInfrastuctureIndex",
"TransportInfrastructureIndex", "CultureAndGreenZoneIndex", "AirQuality"))
            .Append(mlContext.Regression.Trainers.LightGbm());
        // </Snippet10>

        Console.WriteLine("===============  Create  and  Train  the  Model
===============");

        // <Snippet11>
        var model = pipeline.Fit(dataView);
        // </Snippet11>
```

```
        Console.WriteLine("===============       End       of       training
===============");
        Console.WriteLine();
        // <Snippet12>
        return model;
        // </Snippet12>
    }
    private                    static                    LocationTrainerData
ToLocationTrainerData(PromotionLocation location)
    {
        return new LocationTrainerData()
        {
            CountryId = location.CountryId,
            StateId = location.StateId,
            CityId = location.CityId,
            DistrictId = location.DistrictId,
            SocialInfrastructureIndex = location.SocialInfrastractureIndex,
            CommercialInfrastuctureIndex                                    =
location.CommercialInfrastructureIndex,
            TransportInfrastructureIndex                                    =
location.TransportInfrastructureIndex,
            CultureAndGreenZoneIndex = location.CultureAndGreenZoneIndex,
            AirQuality = location.AirQualityIndex,
            Price = location.Price,
        };
    }

    public class LocationTrainerData
    {
        [LoadColumn(0)]
        public float CountryId;

        [LoadColumn(1)]
        public float StateId;

        [LoadColumn(2)]
        public float CityId;

        [LoadColumn(3)]
        public float DistrictId;

        [LoadColumn(4)]
        public float SocialInfrastructureIndex;

        [LoadColumn(5)]
        public float CommercialInfrastuctureIndex;

        [LoadColumn(6)]
        public float TransportInfrastructureIndex;

        [LoadColumn(7)]
        public float CultureAndGreenZoneIndex;

        [LoadColumn(8)]
```

```
        public float AirQuality;

        [LoadColumn(9)]
        public float Price;
    }
}
```

Program.cs

```
using LocationAnalyzer;
using LocationAnalyzer.Cache;
using LocationAnalyzer.ConsumerHandlers;
using LocationAnalyzer.Evaluators;
using LocationAnalyzer.Gateway;
using LocationAnalyzer.Persistance;
using LocationAnalyzer.Persistance.Repository;
using LocationAnalyzer.Services;
using LocationAnalyzer.Settings;
using LocationAnalyzer.Trainers;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

IConfiguration configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("appSettings.json", optional: false)
    .Build();
var serviceCollection = builder.Services;
serviceCollection.AddRedis(configuration);
serviceCollection.AddDatabaseContext(configuration);
serviceCollection.AddConfiguration(configuration);
serviceCollection.AddTransient<IDistributedCache, DistributedCache>();
serviceCollection.AddTransient<IGoogleMapGateway>(sp         =>         new
GoogleMapCacheGateway(
    new GoogleMapGateway(sp.GetRequiredService<GoogleMapsSettings>()),
    sp.GetRequiredService<IDistributedCache>()));

serviceCollection.AddTransient<IPromotionLocationRepository,
PromotionLocationRepository>();
serviceCollection.AddTransient<IMLModelRepository, MLModelRepository>();
serviceCollection.AddTransient<IPlaceRepository, PlaceRepository>();
serviceCollection.AddTransient<IPromotionLocationService,
PromotionLocationService>();
serviceCollection.AddTransient<ITrainer, Trainer>();
serviceCollection.AddTransient<IEvaluator, Evaluator>();
serviceCollection.AddHostedService<PromotionConsumer>();
serviceCollection.AddControllers();
serviceCollection.AddEndpointsApiExplorer();
builder.Configuration.AddConfiguration(configuration);
builder.Services.AddSwaggerGen();

var app = builder.Build();
app.MapControllers();
app.UseSwagger();
```

```
app.UseSwaggerUI(options =>
{
    options.SwaggerEndpoint("/swagger/v1/swagger.json", "v1");
    options.RoutePrefix = string.Empty;
});

RunMigrations(app);
app.Run("http://localhost:6054");


static void RunMigrations(IHost host)
{
    using var scope = host.Services.CreateScope();
    var scopedServiceProvider = scope.ServiceProvider;
    var                                logger                                =
scopedServiceProvider.GetRequiredService<ILogger<Program>>();

    try
    {
        var                                context                                =
scopedServiceProvider.GetRequiredService<LocationAnalyzerContext>();
        context.Database.Migrate();
        logger.LogInformation("Database  migrations  has  been  successfully
run");
    }
    catch (Exception e)
    {
        logger.LogError(e, "Migrations failed");
        throw;
    }
}
```

Сервіс оцінки житла

PromotionConsumer.cs

```
using System.Text.Json;
using Confluent.Kafka;
using PropertyAnalyzer.Domain;
using PropertyAnalyzer.Evaluators;
using PropertyAnalyzer.Persistance.Repository;
using PropertyAnalyzer.Services;
using PropertyAnalyzer.Settings;
using PropertyAnalyzer.Trainers;

namespace PropertyAnalyzer.ConsumerHandlers;

public class PromotionConsumer : BackgroundService
{
    private readonly KafkaSettings _kafkaSettings;
    private readonly ILogger<PromotionConsumer> _logger;
    private readonly IServiceScopeFactory _serviceScopeFactory;

    public PromotionConsumer(
```

```
        KafkaSettings kafkaSettings,
        ILogger<PromotionConsumer> logger,
        IServiceScopeFactory serviceScopeFactory)
    {
        _kafkaSettings = kafkaSettings;
        _logger = logger;
        _serviceScopeFactory = serviceScopeFactory;
    }

    protected override async Task ExecuteAsync(CancellationToken
stoppingToken)
    {
        var config = new ConsumerConfig
        {
            BootstrapServers = _kafkaSettings.BootstrapServers,
            GroupId = _kafkaSettings.Group,
            AutoOffsetReset = AutoOffsetReset.Earliest
        };
        using var consumer = new ConsumerBuilder<Ignore,
string>(config).Build();
        consumer.Subscribe(_kafkaSettings.PromotionConsumerSettings.Topic);
        while (!stoppingToken.IsCancellationRequested)
        {
            try
            {
                var consumeResult = consumer.Consume(stoppingToken);
                var promotionData =
JsonSerializer.Deserialize<PropertyPromotionData>(consumeResult.Message.Val
ue)!;
                await Task.WhenAll(
                    TrainAsync(promotionData),
                    PredictAndSaveReportAsync(promotionData));
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, ex.Message);
            }
        }
    }

    private async Task PredictAndSaveReportAsync(PropertyPromotionData
promotionData)
    {
        using var scope = _serviceScopeFactory.CreateScope();
        var result = await
scope.ServiceProvider.GetRequiredService<IEvaluator>().PredictAsync(promoti
onData);
        await
scope.ServiceProvider.GetRequiredService<IReportRepository>().UpsertAsync(r
esult, CancellationToken.None);
    }

    private async Task TrainAsync(PropertyPromotionData promotionData)
    {
```

```
        using var scope = _serviceScopeFactory.CreateScope();
        var property = await
scope.ServiceProvider.GetRequiredService<IPropertyService>().EnrichAndSaveA
sync(promotionData, CancellationToken.None);
        var properties = await
scope.ServiceProvider.GetRequiredService<IPropertyRepository>().GetAllAsync
();
        await
scope.ServiceProvider.GetRequiredService<ITrainer>().Train(properties.Conca
t(new[] { property }).ToArray());
    }
}
```

PropertyEvaluationController.cs

```
using Microsoft.AspNetCore.Mvc;
using PropertyAnalyzer.Evaluators;

namespace PropertyAnalyzer.Controllers;

[Route("/api/v1/property")]
public class PropertyEvaluationController : ControllerBase
{
    private readonly IEvaluator _evaluator;

    public PropertyEvaluationController(IEvaluator evaluator)
    {
        _evaluator = evaluator;
    }

    [HttpPost("{id}/estimate")]
    public async Task<IActionResult> EstimateLocation([FromRoute(Name =
"id")] int externalId)
    {
        var result = await _evaluator.PredictAsync(externalId);
        return new OkObjectResult(new
        {
            estimation = result.Estimation,
            published_price = result.RealPrice,
            location_price = result.LocationPrice,
            deviation = result.Deviation,
        });
    }


    [HttpPost("evaluation")]
    public async Task<IActionResult> EvaludateLocation([FromBody]int[]
externalIds)
    {
        var evaluationResult = await _evaluator.EvaluateAsync(externalIds);
        return new OkObjectResult(new
        {
            r_squared = evaluationResult.RSquared,
            root_mean_squared_error = evaluationResult.RootMeanSquaredError
```

```
        });
    }
}
```

Evaluator.cs

```
using System.Text.Json;
using Dapper;
using Microsoft.ML;
using Microsoft.ML.Data;
using Npgsql;
using PropertyAnalyzer.Converters;
using PropertyAnalyzer.Domain;
using PropertyAnalyzer.Persistance.Repository;
using PropertyAnalyzer.Services;
using PropertyAnalyzer.Settings;
using PropertyAnalyzer.Trainers;

namespace PropertyAnalyzer.Evaluators;

public class Evaluator : IEvaluator
{
    private readonly IMLModelRepository _mlModelRepository;
    private readonly IPropertyService _propertyService;
    private readonly ConnectionStrings _connectionStrings;

    public Evaluator(IMLModelRepository mlModelRepository, IPropertyService
propertyService, ConnectionStrings connectionStrings)
    {
        _mlModelRepository = mlModelRepository;
        _propertyService = propertyService;
        _connectionStrings = connectionStrings;
    }

    public async Task<EstimationResult> PredictAsync(int externalId)
    {
        var      property      =      (await      GetPropertiesAsync(new[]
{externalId})).First();
        return await PredictAsync(property);
    }

    public async Task<EstimationResult> PredictAsync(PropertyPromotionData
propertyPromotionData)
    {
        var                 property                 =                 await
_propertyService.EnrichAsync(propertyPromotionData,
CancellationToken.None);
        return await PredictAsync(property);
    }

    public async Task<(double RSquared, double RootMeanSquaredError)>
EvaluateAsync(int[] externalIds)
    {
        var properties = (await GetPropertiesAsync(externalIds)).ToArray();
```

```
        var trainerData = properties.Select(p => p.ToLocationTrainerData()).ToArray();
        var mlModel = await _mlModelRepository.GetLastVersionAsync();
        using var memoryStream = new MemoryStream(mlModel);
        MLContext mlContext = new MLContext(0);
        var transformer = mlContext.Model.Load(memoryStream, out var schema);
        IDataView dataView = mlContext.Data.LoadFromEnumerable(trainerData);

        var predictions = transformer.Transform(dataView);
        var metrics = mlContext.Regression.Evaluate(predictions, "Label", "Score");
        return (metrics.RSquared, metrics.RootMeanSquaredError);
    }

    private async Task<EstimationResult> PredictAsync(Property property)
    {
        var predicateInput = property.ToLocationTrainerData();
        predicateInput.Price = 0;
        var mlModel = await _mlModelRepository.GetLastVersionAsync();

        using var memoryStream = new MemoryStream(mlModel);
        MLContext mlContext = new MLContext(0);
        var transformer = mlContext.Model.Load(memoryStream, out var schema);
        var predictionFunction = mlContext.Model.CreatePredictionEngine<PropertyTrainerData, PropertyPrediction>(transformer);
        var predictedPrice = predictionFunction.Predict(predicateInput).Price;
        return new EstimationResult(
            property.ExtraId,
            predictedPrice,
            property.Price,
            property.LocationPrice,
            property.Price / (decimal)predictedPrice);
    }

    private async Task<Property[]> GetPropertiesAsync(int[] externalIds)
    {
        await using var connection = new NpgsqlConnection(_connectionStrings.DimRiaContext);
        var promotions = (await connection.QueryAsync<string>("SELECT data FROM property_promotions WHERE external_id = ANY(@externalIds)", new { externalIds })).ToArray();

        var list = new List<Property>();
        foreach (var promotion in promotions)
        {
            var data = JsonSerializer.Deserialize<PropertyPromotionData>(promotion)!;
            var property = await _propertyService.EnrichAsync(data, CancellationToken.None);
            list.Add(property);
        }
```

```
        return list.ToArray();
    }
}

public class PropertyPrediction
{
    [ColumnName("Score")]
    public float Price;
}
```

PropertyAnalyzerContext.cs

```
using LocationAnalyzer.Persistance.Entities;
using Microsoft.EntityFrameworkCore;
using PropertyAnalyzer.Persistance.Entities;

namespace PropertyAnalyzer.Persistance;

public class PropertyAnalyzerContext : DbContext
{
    public
PropertyAnalyzerContext(DbContextOptions<PropertyAnalyzerContext> options)
        : base(options)
    {
    }

    public DbSet<PropertyDal> Properties { get; set; }

    public DbSet<TrainModelDal> TrainModels { get; set; }

    public DbSet<ReportDal> Reports { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<PropertyDal>()
            .Property(x => x.ExtraId)
            .ValueGeneratedNever();
        modelBuilder.Entity<TrainModelDal>()
            .Property(x => x.Id)
            .ValueGeneratedNever();
        modelBuilder.Entity<ReportDal>()
            .Property(x => x.ExternalId)
            .ValueGeneratedNever();
    }
}
```

PropertyService.cs

```
using PropertyAnalyzer.Domain;
using PropertyAnalyzer.Dto;
using PropertyAnalyzer.Gateways;
using PropertyAnalyzer.Persistance.Repository;

namespace PropertyAnalyzer.Services;
```

```
public class PropertyService : IPropertyService
{
    private const int UA_CODE = 380;
    private readonly ILocationEstimateGateway _locationEstimateGateway;
    private readonly IPropertyRepository _propertyRepository;

    public PropertyService(
        ILocationEstimateGateway locationEstimateGateway,
        IPropertyRepository propertyRepository)
    {
        _locationEstimateGateway = locationEstimateGateway;
        _propertyRepository = propertyRepository;
    }

    public async Task<Property> EnrichAndSaveAsync(PropertyPromotionData
promotionData, CancellationToken cancellationToken)
    {
        var property = await EnrichAsync(promotionData, cancellationToken);
        await     _propertyRepository.UpsertRangeAsync(new[]     {property},
cancellationToken);
        return property;
    }

    public    async    Task<Property>    EnrichAsync(PropertyPromotionData
promotionData, CancellationToken cancellationToken)
    {
        var                 property                 =                 await
_propertyRepository.FindAsync(promotionData.RealtyId, cancellationToken);

        if (property is not null)
            return property;

        var             locationEstimation             =             await
_locationEstimateGateway.EstimateAsync(new LocationEstimateInput()
        {
            CountryId = UA_CODE,
            CityId = promotionData.CityId,
            Longitude = (float) promotionData.Longitude,
            DistrictId = promotionData.DistrictId,
            Latitude = (float) promotionData.Latitude,
            StateId = promotionData.StateId,
        });
        var tags = promotionData.TagUk?.Select(t => t.TagId).ToArray() ??
Array.Empty<string>();
        property = new Property(
            promotionData.RealtyId,
            promotionData.WallType,
            promotionData.FloorsCount,
            promotionData.Floor,
            promotionData.RoomsCount,
            promotionData.TotalSquareMeters,
            promotionData.LivingSquareMeters,
            promotionData.KitchenSquareMeters,
            GetRepairType(tags),
```

```
            tags.Contains("С-мебелью"),
            (decimal)locationEstimation.Estimate,
            (decimal)(promotionData.CurrencyType switch
            {
                "грн" => promotionData.PriceItem / 36.6f,
                _ => promotionData.PriceItem
            }));
        return property;
    }

    private string GetRepairType(string[] tagUk)
    {
        if (tagUk.Contains("С-дизайнерским-ремонтом"))
            return "DesignerRepairType";

        if (tagUk.Contains("С-ремонтом"))
            return "CosmeticRepairType";

        return "RoughRepairType";
    }
}
```

Trainer.cs

```
using System.Diagnostics;
using Microsoft.ML;
using PropertyAnalyzer.Converters;
using PropertyAnalyzer.Domain;
using PropertyAnalyzer.Persistance.Repository;

namespace PropertyAnalyzer.Trainers;

public class Trainer : ITrainer
{
    private readonly IMLModelRepository _mlModelRepository;

    public Trainer(IMLModelRepository mlModelRepository)
    {
        _mlModelRepository = mlModelRepository;
    }

    public async Task Train(Property[] promotionLocations)
    {
        // <Snippet3>
        MLContext mlContext = new MLContext(seed: 0);
        // </Snippet3>

        // <Snippet5>
        Stopwatch stopwatch = Stopwatch.StartNew();
        IDataView                          dataView                          =
mlContext.Data.LoadFromEnumerable(promotionLocations.Select(p             =>
p.ToLocationTrainerData()));
        var model = Train(mlContext, dataView);
        Console.WriteLine($"Train time - {stopwatch.ElapsedMilliseconds}");
```

```
        // </Snippet5>

        using var memoryStream = new MemoryStream();
        mlContext.Model.Save(model, dataView.Schema, memoryStream);
        await _mlModelRepository.InsertAsync(memoryStream.ToArray());
    }

    public static ITransformer Train(MLContext mlContext, IDataView dataView)
    {
        // <Snippet6>
        // </Snippet6>

        // <Snippet7>
        var pipeline = mlContext.Transforms.CopyColumns(outputColumnName:
"Label", inputColumnName: "Price")

.Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName:
"WallTypeEncoded", inputColumnName: "WallType"))

.Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName:
"RepairTypeEncoded", inputColumnName: "RepairType"))

.Append(mlContext.Transforms.Categorical.OneHotEncoding(outputColumnName:
"WithFurnitureEncoded", inputColumnName: "WithFurniture"))
            .Append(mlContext.Transforms.Concatenate("Features",
"WallTypeEncoded", "MaxFloor", "Floor", "RoomsCount", "TotalSquareMeters",
"LivingSquareMeters",      "KitchenSquareMeters",      "RepairTypeEncoded",
"WithFurnitureEncoded", "LocationPrice"))
            .Append(mlContext.Regression.Trainers.LightGbm());
        // </Snippet10>

        Console.WriteLine("===============  Create  and  Train  the  Model
===============");

        // <Snippet11>
        var model = pipeline.Fit(dataView);
        // </Snippet11>

        Console.WriteLine("===============       End       of       training
===============");
        Console.WriteLine();
        // <Snippet12>
        return model;
        // </Snippet12>
    }
}
```

Program.cs

```
using LocationAnalyzer.Persistance.Repository;
using Microsoft.EntityFrameworkCore;
using PropertyAnalyzer;
using PropertyAnalyzer.ConsumerHandlers;
using PropertyAnalyzer.Evaluators;
```

```
using PropertyAnalyzer.Gateways;
using PropertyAnalyzer.Persistance;
using PropertyAnalyzer.Persistance.Repository;
using PropertyAnalyzer.Services;
using PropertyAnalyzer.Trainers;

var builder = WebApplication.CreateBuilder(args);

IConfiguration configuration = new ConfigurationBuilder()
    .SetBasePath(Directory.GetCurrentDirectory())
    .AddJsonFile("appSettings.json", optional: false)
    .Build();
var serviceCollection = builder.Services;
serviceCollection.AddDatabaseContext(configuration);
serviceCollection.AddConfiguration(configuration);
serviceCollection.AddTransient<IPropertyRepository, PropertyRepository>();
serviceCollection.AddTransient<IMLModelRepository, MLModelRepository>();
serviceCollection.AddTransient<ILocationEstimateGateway,
LocationEstimateGateway>();
serviceCollection.AddTransient<IPropertyService, PropertyService>();
serviceCollection.AddTransient<ITrainer, Trainer>();
serviceCollection.AddTransient<IEvaluator, Evaluator>();
serviceCollection.AddTransient<IReportRepository, ReportRepository>();
serviceCollection.AddHostedService<PromotionConsumer>();
serviceCollection.AddControllers();
serviceCollection.AddEndpointsApiExplorer();
builder.Configuration.AddConfiguration(configuration);
builder.Services.AddSwaggerGen();

var app = builder.Build();
app.MapControllers();
app.UseSwagger();
app.UseSwaggerUI(options =>
{
    options.SwaggerEndpoint("/swagger/v1/swagger.json", "v1");
    options.RoutePrefix = string.Empty;
});

RunMigrations(app);
app.Run("http://localhost:6055");

static void RunMigrations(IHost host)
{
    using var scope = host.Services.CreateScope();
    var scopedServiceProvider = scope.ServiceProvider;
    var                                logger                                =
scopedServiceProvider.GetRequiredService<ILogger<Program>>();

    try
    {
        var                                context                                =
scopedServiceProvider.GetRequiredService<PropertyAnalyzerContext>();
        context.Database.Migrate();
```

```
        logger.LogInformation("Database migrations has been successfully
run");
    }
    catch (Exception e)
    {
        logger.LogError(e, "Migrations failed");
        throw;
    }
}
```

```
using Microsoft.ML.Data;

namespace PropertyAnalyzer.Trainers;

public class PropertyTrainerData
{
    [LoadColumn(0)]
    public float WallType { get; set; }

    [LoadColumn(1)]
    public float MaxFloor { get; set; }

    [LoadColumn(2)]
    public float Floor { get; set; }

    [LoadColumn(3)]
    public float RoomsCount { get; set; }

    [LoadColumn(4)]
    public float TotalSquareMeters { get; set; }

    [LoadColumn(5)]
    public float LivingSquareMeters { get; set; }

    [LoadColumn(6)]
    public float KitchenSquareMeters { get; set; }

    [LoadColumn(7)]
    public float RepairType { get; set; }

    [LoadColumn(8)]
    public float WithFurniture { get; set; }

    [LoadColumn(9)]
    public float LocationPrice { get; set; }

    [LoadColumn(10)]
    public float Price { get; set; }
}
```

ДОДАТОК Г. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

Розробка методів і програмних засобів підвищення якості експертної багатокритеріальної оцінки житлової нерухомості

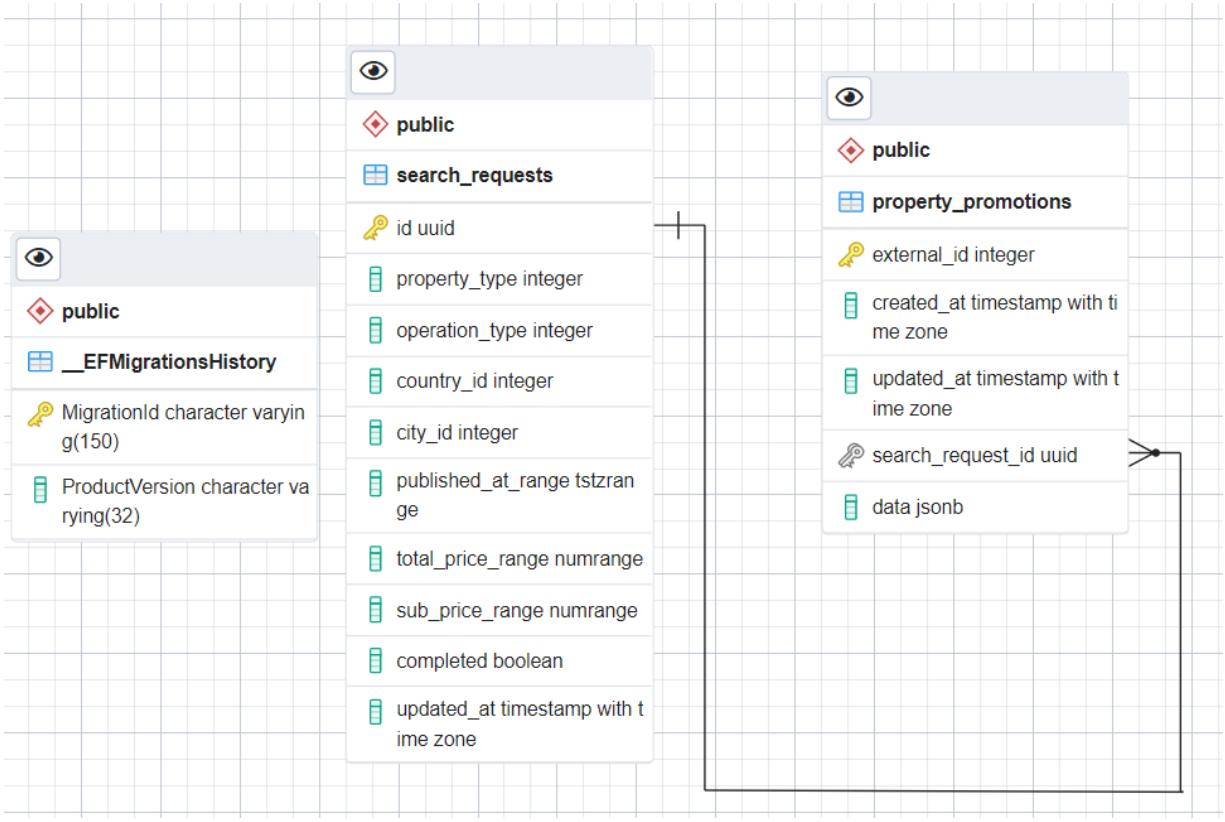

Рисунок Г.1 – Схема бази даних DimRiaFetcher

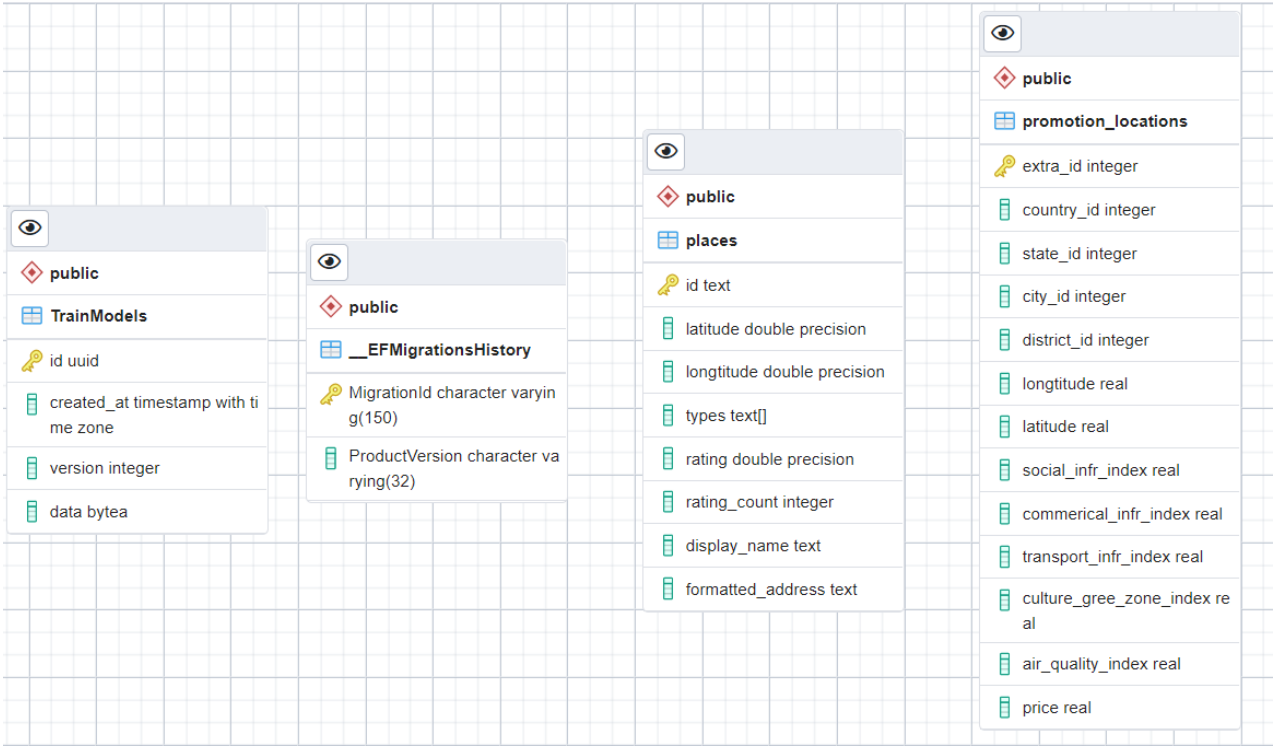

Рисунок Г.2 – Схема бази даних LocationAnalyzer

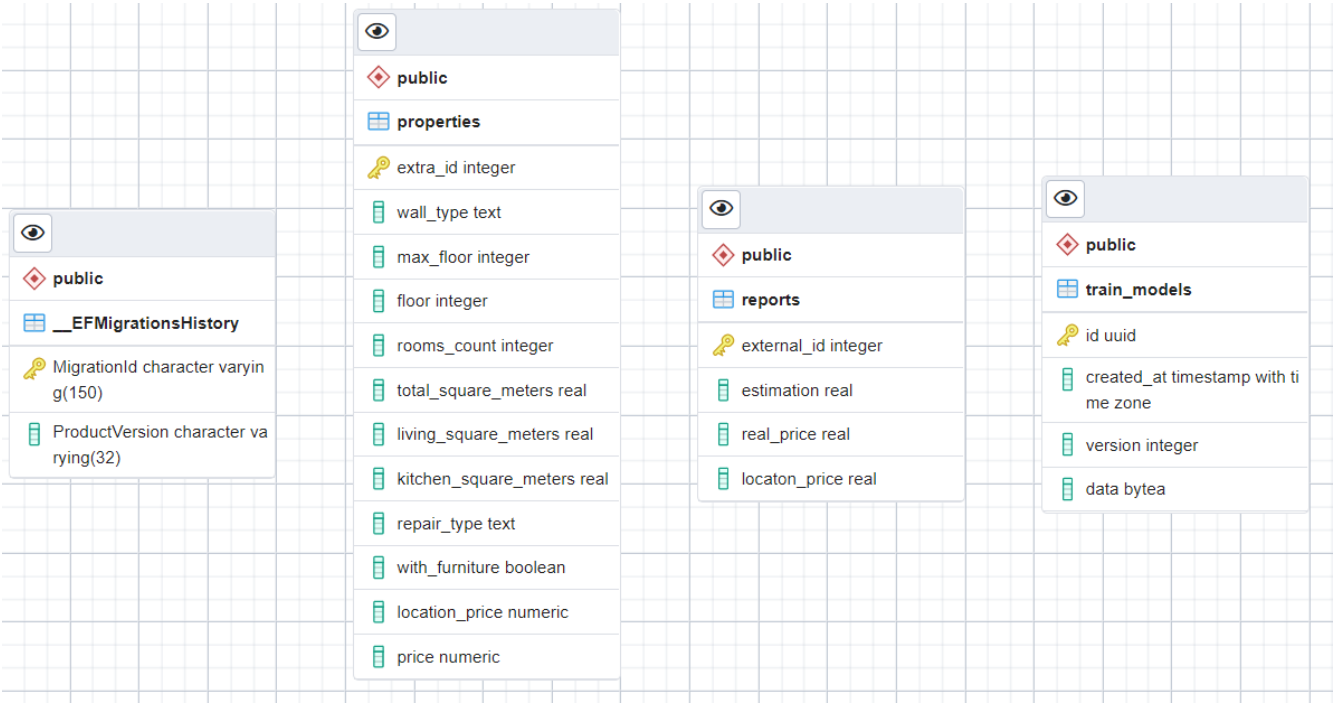

Рисунок Г.3 – Схема бази даних PropertyAnalyzer

Рисунок Г.4 – Титульний слайд

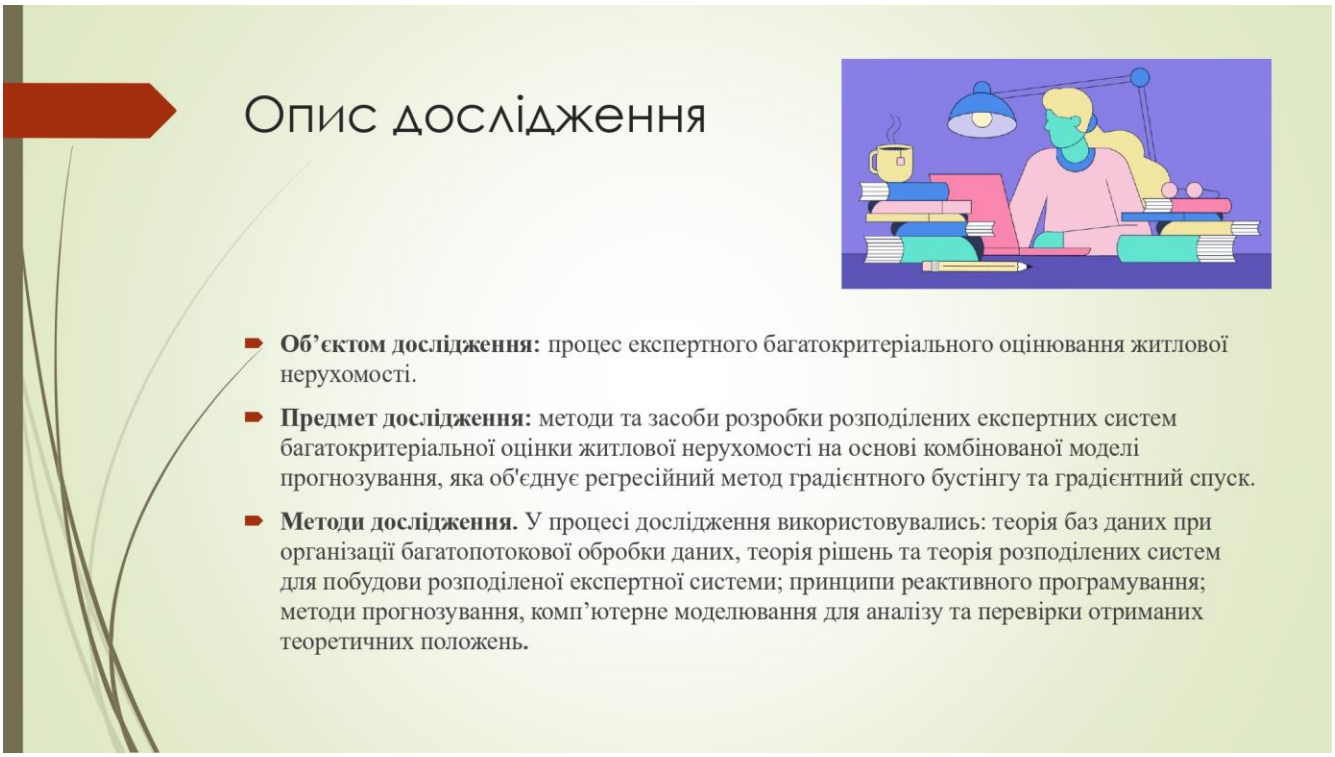

Рисунок Г.5 – Слайд 2. Опис дослідження

Рисунок Г.6 – Слайд 3. Наукова новизна

Рисунок Г.7 – Слайд 4. Модель прогнозування

Рисунок Г.8 – Слайд 5. Архітектура системи

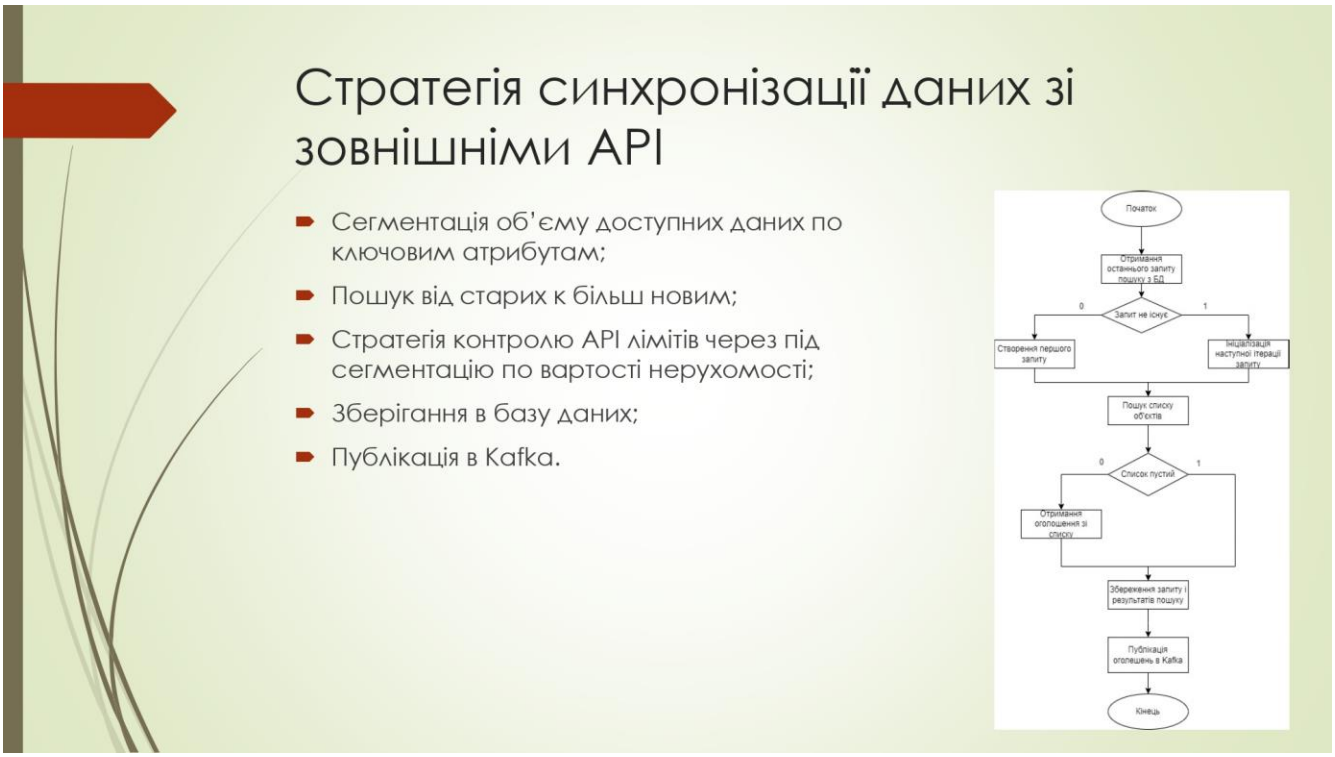

Рисунок Г.9 – Слайд 6.Стратегія синхронізації даних зі зовнішніми API

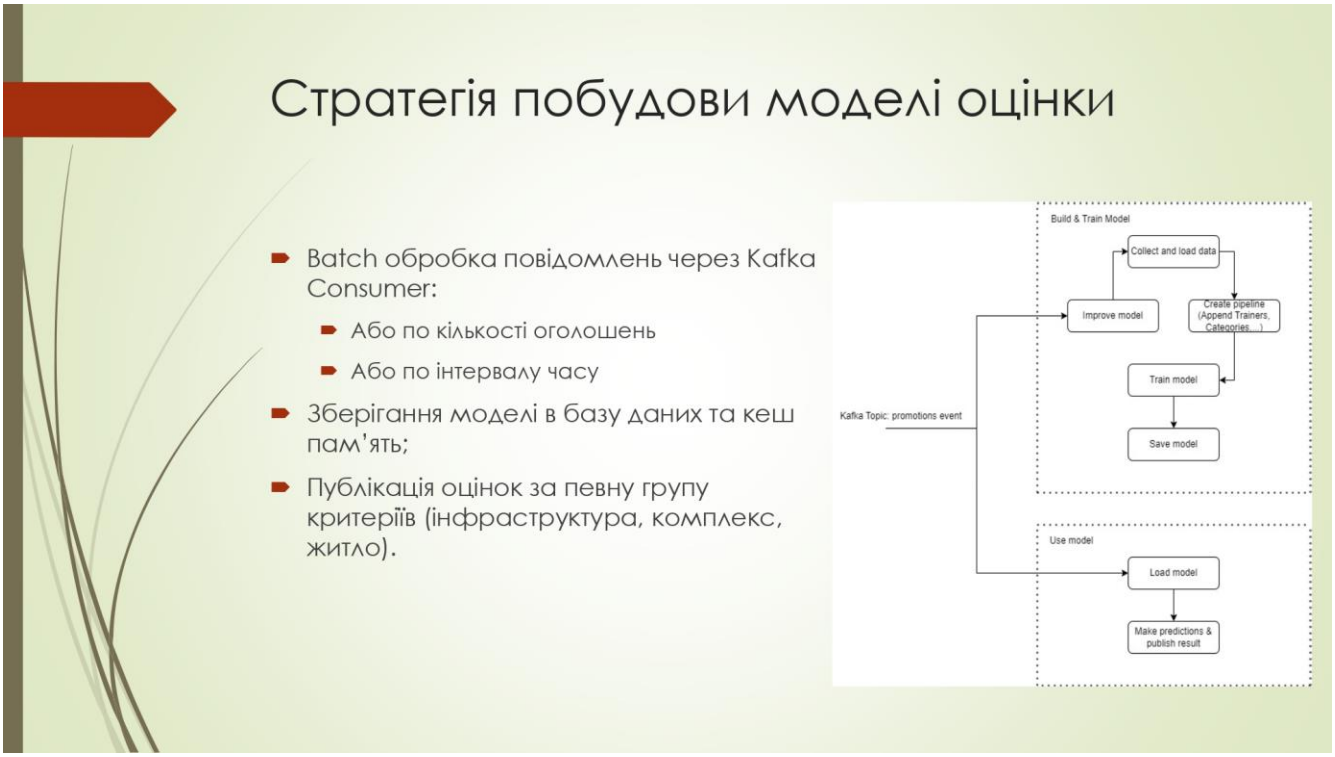

Рисунок Г.10 – Слайд 7. Стратегія побудови моделі оцінки

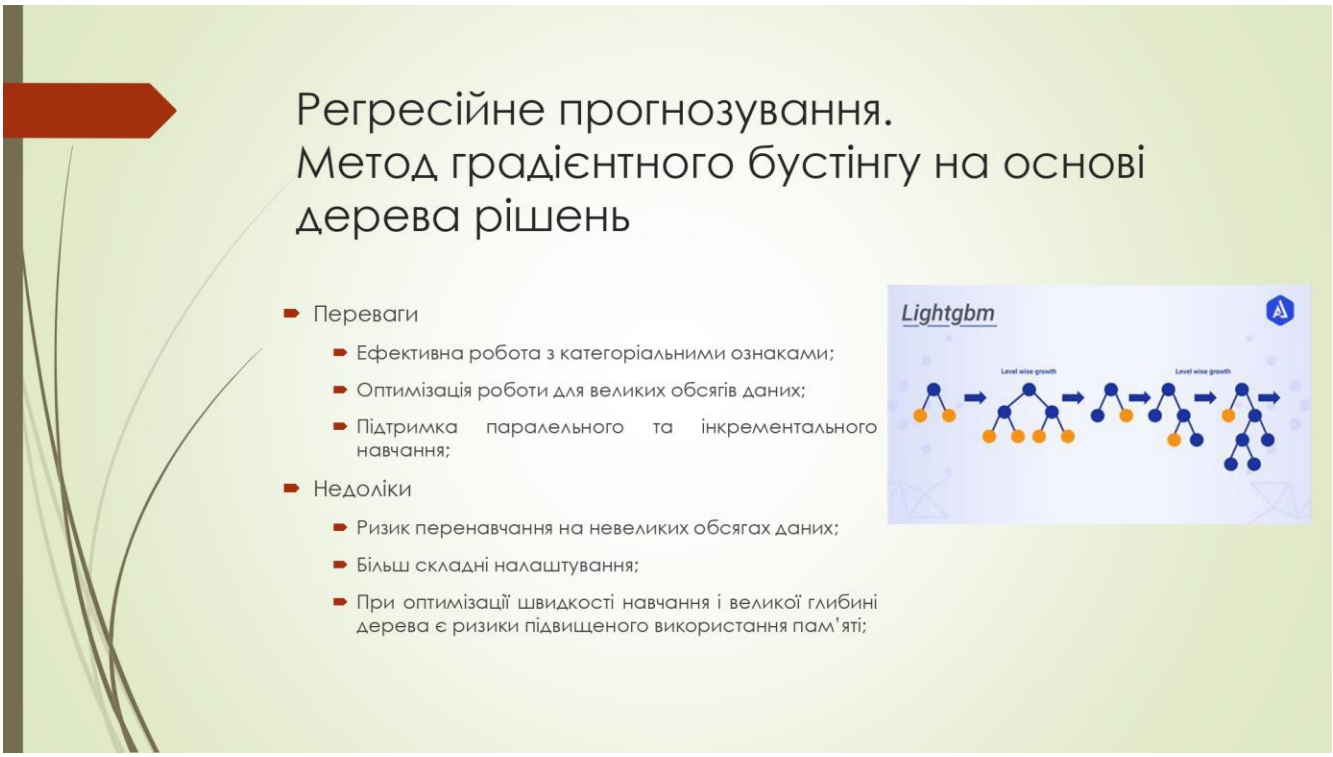

Рисунок Г.11 – Слайд 8. Метод градієнтного бустінгу на основі дерева рішень

Рисунок Г.12 – Слайд 9. Метод градієнтного спуску

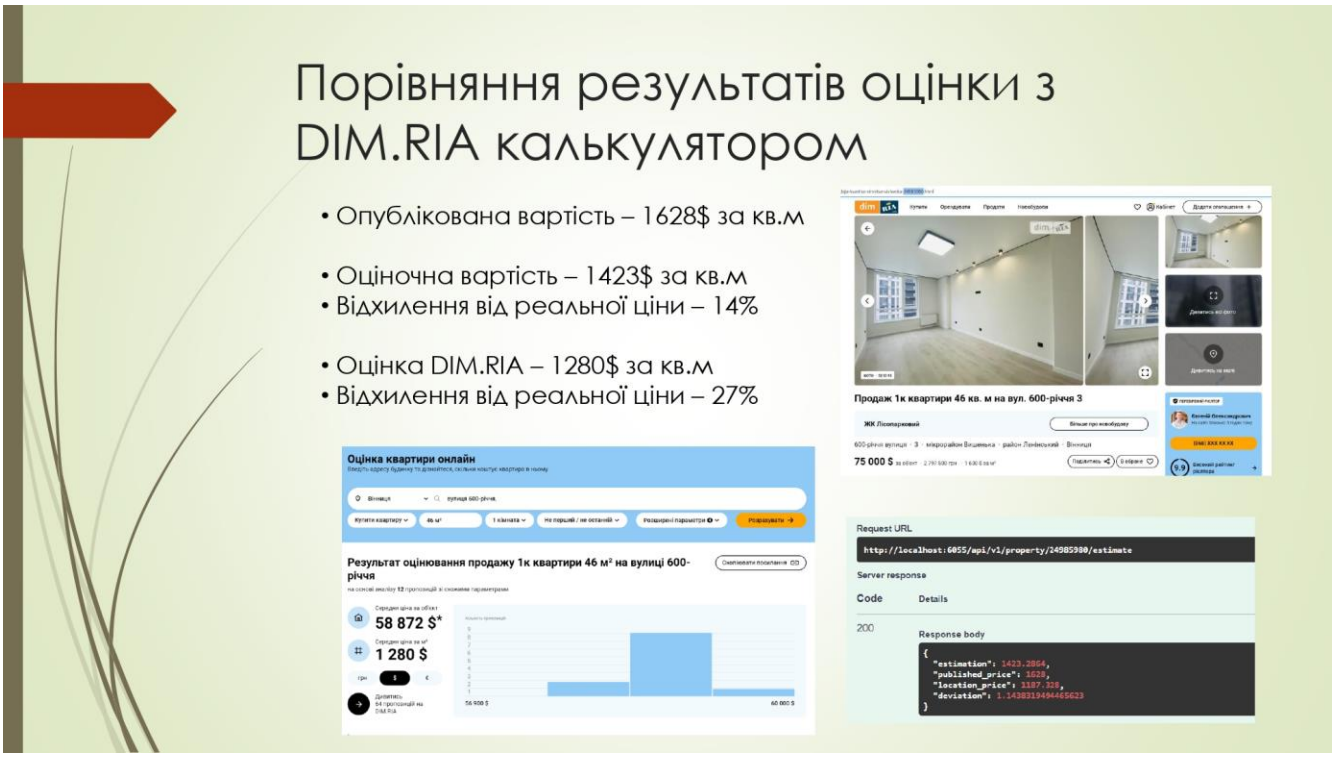

Рисунок Г.13 – Слайд 10. Порівняння результатів оцінки з DIM.RIA калькулятором

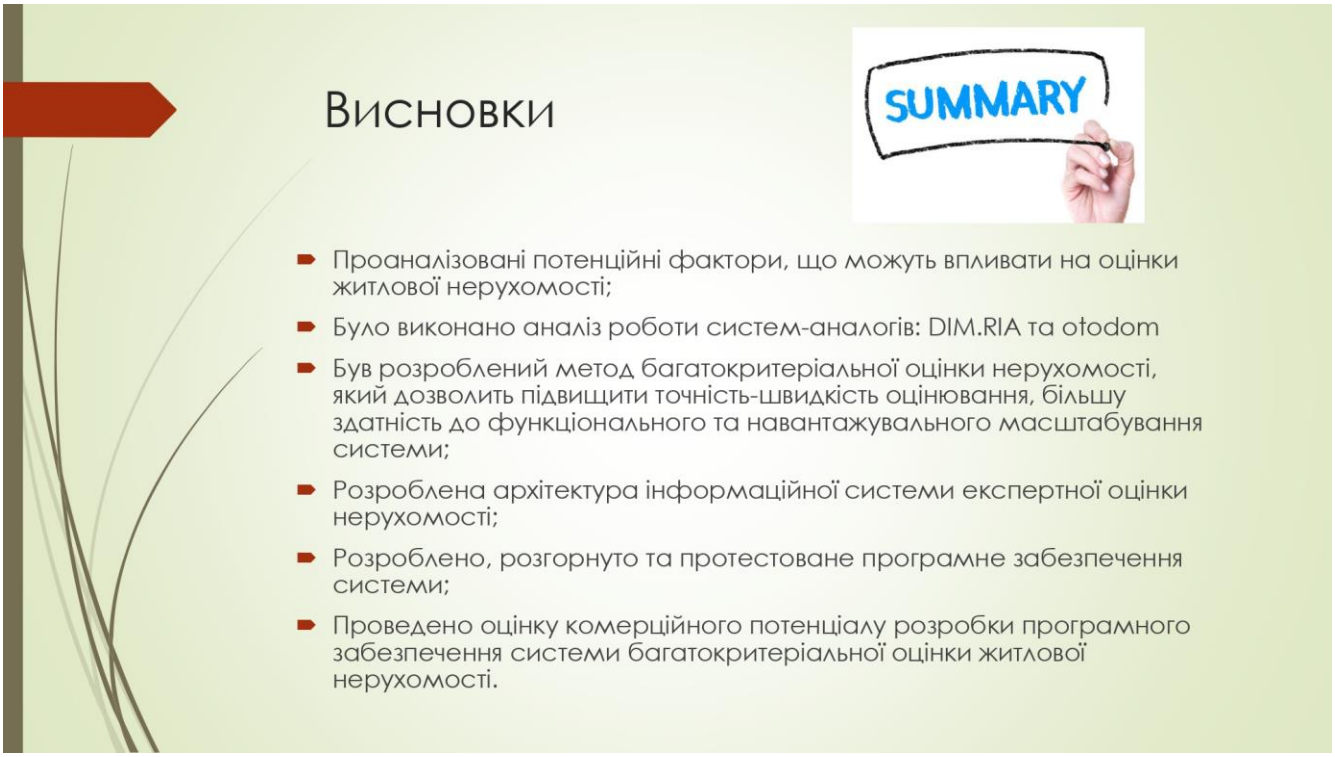

Рисунок Г.14 – Слайд 11. Висновки

Рисунок Г.15 – Слайд 12. Використані програмні засоби

Рисунок Г.16 – Слайд 13. Питання та відповіді